National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre reference

Our file Notre reference

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.

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.

Canada

Generation Of Grey Box Use Cases For Personal Communication Systems using Simple Design Machines

By

Tarun Shah

THESIS

Submitted to the School of Graduate Studies and Research
in Partial Fulfillment of the Requirements for the

Degree of

Master in Computer Science

Department of Computer Science - University of Ottawa
(Ottawa-Carleton Institute for Computer Science)

ISBN 0-612-07876-0

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ABSTRACT

The advantages in representing high level user service scenarios by means of a formal *testable* design notation are *several.* Formalized scenarios provide for a more extensive test coverage of designs both from the user perspective and the system perspective when compared to a non formal method such as one employing natural language. Verifying completeness of design by scenario checking is targeted towards catching design errors. We present guidelines for constructing test cases and scenarios which simplify the process and disambiguate the specification. This thesis defines a formal testable design notation called Simple Design Machines(SDM's) (simplification of Design Machines); which lends itself to the semi-automated generation of grey-box use cases. These grey box use cases, when combined with white and black box test cases, provide for a more comprehensive test suite. Thus, SDM's support design for Software Testability. *Personal Communication Systems* is the application used to validate SDM's. In this thesis, first the need for a testable design language is demonstrated, then different existing testing techniques are reviewed, followed by an overview of the proposed new language (SDM's) and the application used for validation (PCS). SDM's are effective in situations where the system responds to combinations of inputs.

Examples illustrating how to construct SDM's for simple applications are then given. We subsequently define a mapping from PCS communication systems to SDM's, walk through the generation of grey box use cases, validate these use cases and measure coverage for the SDM designs. A variety of different tools are assessed into for feasibility with respect to automatic test case generation. A prototype of the Symbolic Scenario Selector was implemented in C and tested on PCS. Traces are generated for the SDM's and coverage is measured. Inconsistencies in the specification can be detected as the PCS example demonstrated. Finally, we assess advantages and limitations of the approach based on the case study and other experiences.

Keywords : use cases, personal communication systems, simple design machines, state

ACKNOWLEDGMENTS

I wish to acknowledge, with gratitude, the advice and help I have received in preparing this study. I am particularly grateful to my supervisor, Dr. Probert, who helped me choose the thesis topic and provided guidance, support and encouragement during the research.

I would also like to thank Dr. Wali Sabah for providing me with valuable information on Personal Communication Systems, Wireless communications, Intelligent Networks and referring me to more literature on these subjects.

I am also thankful to Michel Racine for providing me lab support and Douglas King for aiding me in the construction of a Symbolic Scenario Selector prototype, especially with information on debugging C Code using dbxtool.

I gratefully acknowledge advice given by Nursumulu Khanaidoo and Cecelia Geldrez with regards to Cause Effect Graphing and Design Machines respectively.

Finally, I wish to thank my parents, brother and sister for giving me the support and opportunity to come to Canada.

TABLE OF CONTENTS

Appendix A : PCS Specification in Mondel
A.1 Explanation of Mondel and Objects
A.2 PCS Specification

Appendix B : Mapping to SDM's
A.1 Explanation of Steps followed
A.2 Mapping to SDM's

Appendix C : Output Runs for Tool
C.1 Explanation of the Tool
C.2 Output Runs

Appendix D : Code Listings for SSS Tool
D.1 Explanation of how code was generated
D.2 Code Listings

LIST OF FIGURES

Chapter1

Introduction

1.1 Motivation For Development of Simple Design Machines (SDM's)

The term "systems analysis" applies to a number of methodological approaches that guide the developer in the identification of system requirements and phases of the development life cycle. The designer is typically concerned with questions such as *"What does the system do? What are the system's output and input? What data must the system store and maintain? What are the important properties of high level system constructs? If the system is "real time", what are the events to which the system must be responsive and what are the associated real time constraints for a response?*

While all system components are of three major types namely *data*, *activity* and *control* components, different types of systems will place different emphasis on each. Designers first focus attention on the system at its "highest level" and then progressively decompose it into more detailed levels until they reach the bottom-most level. When the designer has completed this top - down , layer-by-layer analysis of the system, a hierarchy of system activities in which no activity has more than a single parent is created. [2].

Hatley and Pirbhai[1] also suggest a top down, hierarchical approach. On the other hand, Ward and Mellor[37] suggest that analysis should begin at whatever level the system interacts with the *external environment.* The method of *Simple Design Machines (SDM)* proposed in this thesis supports the hierarchical top down approach, as well as the iterative design approach.

One might argue that there are several specification/design languages available; therefore why do we need SDM's?
Some reasons include:

i) SDM's are used when we have a system that responds to a ***complex combination of conditions and stimuli.*** Such systems are difficult to describe using finite state machines and even more difficult to describe in natural language[38]. This point will be demonstrated in the next chapter.

ii) SDM's are designed to be used specifically for the generation of additional ***grey box use cases***. These are use cases created in the design phase of the life cycle process and are explained in detail in Chapter 2. These are translated into executable use cases and added to white box and black box use cases, and thus help to improve the coverage of the system test suite.

iii) One reason why only a small percentage of industry uses a formal specification method as opposed to natural language is that most specification languages are difficult, complex and hard to read when applied to a realistic system. As Yourdan[40] notes, many development projects were abandoned because it took too long to build a model of a system that would soon be obsolete. SDM's, when kept simple enough are quick and easy to read, analyze and implement.

iv) Grey box use cases (traces) generated from SDM's at design validation time can be used to validate both the design as well as the implemented product. Designers can verify that their system operates according to the ***traces or scenarios*** generated at design time.

v) A *testable* design is one that lends itself to the automatic generation of use cases. We claim and argue in section 4.3 that SDM's when used appropriately provide testable designs.

1.2 Contributions of the Thesis

This thesis defines a formal testable design language called Simple Design Machines(SDM's) (an extension of Design Machines) which lends itself to the semi-automated generation of grey-box use cases and is shown to support design for Software Testability. A survey of other formal specification approaches is performed and an assessment of SDM by direct comparison is done.

An implementation of a support tool , which contains the ability to automatically generate traces is shown. Use of existing commercial tools to support SDM Methodology are explained. In later chapters we also talk about how to use SDM's to support Grey Box Analysis and design inspections. A specific design and test strategy for SDM's which includes fast prototyping and iterative development is developed.

As a case study, we applied this testable design language (SDM) to a medium - sized, realistic application(***Personal Communication Systems)*** and generate grey box use cases from the resulting design. SDM's appear to be particularly well suited to such flexible telecommunication systems. Some defects were found in the Personal Communication System Specification by using the SDM approach.

1.3 Outline of the Thesis

Chapter 2 presents an overview of Design Languages and Methods. We highlight strengths and weaknesses of each approach. Different testing techniques are looked at and the test coverage criteria's are measured.

Chapter 3 presents an overview of Design Machines(DM's) and Simple Design Machines (an extension of DM's). These concepts are applied to a three way call. An introduction to Personal Communication Systems (PCS) is also presented.

Chapter 4 shows the mapping of a natural language specification of Personal Communication Systems to Simple Design Machines. First the PCS specification in natural language and in an object oriented specification language (Mondel) is explained. The mapping methodology to SDM's is then presented. A few simple examples are shown. The details of the mapping process are given in appendices A-D. Grey box traces are generated and the coverage is measured.

Chapter 5 deals with commercial tools for automated mapping and gives a detailed example. Requirements for such a tool are examined. A prototype of a SDM interpreter and analyzer is implemented in C. Mapping of SDM's to SDL(Specification and Description Language) is then explained.

Appendices A-D show the entire mapping process applied to Personal Communication Systems : The PCS Specification in Mondel, conversion to SDM's, generation of grey box traces, application of the SDM tool, and inconsistencies in the PCS specification which were detected with the help of the SDM tool.

Chapter 2

Background and basic definitions

2.1 Basic Context

2.1.1 Software Development Process

Software Development is the application of scientific principles to the orderly transformation of a problem into a working software solution and the subsequent maintenance of that software through the end of its useful life. Royce[35] was the first to coin the phrase *waterfall model* to characterize the series of software development phases.

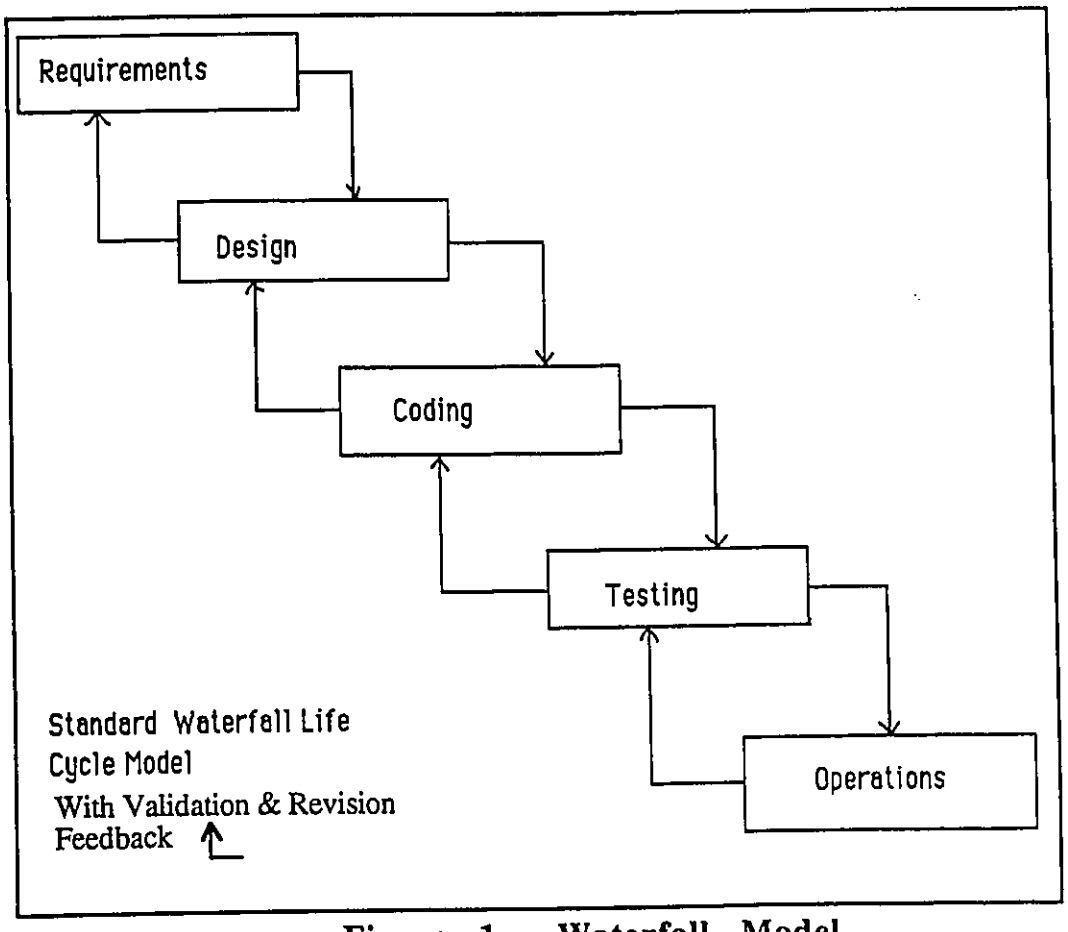


Figure 1 Waterfall Model

The model consists of the following stages:

1. ***Capture Software Requirements*** - This includes analysis of the software problem at hand and concludes with a complete specification of the expected external behavior of the software system to be built.

2. ***Construct Preliminary Design*** - The activity that decomposes the software system into its actual constituent components and then iteratively decomposes these components into successively smaller components.

3. ***Derive Detailed Design*** - The activity of defining and documenting algorithms for each component that will be realized as code.

4.***Develop Code***- The activity of transforming the algorithms defined during the detailed design stage into a computer understandable language.

5. ***Conduct Unit Testing*** - The activity of checking each coded module for the presence of errors.

6. ***Conduct Integration Testing*** - The activity that interconnects sets of previously tested modules to ensure that the sets behave as well as their independently tested module components did.

7. ***Carry out System Testing*** - The activity of checking that the entire software system (i.e.. fully integrated) embedded in its actual hardware environment behaves according to the specification.

Various terms are used to denote the writing of a specification [15]. These are requirements [3], specification[13], system design[36], functional specification[7], external design[39], software requirements specification[23] and input/output perspective[25]. Errors in Specifications are common [5], and are often not detected until after product delivery, and cost far more to fix than if detected earlier[7,15,18]. Therefore it is of extreme importance to find ways to reduce the most common types of errors in specifications : incorrect facts, design omissions, inconsistencies and ambiguities. This is applicable for other process models too, such as fast prototyping, object oriented, spiral or iterative development.

The use of formal methods to specify a product's external behavior reduces the number of errors that may remain latent in the Specification because the associated techniques and tools can alert the specification writers to potential errors.

However, most formal methods are not yet widely accepted. Some simple, yet formal approaches, such as SDM's may be a accepted as a bridge to formalize the design process, a few steps at a time. Most requirement specifications today are written in natural language and range in length from a few pages to thousands of pages.
The size of the document rarely has any relationship to the complexity of the problem. The ***larger documents*** are usually created by the organizations that are attempting to achieve completeness. However, the larger the document becomes, the more difficult consistency and maintenance become. The ideal solution to this would be to use a formal method. ***Verification*** is the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase.***Validation*** is the process of evaluating software at the end of the software development process to ensure compliance with software requirements. It can also be thought of as assessing whether or not we are building the right product.

2.1.2 A Survey of Current Testing Techniques

This section introduces several different testing techniques. Different specification / design methodologies lend themselves to different testing techniques. Good references include [28], [6].

Black Box Testing (Functional Testing)

The form of testing which is based only on software requirements specifications is called "black box testing". ***Black box testing is a software test strategy*** which derives its test specifications and case data from the external specifications and requirements of the system. Methods for black box testing include nominal testing (expected values), random testing and testing at boundary values (both inside and outside the boundaries).It verifies the end results at the system level but does not check on how the system is implemented. It also does not assume that all the statements in the program are executed. A comprehensive test suite should include both black box and white box use cases.

White Box Testing

White box testing is a software test methodology at the module level in which test procedures and use cases (non executable test cases) are derived from the internal structure of the program.It may execute all the statements or branches in the program to check how the system is implemented. Some methods of white box testing are statement coverage, branch coverage and path coverage. There are two types of white box testing. These are ***Static Testing and Dynamic Testing***

In **Static Testing** we need to inspect code, analyze code, symbolically execute code (but do not execute code). This includes Code Inspections, Symbolic Execution, Data Flow Anomaly Analysis. For **Dynamic Testing** we need to exercise code in a controlled manner, to achieve a code coverage objective.

Code Coverage: Two main types of code coverage include ***Control flow coverage and Data Flow coverage.*** The coverage goals include traversing all paths, modules, exercising all pairs of modules, all variables defined and used, data structures filled then emptied, suspicious modules are also thoroughly tested. Control flow testing constructs use cases to cover all statements, branches (decisions) every outcome of each decision. Data flow tests different values for each variable. Incorrect values are also tested.

Higher degrees of white box coverage are: ***Decision/Condition coverage*** where each possible outcome of each condition in a decision and each possible outcome of each decision is covered. ***Multiple Conditions*** coverage where all possible combinations of condition outcomes in each decision are covered. Both cost of testing and degree of coverage have to be considered.

Mutation Testing

The process in mutation testing is that small (<3) changes are made in a module and then the original and mutant modules are compared. This is similar to the idea of error seeding: introduce defects and verify that use cases are comprehensive enough to "catch" the defects (called mutants). This method is more systematic - all single modifications to the code are generated - each single modification yields a new mutant.

The idea is that the test suite is adequate when it suffices to distinguish between the original and any non equivalent mutant. Mutation testing is based on the following two hypotheses:
Competent programmer hypothesis : The module under test has been written by a competent programmer or designer. Therefore, if the module is not correct, it differs from a correct one by at most a few small faults.
Coupling Effect Hypothesis : The test suite that distinguishes all modules from a correct one by only simple faults is so sensitive that it also implicitly distinguishes more complex faults.

Grey Box Design Inspection
Grey Box Design Inspection represents design as a precise sequence of inputs, decisions and actions. To create Grey box use cases we perform the following steps. ***Select*** a black box or white box test method and tailor it to the design representation.
Identify design attributes appropriate to the test method. ***Apply*** the method to generate use case requirements for design coverage. ***Solve*** the use case requirements to yield grey box use cases. ***Validate*** the use cases. Systematically ***inspect*** the design by tracing use cases and monitoring the level of coverage desired.

2.1.3 Design For Software Testability (DFST)

By investing more up-front effort in ***verifying and validating*** their software requirements and design specifications, some projects achieve reduced ***integration and test*** costs, higher software reliability and maintainability, and more user responsive software. A basic objective in Design For Software Testability (DFST) is to identify and resolve software problems and high risk issues ***early on*** in the software life cycle. The main reason for doing this is shown in the next figure.

It shows that, for large projects, ***savings of up to 100:1 are possible*** by finding and fixing problems early in the life cycle. For smaller projects, the savings are more on the order of 4-6:1, but this still provides a great deal of benefit for early investment in DFST activities.[4].

Besides the major cost savings, there are also significant payoffs in ***improved reliability, maintainability, and human engineering*** of the resulting software product. SDM's support DFST in that scenarios can be selected from the resulting SDM. ***These scenarios can be used to validate the formal specification and verify operation of the previous phase.***

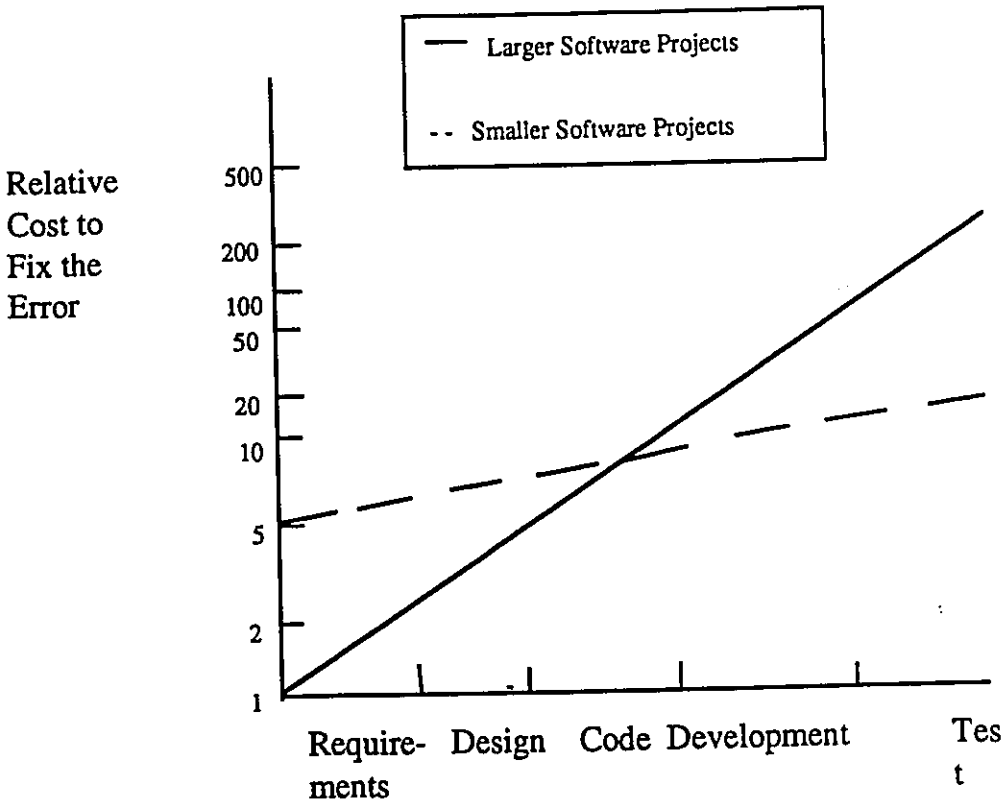


Figure 2: Increase in cost to fix or change software throughout the life cycle.

2.1.4 Criteria for Assessment of System Requirements Specification Methods and Models

When the technique is properly used, the resulting specification should be helpful and understandable to non-computer oriented customers and users. It should be able to serve effectively as the basis for detailed design and testing, provide ***automated checks for ambiguity***, incompleteness and inconsistency, encourage the requirements writer to think and write in terms of external product behavior, not internal product components. Also the technique should provide a basis for ***automated prototype generation***, test generation and be well suited to the particular application. The following criteria have been looked at to compare the various Specification Methods.

1. Accessibility to designers for tools and support
2. Testability
3. Whether or not they support automated checking
4. Whether or not they support the automatic generation of use cases.
5. Ease of learning
6. Ease of use
7. Appropriate applications each can be used for
8. Prototype generation capability
9. Ability to represent normal cases Vs exception cases
10. Organizational assistance provided

The above criteria were used by A.M Davis[2] for his comparison. The Specification Method and Models that are looked at are the following :

1. Natural Language
2. FSM (Finite State Machines)
3. PDL (Program Description Language)
4. SDL (Specification and Description Language)
5. Decision Table and Decision Trees
6. StateCharts
7. Other Case Models
8. Object Oriented Design

These were selected as they are the most common and are most related to the SDM Model, which we present in Section 2.3 and Chapter 3.

2.2 Models and Methods Currently in Use

2.2.1 Natural Language

Natural Language is simply free form English and is presently the most widely used specification language. It is easy to use and requires no prior skill. It is obviously understandable to computer naive personnel but with no formality there is no basis for design and test. No processor is available to perform any kind of checking and should be used for any application where misinterpretation of requirements is acceptable or at least non critical.A.M.Davis[2].

2.2.2 Finite State Machines

A finite state machine (FSM) is a hypothetical machine that can be in only one of a given number states at any specific time. In response to an input, the machine generates an output and changes state. Both the output and the new state are purely functions of the current state and the input. A.M Davis[2].

There are two notations commonly used to define FSMs: state transition diagrams (STD) and state transition matrices (STM). When STDs are used , a circle denotes a state, a directed arc connecting two states denotes the ***potential to transition between the two*** indicated states, and the label on the arc (which has two parts separated by a slash) denotes the input or event that triggers the transition and the output with which the system responds. The second way to describe the behavior of a finite state machine is the STM. In an STM, a table is drawn with all possible states labeling the rows and all possible stimuli labeling the columns. FSM's have been used effectively for requirements specifications for ***telephony applications***. They also serve as the underlying model for many of the techniques that follow, for example, SDL.

2.2.3 Program Design Language

Program Design Language is the standard for specifying detailed designs for software modules. PDL, also called structured English and psuedocode, is simply free-form English with special meanings for certain key words. The most commonly used PDL today is processed by a PC based tool called PDL[10].

Example:

```
1. IF elevator  moving
2. THEN loop
3. ELSE
4.  IF reached  requested  floor
5.   THEN OPEN_DOOR
6.      ELSE  Signal Alarm
7.   ENDIF
8.  IF  time  limit  exceeded
9.  THEN CLOSE DOOR
10     ELSE  OPEN DOOR
11   ENDIF
12  IF  open  door  button  pressed
13   THEN OPEN DOOR
14      ELSE  put elevator in motion
15   ENDIF
16  ENDIF
```

Normal scenarios and exception scenarios are hard to describe using PDL. For example, in the above example the order is arbitrary. Being little more than natural language, the lack of formality provides no basis for testability. Most importantly no automated analyzer can check detect semantic errors.

2.2.4 SDL (Specification and Description Language)

The ***Specification and Description Language*** (SDL) was developed by CCITT for the external behavior and internal design of telephone switching systems. The requirements of standardized OSI protocols should be specified using a formal description technique in order to assure an unambiguous description. SDL as an FDT offers this possibility, with respect to behavioral aspects. Since the conformance of implementations are determined from tests based on the specification, the requirements must be specified such that they are testable.

In order to formally reason about the external behavior of SDL systems, the behavior of an SDL system has to be defined relative to how the events it may perform are observed in the environment of the system.

With the term ***testability of an implementation***, we refer to the ability to determine by testing if the implementation satisfies the requirements expressed in its associated specification.
The semantical properties of SDL considered to affect the testability of implementations derived from SDL specifications are:

- ***Asynchronous communication***
- ***The time model***
- ***Non determinism***

In SDL , the communication of signals between processes in the system and the environment are performed asynchronously. An SDL system may also internally discard signals sent to it.

SDL is used in the analysis and specification of functionality of systems. This is done by making SDL systems that are models of existing or forthcoming real world systems. SDL systems are specified in SDL system specifications. Reed[34].

As part of the modeling process, components of the real world systems are identified and selected properties are described. These components are modeled by instances as parts of the SDL system. The classification of components into categories and subcategories is in the SDL system represented by types and subtypes of instances.

Limitations of SDL include :
1. With respect to requirement capture, other languages may be more suitable due to the delimiting behavior of SDL. SDL is not intended for describing general parameters of systems, like capacity and weight.
2. SDL contains no specific construct for user communication. A System whose main purpose is to maintain an advanced user interface is highly reactive, but may be more appropriately specified in languages with direct connections to advanced window management facilities, advanced file handling capabilities and so on.

3. SDL's are quite difficult to learn and cannot adequately describe normal use cases Vs exception cases which is partially due to complicated notations and semantics.

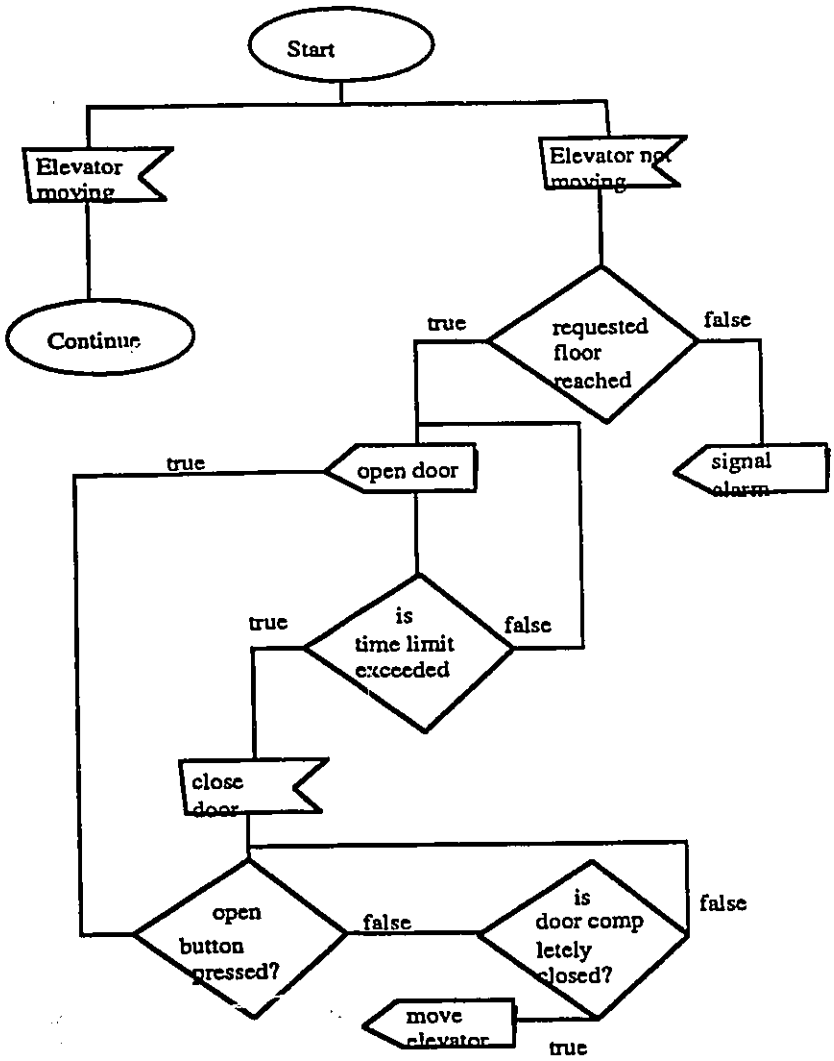


FIGURE 3 Simplified SDL Representation of Elevator

2.2.5 Decision Tables and Decision Trees

Sometimes there is a need to describe the required external behavior of some aspect of a system when the FSM approach makes no sense. One simple solution is the decision table. Their use and capabilities were recently explored by Chvalovsky[14] and Moret[30].

One column for every combination of condition outcomes

List all conditions that influence decisions

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Condition 1	y	y	y	y	n
Condition 2	y	y	n	n	y
Condition 3	y	n	y	n	y
Action 1	x	x	x		
Action 2		x	x	x	
Action 3					
Action 4	x			x	x

All combination of answers

List all possible decisions

Fill in with the actual required system behavior

Figure 4 Decision Table

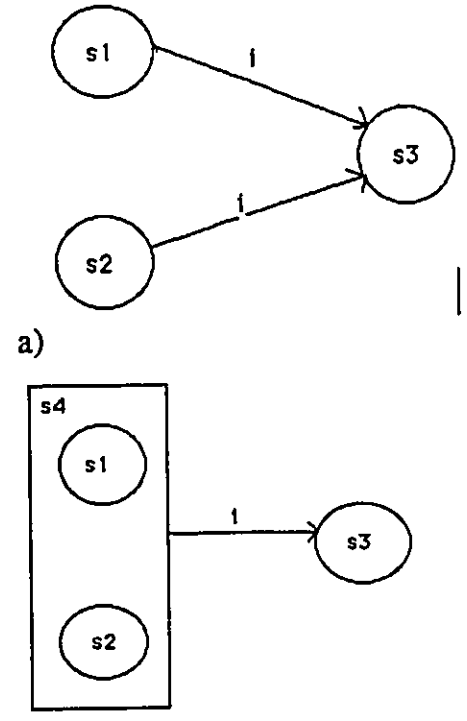
To construct a decision table, first draw a row for each condition that will be used in the process of making a decision. Next draw a column for every possible combination of outcomes of those conditions (if there are n conditions and each has a binary result, then there will be 2^n columns. Thus decision tables can become awkward and unwieldy even for small values of n). Finally, add rows at the bottom of the table for each action (or response) that you may want the system to perform (or generate), and fill in the boxes to reflect which actions you want performed for each combination of conditions.

A decision tree captures the same information as a decision table but is graphical rather than tabular. Basically it is a flow chart without loops and without fan-in (i.e. two arrows pointing to the same node).

The decision tree often takes up more space than the corresponding decision table not because of the blank space required between the nodes, but because the decision tree captures the order of ***evaluating conditions,*** it is possible to save considerable space on some problems due to the all-encompassing effect of some conditions. Decision tables can be easily be automated with any s***tandard spreadsheet package.***

2.2.6 Statecharts

Statecharts, extensions to Petri nets (a version of finite state machines) were proposed by Harel[21]. Statecharts make it easier to model complex real time system behavior without ambiguity. The extensions provide a notation and a set of conventions that facilitate the ***hierarchical decomposition*** of FSMs and a mechanism for communication between concurrent FSMs.



a)

b)

FIGURE 5 : The Superstate extension to FSM's

The next simple extension is the superstate. The superstate can be used to aggregate sets of states with common transitions.

For example, let us assume that we have a situation in which two states, S1 and S2 both transition to state S3 upon the same stimulus, i. Using regular STDs, figure 4a) would result. With the superstate concept, however, we can use the shorthand notation shown in figure 4b). Harel also introduces the concept of the ***default entry state***. This is the subordinate state of a superstate into which the FSM enters if no other subordinate state is specified as the next state. Again, as in SDL there is a considerable learning curve involved which makes the notation impractical to use [1].

2.2.7 Other Techniques

In addition to the above techniques which are straight forward to compare with SDM's, there are some additional techniques in common use which we mention for the sake of completeness.

Data Flow Diagram

A data flow diagram (DFD) is a visualization of the data interaction that the overall system or some lower-level part of it has with other activities, whether internal or external to the system. On a DFD a reader will find graphic representations of external entities, processes, data flows, and data stores interconnected to show the progressive transformation of data.

The first DFD to be created represents the ***target system at its context level.*** The emphasis, instead, is on the types of data flows with which the system interacts: both those that enter the system from their source and those that travel from the system to their destination. Once client and developer reach a consensus on the scope of the target system, the developer can be proceed to identify the data-process interaction in progressively finer detail. Each resulting DFD, at what are called the ***"intermediate levels"***, is an elaboration or explosion of data-process interaction at the prior level. Before proceeding with an explosion of each of these processes, the analyst must balance the child diagram against the parent. To do so, the analyst must make sure both that the "data in" at the child level is identical with or adds up to the "data in" at the parent level and that the "data out" at the ***child level*** adds up to the "data out" at the parent level. [2]. After completing "level balancing" the analyst explores each process on the balanced diagram into its own DFD, showing the inner workings of each.

Eventually the analyst reaches a level of detail at which it is no longer useful to decompose the functions and the data with which they interact.

Data Dictionary

As the analyst completes each data flow diagram, he / she should employ a second tool of structured analysis, the data dictionary. The data dictionary is a repository, manual or computer based, containing information about the various data objects appearing on each DFD. It should include an entry for each data object identified. For instance, for each data store there will be an entry that specifies ***its name, all of its aliases, narrative text describing it, its contents, and the names of all processes that access it.*** Data flows need not decompose immediately to data elements. Very often the components of a data flow may be what was called above a data group i.e. named clusters of other data groups and / or data elements [2].

Structured Analysis and Design Technique

An earlier version called "Structured Analysis and Design Technique" encourages the analyst to begin by asking about the objectives of a system (the "why" question) before progressing to the major components of a system (the "what" question) and eventually determining the implementation technique. The technique has been used for planning as well as for analysis and design. Like all versions of structured analysis, SADT suggests that to understand a system you begin at the top and progress through the many layers from conceptual abstractions to concrete components. Data are classified as ***"consumed" (i.e.. data transformed by activity) and "non-consumed" i.e. data that control the activity but are not transformed.*** The analyst is also expected to identify the "mechanism" that will perform each activity [2].

Yourdan-DeMarco

Yourdan founded a firm of consultants who developed and have taught a generic form of structured analysis to organizations throughout the world. The technique advocates a top-down approach in which the analyst is led to ***decompose the system and its functions through levels of increasing detail.*** While it was initially process-oriented, more recent proponents of this approach have introduced the technique of information modeling to provide a better balance between the aspects of data and activity.

Gane and Sarson

This pair co-wrote a study of a version of structured analysis that suggests a slightly different emphasis from the Yourdon-DeMarco approach and a different graphic notation. This approach pays more attention than does the Yourdon-DeMarco approach to the identification of the data components of a system. What Yourdon-DeMarco call a data group, Gene and Sarson call a data structure. The two terms have the same meaning. In addition to the four tools already presented, Gane and Sarson suggest the use of a ***data access diagram to describe the structure and contents of data stores in the system.***
Each data store is viewed as collecting several design component types, each of which is related in some way to one or more of the other entities in the data store. The data access diagram pictorially presents the different entities in a data store and the access paths that link them. [2].

Ward and Mellor

Ward and Mellor were the first to publish their techniques for the analysis of "real time - embedded" systems. The following object types were defined. ***Process :*** Functions or activities in the system that transform or manipulate data in some way. Processes exchange data either with other processes, stores of data, or sources / destinations outside the boundaries of the system. [2].

Data flow: Data that pass from some source either internal or external to the system, to destinations, either internal or external to the system.*Data Store:* A place where data is held for later transformation or for reference.*External Design component :* Some activity outside the boundary and control of the target system that interacts with the system by means of data. Ward and Mellor advocate building a series of models during the analysis and design phases of real time system development. [38].

2.2.8 Object-Oriented System Design

Motivation For Object-Oriented Design

Humans tend to think in terms of models of the real world in which "objects" are the unit of consideration. It is therefore natural to have objects within the ***specification of a system to be developed.*** Using objects as building blocks for specifications and programs provides for information hiding. Only a fixed set of operations and functions are externally visible, internal aspects cannot be used for the design of other objects. The ***inheritance*** structure of object classes (found in most object-oriented languages) is a useful concept for the structuring of complex specifications and programs. The concept of a class of ***"persistent" objects corresponds to the notion of entities stored in the database.*** Current object-oriented design methods usually comprise a certain number of (iterative) design phases.

Though not all practitioners agree on the number and the denomination of these phases, all come up with basically the same philosophy. A ***preliminary phase*** first consists of the general problem area, of the specific aspects we want to handle, and also on the aim of the expected design; this phase is an informal one, but it should lead the designer to separate the different aspects included in the application, and also to precisely define the intended purposes of the model to be elaborated.

A ***second phase*** consists of the identification of the key components of the so-called "application domain"; this phase takes as input any information describing the functionality's to be provided, and should produce as a result, a set of entities, together with relationships among them, that can be easily mapped onto object oriented language constructs. The ***third phase*** is concerned with the allocation of the functions to be provided as operations to be offered by objects identified in the previous phase; some functions will also uncover some new objects which must be integrated in the application domain. The ***last phase*** is concerned with the definition of the behaviors of the objects; these behaviors consider the possible sequences of operation calls as well as the necessary processing associated with each operation; complex objects can be specified as smaller "applications".

Standard Features Of Object Oriented Languages

They are the following: ***Object types, instances and classes.*** A specification defines a certain number of object types. Each newly created object instance belongs to a given type which defines its properties. Using the inheritance structure discussed below, a given type definition may be used to define more specific subtypes of objects. We say that an object instance belongs to the object class A, if it either belongs to the type A or one of the subtypes of A. ***Attributes:*** An object type definition may include a certain number of named attributes. This means that each object instance of that type will have fixed references to other object instances, one for each attribute. ***Operations:*** The definition of a type may include the declaration of named operations (sometimes called "methods") which may be invoked by other objects or object instances of that class.

Typing: The OOL, Model supports strong type checking based on the declared object types.Generic classes (i.e. with type parameters) are also supported. ***Behavior:*** The execution of an operation implies in general certain state changes and possibly the execution of other operations on other objects which are known to the object which executes the operation.
Object Oriented languages are becoming popular but are not well controlled or easy to manage. [8].

2.3. Simple Design Machines (SDMs) and Example

2.3.1 SDM's : General Idea

Simple Design Machines (SDM's) is an extension of Design Machines, which is useful for representing high level design scenarios required to generate grey box use cases. These are use cases created in the design phase of the life cycle process. They are added to white box and black box use cases, and thus help to improve the coverage of the system test suite.
SDM's, when kept simple enough are quick and easy to read and implement. Grey box use cases (traces) generated at design time can be used to validate both the design as well as the implemented product. Designers can verify that their system operates according to the ***traces or scenarios*** generated at design time.

Whatever state it is in, the FSM reacts to the input given to it and does not care how it got to that state. FSM's can be in only one of a given number of states at any specific time. Both the output and the new state are purely functions of the current state and the input. A state in SDM's is dependent on the path to the node as well as the node itself.

In SDM's, on the other hand, the action from the state is dependent on its context within a particular service or functional path. Each design purpose is captured by the design path and all corresponding exception paths. The context of the system state is important. Even if the design is incomplete, the checking of that design will be complete *upto* the selected testability level.

SDM's are used when we have a system that responds to a ***complex combination of stimuli.*** Such systems are difficult to describe using finite state machines and even more difficult to describe in natural language[2].

2.3.2 SDM's : An Example - The Elevator

Let's consider a small example. If we want to specify the external behavior of a software controlled elevator door on an elevator control system, we would agree that :

- If the elevator is stopped at a floor and it has reached the requested floor, the elevator doors should open.

- If the door is CLOSING and the OPEN DOOR button is pressed, the elevator doors should OPEN.

- If the time limit is exceeded, the elevators doors should close.

We would agree that the Specification should include these three statements, yet what if a combination of the conditions is true? For example, what do we do when the time limit is exceeded and the OPEN DOOR button is pressed? See Figure 2, Chapter 3 for more details. In general, the required system responses to all combinations of these stimuli are difficult to describe using a finite state machine, and even more difficult to describe in English. SDM's on the other hand can describe it easily and naturally.

Notice that we are always able to decide on a yes / no transition for every state. Decision nodes are therefore testable. SDM's deal first with the normal cases. As in the above example we have a normal use case (vertical path) which is executed most frequently and then exception cases (those that occur less frequently, represented as alternative answers). This corresponds to a natural, common design process, and is also well suited to use case based design inspections. Note : SDM's provide more information than system flowcharts. SDM's provide normal vs exception use cases, capture context information, provide for hierarchical decomposition and lend themselves to grey box testability.

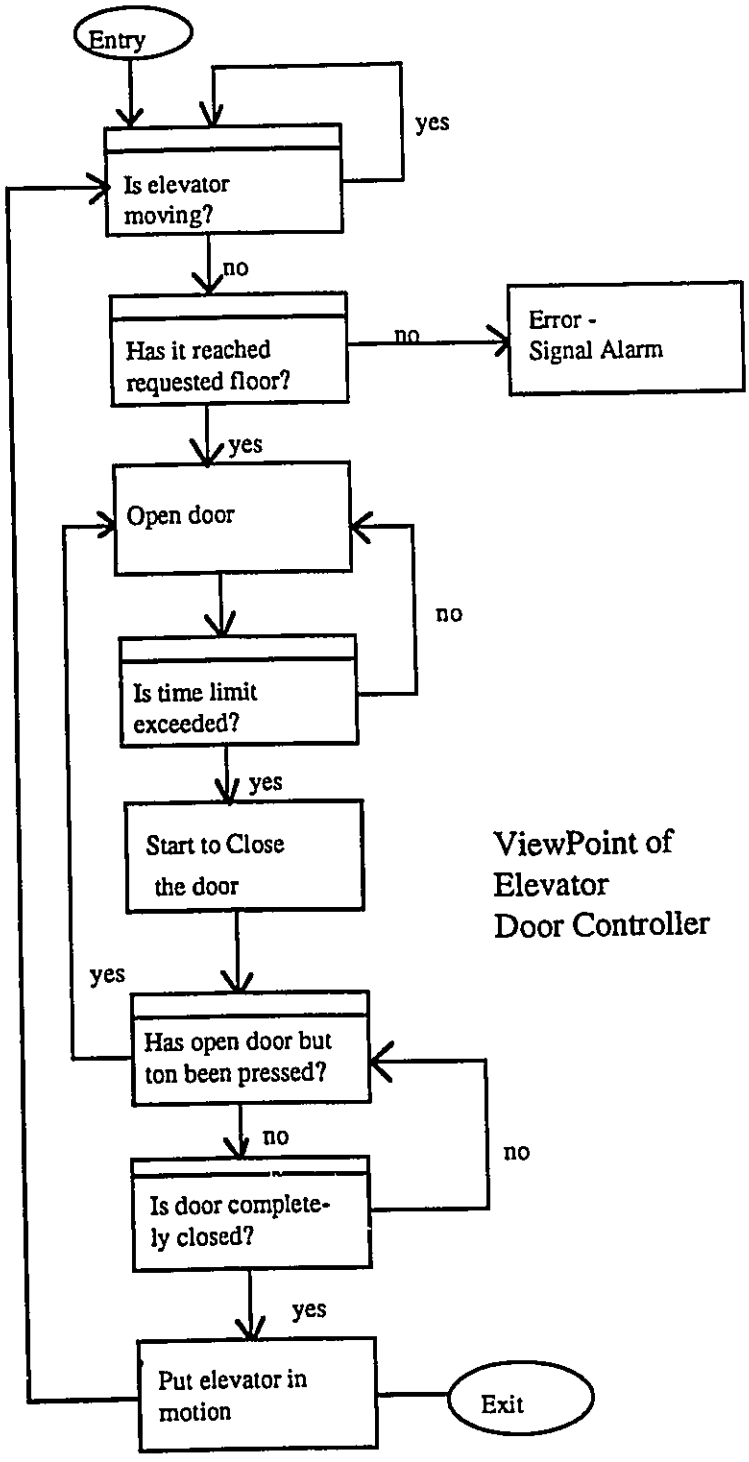


FIGURE 6. An SDM for an Elevator Door Control

2.3.3 Comparison of SDMs with FSMs

Let's consider the Elevator example described in section 2.3.2 as modeled by an FSM.

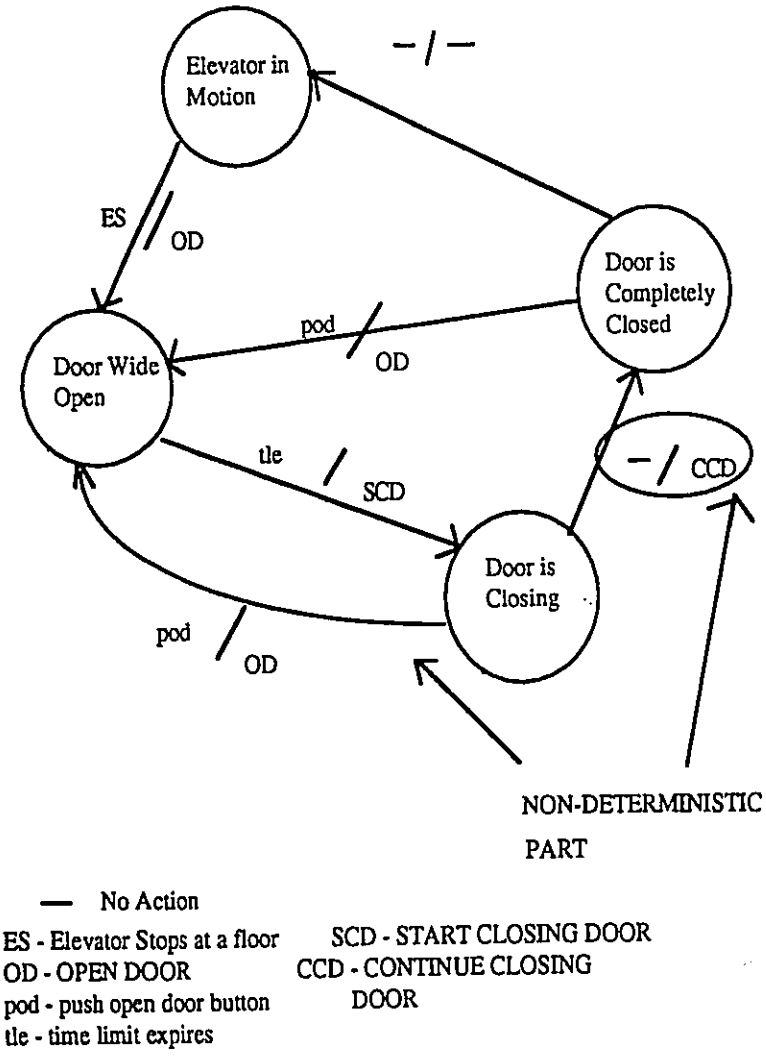


Fig 7 Finite State Machine for Elevator Problem

The non-deterministic part in the figure above shows the transition from "Door is Closing" to "Door is Completely Closed".

Consider the state "Door is Closing". The state may be exited in one of two ways. The two exiting transitions are concurrent ("pod" is an interrupt event) and therefore modelling the transition "closing --> completely closed" as a non-deterministic action does not capture the broader context.

Modelling the transition event as "no action" is even worse in terms of hiding the interrupt semantics.
The case where the elevator is not moving and the open button is pressed could be represented by adding a new state "Elevator Not in Motion" and having an outgoing arc where the output transition is the open button pressed.
Something more difficult to represent would be something like "timeout occurs" and "OPEN DOOR button is pressed". In [2], A. Davis states that "since this requires a *combination of conditions and inputs,* it would be more difficult to represent using FSMs" Using SDM's , we are trying to keep the design simple and scenario directed yet capture the context.

It is also difficult to establish differences between normal cases and exception cases as opposed to SDM's. FSM's represent states succinctly with no obvious relation to scenarios and meaning. In SDM's, scenarios are explicitly represented by a topographical layout. For example, in Figure 6, the normal use case is clearly indicated as the path taken from top to bottom. This is the path from "Is elevator moving?" to "Put elevator in motion".

A "State" in SDM's is different from a "State" in FSM's. The word "state" is used much the same way its used in ordinary english, as in "state of the union" or "state of health". The Oxford English Dictionary defines "state" as : "A combination of circumstances or attributes belonging for the time being to a person or thing".
A moving automobile whose engine is running can have the following states with respect to its transmission:
1. Reverse Gear
2. Neutral Gear
3. First Gear
4. Second Gear
5. Third Gear
6. Fourth Gear

States are represented by nodes. States are numbered or may be identified by words or whatever else is convenient.
Whatever is being modeled is subjected to inputs. As a result of those inputs, the state changes, or is said to have made a transition. Transitions are denoted by links that join the states. The input that causes the transition are marked on this link, that is the inputs are link weights. There is one outlink from every state for every input. If several inputs in a state cause a transition to the same subsequent state, instead of drawing a bunch of parallel links we can abbreviate the notation by listing the several inputs as in : " input1, input2, input3..." . A finite state machine is an abstract device that can be represented by a state graph having a finite number of states and a finite number of transitions between states.
An SDM also has a finite number of states and a finite number of transitions between states., but in SDM's, each state is dependent on the path to the node as well as the node itself.[6][7]. State graphs don't represent time - they represent sequence. A transition might take microseconds or centuries; a system could be in one state for milliseconds and another for eons, or the other way around; the state graph would be the same because it has no notion of time.

2.4 Comparisons Of Models and Methods

The following is a comparison of techniques for the ***Specification of External System Behavior.*** They are summarized in Table 1. *[2].*

Natural Language : They are understandable to computer naive personnel, no processor available to perform checking and can be used for any application where misinterpretation is acceptable.

Finite State Machines : They require some understanding, the formal model enables one to unambiguously define intended product behavior and thus serve as a basis for both design and test. Static errors can be detected. Can be used for any real time system that is relatively small.

PDL : Almost as easy to read as natural language as most of it is natural language. Static structural errors can be detected. Also used for any application where misinterpretation is acceptable.

SDL : They require some training, potential exists for static and behavioral error checking. SDL was developed specifically for telephone switching systems, but could be used in a range of real time applications.

Petri-nets : They are quite difficult to grasp, potential exists for protocol error checking but can be useful in applications where synchronization is critical to the specification of external behavior.

SDM : They can be quickly and easily understood, have potential for inexpensive (automatic) static checking and can be applied to systems where the application responds to a combination of inputs.

CRI-TERIA	Natural Language	FSM's	SDM's	PDL	SDL	Petri Nets
Access-ability to Designers	Yes	Yes	Limited	Yes	Yes	Limited
Test-ability	None	For certain applica-tions	Possible to unambiguously define product behavior	None	For certain applica-tions	For certain applica-tions
Supports Auto-mated Checking	None	Static, structural and behavioral errors can be found.	Semi automated checking possible	None	Static, structural and behavioral errors can be detected.	Static, structural and behavioral errors can be detected.
Supports generation of use cases	None	Yes	Yes	None	Yes	Yes
Ease of learning	Yes	Requires some time	Yes	Yes	Not easy	Not easy
Ease of use	Yes	For some applications	Yes	Yes	Not easy	Not easy
Prototype Generation	None	Yes, using tools	None	None	None	None
Normal vs exception cases	None	None	Yes	None	None	None
Appropriate Applications	Application where misinterpretation is ok	Application which does not warrant SDM's	Applications with multiple viewpoints	Application where misinterpretation is ok	Telephone feature oriented systems	Those in which synchrony is important
Organizational Assistance	Paragraph and Chapter	The machine	Methodology defined	The module	The stimulus response sequence	The net

TABLE 1

We have seen some of the shortcomings of a variety of design methods based on a small scale problem (simplified elevator controls), and have made some claims about SDM's which can easily be supported for small scale problems. In the rest of the thesis, we shall give details of the Grey Box SDM Validation Method and apply it to a much more realistic example, Personal Communication Systems (PCS).

Chapter 3

SDM's and Grey-Box Design Validation

3.1 Background: Design Machines

3.1.1 Introduction to Design Machines (DM's)

The concepts of design machines and Grey Box Design Inspection were originated by Probert[32] and subsequently developed by Geldrez[12]. A Design Machine is essentially a directed graph with labeled nodes and labeled arcs, representing a system design. *Nodes* correspond to descriptions of the current states of the entities involved in the communication system. *Arcs* correspond to transitions between these nodes (changes in the system state). The idea of a DM originated from the concept of Design Automation, which was introduced to facilitate the creation of *testable designs*. testable designs enhance the testing of an implementation of that design.

DMs can model both ***sequential*** and ***concurrent*** behavior, and can model recursion and non determinism when required. A DM can invoke another DM by referencing a node of that DM. DMs also have specific points of return from DM invocations similar to modular programming. This allows DMs to follow a hierarchical structure that permits designers to develop a design via successive refinement. A *design component* is viewed as having some kind of input and some kind of output, and must *interact with other design components*.

3.1.2 Design Machine Constructs

A DM consists of two high level constructs : nodes and transitions.

NODES:

Nodes describes the *states of the design component* involved in the communication system, *messages exchanged* between *design component* s; *actions* performed by design component or error / recovery situations of design component. Nodes can be of four types *(State, I/O event, Action and Error)* and can coexist with other nodes of the same type via a Boolean operator : " and, or, not, parallel". Each node type can be classified as an assumption (pending design decision) or a Boolean valued Status (query on status of a system or environment). "Assumption" or "status" names the *class* of a node. The types of nodes are:

State Nodes :
This type of node describes the internal state of the design component involved in the communication system, e.g. ' ready to send' or 'ready to receive'. As the design becomes more refined a State node may describe the state of a sub-component of the design, e.g. "Buffer full" in the case of a buffer or 'eof File_name' in the case of a file. These nodes have a yes / no boolean clause. A State Node can be considered a passive description of an design component. That is, it does not cause a ***change or alteration*** to the design component itself.

I/O Event nodes:
These are actual messages exchanged among design component. Design component can send and receive messages. The source and receiver may be specified if required, e.g. ' send Msg to Design component_X'.
In the case of an design component broadcasting a message to several design component, ali receiver design component should be specified explicitly or in some kind of notation, e.g. ' send Msg to Design component1, ..., Design component5'.

Error Nodes:
Error Nodes describe *error situations*. An error can occur if a message is lost or incorrect. A message can be lost or distorted in a channel between communicating design component. A message is incorrect if it does not correspond to the actual state (State Node) of the design component.

An error can also occur if an design component performs an action that is *invalid* (does not correspond to its state) or does not perform an action that should have been performed according to the state of the design component.

Action Nodes :
Action Nodes correspond to actions performed by design component, e.g. 'set buffer to empty', and 'read data'. Action Nodes are also used to indicate the exit or termination point of a DM. An Action Node acts upon a design component. That is, it causes a change or alteration to the design component.

TRANSITIONS
Transitions are *arcs that take the DM to the next Node.*
Two types of Transitions are introduced : Simple and Boolean Transitions. Simple Transitions originate from Assumption Nodes and Boolean Transitions originate from Status nodes.

Boolean :
These are labeled arcs, which describe the next node to be taken, depending on affirmative or negative evaluations for the current node and any additional information necessary for transition as indicated by the label eg. "timeout" is an additional label. This additional label can be combined via 'and', 'or', 'not', 'parallel' with other labels. We will refer to a Boolean Transition as one of a pair of outgoing arcs, where one arc corresponds to the affirmative response to current node and the other arc to the negative response to the current node. Each Status Node has a pair of outgoing arcs.

Simple :
These are ***unlabelled arcs***, that describe unconditional flow of control from a node to the next node. The figure 8 describes a graphical summary of the constructs used in a DM. The addition of a bar in the leftmost upper corner of a node denotes an Assumption node.

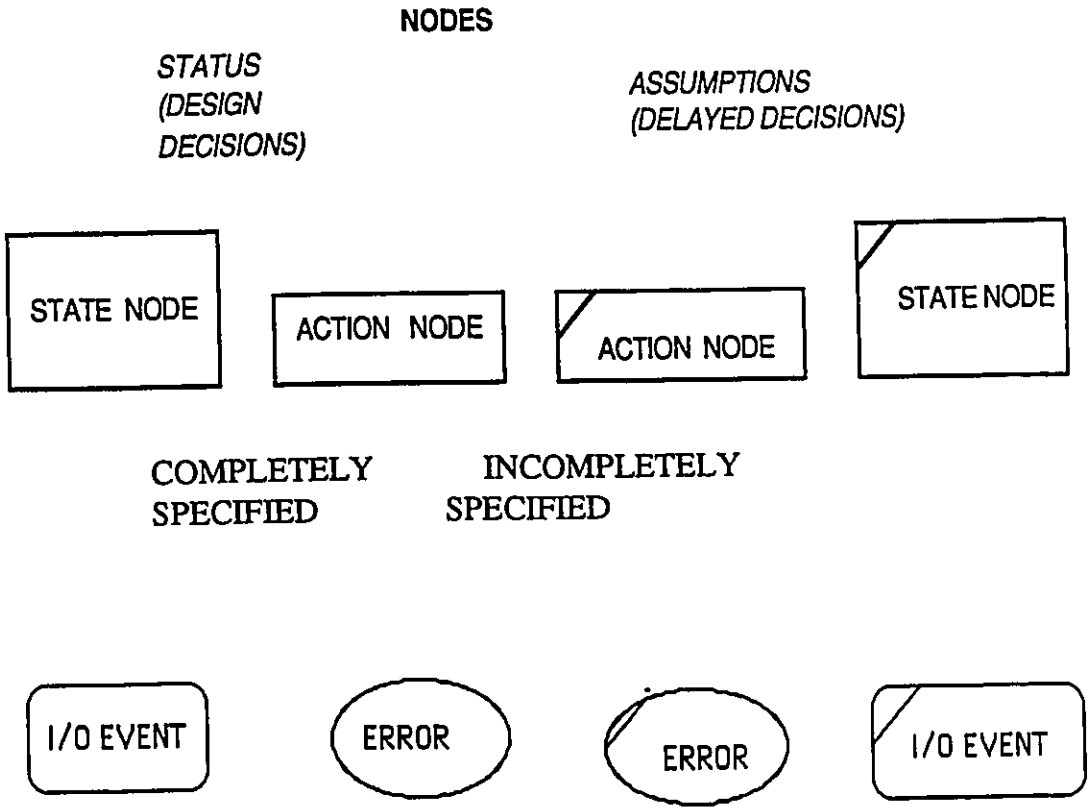


FIGURE 8 : Nodes and Transitions

DMs are ***descriptive.*** Each construct describes a step in the design of a system. This description is facilitated by a list of *keywords* that a designer should use, in order for his/her design to be more readable. The list of keywords to be used is not limited. Each designer should ***create his own list*** according to the specific application. This should be a normal output of the domain analysis process.

A DM should be ***flexible, understandable, extensible*** and able to describe the behavior of a system in a high level manner, supporting both abstraction and successive refinement. When describing a design, a designer needs to have the flexibility of completely specifying some design decisions and intentionally ***delaying*** the complete specification of others. If a designer encounters a design issue that leads to indecision, he should not have to delay further development of the design, awaiting investigation of issues which have only a limited design impact and should have the flexibility to assume a reasonable behavior of the system and continue development of the rest of the design. Later, after proper investigation of the design issue, the designer can revisit and adjust the design appropriately. For this situation, a node class called Assumption is introduced. These are used when the designer wishes to delay a decision about what steps should be taken in the event of a specific state node, action node, error node or I/O event node occurring or not. An assumption node *allows the designer to represent the occurrence of a system event, but defers the complete specification of an appropriate systems response.*

3.1.3 Characteristics and Properties of Design Machines

DMs are *flexible, executable and capable of describing the behavior of a system in increasing levels of detail.* DMs can invoke other DMs, thus allowing a modular hierarchical structure that permits designers to develop a design by means of successive refinement, where design decisions can be delayed if desired. A DM is *traceable* (recorded paths of execution). It can include recursion, can model sequential or concurrent behavior and can express non - determinism when desired.

However, DMs provide more descriptive capabilities than is necessary for most high-level requirements analysis and preliminary design representation and validation. For this reason, we simplified the representation model, and denote this representation Simple Design Machines or SDMs discussed in the next section.

3.2 Introduction to Simple Design Machines (SDM's)

One of the primary disadvantages of using SDL, DM's and other formal specification/ design languages is the fact that these languages are in themselves quite complex to learn. Designers therefore tend to write specifications in ***natural language*** which is more error prone.

The specification / design languages should therefore be kept sufficiently simple and not be inundated with too much syntax/semantics so as to make the language too complex to use. Enough functionality is however desired so as to provide *unambiguous design scenarios* which will aid in the generation of grey box use cases. Keeping this in mind , we propose the concept of *Simple Design Machines - a restriction of Design Machines, with fewer constructs and less functionality than DM's.* SDM's consist of only *decision nodes, action nodes* and *invocation nodes*.There are still two modes of evaluation for SDM's: ***Immediate Evaluation*** *and* ***Delayed Evaluation.***

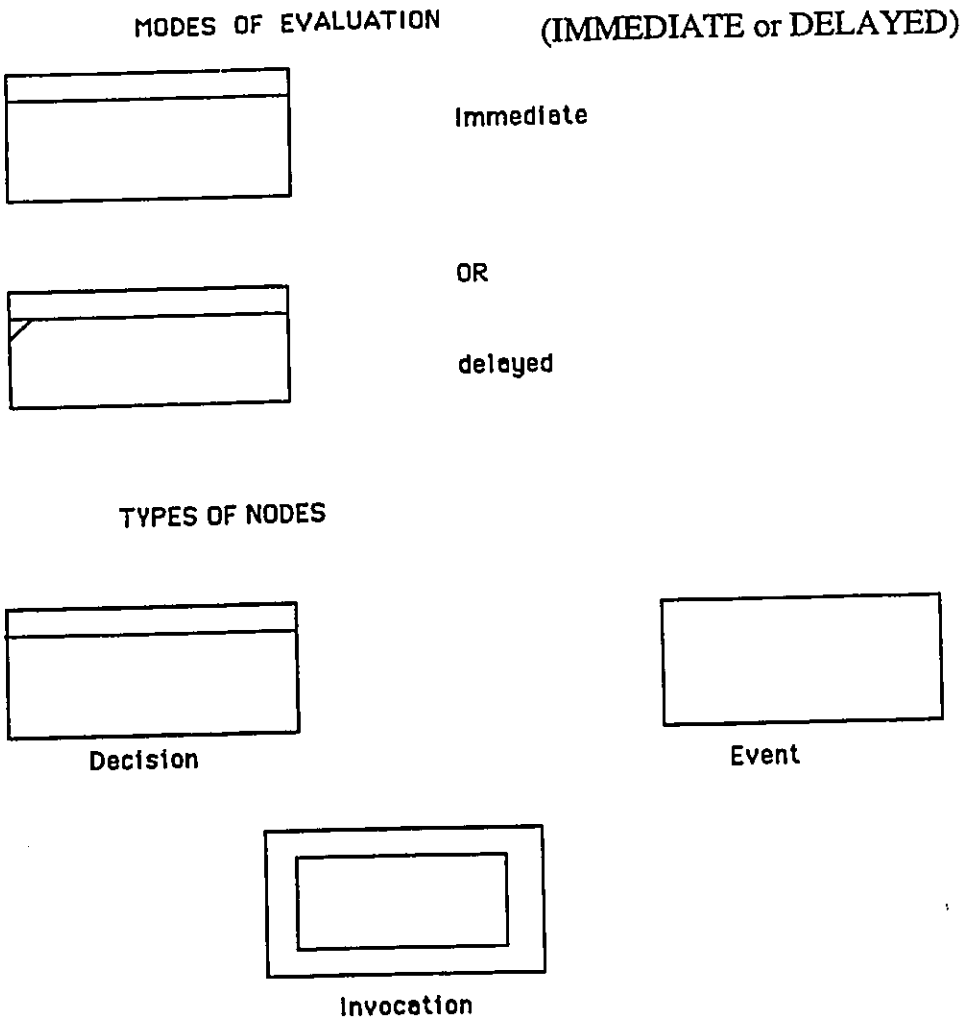


FIGURE 9 Modes of Evaluation

Decision nodes have yes and no transitions. Event nodes perform some kind of input, output or action. Invocation nodes invoke a lower level SDM. In the SDM context, three important concepts are relevant: a process, a design component and a role. The interaction of these three concepts defines a communication system or a set of communication services (in the telecommunication domain).

Example

In telephone systems, many switches provide such services as Three Way Call. This service allows three phones (parties) to be connected among each other. In terms of SDMs concepts, 3WC is viewed as follows:

Process : The Three Way Call Process
Design component : Phones A, B, C
Roles : Dial(er), Flash(er), Disconnect(er)

*A **process** is a self contained unit with a functional goal.* It describes in a few terms what is to be designed from a user's perspective. Processes are composed of entities and it is the interaction among these entities that describes the goal of the process. Geldrez[12].The interaction of entities is described in terms of roles that these design component can take.

The "behavior" requirements define the functions of the system. They describe all the inputs and outputs to and from the system as well as information concerning how the inputs and outputs interrelate. In other words, we need to define completely the transform function of the system software being specified. This description of how inputs map into outputs are typically called "behavioral description" or "operational specifications". The nonbehavioral requirements define the attributes of the system as it performs its job. They include a complete description of the system's required levels of efficiency, reliability, security, maintainability, portability, visibility, capacity, and standard compliance , for example.

A role is a set of behaviours which are intended to accomplish a function or service. If the system is said to support a particular role, then the system must provide the set of behaviours for that role. An entity is a physical concept as opposed to roles which are considered abstract concepts. Entities subscribe to roles.
An entity can take different roles *sequentially*. Many times a role is too complex and must be broken down into different components. When this occurs, the role becomes a new process and is broken down into different design component that will take new roles. This process of successive refinement characterizes the development of a design in the *SDM formalism.*

3.2.1 Characteristics and Properties of Simple Design Machines

SDMs are *flexible, executable and capable of describing the behavior of a system in increasing levels of detail.* In fact, the state contents are allowed to be free text in the initial stages of the SDM Methodology and are not required to have a formal semantics. SDMs can invoke other SDMs, thus allowing a modular hierarchical structure that permits designers to develop a design by means of successive refinement, where design decisions can be delayed if desired. A SDM is *traceable* because it is possible to record the paths of execution. As for example in the elevator example we could generate a trace from top to bottom which would look something like this:

(elevator not moving, requested floor not reached, signal alarm.)
An SDM can be recursive, can model sequential and/or concurrent behavior and can express non determinism when desired.

NOTE : State of an SDM <> State of an FSM. A State in SDM's contains both information for that state as well as information for the path leading to that state. An SDM has a finite number of states and a finite number of transitions between states., but in SDM's, each state is dependent on the path to the node as well as the node itself. This is unlike the state for FSM's. Refer to Section 2.3 for more details.

3.2.1.1 Hierarchical Structure Of SDMs

SDMs can be invoked by and can invoke other SDMs, thus creating a hierarchical structure for SDMs. This hierarchical structure allows designers to develop a design with successive refinement. Communication between SDMs in this hierarchical structure is achieved through parameter passing. The graphical representation for the hierarchical structure for SDMs follow simple rules. For example in the elevator example in Chapter 2, actions like OPEN DOOR and CLOSE DOOR are high level actions (level 1). Both these could have lower levels where these actions could be described in more deatil.

3.2.1.2 Notation

This section describes the definitions and notations relevant in the graphical representation of the invocation of the SDMs.
The major components of the hierarchical structure of a Complete SDM are the *block* and the SDM. Each of these components has a level. *The level of a component corresponds to the level of refinement of the design.*
Level 1 corresponds to the lowest level of refinement and describes the design in a high level manner. The level of a component is indicated as a rectangle at the top right corner of the component. SDMs have graphical representation of their types.
When an invocation node is encountered, the SDM with the name of the invocation node has the graphical representation of the type of the status invocation node (decision, I/o event). A *block* is a group of SDMs at the ***same level.*** A block is the outer shell of a collection of SDMs. It is denoted as a dashed line square or rectangle containing the SDMs that have the same level. When a new level is created, a new block is created.Blocks do not have a name or label at the top left hand corner but have a Level. ***An SDM always has a level.*** In order to avoid crowding the representation of a design, templates of a SDM can be used when SDMs have the same behavior (same SDMs) at a specific level.

Templates are drawn such that if SDM 'Y' is a template of SDM 'X', then SDM 'Y' is drawn behind SDM 'X' with only the name and variables of SDM 'Y' visible.

3.2.1.3 Level Constraints on Components

As said earlier, the level of a complete SDM (design) corresponds to the level of refinement of the design. Level 1 always corresponds to the lowest refinement of a design (high level design). When an invocation node is encountered, a new level is created. Notice that the case of a block, the Block can have a maximum of depth two in sublevels.

```
*****************************************************************
*************************
1. Level := 1;
2. create a block with level 'Level'
    2.1) label each process at this level as
            Level.1, Level.2 , ....., Level.n          {assume 'n' processes}

          { level 'Level' has now 'n' sublevels}
    2.2) label each non concurrent SDM at this level as
            Level.n+1, Level.n+2, ....., Level.n+m   {assume 'm' SDMs}
            {level 'Level' has now 'n+m' sublevels}
  2.3) For x := 1 to 'n' do
              label the concurrent SDMs in Process 'x' as
            Level.x.1, Level.x.2, ....., Level.x.d     {assume Block 'x'
has 'd' SDM'}
              { sublevel ' Level x' has now 'd' sublevels}
3.  (If at least one invocation node is encountered at level 'Level' in a
nonconcurrent
    SDM (SDM not contained in a process) OR
     ( If at least one Invocation Node is encountered at level 'Level' in
a  Process)
   THEN  Level := Level + 1
          Go to 2
4   ELSE  Terminate

*****************************************************************
*************************
```

ALGORITHM FOR LEVELING OF BLOCKS

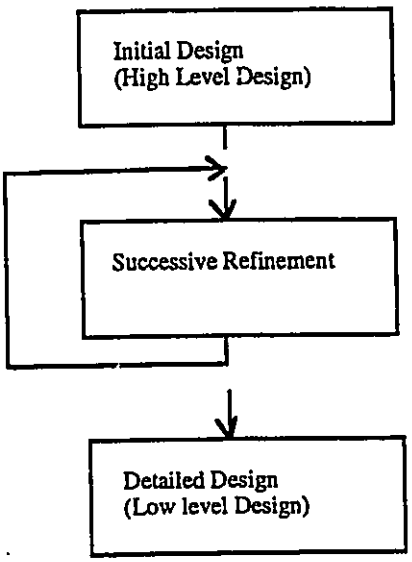
3.2.1.4 Invocation Constraints

SDMs can invoke other SDMs. A SDM can invoke a SDM contained only in the next ***deeper*** level. This is because the term invocation is related to more detail in the design. The invoked SDM corresponds to a set of steps, situated at a higher level, from which execution continues. This set of steps can be in the form of any of the constructs previously described. The invocation of SDMs must follow either of the two rules given below

Rule 1: An SDM located at level N can only be invoked by one SDM at level N-1, but a DM at level N-1 can invoke many DMs at level N.
Rule 2: If a SDM located at a Level N is invoked by many SDMs at level N-1, then the DMs at level N-1 must have the same behavior. The SDM at level N corresponds to the SDM for a role and must be identified with the word *role* before the *SDMs name.*

Guidelines for Designing via Simple Design Machines

The process of design with SDMs is very intuitive and relies on the designer's ***interpretation of the given specification.*** This section does not provide a precise or a formal model of deriving designs from specifications, but rather a set of guidelines that can facilitate this process.These guidelines are not hard and fast, they are subject to the designer's needs such as the level of detail the design should have.



In the initial design (high level design) phase the designers should identify the following:

1.***Identify the process*** to be designed. A name should be given to the process and the design will be referenced throughout the design process by this given name.

2. ***Decompose the design*** into units. This is done by identifying the design components that play a direct role in the description of the process.
3. ***Identify roles*** that the design component can take to describe the process.

3.2.2 Traces and Traceability

Another positive aspect of SDMs is their traceability. *Traces are recorded paths of execution, which enhance readability of SDMs*. Traces are affected by the recursion, concurrency and non - determinism of SDMs. Traces are flexible in the sense that they can be bounded by a specific level of the SDM (Bounded Trace) or can be extended to the overall design (Unbounded Trace) in the case of a CSDM (Complete SDM) with more than one level. A Complete SDM with only one level has only Bounded Traces.

For the purpose of discussion, the following terms are defined: *Bounded Trace*: Trace that does not contain a cycle. *Unbounded Trace*: Trace that contains a cycle. SDMs can have both bounded and unbounded traces. If a SDM has only Bounded Traces, it is possible to list all possible acyclic paths from the beginning to the end of the SDM. However when a **loop** is encountered the SDM has **Unbounded Traces.** One resulting constraint of SDM's is that if a complete SDM has only one level, it must generate only bounded traces. This is both a good design principle and a limitation of the tool.

To summarize, in SDM's the action from the state is dependent on its context within a particular service or functional path. Each design purpose is captured by the design path and all corresponding exception paths. The context of the system state is important. Even if the design is incomplete, the checking of that design will be complete *upto* the selected testability level.

3.3 The SDM Design Methodology and Grey Box Design Validation

This chapter describes and illustrates the Sdm Methodology.In the process we will show how ***Sdm's support both testability and traceability.*** PCS is the case study application used to bring out these characteristics. Section 4.1 describes the Sdm Methodology, Section 4.2 describes the PCS Specification in detail, Section 4.3 deals with other functions such as the Simple Call and Response. Lastly, Section 4.4 deals with the mapping of PCS to Sdm's and the evaluation of the Sdm technique.

This section describes a step-by-step approach to using SDM's.

Step 1 - Obtain reasonably precise, complete, structured natural language specification.
Step 2 - a) Derive corresponding SDM's from the spec in Step 1. First, describe the high level normal use cases,
b) then go back and describe the high level exception cases. This is a type of iterative design technique.
Step 3 - Decompose the high level invocation states into lower levels. This follows a hierarchical top down approach.
Step 4 - Generate traces and validate the specification. Update and reiterate if necessary.

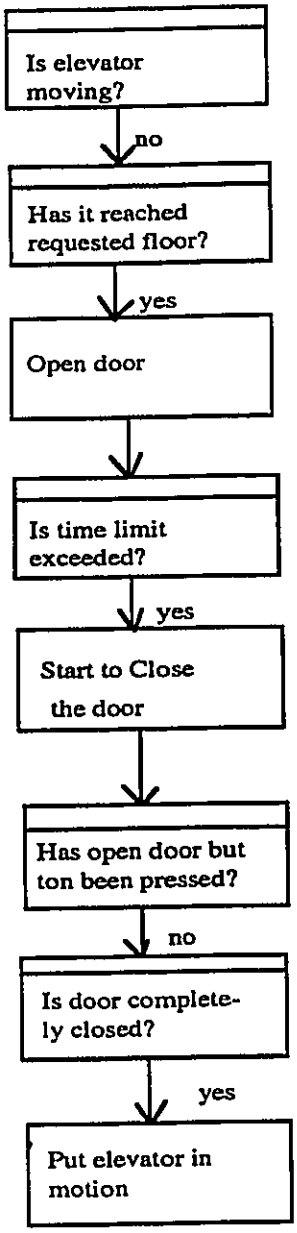
Example : Elevator

Step 1 - Obtain reasonably precise, complete, structured natural language specification.

- If the elevator is stopped at a floor and it has reached the requested floor, the elevator doors should open.

- If the door is CLOSING and the OPEN DOOR button is pressed, the elevator doors should OPEN.
- If the time limit is exceeded, the elevators doors should close.

Step 2 - a)Derive corresponding SDM's from the spec in Step 1. First, describe the high level normal use cases.
We first construct the SDM with the normal use cases. The normal initial state consists of the elevator user in a stationary elevator. It would look something like that shown in Figure 10.



ViewPoint of
Elevator
Door Controller

FIGURE 10. An SDM for an Elevator Door Control

The normal use case is a typical example of how an elevator generally operates.

If the elevator is not moving then we check if it has reached the requested floor.

If it has, then we open the door.

We check if the time limit has been exceeded, if yes, we start to Close the door.

If the OPEN DOOR button is pressed we check to see if the door is completely closed.

If the door is completely closed we put the elevator in motion.

Step 2b) Go back and describe the high level exception cases.

Now we examine the exception cases one by one.

If the elevator is not moving and it has not reached the requested floor, then we must signal alarm. This is not defined in the specification, as it was incomplete, so the designer must go back to the customer and make sure that's what he wanted.

Similarly, if the door was starting to close and the OPEN DOOR button was pressed, the door would have to be opened.

If the OPEN DOOR button had not been pressed, but the door was not completely closed, then we have to keep checking for the OPEN DOOR button being pressed until the door was completely closed.

If the OPEN DOOR was pressed then we go back to OPEN DOOR

After we complete the last state which is to put the elevator in motion, we loop back to the first state which checks if the elevator is in motion.

An enhanced SDM (exceptions included) will look like this:

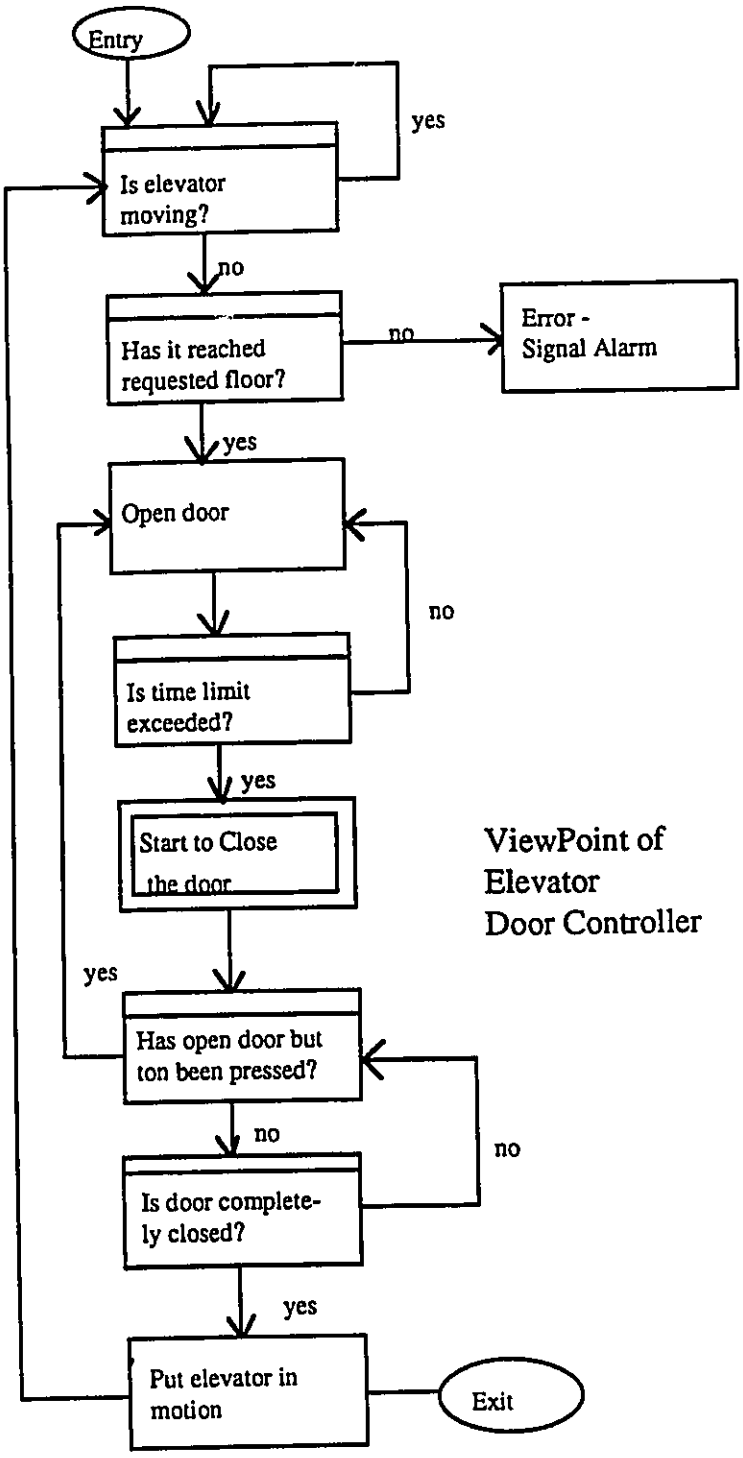


FIGURE 11. An SDM for an Elevator Door Control

Step 3 - Decompose the high level invocation states into lower levels. (hierarchical top down approach)

Consider the invocation state "Start to Close the Door". Now we can describe this state with a greater level of detail

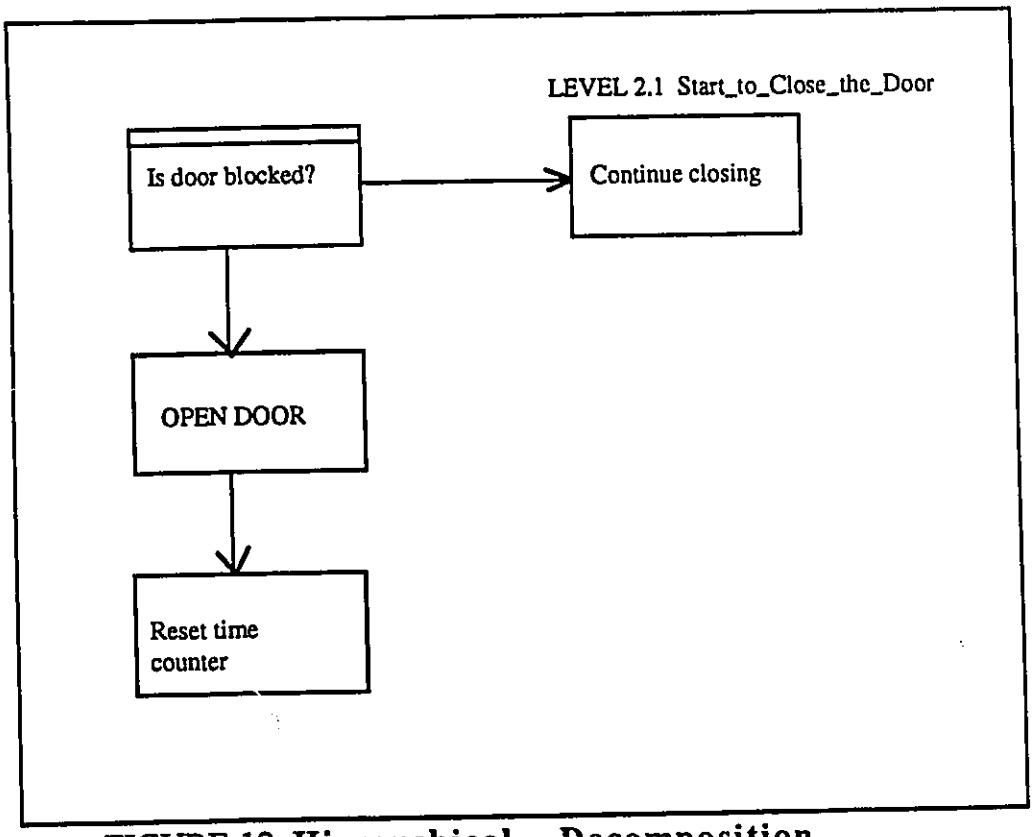


FIGURE 12 **Hierarchical Decomposition**

Follow this process for each invocation state that needs decomposition until it need not be decomposed any furthur.

3.4 Grey Box Design Validation

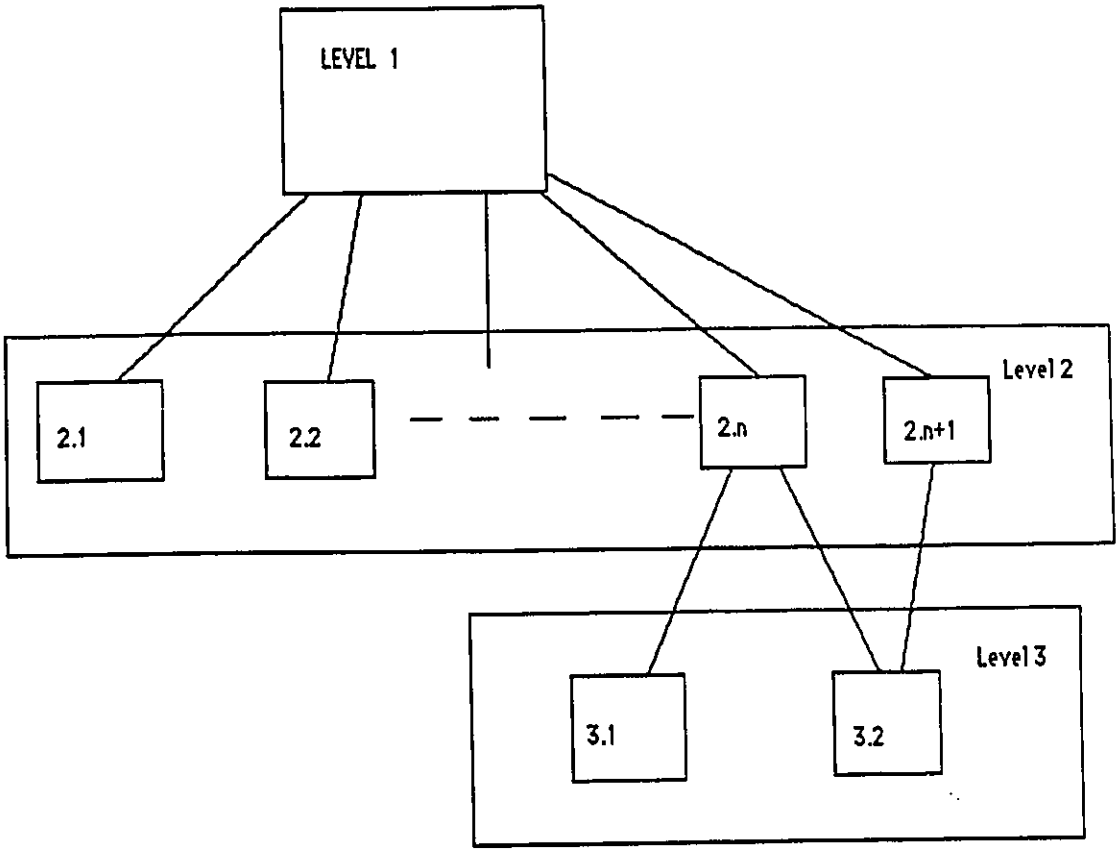
The next step in the process is so generate traces. The traces are then validated against the specification in natural language to detect inconsistencies.

The following is a Generation Strategy for Grey box use cases.

1. Level-by-Level : Symbolically execute each SDM independently. Each level is executed independently and use cases(traces) are generated for that level.

2. Depth First : Execute each SDM path of a SDM tree.
The SDM path is executed from root (level 1) to the leaf (the highest level number for that path). Traces are generated and executed. This is described in the following figure.
Execute each path from the root to the leaf, in a depth first fashion, so that each external branch is executed at least once. This would provide test cases with close to 100% coverage. These would consitute grey box test cases. The reason for choosing a depth first approach is as follows: Deep design paths should be formed on success (normal) paths first, and exception paths clustered with normal paths second. This lends itself to a depth first approach with maximum coverage. No loops are supported in this analysis.

GREY BOX TESTING STRATEGY



1. Execute each SDM independently ie. execute each level independently.

2. Execute each path from the root to the leaf, in a depth first fashion, so that each external branch is executed at least once. This would provide test cases with close to 100% coverage. These would constitute grey box test cases. The reason for choosing a depth first approach is as follows : Deep design paths should be formed on success (normal) paths first, and exception paths clustered with normal paths second. This lends itself to a depth first approach with maximum coverage. No loops are supported in this analysis.

FIGURE 13: Grey Box Design Inspection Strategy

The following example traces are some of those generated. Refer to the complete SDM for the elevator example.

Trace #1 is a normal use case.

Trace #1
elevator not moving
reached requested floor
open door
time limit exceeded
start to close the door
OPEN DOOR button is not pressed
DOOR is completely closed
put elevator in motion

Trace #2 is an exception use case

Trace #2
elevator not moving
has not reached requested floor
Error - signal alarm

This strategy yields the maximum coverage. Once these traces are generated, we go back to the natural language specification and validate it against the traces. If there are inconsistencies, the designer must go back to the customer and verify how the system should operate. Then, changes can either be made to the specification or to the resulting SDM.

3.5 Difference Between SDM's and DM's

The following table summarizes the key syntactic/semantic distinctions between DM's and SDM's.

Criteria	DM's (Design Machines)	SDM's (Simple Design Machines)
Types of nodes	Four types - State Nodes, I/O Event nodes, Error Nodes and Action nodes.	Three types - Decision, Event or Invocation.
Transitions	Can be combined with "and", "or", "not" labels.	Cannot be combined.
Number of rules on Design Methodology	>20	<10
Ease of use	More complex	Simple to use
Readability of Design	Not as easy to read	Easy to read
Concurrency	Supported fully	Not fully supported
Normal vs Exception cases	Not explicitly indicated	Explicitly indicated
Derivation of Behaviour Scenarios	Not automated	Semi-automated, obvious

FIGURE 14 - COMPARISON BETWEEN SDM's AND DM's

One of the primary disadvantages of using DM's and other formal specification/ design languages is the fact that these languages are in themselves quite complex to learn. Designers therefore tend to write specifications in ***natural language*** which is more error prone, or in detailed pseudo-code which is difficult to verify and validate.

The specification / design languages should therefore be kept sufficiently simple and not be inundated with too much syntax/semantics so as to make the language too complex to use which was a problem with DM's.

Enough functionality is however desired so as to provide *unambiguous design scenarios* which will aid in the generation of grey box use cases for design inspections.

Keeping this in mind , we proposed the concept of *Simple Design Machines - a restriction of Design Machines, with fewer constructs and less functionality than DM's.* SDM's consists of only *decision nodes, action nodes* and *invocation nodes*.There are still two modes of evaluation for SDM's: ***Immediate Evaluation and Delayed Evaluation.*** They have an advantage over Dm`s as they emphasize normal vs exception case based development. In this kind of development system, the normal (most frequent) response of the system is first described, followed later by the exception (less frequent) operation of the system.

Other distinctions between DM's and SDM's relate to the fact that SDM's are a subset of DM's. SDM's have fewer constructs and less functionality than DM's but are easier to use and can be used to generate unambiguous design scenarios, semi-automatically. For example, refer to the interpreter in Appendix C. It assists in the semi-automatic generation of use cases.

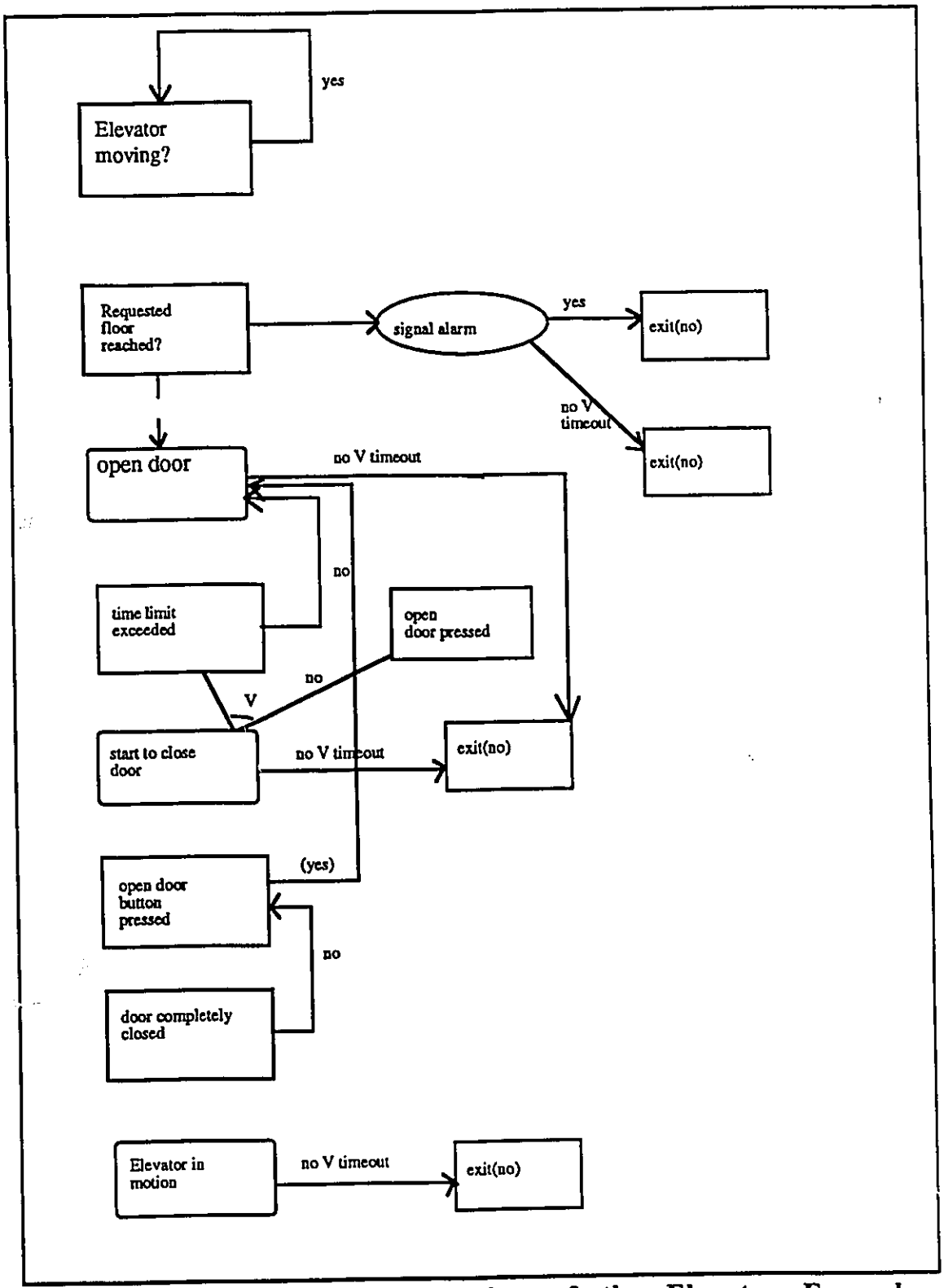


FIGURE 15 DM Representation of the Elevator Example

As we can see from the representation of the elevator example using DM's, they are more complex, have more types of nodes , they are harder to read and debug. They do not readily show normal vs exception cases. Also the concept of having outgoing arcs from an action node is in itself contradictory, which makes it hard to automatically generate scenarios.

Chapter 4

Case Study : Grey Box Design Validation of PCS Specification

4.1 Case Study Plan

In this chapter grey box design validation of the PCS Specification is performed. We start with the natural language specification for PCS, augment it with the Mondel Specification, build or buy an SDM support tool, derive PCS SDM's, generate grey box use cases, apply this to the PCS Specification and detect problems if any. We then measure costs (size, time) and estimate the cost effectiveness of applying the SDM Grey-Box method to PCS in the Case Study. As the CRIM-BNR Natural Language Specification is quite ambiguous, we also refer to the Mondel specification of PCS which is defined in Appendix A. PCS was picked as the application under consideration because it is fairly interesting, large and complex.

4.2 PCS in Natural Language and Mondel

4.2.1 Introduction to PCS

Envisioning that more than 50 percent of the tele-traffic will be associated with cordless or mobile terminals by the year 2000, the Commission of the European Communities (CEC) launched an ambitious research initiative under the Research on Advanced Communications for Europe (RACE) program in 1988. The program's purpose was to study the enabling techniques for creating a third generation mobile system by the turn of the century.

Personal Communication systems (PCS) *is a concept of telephone network services and operation which, in its full implementation, dissociates the service offering to the subscriber from the physical location of the service and access technology over which the service is delivered, and divests responsibility for service configuration and location from the telco to the subscriber.*

In PCS the subscriber service consists of a personal subscriber number with, optionally, a ***personal service profile*** which indicates what advanced calling features such as call waiting, call forwarding and calling number delivery are subscribed to. *Service delivery* is a software operation which consists of associating the subscriber service to a particular port in the network. Service configuration and location are controlled by the subscriber. From his terminal equipment, the subscriber can activate or deactivate calling features and as he moves, relocate his service to the new port. This is accomplished in the network by downloading calling features to the subscriber port, and by updating the subscriber number via *port translation* if appropriate.

Service in PCS need not imply exclusive ownership of the port supporting it. Multiple, independent services may coexist on a common port. Service also need not be static. Relocation may be frequent or dynamic; e.g. on a daily basis from home to office or through a mobile wireless access. With a cellular phone, relocation is occuring dynamically as the user moves from sector to sector. PCS relies on widespread penetration of digital switching technology, common channel signaling and databases for subscriber number ***translation and service profile download.*** [24].

Consider a social insurance number valid only for one government service, or a credit card number valid only at one outlet of a specific goods or service provider. This would not be very useful indeed as the average person would require 50 social insurance numbers and 200 credit cards to get the equivalent of what exists today. But this is the way the telecommunication industry operates. The telephone service of the subscriber : telephone number, call features, data fill of the features, etc. is permanently attached to a physical location in the telephone network. Personal Communication Service (PCS) ***dissociates the service of the subscriber from the physical location*** where the service is delivered. To achieve this goal, PCS relies on the following service principles:

Service Portability:

PCS uses the same concept as that of the "username" in computer systems or the social security number with the government.

The public id design component (i.e. directory number) and service of the subscriber refers to his service, not to the service at a given location in the network. Example : If a user has a call waiting option set, this option is true wherever the user is located.

Customer Control

From his telephone set or other appropriate terminal, the subscriber directly controls his service configuration. This includes both the call features and the location. Example : He may set all incoming calls to his work and all outgoing calls to home.

Access Independence

The subscriber's service i.e. the directory number and call services, is independent of the access technology over which it is delivered, although some calling features may be masked by limitations of some specific access technologies or terminals. The model I've used to represent this is ***objects in a domain*** [33]. Example : The user might not be able to receive email at home due to lack of a modem.

Four Key Attributes of a PCS user

In a traditional telephone service, a single telephone number is assigned together with a particular ***feature profile*** to a single physical access. The telephone number and feature profile in this context refer not only to the subscriber but also to the location where the service is delivered. The first 3 digits of the telephone number define the serving switch and the last four digits define the specific line equipment to which the physical loop termination is attached. PCS generalizes the traditional service described above by dissociating the service i.e.. the telephone number and feature profile, from the location where the service is delivered, and by transferring from the telco to the subscriber the responsibility for service configuration and location.
The fundamental change from traditional telephone service to PCS are:
1. ***The service unit becomes the "person", not the line, independently of geographical location and numbering plan restrictions.***
2. ***The service becomes also under the direct control of the subscriber, not the telco.***

To avoid confusion with the traditional telephone service , the following definitions are made. [31].

Subscriber Number (SN):

The public directory number which refers to a given subscriber and which is the billing reference for the telco. It may use the same seven digit numbering plan as is used today, but, as opposed to the conventional telephone number, it does not refer to any location in particular. The SN is the equivalent of the "user name" in computer systems. From the subscribers' perspective , the SN is equivalent to a telephone number today.

Personal Identification Number (PIN):

The private number used by a subscriber to gain access to controllable aspects of his service. It has the conventional meaning of a PIN as in an automated teller machine at banks, and is loosely equivalent to a "password" in computers. Using the PIN , the subscriber may also be provided with the capability to access network - wide services which may require subscriber verification (e.g.a functional equivalent to use of calling cards). [20]. This number may initially be assigned by the telco or maybe changed dynamically by the subscriber, depending on the implementation.

Feature Profile(FP)

The set of features assigned to a given subscriber at any given point in time. It lists the status (activated / deactivated) and related data of all the features whose usage is permitted. As described above, the subscriber may be provided with the ability to ***activate and deactivate features*** (e.g.. call waiting, toll denial, distinctive ringing, etc.) from his FP using a PIN. [20].

Network Port (NP)

This is the physical address referring to the loop termination of a given subscriber in the network. It is the generalization of the line equipment number on the switch and must be unique within the service portability area. The NP may also be a "virtual address", for instance , a voice mailbox. This allows subscribers to temporarily leave the physical network without disappearing as billable and addressable (i.e.. dialable) design component. From the telco internal network operation perspective, ***the NP is the equivalent of the telephone number today.*** [26].

It was decided to ignore in the specification the aspects related to the ***service profile management (feature profile)*** of PCS as they are modified by the user and there is little system input.

The definition of the PCS services in terms of user's mobility was stated in the Problem Definition. "Relocation" is the movement of the user from one location to another. There are mainly two types of relocation : **1) relocation of outgoing calls**
2) relocation of incoming calls

1. RELOCATION OF OUTGOING CALLS

The Relocation of outgoing calls allows a user to use Network Access Points (NP) to perform calls charged to his account. Relocation is the term used when the user locates from one NP to another. By using the relocation of outgoing calls, the user is responsible for any access to the NP without identification, to perform calls during the relocation period. The feature profile of the user becomes active and replaces the default feature profile of the NP for all the outgoing calls made on the NP possessed by the user via relocation. Three functions were assigned to the relocation of outgoing calls: ***The session,, set, Unexpected "sudden" termination of the session.***

The Session of Outgoing calls

The outcall session allows a PCS User to take possession of an NP for a certain period of time of a certain type of service.
The user becomes responsible for any access to the NP during the relocation period. Example : User Smith takes possession of the NP for eight hours. During that time, Jones obtains access to the NP and makes a long distance call. The call will be billed to Smith. An outcall session starts with explicit invocation of a PCS function (start_outcall_session) and terminates with the invocation of the PCS function (terminate_outcall_session) or automatically when the duration of the session expires. Since the session can be terminated explicitly, the duration is sometimes undetermined at the start of the session.

When the session terminates, ***the owner of the NP becomes responsible for any access to the NP and the default feature profile of the NP is reactivated.***

A user can invoke as many outcall sessions as he wants, on different NP and for various types of services. The sessions will all be active simultaneously. However, there can't be more than one session for an NP at a time. Also, the invocation of a new session causes the termination of all the sessions started previously on the NP for the type of service of the new session.

A PCS user can invoke or terminate an outcall session from any NP even if the session is on a different NP than any one for which the session is active. The owner of the NP must be able to grant or refuse access for users using the *default feature profile*. In our example, the owner of the NP, Smith, must have granted access to user Jones, and therefore should be responsible for Jones' call.

The Set of Outgoing Calls

The set of outgoing calls allows a PCS user to take possession of an NP, for a certain type of service within a period of time delimited by a particular action according to the type of NP on which it takes place.It's been defined that the duration of the "serie" is ***delimited by the duration of the NP activation*** e.g. all the time for which a conventional phone is picked up. The definition of the duration of an NP activation must be determined for each NP type (for the fax, this definition will be different). The user becomes responsible for the use of an NP where he becomes relocated for the duration of the serie, in the case of conventional telephones, *it will be as long as the user does not hang up.* A "serie" begins with the explicit invocation of a PCS function (start_serie) and terminates when its delimiting action occurs. For example, when the user hangs up, in the case of conventional telephones.

It is possible to perform several calls during a "serie". The user should not hang up when a call is terminated, only the ***callee must hang up when the call is terminated.*** The user should enter a significant code to signal that the serie continues, without hanging up and he should enter a new number for another subscriber. If a serie is invoked for an NP and there's an active session already for that NP, the activation of the serie takes place ***"override"*** of the session for the duration of the serie only; it does not terminate the outcall session. The session resumes after the termination of the serie. The serie is PCS version of the *calling cards existing at Bell Canada.*

The Unexpected Termination of an Outcall Session

The function allows the owner of the NP to terminate all sessions taking place at the NP under it's responsibility. ***An ambiguity arises however.It will appear clearly during the DM analysis of the behavior (next step of the methodology).*** A user is not informed of the termination of its outcall session when the owner decides to do so. There is no way of informing him, unless he provides explicit identification. Therefore, it is possible for him to use the NP again, thinking he's still responsible for it. The owner of an NP can ***invoke the termination function*** of outcall sessions from any NP.

2. RELOCATION OF INCOMING CALLS

The relocation of incoming calls allows the user to indicate a destination that is different from the default one (represented by the relocation <home location> in the domain). This destination will be the ***"routing point"*** for his calls for a given service type. The feature profile of this user becomes active, in relation with the communication environment that characterizes the NP on which the relocation takes place. This feature profile is called "effective feature profile" in the domain, and is in relocation with the relation *<incall location> on which it depends.* Hence, a user can receive these calls according to his description of the service at any NP by using this operation. Two operation were assigned to the incoming calls "incalls" :

The incall session and the unexpected termination of the session.

The Incall Session

The incall session allows the PCS user to relocate the destination of the calls that are assigned to him for a certain service type and for a period of time that he can specify at the start of the session.An incall sessions starts by an explicit invocation of the PCS function (start_incall_session). The session terminates by invoking explicitly the PCS function (terminate_incall_session) or automatically when the duration of the session expires.

Since the session can be terminated explicitly, its duration can be undetermined at the start of a session.

When the session terminates, the destination of the calls forwarded to the user is redefined as the default address <home location> for the type of the given service. A user can start one and only one incall session at a time, for a given service type. This is because the multiple delivery point management for a call can be difficult due to the geographical relocation of the points. It is possible however for a user relocate for incall sessions at several locations simultaneously if the routing points respect certain criterion's.

Even this case causes difficulty in the sense that for each start of a session, there should be a verification of constraints. The information management is difficult to justify with regard to advantages in the case of ***multiple delivery points***. There is no constraint imposed on the number of visitors on an NP i.e. on the number of active incall sessions for a given NP. The owner of an NP must be able to control the access to the NP it is responsible for, and therefore should be able to grant or refuse access to the users using the **default feature profile.**

The Unexpected Termination of Incall Sessions

This function allows the Owner of an NP to terminate all incall sessions taking place on the NP it is responsible for. An ambiguity arises however in the service. It will appear clearly during the SDM analysis of the specification.

(A user is not informed of the termination of the incall session if the owner of the NP decides to terminate. There is no way of informing the user, unless the owner of the NP provides explicit notification. Thus , it is possible for a user to be unaware that calls can no longer be forwarded to him.)
Moreover, the owner of the NP can invoke the termination function of incall sessions from any NP.

4.2.2 Small Examples of PCS

4.2.2.1 Normal PCS Use Case

In the following figure 16 we see a typical PCS scenario from a user perspective. An idle user goes off-hook. The system is now ready. He now has an option to either use the PCS system or simply make a normal call. In the case he decides to make a normal call, the user dials, if it rings then we established a connection and go on hook when the user is done. If it is busy then the user hangs up.
If the user wants to use the PCS functionality, then he enters the PCS code and authentication and gets into PCS mode. Once in the PCS mode the user has the option of either starting an outcall session, terminating an outcall session, starting an incall session, terminating an incall session, terminating some other outcall session or terminating some other incall session. When done, the user goes on hook. If the user wants to start_serie, he dials serie, establishes serie and can either continue or hang up when finished.

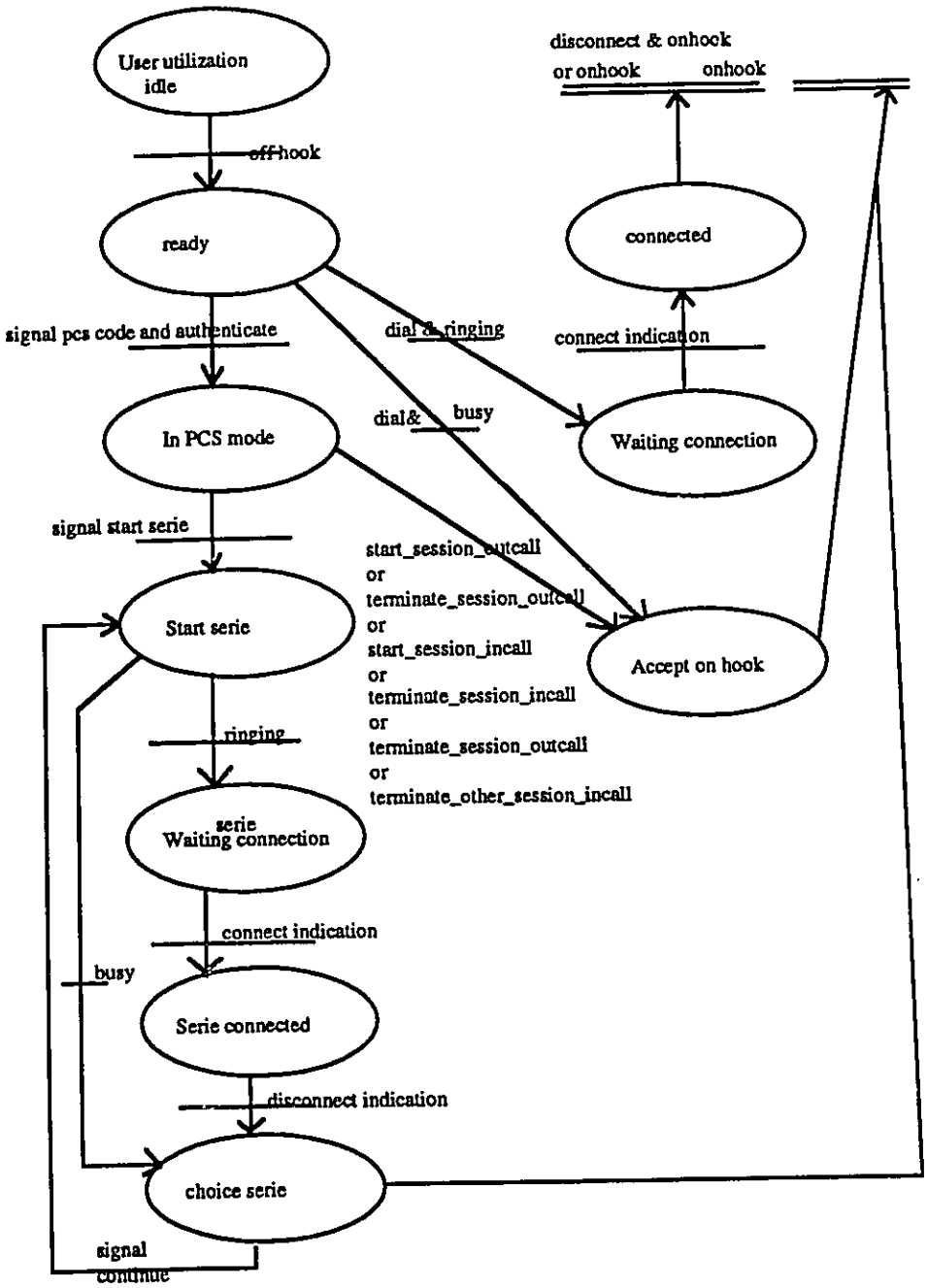


FIGURE 16 PCS Example

4.2.2.2 Simple PCS: The Electronic Professor

Professor P has an office at Home with a phone(HP) / fax(AF) and an office at the University with a Secretary(S), fax(UF), and a research assistant (RA). His University calls to his phone (UP) are forwarded to his secretary's phone SP. He has a conference call feature, and auto / manual options on both faxes. He grants long distance priveleges to his Secretary, but not to his RA or UF.

In this case study, we discover that natural language was not enough and consulted a more formal technique called Mondel (an Object Oriented Specification Language). For the sake of completeness we have included this reference in Appendix A.

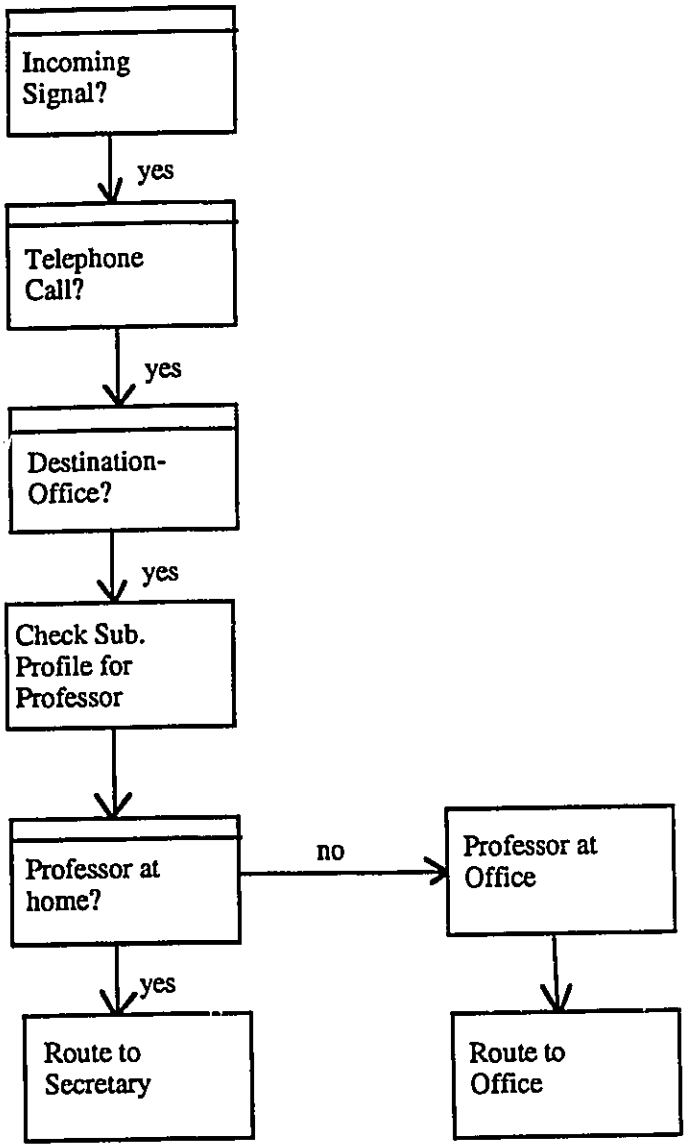


FIGURE 17 Normal Scenario For Electronic Professor

4.3 Constructing the PCS SDM

Consider a PCS scenario such as a user logging on. He first enters his PIN number. He then enters his Subscriber Number. If both these are correct, he then enters into the PCS system, and has several options:
1. Start Outcall Session
2. Terminate Outcall Session
3. Start Other Outcall Session
4. Terminate Other Outcall Session
5. Start Other Incall Session
6. Terminate Other Incall Session
7. Start Serie
8. Terminate Serie
9. Call
10. Respond.

The SDM at the highest level (Level 1) will denote the user logging on, entering the system, with several options presented to him. Each of these options would represent a separate sublevel (at higher level numbers). Every decision node has a yes / no transition. The "yes" transitions are not explicitly mentioned. The first state is the one where the system checks whether the customer is requesting a service. If not, we stay in the same state. If the customer is requesting a service, then the PCS system is invoked.

The customer then enters his assigned subscriber number and PIN number. If these are incorrect, the customer is informed and a hang-up is accepted. If these are correct the customer is faced with one of the ten options mentioned above. If an invalid function is chosen, a hang-up by the user is accepted. When one of the ten options are chosen, the corresponding invocation node is executed. This would be a SDM at a lower level (although with a higher level number). The SDM modeled for the first level is as shown. The entire mapping process for Personal Communication Systems is explained in detail in Appendix B.

4.4 Performing Grey Box Validation

Grey box design validation identifies and represents normal and exception use cases by looking at the design detail. Traces are generated which are used to validate the specification.
Examples of Grey Box traces. The coverage goal was branch coverage, which meant that every branch had to be executed at least once.

4.4.1 Derive a complete set of traces for each Level

Level-by-Level traces for Level 1
Step 1 : Complete Set of Traces for LEVEL 1

Trace 1

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session
Do_not_start_incall_session
Do_not_terminate_incall_session
Do_not_terminate_other_incall_session
Do_not_terminate_other_outcall_session
Do_not_do_serie
Do_not_call
Do_not_respond
Invalid

Trace 2

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_not_valid
Error_in_SN
Accept_hangup

Trace_3

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_not_valid
Error_in_PIN
Accept_hangup

Trace_4

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Start_outcall_session
Call_StartOutcallSession
Accept_hangup

Trace_5

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Terminate_outcall_session
Call_TerminateOutcallSession
Accept_hangup

Trace_6

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session
Start_incall_session
Call_StartIncallSession
Accept_hangup

Trace_7

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session
Do_not_start_incall_session
Terminate_incall_session
Call_TerminateIncallSession
Accept_hangup

Trace_8

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session
Do_not_start_incall_session
Do_not_terminate_incall_session
Terminate_other_incall_session
Call_TerminateOtherIncallSession
Accept_hangup

Trace_9

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session
Do_not_start_incall_session
Do_not_terminate_incall_session
Do_not_terminate_other_incall_session
Terminate_other_outcall_session
Call_TerminateOtherIOutcallSession
Accept_hangup

Trace 10

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session
Do_not_start_incall_session
Do_not_terminate_incall_session
Do_not_terminate_other_incall_session
Do_not_terminate_other_outcall_session
Do_serie
Call_Do_serie
Accept_hangup

Trace 11

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session
Do_not_start_incall_session
Do_not_terminate_incall_session
Do_not_terminate_other_incall_session
Do_not_terminate_other_outcall_session
Do_not_do_serie
Call
Call_Call
Accept_hangup

Trace 12

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Do_not_start_outcall_session
Do_not_terminate_outcall_session

Do_not_start_incall_session
Do_not_terminate_incall_session
Do_not_terminate_other_incall_session
Do_not_terminate_other_outcall_session
Do_not_do_serie
Do_not_call
Respond
Call_respond
Accept_hangup

Step 2. Execute each SDM path of a SDM tree.
Depth First Traces
(Only 1 trace is shown)
Trace 11

Customer_is_requesting service
Invoke_PCS
Subscriber_number_is_valid
PIN_is_valid
Start_outcall_session
Get_SN
Outcall_session_not_active
Get_valid_NP
Off_hook_SN
Busy_signal
Display_ErrMsg
Call_AcceptHangup

4.4.2 Execute SDM on Traces

Using the use cases from the previous section, we then execute the SDM. The execution of the SDM can be done either by using a semi-automated tool like the Symbolic Scenario Selector (implemented in the Appendices) or one could take a use case and generate from it test cases.

4.4.3 Assess Coverage and Correctness

Since the aim of the trace generation process was to achieve decision/condition coverage, this was quite easy to do as every decision node had been executed at least once.

The inconsistencies found in the specification were the following :

An outcall session starts with the explicit invocation of a PCS function (start_outcall_session) and terminates with the invocation of the PCS function (terminate_outcall_session). Since the session is terminated explicitly, the ***duration is sometimes undetermined*** which could cause a problem in the implementation.

This is similar to the party at the other end not hanging up.
According to the specification, a user can invoke as many outcall sessions as he wants, on different NP's and for various types of services. The sessions can all be active simultaneously. But according to the specification, there cannot be more than one session for an NP at a time. This is contradictory.Another ambiguity caught by SDM's is the fact that the ***owner of the NP can terminate all sessions taking place at the NP under its responsibility***. The user is not informed of the termination of the outcall session when the owner decides to do so.
As a result, there is no way of informing him, unless explicit identification is provided, which would then make it possible for him to use the NP again, thinking he was still responsible for it. 86 traces were generated in total for the case study.

The next step recommended in the Grey Box Validation process is to validate the actual implementation or code of the system, using the same traces.

4.5 Results / Recommendations of Case Study

4.5.1 Benefits Observed

The process of applying the SDM method was found to be both systematic and scenario based. Some bugs were also found by using this method. The normal vs exception scenarios are readily apparent which would make the design both easier to read as well as understand.

4.5.2 Costs, Limitations and Benefits Analysed.

Step 1:
Obtain reasonably precise, complete PCS specification.
Cost of step
The cost of this step is similar to writing a detailed functional specification in a corporation. The actual cost would be proportional to the experience and level of understanding of the product by the author. ***Natural language was used to write the functional specification that I used.*** The cost for this step is quite low, but it is one of the most important, as errors caught earlier in the life cycle are less costly to fix.

Step 2:
Apply commercially available tools or the Symbolic Scenario Selector to the specification in order to derive the corresponding SDM's.

Cost of step
Cost of applying the tool to natural language is lower than that of applying it to a formal language. Let us compare the costs between using the SSS and an SDL tool. For SDL, designing the finite state machine and verifying it for proper syntax and semantics is more difficult and time consuming than that for SDM's. ***That is because the syntax, semantics, error checking of SDM's are not as stringent as that of SDL.*** The complexity and amount of SDL constructs is also much larger which makes this step more costly. It is however, also more accurate. Code can also be generated automatically from the FSM's for SDL. This is not possible for SDM's as they are not completely a formal language.

Step 3:
Generate Traces or use cases from the SDM. Measure the effectiveness of using the approach.

Cost of step
The traces that are generated from the SDM can be used as use cases. User directed scenarios are generated from which the designer can verify that the implemented product works according to the use case / trace generated. These use cases are the grey box use cases used to uncover inconsistencies in every phase of the life cycle. Effectiveness and coverage of these use cases can be performed by static or dynamic checking. Decision coverage, condition coverage techniques can also be applied to verify that each branch has been executed at least once.
In the appendices, the traces are generated so that each branch of each node has been executed at least once.

4.6 Results of Case Study

SDM's are used when we have a ***system that responds to a combination of stimuli.*** This is difficult to describe using finite state machines and even more difficult to describe in natural language. This is one of the main advantages of using SDM's. ***SDM's , which generate grey box use cases, are to be used specifically for the generation of additional grey box use cases.*** This is in addition to white box and black box use cases, which would provide a more comprehensive test suite. Most other specification languages have been formed primarily to create a formal unambiguous method of specifying requirements of a system. SDM's when kept simple enough are quick and easy to read and implement. SDM's lend themselves more to use in communication systems as opposed to real time systems as inter process communication is kept to a minimum.

Also, grey box use cases (traces) generated at design time can be used to validate both the design as well as the implemented product. Designers could verify that their system operates according to the trace or use case generated at design time. We have seen the application of a testable design language (SDM) to a medium sized application (Personal Communication Systems) and the generation of grey box use cases from the resulting design. User directed scenarios are effectively described in this manner. It must be emphasized that the SDM approach would work best for applications where the system responds to a combination of inputs. ***Finite state machines would be better suited to systems that responds to a several independent stimuli.***

One disadvantage with SDM's is that because they are relatively simple - with simple constructs, they are not suitable for representation of real time, embedded systems or those that require inter process communication. ***SDM's are suitable to describe user directed scenarios. System level scenarios are better described using finite state machines.*** A large infrastructure of documents and tools does not exist for SDM's as it does for SDL and other specification languages.
As the size of the SDM increases, the chance finding an incorrect transition decreases, unless that branch is executed in a trace. SDM's cannot be effectively used to describe processes and the passing of information between these processes. It is used mainly to describe scenarios. Also SDM's cannot pass parameters between different levels of hierarchy, as that would increase the complexity and make the output traces harder to read. SDM's are ideal for those applications that are decision oriented. ***Real time applications, those with strong data intensity, complex real time applications that are stimulus oriented or feature oriented are best described using other specification languages.*** Based on the Case Study, the inconsistencies in the previous section should be looked into and fixed.

The following chapter describes techniques and tools as well as the working of a designed prototype which validates traces and measures the coverage. These traces are used to validate the Natural Language Specification. Often cases that are not specified are examined and ambiguities are detected.

Chapter 5

Development Of Tool Support For SDM Method

This chapter can be divided into two sections.
1. Existing Commercial Tool Support for SDM's and
2. A New, Proposed Prototype.

5.1 Existing Commercial Tool Support for SDM's

5.1.1 MacDesigner and MacAnalyst

MacDesigner and MacAnalyst are an integrated family of CASE tools which support the software development life cycle. They have been successfully applied in the field to a wide variety of system analysis and software development projects throughout the United States and abroad. MacAnalyst is an integrated application for structured analysis, data modeling and prototyping. This product supports industry standards, structured analysis techniques including the data dictionary. Screen prototyping allows the designer to quickly draw screens including ***fields, tables, menus, and buttons*** and then execute the live prototype. Extensive Verification and balancing reports ensure consistency and completeness.

MacDesigner supports structured design techniques for new systems and documentation of existing software. Its structured design capabilities include structure charts and module descriptions. ***Object - oriented design support includes inheritance diagrams and object communication diagrams.*** Other features include tree diagrams, data dictionary, requirement database with traceability and extensive verification and balancing reports. The feature in the MacAnalyst that can used to map PCS into SDM's are the State Transition Diagrams (STD).

Although it can take on many graphical forms depending on the methodology followed, the basic concept behind a STD is simple. It consists of ***States*** of the machine, ***Events*** which can occur when the machine is in a given state, ***Actions*** that take place as the result of an event occurrence. ***States*** describe the behavior, or operating mode, of the system when a given set of circumstances exist. When a system is in a given state, the occurrence of certain combinations of inputs, called events, will cause the system to change state, produce certain outputs, or in most cases, do both. The outputs of the system which are produced when an event occurs are called actions. Successful application of any modeling technique requires a clear understanding of the system domain. ***An STD should include all of the states, events and actions important to the system being modeled.*** Those not directly involved with the functions being modeled should be excluded. The STD shown above represents states as named boxes. Transition lines attach the machine states, and document the events and actions that are associated with the transfer from one machine state to another.

5.1.1.1 The Palette of Tools

On the left hand side of the STD window there is a palette of tools for creating and editing an STD. A tool can be selected by clicking on it.

SELECTION ARROW

The selection arrow is used for selecting objects in the STD window. Individual objects can be selected by clicking on them. A group of objects can be selected by enclosing them in a selection box made by pressing the mouse button at one corner, dragging to the other corner and releasing the mouse button. Multiple objects can also be selected by keeping the Shift key depressed while clicking on each object. The selection arrow is used for moving objects in the diagram by placing the arrow on the object, pressing the mouse button and dragging the object to a different location. It can be used to resize caption rectangles by grabbing the inside of the right most knob of a caption and dragging it to the right or left to achieve the desire text rectangle size.

State and subdiagram objects can also be resized if the document defaults allow it. The selection arrow can be used to quickly explode a ***subdiagram symbol*** to a child STD by double - clicking on it.Press the Shift key while double - clicking to move to a higher level STD.

CAPTION TOOL

The ***Caption Tool*** is used to create a caption in the STD window by clicking at the location you want the caption to be placed. An attribute dialog will be presented on the desktop, allowing you to type in the text of the caption and set other options such as text style and justification.Pressing Return or clicking OK will remove the dialog and place the caption into the diagram. Once a caption has been created it can be moved or have its display rectangle resized by using the selection arrow.

STATE TOOL

The State tool is used to create a state by clicking at any location in the STD window. The attribute dialog allows the state name and displayed name to be defined. The state name is echoed in the "Displayed As" edit box when the state is initially defined. Any subsequent changes to the state name or its displayed as name will not affect the other name.

SUBDIAGRAM TOOL

The Subdiagram tool is used to create a symbol which explodes to a child STD. When clicking the tool in the STD window, the attribute dialog allows the subdiagram symbol to be named. This name becomes the default title in the child diagram. If a child diagram already exists, its name may be selected from the menu associated with the ***Explodes To field***. If no diagram is designated in the "Explode To" field, a new child diagram is created. The dialog also allows the number of the child diagram to be entered.

TRANSITION TOOL

The Transition tool is used to associate states and subdiagrams on an STD.

The attribute dialog allows a transitions event name, displayed as name, action name, and termination name to be defined. The event name is echoed in the ***Displayed As edit box*** when the transition is initially defined. Any subsequent changes in the event name or its displayed as name will not affect the other name. A transition is created by clicking with the Transition tool to create vertices of the transition or double-clicking outside of an object to terminate the transition.

When clicking on a state or subdiagram, the transition becomes logically bound to that object. When the end of a transition is not bound to an object on the current STD window, the termination name will be displayed at the end of the transition. Once created , the vertices of a transition, the termination name, or the event - action text can be dragged to other positions. Pressing the ***Option key*** while drawing forces lines to be drawn perpendicular. Note that smoothing will override the perpendicular line option, and the transition will be smoothed as soon as it is terminated.

PATH TOOL

The Path tool is used to create a document path symbol. The document path symbol is a navigation tool which is used to easily switch between different documents in the default folder. Double-clicking on the path symbol will open the document to which the path symbol is linked, and that documents window will become the active window. The path dialog box allows several path definition items to be entered. ***The first is a document name, which is the name which will be displayed along with the path symbol.*** The name which is entered does not have to be the same name as the file which the path symbol is linked to. The Type box allows you to define what type of document , design, analysis, ERD, or STD, the path symbol is to be linked to. The Explodes To item allows you to select the name of the document and title of the diagram the path symbol is to be linked to.

5.1.1.2 Objects in the STD Window

STD Window Objects
The STD window can contain the following objects:

Captions

Captions consist of one or more lines of explanatory text and can be drawn in various text fonts, styles and sizes. The text of a caption is enclosed within an invisible display rectangle which can be adjusted in size. Captions can be bordered.

States

A State is drawn as a rectangle containing a name and displayed name which represents a unique system state. Each state in a state model document has a unique name and is defined as an entry in the data dictionary. Additional information about a state may be defined in the Definition field of the data dictionary.

Subdiagram

A subdiagram symbol is drawn as double rectangles with an associated name and number and indicates a subordinate child STD. The child diagram is accessible by double-clicking on the subdiagram symbol with the selection arrow, while double-clicking on the child diagram with the Shift key pressed will return you to the original parent diagram. The name and number of a subdiagram symbol becomes the default name and number of the child STD.

5.1.1.3 Transitions

Transitions are lines which attach state and subdiagram symbols. They represent events and actions that take place in the system. Transitions can have two or more vertex points. Each transition can have an event name, action name, and termination object name.In some cases, event and action names are expressions and equations, respectively. They normally consist of a string of control flow names, operators, and logical conditions. ***A Transition with one end unattached represents a transition to an object on the parent diagram.*** The termination object name gives the name of the object on the parent diagram to which this transition is attached. The termination object name is always positioned next to the unattached end of a transition. A transition with both ends unattached is invalid and has no meaning.

5.1.1.4 Paths

The path symbol is a navigation tool which is used to easily switch between different documents in the default folder. Clicking on the path symbol will open the document to which the path symbol is linked, and that documents window will become an active window.

5.1.1.5 STD Window Object Names

Object names are stored in the data dictionary by typing them into the Dictionary window, by using the Automatic Dictionary Entry option or the merge command. Since object names become dictionary entries they must be single words and follow the ***naming conventions*** used for dictionary entry names. MacAnalyst will automatically purge spaces and illegal characters in object names. In addition to the objects name, most objects also have a displayed name which by default is the same as the object name. Name is the name actually drawn on the diagram and can be edited to be anything the user desires using the Attribute dialog. For example, some users may choose to use multiple words separated by spaces to more fully describe the object.

5.1.1.6 Leveling and Balancing an STD

An STD is typically drawn as one large, flat diagram extending across several horizontal and vertical pages. To support complex diagram, MacAnalyst allows part of a diagram to be placed on a child STD under the subdiagram symbol. Leveling refers to the technique of expanding a subdiagram symbol into a more detailed child diagram. This technique should be repeated as many levels deep as necessary.

Since a lower level diagram is simply an explosion of its parent subdiagram symbol, all transitions into the parent subdiagram must also leave the lower level diagram. ***Ensuring conservation of transition between a parent subdiagram and its lower level diagram is called balancing.*** Use the mouse to move between levels in the STD state model. Double-clicking on a subdiagram symbol is used for exploding to its lower level diagram. The Collapse Objects command on the Option menu can be used to level an existing diagram.

This command extracts a selected group of objects from a parent diagram, creates a ***new child diagram***, and puts the extracted objects on it. A parent subdiagram symbol is created on the parent diagram to represent the lower level child diagram.

5.1.1.6 Tables

Types of Tables

MacAnalyst supports four types of state tables.The Table Type command on the Format menu allows you to choose from one of the four table types - ***Decision Table, Process Activation Table, State Transition Matrix, and State TransitionTable***. The one to closely map SDM's is the State Transition Table.

State Transitions Tables

The State Transition Table contains exactly the same information as a state transition diagram or a state transition matrix. A state transition table is used to define control actions in a sequential state machine. A state transition table is broken down into four columns, which are labeled in the header row along the top of the table. The left-most column lists all of the states the machine can be in which are related to the CBar the table is defining.The next column lists all of the events which apply to each state, i.e. there may be one or more events listed for each machine state.
As in the state transition matrix, the control flows coming into the CBar are the events being input to the table.The third column lists any actions which may occur as a result of an event. Once again, the ***control flows going out of the CBar are the actions being output by the table.*** The right most column defines the state the machine will transition into once an event has occurred.

One of MacAnalyst's most powerful features is the ability to check a model for errors. The two types of errors which typically occur when creating STD diagrams and tables are those of inconsistency and incompleteness in the diagrams and an imbalance of connections between STD levels. MacAnalyst provides flexible checking of both types of diagramming errors using the ***Verify Diagram, Verify Table, and Balance Diagram*** commands from the Report menu.

Individual error checks can be selectively enabled or disabled using checkboxes on the dialog presented by the Customize Verify Reports command. These commands are only enabled when a text document is available in the Text window for displaying their output. When the Verify Diagram, Verify Table, or Balance Diagram command is chosen, a dialog is presented to specify which diagram should be checked. For example, Verify Diagram presents the following dialog allowing the current STD to be checked and - or lower child STDs to be checked. This command checks for the following inconsistencies and incompleteness in STDs.Blank state, event, action, connection terminationor subdiagram names, States or subdiagram without transitions, duplicate state names, unattached transitions, paths with no explosion defined

Verify Table

This command checks for the following inconsistencies and incompleteness in Tables. Cells which have no text associated with them and unnamed dictionary entries. This command checks that the transitions into an STD are the same as those into its parent subdiagram symbol. Likewise, the transitions leaving the STD must also balance with those leaving its parent subdiagram. For example, a transition entering the subdiagram symbol from StateX on the parent STD would also enter the child STD. On the child STD, one end of the transition would be unattached and have the termination name, State X.

5.2 New Proposed Tool - The Symbolic Scenario Selector (SSS)

5.2.1 Need For the Symbolic Scenario Selector

There is an increased emphasis on automating the software development process and a shift towards the development of integrated tools, techniques and methodology in a software development environment. A software development environment in this context is defined as the ***process, methodology and automated tools*** required to develop a software system. The three major components in a software development environment are: ***Software Paradigm, Methods and Techniques and Automated Software Tools.***

The *software paradigm* is the process of developing a software system and it is fundamental to all software development environments. Its function is to describe a model of a sequence of events to create a software product.

The phases of the software life cycle are

Specification (Black Box Testing)

Design (Grey Box Design Inspection)

Code (White Box Testing)

What we need is a set of integrated tools to automate this methodology. Automation of a well integrated and comprehensive methodology will significantly enhance and increase the productivity of an environment. *SDL is an artificial language* designed for the purpose of writing specifications as opposed to natural languages which are multi - purpose and evolve in an ad - hoc way. It is possible to write an SDL spec. with precise meaning and no ambiguities. *SDL lends itself to test generation.* Now, we move to the next phase, Design. In previous sections, we saw the need for Simple Design Machines which was *to facilitate the creation of testable designs, thus enhancing the testing of the implementation of that design and also supporting system design for testability.* What we are striving for here is **Design Automation.** Therefore, it is appropriate at this stage, to introduce a tool which enables the user to

1. ***Create SDM***

2. ***Execute SDM***

3. ***Provide Tracing and Coverage Information.***

This tool is called the SYMBOLIC SCENARIO SELECTOR (SSS from now on). The SSS is a lot like most of the standard editors with the above 3 options provided. The SSS will be explained in detail in the next section. The designer has to create (capture) design scenarios and execute invalidated traces. He can do this with the SSS to create design scenarios.

The *SDM kernel can also be mapped onto the existing SDL kernel.* The shell would contain the SSS, interprocess mail, semaphores apart from other features. It is to be kept simple, yet effective, as one of the main problems with SDL is that programmers don't want to have to study the language and constructs. They rather write the specifications in natural language.
The key factor to the successful use of SDM is that it is well accepted and well understood by the designers. We should not have too many constructs, rules, constraints which will hamper wide use.

The *SSS is a highly automated syntax driven graphical tool.* Placement of symbols and lines that generate symbols are always generated automatically, but the user can redefine both symbol and line positions. Entry of text is directed to selected symbol and is automatically justified and wrapped according to symbol type. The *internal data structure used is an array.* Many pages can be contained in one document. The size of the document is limited only by available working memory. On the diagram being edited a syntax check and limited semantic check can be performed. The user steps through the entire simple design machine, answering yes / no to specific nodes in the SDM. (s)he can change the input area, intermediate area, or output area, depending on the state (s)he is in.

5.2.2 Creation of SDM

Ideally, the SSS is a graphical editor for process, service, procedure and macro diagrams. It is a highly automated syntax driven graphical tool. Placement of symbols and lines that connect states or nodes are always generated automatically, but the user can redefine both symbol and line positions. Entry of text is directed to the selected symbol and is automatically justified and wrapped according to symbol type. The internal data structure of the graphical editor is created as an array. There are six arrays in total. *One for the node number, Description of the node, Transition to node number, when 'yes' branch is taken, Action to be performed, if any, on the 'yes' branch, Transition to node number, when 'no' branch is taken., Action to be performed, if any, on the 'branch'.* The above are for **Boolean transitions**. For simple transitions, values 3,5 will be the same, and values 4,6 will be the same. Many pages can be contained in one document.

This is limited by available working memory.
A possible view of an SSS is shown on the next page.

5.2.3 Possible View Of Symbolic Scenario Selector

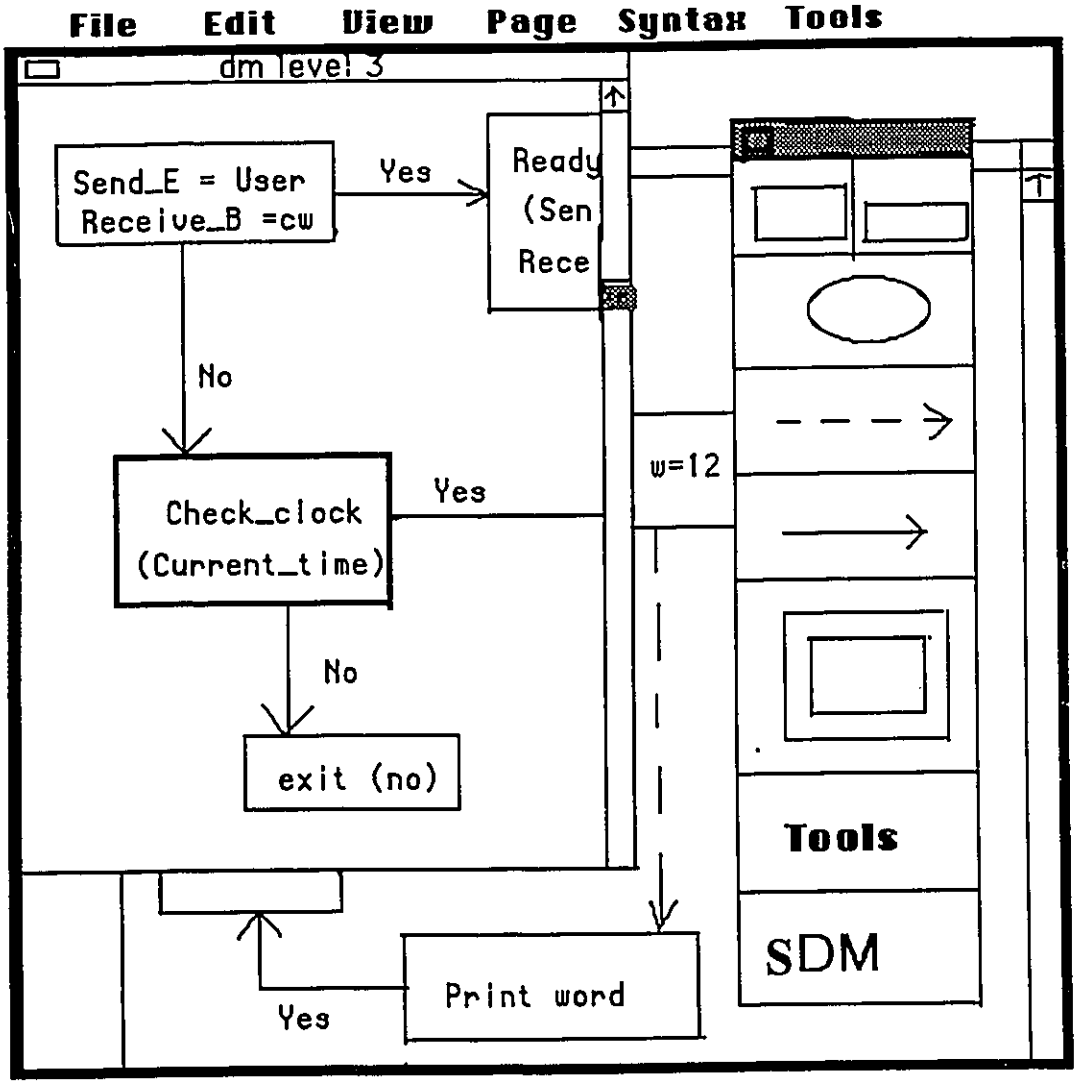


FIGURE 18 View of SSS

On the diagram being edited, a syntax check and a limited semantic check can be performed. The semantic checks that can be included in the SSS are : states that cannot be reached, no state corresponding to the next state, missing or multiple connectors, no outconnector corresponding to an inconnector, errors in input and save lists, absence of start symbol. If an error is found, there are two ways to inform the user. If the erroneous text is small enough to fit in a dialog window, it is shown with the erroneous token highlighted. In that case, the user can correct the text and proceed with the syntax check.

If the text, in which the syntax error is found, is not small (for example in text symbols), the symbol containing it is selected and the error token is highlighted there. The actual syntax error is usually before the token found as the one with the error, but the information should be adequate for locating it. *The SSS, for creating of system structure descriptions is a separate application that is used as a shell through which editing of processes and services is provided.*

5.2.4 Constraints

The SSS also automatically verifies that both the static as well as dynamic constraints of the SDM are not being violated. If it is, then appropriate error messages are produced. The constraints on SDMs are divided into static and dynamic. *Static constraints deal with the connection of nodes and arcs. Dynamic constraints deal with the execution and invocation of SDMs. Thus, the SSS automatically checks for conformance to the constraints, generating error messages and appropriate warnings as previously explained.*

5.2.5 Execution Behavior of the SSS

The SDM can be executed. The user traverses through every node and transition, changing the ***input area, intermediate area and output area,*** if and when he wants to. In this way, the user checks if the SDM conforms to expected design. Values of defined variables can also be printed out at any time. The user is expected to input yes / no at every Boolean transition and make changes to the input, intermediate and output areas him(her)self.

The SSS simply complies with the input given by the user, and traverses corresponding paths. Since distinct nodes have distinct English - like labels, the execution of a SDM is determined by the placement and values of its constructs. Nodes that behave as Normal nodes are executed in linear sequence. That is, the initial Node to begin execution is the first Node listed. Following the appropriate transition will take the SDM to the next node. When a node that behaves as an Invocation node is encountered, ***transfer of control*** is passed to the SDM with the name of the Invocation node. When an Exit Node is encountered, an exit from the current SDM is executed and control is transferred back to the SDM that invoked the ***current SDM***.

A possible view of the SSS - in Execute mode is given below.

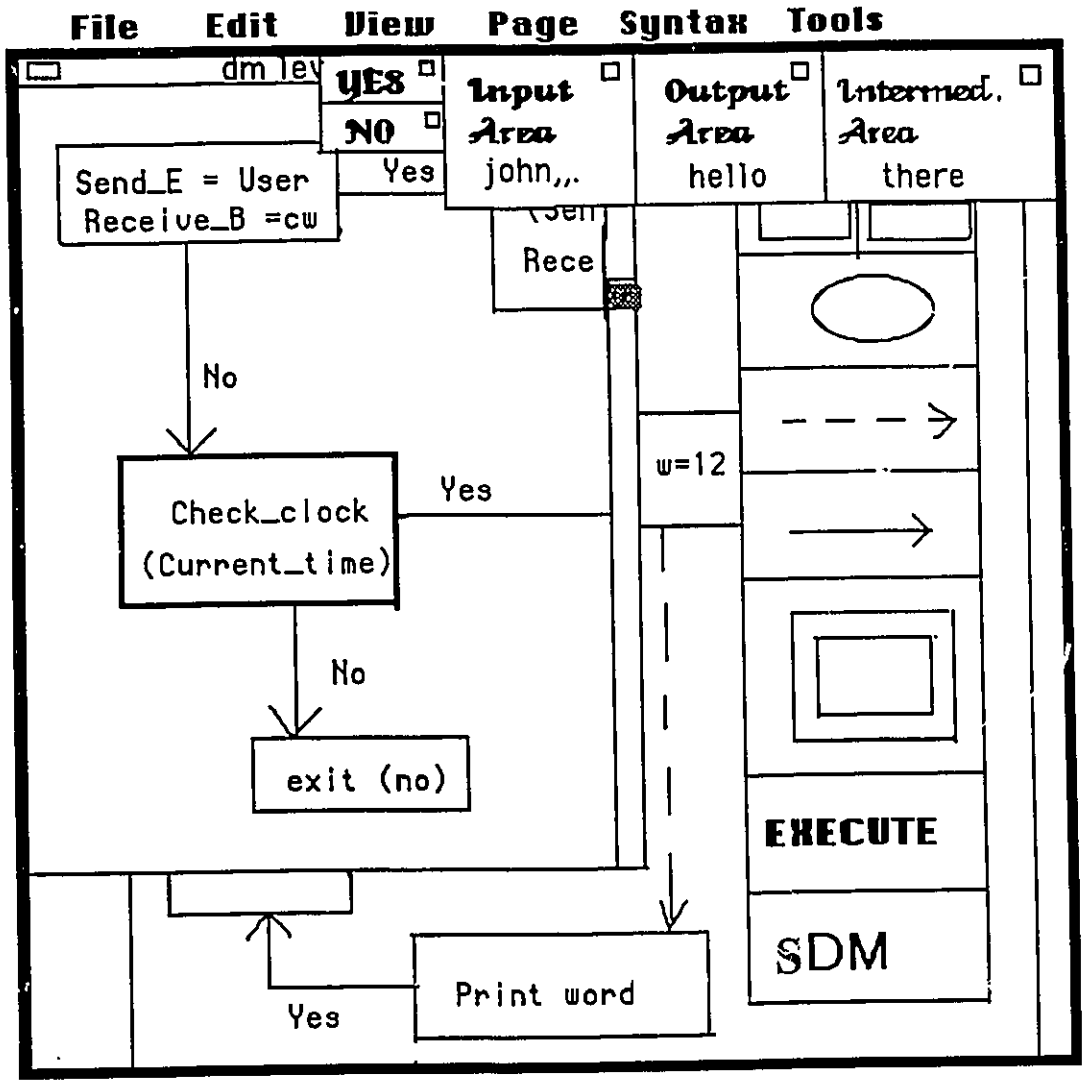


FIGURE 19 SSS - Execute Mode

5.2.6 Trace Capabilities in SDM'S

Another aspect implemented in DMs is the trace capability. *Traces are recorded paths of execution, which allow readability of SDMs. They are affected by the recursion, concurrency and non determinism of SDMs. Traces are flexible in the sense that they can be bounded by a specific Level of the SDM, Bounded Trace, or can be extended to the overall design Unbounded Trace, in the case of a Complete SDM with more than one level.* A Complete SDM with only one Level has only Bounded Traces. For the purpose of the discussion, the following terms are defined:Session : Indicate one execution path from start to end of a SDM (Complete SDM) Finite Trace : Trace that does not contain a cycle.Infinite Trace : Trace that contains a cycle. When dealing with sequential SDMs, a Session has exactly one finite trace. The set of Sessions is finite, since it is possible to list all possible paths from beginning to end of the SDM. When dealing with recursion, concurrency and non determinism, a Session has more than one trace.

These traces can be finite or infinite. The set of sessions is infinite. The reason why a session contains more than one trace is because when we encounter concurrency, the Trace is divided into a number of SDMs running simultaneously. ***That is, the Trace is split into the number of terms present in the 'parallel' expression.*** The SSS provides coverage information to the user. The user has options similar to branch coverage, condition coverage and decision coverage. The SSS prompts the user for input and depending on yes/no answers, takes the appropriate paths. User has the responsibility of changing input, output, intermediate areas. Thus , by modifying yes/no answers to nodes, the user can alter the path taken and attain 100 % coverage. Once, coverage information is requested by the user - those branches that were traversed are highlighted, so the user knows which branches were / were not traversed and make appropriate changes to the input area. ***The trace capability and coverage information, also provides the user with information as when to stop testing the design.***

A possible view of the SSS - in Trace mode is given below.

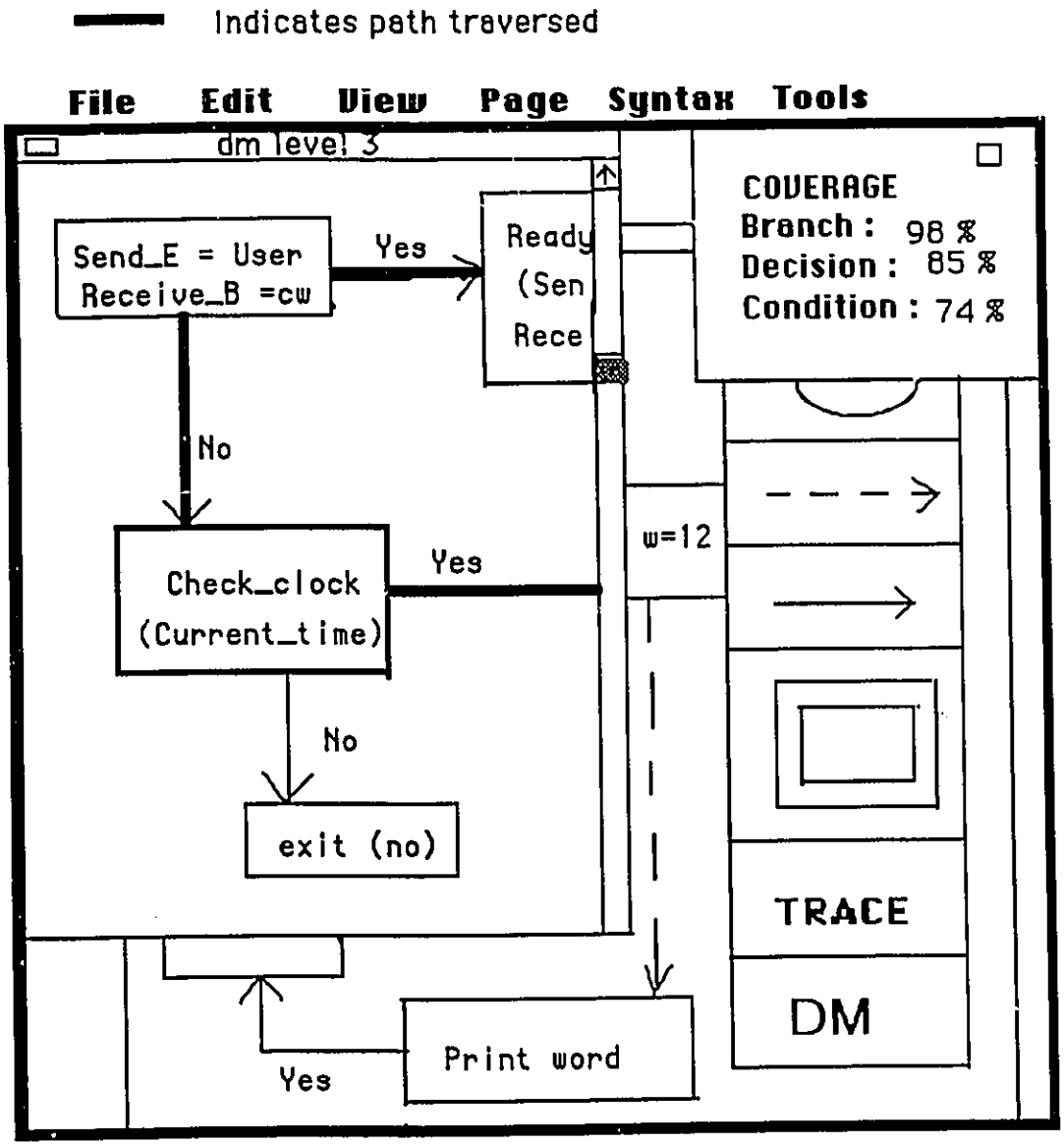


FIGURE 20 SSS in Trace Mode

5.2.7 Implementation of SSS Prototype

A prototype of the SSS has been implemented in C. It has been written to work for any simple design machine. Ideally, the SSS should have three basic functions

1. Create a SDM

2. Execute a SDM

3. Tracing a SDM

In the prototype, all three phases have been implemented. The creation of an SDM , ideally, would require a graphical user interface from which information would be retrieved. However, for this thesis, a command line version is implemented. Data - in a predefined form is read from a file.
e.g..
1 Get_next_character 2 2

The program reads this as

state : 1
description : Get_next_character
transition on yes : state 2
transition on no : state 2

Then there will be another line of data for state 2, in the file.*As mentioned earlier, the data could be read from a file or in the case of a Graphical User Interface, from the screen.* The user inputs data in a file, as explained above. The program reads in every line in the file and stores them in predefined array locations called
1. state_id [i] ----> stores all the ' state id's'.

2. eng_st_des [i] ----> stores all the 'english state descriptions'.

3. nsify [i] ----> stores all 'next transition if yes'.

4. nsifn [i] ----> stores all 'next transitions if no'.

Starting from state 1, the program prints the 'english state description' and asks the user to either *a) input yes/no* OR*b) if the **transition on 'yes' is the same as the transition on 'no' for that state - the program asks the user to make changes to the input area, output area or buffer.*** When the transitions are the same for yes and no - this implies that it is a simple transition, and usually we make changes to the input, output, buffer areas. This is iteratively performed. When changes have to be made to the input / output / buffer areas, the previously stored data is printed out, so that the user doesn't have to keep track. All relevant data is stored in a database where a every state has an associated field, indicating whether it is a decision, event or invocation node. Appropriate execution is performed depending on the type of node. Inconsistencies can therefore be detected.

Every time a state (node) is executed, an appropriate counter is incremented. At the end (once an exit node is reached), the program prints out counts for every state (node). With this, the user knows which states (nodes) have been executed and which haven't. Once all the nodes have been exercised at least once, he could stop testing - whether the SDM conforms to the expected design or not. An industrial strength interpreter will have more sophisticated coverage capabilities like branch, decision, condition coverage.

Appendix B lists the SDM for PCS and Appendix C lists the output traces for the SDM's in Appendix B. The code for the prototype has been written in such a way as to retrieve SDM information from a file instead of from the graphical editor. Once trace information is requested from the prototype, the user is asked to input the filename. The file is then executed and traces are generated.e.g. of the file :

```
I1 Start_Outcall_Session  2 3
2 NP_Active 7  9
```

When broken down, the first line means
State I1 is Start_Outcall_Session.
I denotes the node to be an invocation node.
If the user wants to start the outcall session , then we go to state 2 else to state 3.
If the node is not an invocation node the both the yes and no branches have the same value.
e.g.

I1 Accept_hangup 3 3

Traces are generated by executing this file, line by line. Each line is executed starting from the first. Number of lines executed / total number of lines in the file * 100 gives the trace percentage.

5.3 Comparison of Existing Tools Vs New SSS Tool

Appendix C lists traces for part of the SDM mapping for PCS in Appendix B. The issue I came across while trying to build a tool, is the bit mapped feature of all graphical editors. For a high level graphical editor, each mouse click would have an interpretation in terms of what the user is trying to achieve. If the user is trying to create an SDM or execute or trace an SDM, all he/she needs to do is to click on an options panel. To create an SDM, the basic constructs will be provided, all the user needs to do is to click on the specific construct he/she is looking for and thereby create the SDM.To do this would require extensive coding in either XRUNNER or XVIEW for the graphical editor which is out of scope for this thesis. This is why I have a simple command line version to do this. ***My aim is more to show the methodology of an industrial type of graphical editor.*** The MacAnalyst and Designer use STD's or State Transition Diagrams which has a different syntax than that of SDM's. ***SDM's and STD's are quite similar in their logical breakdown which is why they can be used.***

Chapter 6

Conclusions And Suggestions For Further Research

In this thesis we have seen that the advantages in representing high level user service scenarios via a semi - formal testable design language are several. This provides for a more ***extensive coverage of designs*** both from the ***user perspective*** and the ***system perspective***. We are also aware that errors caught at specification / design time are less costly than those uncovered after coding. ***Inconsistent scenarios can be detected earlier on in the development cycle*** if we use grey box use cases for design. This thesis used a semi-formal testable design language called ***Simple Design Machines(SDM's)*** , an extension of Design Machines - which lends itself to the generation of grey box use cases. The grey box strategy given here is automatable, and appears to be effective for validation of object-oriented designs (for example GUI's) as well as for PCS, but this is an area for furthur study.

First, the basic idea behind SDM's were presented, and various other approaches for design capture and validation of specifications were surveyed and compared to SDM's. Next, SDM's were defined and illustrated on a simple example. Associated with SDM's is a straight forward design synthesis technique, they represent normal use cases first, then fill in exception handling transitions. This approach gives a method of generating discriminating use cases called grey box analysis. In grey box validation of specifications, the SDM is traced in two ways.

i) each level independently
ii) all depth first (acyclic) paths

To test the validity of the SDM/ Grey Box approach, we applied it to a case study of significant complexity, namely SDM/ Grey-box Validation of natural language specifications for Personal Communication Systems (PCS).

PCS was chosen as a realistic case study because it is fairly large, complex and interesting. PCS Systems also respond to combinations of inputs which make it ideal to represent using SDM's. The goal of the case study was to see if inconsistencies could be found in the natural language specification for PCS using SDM's. To facilitate the case study, we considered two approaches, namely the use of existing tools versus building of an appropriate tool for SDM representation and trace management. The design decisions and resulting prototype called the Symbolic Scenario Selector are described in Chapter 5. The prototype SSS, in addition to expediting the Case Study, provided support for proof-of-concept of the SDM/Grey-Box Validation approach.

The Case Study supported our hypothesis that SDM/Grey Box Validation is useful for understanding natural language specifications from a design perspective, to a point where some errors and ambiguities can be (and were) detected in the natural language specification.

A number of areas and extensions exist for furthur study. A Graphical User Interface needs to be added, to make the SSS easier and more effective to use. Since the introduction to SDM has to be a gradual process, we should place special attention to on-line help and training aids. One application used could be Hypercard. Hypercard makes it easy to ***customize and program*** Macintosh graphic user interfaces. This means that training material can contain text and drawings describing various elements of SDMs. Other parts can contain elements of SDM recommendations and guidelines, ***all interconnected on numerous points for easy navigation through these documents.*** We will need to implement links which will enable the user of graphical editors to access needed hypercard stacks from the editors.

One stack contains help material that describes the use of the editors. Other stack material should be used for SDM training material. Hypercard could be used for other things as well. Since it is used by Oracle relational databases for Macintosh machines as a user interface, we could try and integrate those elements in our tool set.

A process in a SDM could be mapped onto a process in SDL, if there is a direct mapping between extended finite state machines and SDM's. The concept of a process is similar both in SDM's and SDL. It is the design component and the transitions between the design component that differs. In SDM's these are nodes and yes/no arcs while in SDL these are design component and communication channels. What needs to be investigated is how SDM's can model transitions each of which is firable on reception of a specific signal.

In conclusion, I would like to stress that SDM's would not be ideal for every type of system. ***Only those applications that respond to complex combinations of context, conditions and stimuli are SDM's most effective..***
In SDM's, the action from the state is dependent on its context within a particular service or functional path. Each design purpose is captured by the normal design path and all corresponding exception paths. The context of the system state is important. Even if the design is incomplete, the checking of that design will be complete *up to* the selected testability level. In SDM's , the history and topographical position of the state is important. How it got to that state depends on all the previous states. FSM's do not capture this.

More experience and tool support is needed for SDM's to make it an effective methodology for practical use in industry.

REFERENCES

[1] A. Ghedamsi, R. Dssouli, G.v.Bochmann (Univ. of Montreal, Canada) "Automated Test Case Selection based on Test Coverage Metrics" - 5th International Workshop on Protocol Test Systems (IWPTS '92), September 77-88, 1992, Montreal, Quebec, Canada.

[2] A.M Davis, " A Comparison of Techniques for the Specification of External System Behavior" Communications of the ACM, September 1988, pp.1098 - 1115.

[3] Alford, M. A requirements engineering methodology for real time processing requirements. IEEE Trans. Softw. Eng. 3, 1 (Jan. 1977), 60-69.

[4] Barry W. Boehm, TRW, "Verifying and Validating Software Requirements and Design Specifications", IEEE Software, Volume 1, Number 1, January 1984, pages 75-88.

[5] Basili, V., and Weiss, D. Evaluation of a software requirements document by analysis of change data. In Proceedings of the 5th International Conference on Software Engineering (San Diego, CA, March 9-12). pp. 314-323, IEEE Press. Wash., DC, 1981.

[6] Beizer, Boris. Software Testing Techniques, Second Edition, Van Nostrand Reinhold, New York, 1990

[7] Berzins, V., and Gray, M. Analysis and design in MSG 84: Formalizing functional specifications. IEEE Trans. Softw. Eng. 11, 8 (Aug. 1985), 657-670.

[8] Boehm, B. Software Engineering. IEEE Trans. Comput. 25, 12 (Dec. 1976), 1226-1241.

[9] Buhrke, R., et. al. Design choices for a large PCM switch with multiprocessor control. In Proceedings of the International Switching Symposium(May 1979). pp 134 - 146.

[10] Caine, S., and Gordon, E. PDL - A tool for software design. In Proceedings of the AFIPS National Computer Conference, Vol 44 (Anaheim , CA, 1975), AFIPS Press. Montvale, New Jersey, pp. 271-276.

[11] CCITT: Draft Recommendation E.202, "Network Operational Principles for Future Public Mobile Systems and Services," TD 11 (Rev. 1), WPII/1 Joint Rapporteurs Meeting, Cantebury, UK, Oct 14-18, 1991.

[12] Cecelia Geldrez, Design Machines, University of Ottawa - Masters Thesis

[13] Charette, R. Software Enginnering Environments. McGraw Hill, New York, 1986. pp. 233 - 249.

[14] Chvalovsky, V. Decision tables. Softw. Pract. Exper. 13 (1983). 423-429.

[15] Daly, E. Management of software development. IEEE Trans. Softw. Eng. 3, 3 (May 1977), 230-242.

[16] Davis, A. A taxonomy for the early stages of the software development life cycle. J. Syst. Softw. 8,4 (Sept. 1988).

[17] ETSI Technical Report: DTR NA - 70210, Network Aspects: Universal Personal Telecommunication (UPT) Suppl. Serv. Feb 1992.

[18] Fagan, M. Design and Code inspections and process control in the development of programs. IBM Report IBM-SDD-TR-21.572. Dec. 1974, IBM, Bethesda, Maryland.

[19] Ferenc Belina , Dieter Hogrefe, "The CCITT - Specification and Description Language SDL" - North-Holland, Computer Networks and ISDN Systems 16. Pages 311 - 340.

[20] H. Auspurg, "Mobile Telephone: an ideal IN Application", Telecom Report 12, No. 5, 1989.

[21] Harel, D. StateCharts: A visual formalism for complex systems. Scientific Computing Programme 8 (1987).

[22] Hatley, D. and Pirbhai, I., Strategies for Real Time System Specification.New York: Dorset House, 1987 pp 23- 34.

[23] IEEE A guide for software requirements specifications. IEEE / ANSI Standard 830-1984. Institute of Electrical and Electronics Engineers, New York , 1984.

[24] J. Regnier, W. H. Cameron - Personal Communication Services - The New POTS, Bell Northern Research.

[25] Kerola, P., and Freeman, P. A comparison of life cycle models. In Proceedings of the fifth International Conference on Software Engineering (San Diego, CA, March 9-12), pp. 90-99, IEEE Press. Wash., DC, 1981.

[26] M. Di Concetto, G. Marino, E. Merli & F. Zizza (Italtel SIT & CRL, Italy),"An Approach to the Test of an ATM based Signalling Application" - 5th International Workshop on Protocol Test Systems (IWPTS '92), September 28-30, 1992, Montreal, Quebec, Canada.

[27] M. McAllister, S.T Vuong &, J. Alillovic-Curgus (UBC Canada), "Automated Test Case Selection based on Test Coverage Metrics" - 5th International Workshop on Protocol Test Systems (IWPTS '92), September, 64-76, 1992, Montreal, Quebec, Canada.

[28] Marick Brian, The Craft of Software Testing - Subsystem Testing Including Object Based and Object Oriented Testing, Prentice Hall Series in Innovation Technology, 1995.

[29] Michael Koblentz (Bellcore , USA)
"Issues in Broadband ISDN Testing" - 5th International Workshop on Protocol Test Systems (IWPTS '92), September 77-88, 1992, Montreal, Quebec, Canada.

[30] Moret, B. Decision trees and diagrams. ACM Computing Survey 14, 4 (Dec. 1982), 593-623.

[31] "NTT: Call the Person, not the Phone", Telephony, March 5, 1990. pp 23 - 34.

[32] Probert, Geldrez, Design Machines, University of Ottawa, Ontario, Canada.

[33] Probert, Software Quality Engineering, Volume 1-5, University of Ottawa, July, 1994.

[34] R.J. G. MacNamee, "UMTS - A Personal and Mobile Communication Service ", ITU Com., Geneva, 1989.

[35] Reed Rick, Faergemand Ove : SDL '91 Evolving Methods, Proceedings of the Fifth SDL Forum Glasgow, Scotland, UK, 29 September - 4 October, 1991.

[36] Reline, S., and Gordon, E. PDL - A tool for software design. In Proceedings of the AFIPS National Computer Conference, Vol 44 (Anaheim , CA, 1975), AFIPS Press. Montvale, New Jersey, pp. 271-276.

[37] Royce, W. Managing the development of large software systems: Concepts and techniques. Reprinted in Proceedings of the ninth International Conference on Software Engineering (Monterey, Calif., March 30 - April 2nd), pp 328-338, IEEE Press. Wash., DC, 1987.

[38] Ward, P. and Mellor, S., Structured Development for Real Time Systems Englewood Cliffs, N.J: Prentice Hall, 1985. pp 56 - 67.

[39] Wasserman, A. Extending State Transition Diagrams for the specification of human computer interaction. IEEE Trans. Softw. Eng. 11, 8 (Aug. 1985) 669-713.

[40] Yourdan, E. and Constantine, L., Structured Design, Englewood Cliffs, NJ, Prentice Hall, 1979.

APPENDIX A :

PCS SPECIFICATION

IN MONDEL

This appendix gives a formal, high-level specification of the major objects, attributes and operations in PCS and supports some of the discussion in sections 3.3 and 4.2.

The PCS Specification has been written in Mondel, an Object Oriented Specification Language. Three objects defined are

1. Entity
2. Relationship
3. User.

The User performs the operation.
The following operations are possible:

1. Start_Outcall_Session
2. Terminate_Outcall_Session
3. Terminate_Other_Outcall_Session
4. Start_Incall_Session
5. Terminate_Incall_Session
6. Terminate_Other_Incall_Session
7. Do_Serie
8. Call
9. Respond
10. Hang_up
11. Signal_Continue
12. Dial

The following are internal functions:

1. Send_M (Send Message)
2. Receive_M (Receive Message)
3. Connect_Response
4. Disconnect_Indication
5. Response_Indication
6. Outcall_Session_Time_Out
7. Incall_Session_Time_Out.

The following definitions will also prove useful

NP : port (equivalent to telephone location)
SN : Subcriber Number (equivalent to telephone number but not specific to particular telephone)
PIN : The private number used by a subscriber to gain access to controllable aspects of his service.
FP : Feature Profile. The set of features assigned to a given subscriber at any given point in time.

Each Operation is subdivided into smaller operations or functions. Parameters are passed between these functions. Similar to any programming language.

The object USER has the following attributes:
name
operations with parameters
user responses
(parallel or sequential mode)
Behaviours are defined by procedures which may use other procedures. eg. user may respond with Outcall_Session_Idle which is defined by a procedure (O_S_I) which uses a reference procedure valid np.
Parameters (eg.NP) are defined for procedures. Relationships that group parameters according to a functional purpose are also defined. Additional functions are provided which allow the description of normal use cases and exception cases.

```
/*****************************************************************************

          PCS  SPECIFICATION

*****************************************************************************/

unit spec =

type ENTITY = persistent endtype ENTITY


type RELATIONSHIP = persistent endtype RELATIONSHIP


type USER = persistent with

  name : string;

  operation

              (***** Np           : the port **********)
              (***** Sn           : the subscriber number*****************)
              (******Sn_Calling   : the subscriber number of the caller)
              (***** Pin          : the access PIN ******************)
              (***** Active_Np    : the active port *********)
              (***** Service_Type : the service type **********)
              (***** Duration     : the duration ****)
              (***** Action_Sn    : the SN of the action ***********)


       Start_Outcall_Session

       ( Np           : string ;
         Sn           : string ;
         Pin          : string ;
         Action_Np    : string ;
         Service_Type : string;
         Duration     : integer
        ) atomic ;


       Terminate_Outcall_Session

       ( Np           : string ;
         Sn           : string ;
         Pin          : string ;
         Action_Np    : string ;
         Service_Type : string
        ) atomic ;

       Terminate_Other_Outcall_Session

       ( Np           : string ;
         Sn           : string ;
         Pin          : string ;
         Service_Type : string
        ) atomic ;

       Start_Incall_Session

       ( Np           : string ;
         Sn           : string ;
         Pin          : string ;
         Action_Np    : string ;
         Service_Type : string ;
         Duration     : integer
        ) atomic ;

       Terminate_Incall_Session

       ( Np           : string ;
         Sn           : string ;
         Pin          : string ;
         Service_Type : string
        ) atomic ;

       Terminate_Other_Incall_Session

       ( Np           : string ;
         Sn           : string ;
         Pin          : string ;
         Action_Sn    : string ;
         Service_Type : string
        ) atomic ;

       Do_Serie

       ( Np           : string ;
         Sn           : string ;
         Pin          : string ;
         Service_Type : string
        ) atomic ;

       Call

       ( Np           : string ;
         Sn_Calling   : string ;
         Sn           : string ;
         Service_Type : string
        ) atomic ;

       Respond

       ( Np           : string
       ) atomic;




             (***** Np           **************)
             (***** Fonction     ************)
             (***** Sn           *****************)
             (***** Message      **************************)


       Hang_Up

       ( Np           : string ;
         Fonction     : string
        ) atomic ;

       Signal_Continu

       ( Np           : string
        ) atomic ;

       Dial
```

```
        ( Np          : string ;
          Sn          : string
         ) atomic ;

        Send_M

        ( Np          : string ;
          Message     : string
         ) ;



              (***** Np           :  ************)
              (***** Sn           :  **************)
              (***** Service_Type :  ***********)
              (***** Message      :  **************)
              (***** Response     :  *****)
              (******Sn_billed    :  ******************)


        Receive_M

        ( Np          : string ;
          Message     : string
         ) ;

        Connect_Response

        ( Np          : string ;
          Response    : string ;
          Sn_billed   : string
         ) ;

        Disconnect_Indication

        ( Np          : string
         ) ;


        Response_Indication

        ( Np          : string
         ) ;



        Outcall_Session_Time_Out

        ( Np          : string ;
          Sn          : string ;
          Service_Type : string
         ) ;


        Incall_Session_Time_Out

        ( Sn          : string ;
          Service_Type : string
         );


behavior

        parallel
                Outcall_Session_Idle;
        and
                Incall_Session_Idle;
        and
                Outcall_Session_Abort;
        and
                Incall_Session_Abort;
        and
                Call_Idle;
        and
                Response_Idle;
        and
                Serie_Idle;

        end; (parallel)


where

        procedure valid_np( NpValue : string ): NP =

                ifexist np : NP suchthat np.value = NpValue
                then
                   return np;
                else
                   return nil ;
                end;

        endproc valid_np



        procedure Outcall_Session_Idle =

                accept
                        Start_Outcall_Session

                        do
                           define np = valid_np( Np )
                           in
                              if np = nil
                              then
                                 writeln "  " ;
                                 writeln "START_OUTCALL_SESSION INVALID (PORT IN
                                 return;
                                 Outcall_Session_Idle ;
                              else
                                 define Etat  = np!Off_Hook(self)
                                 in
                                 if Etat  = "busy"
                                 then
                                    writeln "  " ;
                                    writeln "START_OUTCALL_SESSION INVALID (PORT
                                    return;
                                    Outcall_Session_Idle ;
                                 else
                                    np!Signal_Pcs_Code;
                                    define Result_Authentication = np!Signal_Aut
                                    in
                                    if Result_Authentication = 1
                                    then
                                       writeln "  " ;
                                       writeln " START_OUTCALL_SESSION INVALID (
                                       return ;
                                       Accept_Hang_Up_Start_Outcall_session
                                           (Np,Action_Np,Sn,Service_Type,"Abort"
```

```
                                else
                                  if Result_Authentication = 2
                                  then
                                     writeln "  " ;
                                     writeln " START_OUTCALL_SESSION INVALID
                                     return;
                                     Accept_Hang_Up_Start_Outcall_session
                                        (Np,Action_Np,Sn,Service_Type,"Abort
                                  else { Result_Authentication = 0 }
                                     define Result_Signal = np!Signal_Start_
                                                          ( Action_Np, Service
                                     in
                                     if Result_Signal = 1
                                     then
                                        writeln "  " ;
                                        writeln " START_OUTCALL_SESSION INVA
                                        return;
                                        Accept_Hang_Up_Start_Outcall_session
                                          (Np,Action_Np,Sn,Service_Type,"Abo
                                     else
                                        if Result_Signal = 2
                                        then
                                           writeln "  " ;
                                           writeln " START_OUTCALL_SESSION I
                                           return ;
                                           Accept_Hang_Up_Start_Outcall_sess
                                            (Np,Action_Np,Sn,Service_Type,"A
                                        else
                                          if Result_Signal = 3
                                          then
                                             writeln "  " ;
                                             writeln " START_OUTCALL_SESSION
                                             return;
                                             Accept_Hang_Up_Start_Outcall_se
                                                (Np,Action_Np,Sn,Service_Typ
                                          else
                                            if Result_Signal = 4
                                            then
                                               writeln "  " ;
                                               writeln " START_OUTCALL_SESSI
                                               return;
                                               Accept_Hang_Up_Start_Outcall_
                                                 (Np,Action_Np,Sn,Service_Ty
                                            else
                                               writeln " " ;
                                               writeln " START_OUTCALL_SESSI
                                               return;
                                               Accept_Hang_Up_Start_Outcall_
                                                 (Np,Action_Np,Sn,Service_Typ
                                            end; { if Result_Signal = 4 }
                                          end; { if Result_Signal = 3 }
                                        end; { if Result_Signal = 2 }
                                     end; { if Result_Signal = 1 }
                                     end; {define Result_Signal }
                                  end; { if Result_Authentication = 2 }
                                end; { if Result_Authentication = 1 }
                                end; { define Result_Authentication }
                              end; { if Etat }
                              end; { define Etat }
                           end; { if np = }
                        end ; { define }

              end; {accept}

       endproc Outcall_Session_Idle
```

```
       procedure Accept_Hang_Up_Start_Outcall_session( port       : string ;
                                                       concern_np : string ;
                                                       concern_sn : string ;
                                                       concern_st : string ;
                                                       mode       : string ) =

                 accept
                     Hang_Up
                      provided Np       = port and
                               Fonction = "Start_Outcall_Session"
                      do
                       define np = valid_np(port)
                       in
                          np!On_Hook;
                       end; { define }
                       writeln "   ";
                       writeln " HANG_UP ACCEPTEDD" ;
                       writeln "   " ;

                       if mode = "Abort"
                       then
                         return ;
                         Outcall_Session_Idle;
                       else
                         return ;
                         parallel
                            Outcall_Session_Idle;
                         and
                           Outcall_Session_Active( concern_np, concern_st, co
                         end; {parallel}
                       end; { if mode }

                 end; { accept }

       endproc  Accept_Hang_Up_Start_Outcall_session


       procedure Outcall_Session_Active( concern_np : string ;
                                         concern_st : string ;
                                         concern_sn : string ) =

              accept
                     Terminate_Outcall_Session
                     provided concern_np = Action_Np and
                              concern_sn = Sn and
                              concern_st = Service_Type
                     do
                        define np  = valid_np( Np )
                        in
                           if np = nil
                           then
                              writeln "  " ;
                              writeln "TERMINATE_OUTCALL_SESSION INVALID (POR
                              return;
                              Outcall_Session_Active(concern_np,concern_st,co
                           else
                              define  Etat =  np!Off_Hook(self)
                              in
                              if Etat = "busy"
                              then
                                 writeln "  " ;
                                 writeln "TERMINATE_OUTCALL_SESSION INVALID (
                                 return;
                                 Outcall_Session_Active(Action_Np,Service_Typ
                              else
```

```
                        np!Signal_Pcs_Code;
                        define Result_Authentication =  np!Signal_Au
                        in
                        if Result_Authentication <> 0
                        then
                           writeln *  * ;
                           writeln * TERMINATE_OUTCALL_SESSION INVAL
                           return;
                           Accept_Hang_Up_Terminate_Outcall_session(
                        else { Result_Authentication = 0 }
                           define Result_Signal =  np!Signal_Termina
                                          ( Action_Np, Service_Type
                           in
                           if Result_Signal <> 0
                           then
                              writeln *  * ;
                              writeln * TERMINATE_OUTCALL_SESSION IN
                              return;
                              Accept_Hang_Up_Terminate_Outcall_sessi
                                  (Np,concern_np,concern_st,concern_
                           else { Result_Signal = 0 }
                              writeln *  * ;
                              writeln * TERMINATE_OUTCALL_SESSION  A
                              return;
                              Accept_Hang_Up_Terminate_Outcall_sessi
                                 (Np,concern_np,concern_st,concern_s
                           end ; { if Result_Signal <> 0 }
                           end; { define }
                        end; { if Result_Authentication <> 0 }
                        end; { define }
                     end; { if Etat }
                     end; { define }
                  end; {  if np }
               end; { define }
      or
            Outcall_Session_Time_Out
            provided concern_np = Np and
                     concern_st = Service_Type and
                     concern_sn = Sn
            do
                    writeln * *;
                    return;

      end; {accept}

endproc Outcall_Session_Active


procedure Accept_Hang_Up_Terminate_Outcall_session( port       : string
                                                    concern_np : string
                                                    concern_st : string
                                                    concern_sn : string
                                                    mode       : string

         accept
             Hang_Up
              provided Np       = port and
                       Fonction = "Terminate_Outcall_Session"
              do
               define np = valid_np(port)
               in
                  np!On_Hook;
               end; { define }
               writeln *   *;
               writeln * HANG_UP ACCEPTED" ;
               writeln  *   * ;
               if mode = "Abort"
                then
                 return ;
                 Outcall_Session_Active(concern_np,concern_st,concern_
                else
                  return;
                end;

          end; { accept }

endproc  Accept_Hang_Up_Terminate_Outcall_session


procedure Incall_Session_Idle =

        accept
                Start_Incall_Session

                do
                   define np = valid_np( Np )
                   in
                      if np = nil
                      then
                         writeln *  * ;
                         writeln "START_INCALL_SESSION INVALID (PORT INV
                         return;
                         Incall_Session_Idle ;
                      else
                         define Etat = np!Off_Hook(self)
                         in
                         if Etat  = "busy"
                         then
                            writeln *  * ;
                            writeln "START_INCALL_SESSION INVALID (PORT
                            return;
                            Incall_Session_Idle ;
                         else
                            np!Signal_Pcs_Code;
                            define Result_Authentication = np!Signal_Aut
                            in
                            if Result_Authentication = 1
                            then
                               writeln *  * ;
                               writeln * START_INCALL_SESSION INVALID (A
                               return;
                               Accept_Hang_Up_Start_Incall_session
                                   (Np,Sn,Service_Type,"Abort");
                            else
                              if Result_Authentication = 2
                              then
                                 writeln *  * ;
                                 writeln * START_INCALL_SESSION INVALID
                                 return;
                                 Accept_Hang_Up_Start_Incall_session
                                    (Np,Sn,Service_Type,"Abort");
                              else { Result_Authentication = 0 }
                                 define Result_Signal = np!Signal_Start_
                                                ( Action_Np, Service
                                 in
                                 if Result_Signal = 1
                                 then
                                    writeln *  * ;
                                    writeln * START_INCALL_SESSION INVAL
                                    return;
                                    Accept_Hang_Up_Start_Incall_session
                                      (Np,Sn,Service_Type,"Abort");
```

```
                                else
                                   if Result_Signal = 2
                                   then
                                      writeln "  " ;
                                      writeln " START_INCALL_SESSION IN
                                      return;
                                      Accept_Hang_Up_Start_Incall_sessi
                                       (Np,Sn,Service_Type,"Abort");
                                   else
                                     if Result_Signal = 3
                                     then
                                        writeln "  " ;
                                        writeln " START_INCALL_SESSION
                                        return;
                                        Accept_Hang_Up_Start_Incall_ses
                                          (Np,Sn,Service_Type,"Abort")
                                     else
                                        if Result_Signal = 4
                                        then
                                           writeln "  " ;
                                           writeln " START_INCALL_SESSI
                                           return;
                                           Accept_Hang_Up_Start_Incall_
                                             (Np,Sn,Service_Type,"Abort
                                        else
                                           writeln " " ;
                                           writeln " START_INCALL_SESSI
                                           return;
                                           Accept_Hang_Up_Start_Incall_
                                            (Np,Sn,Service_Type,"Accept
                                        end; { if Result_Signal = 4 }
                                     end; { if Result_Signal = 3 }
                                   end; { if Result_Signal = 2 }
                                end; { if Result_Signal = 1 }
                                end;
                             end; { if Result_Authentication = 2 }
                           end; { if Result_Authentication = 1 }
                           end;
                         end; { if Etat }
                         end;
                      end; { if np = }
                   end ; { define }

         end; {accept}

endproc Incall_Session_Idle



procedure Accept_Hang_Up_Start_Incall_session(  port       : string ;
                                                concern_sn : string ;
                                                concern_st : string ;
                                                mode       : string ) =

            accept
                Hang_Up
                 provided Np       = port and
                          Fonction = "Start_Incall_Session"
                 do
                  define np = valid_np(port)
                  in
                     np!On_Hook;
                  end; { define }
                  writeln "   ";
                  writeln " HANG_UP ACCEPTED" ;
                  writeln "   " ;
                  return ;
                  if mode = "Abort"
                  then
                    return;
                    Incall_Session_Idle;
                  else
                    return;
                    parallel
                       Incall_Session_Idle;
                    and
                      Incall_Session_Active( concern_st, concern_sn);
                    end; {parallel}
                  end; { if mode }
               end; { accept }

endproc  Accept_Hang_Up_Start_Incall_session



procedure Incall_Session_Active(  concern_st : string ;
                                  concern_sn : string ) =

        accept
                Terminate_Incall_Session
                provided concern_sn = Sn and
                         concern_st = Service_Type
                do
                   define np  = valid_np( Np )
                   in
                      if np = nil
                      then
                         writeln "  " ;
                         writeln "TERMINATE_INCALL_SESSION INVALID (PORT
                         return;
                         Incall_Session_Active(Service_Type,Sn) ;
                      else
                         define Etat =  np!Off_Hook(self)
                         in
                         if Etat = "busy"
                         then
                            writeln "  " ;
                            writeln "TERMINATE_INCALL_SESSION INVALID (P
                            return;
                            Incall_Session_Active(Service_Type,Sn);
                         else
                            np!Signal_Pcs_Code;
                            define Result_Authentication =  np!Signal_Au
                            in
                            if Result_Authentication <> 0
                            then
                               writeln "  " ;
                               writeln " TERMINATE_INCALL_SESSION INVALI
                               return;
                               Accept_Hang_Up_Terminate_Incall_session(N
                            else { Result_Authentication = 0 }

                               define Result_Signal =  np!Signal_Termina
                                             ( Service_Type )
                               in
                               if Result_Signal <> 0
                               then
                                  writeln "  " ;
                                  writeln " TERMINATE_INCALL_SESSION INV
                                  return;
                                  Accept_Hang_Up_Terminate_Incall_sessio
                                      (Np,concern_st,concern_sn,"Accept"
```

```
                                else { Result_Signal = 0 }
                                   writeln "  " ;
                                   writeln " TERMINATE_INCALL_SESSION  AC
                                   return;
                                   Accept_Hang_Up_Terminate_Incall_sessio
                                      (Np,concern_st,concern_sn,"Accept")
                                end ; { if Result_Signal <> 0 }
                                end;
                             end; { if Result_Authentication <> 0 }
                             end;
                          end; { if Etat }
                          end;
                       end; {  if np }
                    end; { define }
        or

                Incall_Session_Time_Out
                provided
                        concern_st = Service_Type and
                        concern_sn = Sn
                do
                       writeln " ";
                       return;

        end; {accept}

endproc Incall_Session_Active


procedure Accept_Hang_Up_Terminate_Incall_session(  port       : string
                                                    concern_st : string
                                                    concern_sn : string
                                                    mode       : string

            accept
                Hang_Up
                 provided Np       = port and
                          Fonction = "Terminate_Incall_Session"
                 do
                  define np = valid_np(port)
                  in
                     np!On_Hook;
                  end; { define }
                  writeln "   ";
                  writeln " HANG_UP ACCEPTED" ;
                  writeln "   " ;
                  if mode = "Abort"
                  then
                   return ;
                   Incall_Session_Active(concern_st,concern_sn) ;
                  else
                   return;
                  end;

            end; { accept }

endproc  Accept_Hang_Up_Terminate_Incall_session



procedure Outcall_Session_Abort =

        accept
                Terminate_Other_Outcall_Session

                do
                define np = valid_np( Np )
                in
                   if np = nil
                   then
                      writeln "  " ;
                      writeln "TERMINATE_OTHER_OUTCALL_SESSION INVALI
                      return;
                      Outcall_Session_Abort ;
                   else
                      define Etat = np!Off_Hook(self)
                      in
                      if Etat  = "busy"
                      then
                         writeln "  " ;
                         writeln "TERMINATE_OTHER_OUTCALL_SESSION INV
                         return;
                         Outcall_Session_Abort ;
                      else
                         np!Signal_Pcs_Code;
                         define Result_Authentication = np!Signal_Aut
                         in
                         if Result_Authentication = 1
                         then
                            writeln "  " ;
                            writeln " TERMINATE_OTHER_OUTCALL_SESSION
                            return;
                            parallel
                              Accept_Hang_Up(Np,"Terminate_Other_Outc
                            and
                              Outcall_Session_Abort ;
                            end;
                         else
                           if Result_Authentication = 2
                           then
                              writeln "  " ;
                              writeln " TERMINATE_OTHER_OUTCALL_SESSI
                              return;
                              parallel
                                Accept_Hang_Up(Np,"Terminate_Other_Ou
                              and
                                Outcall_Session_Abort ;
                              end;
                           else { Result_Authentication = 0 }
                              define Result_Signal = np!Signal_Termin
                                                     (Service_Type )
                               in
                              if Result_Signal = 1
                              then
                                 writeln "  " ;
                                 writeln " TERMINATE_OTHER_OUTCALL_SE
                                 return;
                                 parallel
                                   Accept_Hang_Up(Np,"Terminate_Other
                                 and
                                   Outcall_Session_Abort ;
                                 end;
                              else
                                 if Result_Signal = 2
                                 then
                                    writeln "  " ;
                                    writeln " TERMINATE_OTHER_OUTCALL
                                    return;
                                    parallel
                                      Accept_Hang_Up(Np,"Terminate_Ot
                                    and
                                      Outcall_Session_Abort ;
```

```
                                    end;
                                 else
                                    writeln " " ;
                                    writeln " TERMINATE_OTHER_OUTCALL
                                    return;
                                    parallel
                                      Accept_Hang_Up(Np,"Terminate_Ot
                                    and
                                      Outcall_Session_Abort ;
                                    end;
                                 end; { if Result_Signal = 2 }
                              end; { if Result_Signal = 1 }
                              end;
                           end; { if Result_Authentication = 2 }
                        end; { if Result_Authentication = 1 }
                        end;
                     end; { if Etat }
                     end;
                  end; { if np = }
               end ; { define }

      end; {accept}

endproc Outcall_Session_Abort


procedure Incall_Session_Abort =

      accept
            Terminate_Other_Incall_Session

            do
               define np = valid_np( Np )
               in
                  if np = nil
                  then
                     writeln "  " ;
                     writeln "TERMINATE_OTHER_INCALL_SESSION INVALID
                     return;
                     Incall_Session_Abort ;
                  else
                     define Etat = np!Off_Hook(self)
                     in
                     if Etat  = "busy"
                     then
                        writeln "  " ;
                        writeln "TERMINATE_OTHER_INCALL_SESSION INVA
                        return;
                        Incall_Session_Abort ;
                     else
                        np!Signal_Pcs_Code;
                        define Result_Authentication = np!Signal_Aut
                        in
                        if Result_Authentication = 1
                        then
                           writeln "  " ;
                           writeln " TERMINATE_OTHER_INCALL_SESSION
                           return;
                           parallel
                             Accept_Hang_Up(Np,"Terminate_Other_Inca
                           and
                             Incall_Session_Abort ;
                           end;
                        else
                          if Result_Authentication = 2
                          then
                             writeln "  " ;
                             writeln " TERMINATE_OTHER_INCALL_SESSIO
                             return;
                             parallel
                               Accept_Hang_Up(Np,"Terminate_Other_In
                             and
                               Incall_Session_Abort ;
                             end;
                          else { Result_Authentication = 0 }
                             define Result_Signal = np!Signal_Termin
                                                  (Action_Sn,Service_T
                             in
                             if Result_Signal = 1
                             then
                                writeln "  " ;
                                writeln " TERMINATE_OTHER_INCALL_SES
                                return;
                                parallel
                                 Accept_Hang_Up(Np,"Terminate_Other_
                                and
                                 Incall_Session_Abort ;
                                end;
                             else
                                if Result_Signal = 2
                                then
                                   writeln "  " ;
                                   writeln " TERMINATE_OTHER_INCALL_
                                   return;
                                   parallel
                                     Accept_Hang_Up(Np,"Terminate_Ot
                                   and
                                     Incall_Session_Abort ;
                                   end;
                                else
                                   writeln " " ;
                                   writeln " TERMINATE_OTHER_INCALL_
                                   return;
                                   parallel
                                       Accept_Hang_Up(Np,"Terminate_
                                   and
                                       Incall_Session_Abort ;
                                   end;
                                end; { if Result_Signal = 2 }
                             end; { if Result_Signal = 1 }
                             end;
                          end; { if Result_Authentication = 2 }
                       end; { if Result_Authentication = 1 }
                       end;
                    end; { if Etat }
                    end;
                 end; { if np = }
              end ; { define }

      end; {accept}

endproc Incall_Session_Abort


procedure Call_Idle =

      accept
            Call
```

```
                do
                   define np = valid_np( Np )
                   in
                      if np = nil
                      then
                         writeln "  " ;
                         writeln "CALL INVALID (PORT INVALID) " ;
                         return;
                         Call_Idle;
                      else
                         define Etat = np!Off_Hook(self)
                         in
                         if Etat  = "busy"
                         then
                            writeln "  " ;
                            writeln "CALL INVALID (PORT IS OCCUPIED) " ;
                            return;
                            Call_Idle ;
                         else

                            define Result_Signal = np!Dial( Sn,Sn_Callin
                            in
                            if Result_Signal = 1
                            then
                               writeln "  " ;
                               writeln " CALL INVALID (  SN INVALID) ";
                               return;
                               Accept_Hang_Up (Np,"Call");
                               Call_Idle ;
                            else
                              if Result_Signal = 2
                              then
                                 writeln "  " ;
                                 writeln " CALL INVALID (SERVICE INVALID
                                 return;
                                 Accept_Hang_Up(Np,"Call");
                                 Call_Idle ;
                              else { Result_Signal = 0 }
                                return;
                                parallel
                                   Call_Waiting_Connection_Response( Np)
                                and
                                   Call_Idle;
                                end;
                             end; { if Result_Signal = 2 }
                           end; { if Result_Signal = 1 }
                           end;
                         end; { if Etat }
                         end;
                      end; { if np = }
                   end ; { define }

          end; {accept}

endproc Call_Idle



procedure Call_Waiting_Connection_Response( concern_np : string ) =

          accept
                  Connect_Response
                  provided concern_np = Np and
                          Response = "connection_rejection"
                  do
                     writeln "  ";
                     return;
                     Accept_Hang_Up( concern_np, "Call" );
          or

                  Connect_Response
                  provided concern_np = Np and
                          Response = "connected"
                  do
                    writeln "  ";
                    writeln Sn_billed ;
                    writeln "   " ;
                    return;
                    Call_Application_Connection_Waiting(concern_np);

          end; {accept}

endproc Call_Waiting_Connection_Response


procedure Call_Application_Connection_Waiting( concern_np : string ) =

          choice
             accept
                  Response_Indication
                  provided concern_np = Np
                  do
                    writeln  "  " ;
                    return;
                    Call_Connection( concern_np );
             end; { accept }
          or
                  Accept_Hang_Up(concern_np,"Call");
          end; {choice}

endproc  Call_Application_Connection_Waiting


procedure Call_Connection( concern_np : string )  =

          choice
              accept
                  Disconnect_Indication
                  provided concern_np = Np
                  do
                    writeln  "  ";
                    return;
                    Accept_Hang_Up( concern_np, "Call" );
              end; { accept }
           or
              accept
                  Send_M
                  provided concern_np = Np
                  do
                     define np = valid_np(concern_np)
                     in
                       np!Send_Message(Message);
                       writeln "  " ;
                       writeln " MESSAGE ENVOYE    " ;
                       return;
                       Call_Connection(concern_np);
                      end;
```

```
            end; { accept }

        or
            accept
               Receive_M
               provided concern_np = Np
               do
                    writeln "   " ;
                    writeln Message ;
                    return;
                    Call_Connection(concern_np);
            end; { accept }
        or
                Accept_Hang_Up(concern_np,"Call");

        end; {choice}

endproc Call_Connection



procedure Response_Idle =


        accept

                    Respond
                    do
                        define np = valid_np( Np )
                        in
                           define Etat = np!Off_Hook(self)
                           in
                           if Etat = "busy"
                           then
                              writeln "  ";
                              writeln " RESPOND INVALID (THE PORT IS OCCU
                              return ;
                              Response_Idle ;

                           else
                              writeln "  " ;
                              return;
                              parallel
                                 Response_Connection( Np );
                              and
                                 Response_Idle;
                              end;
                           end ; { if Etat = busy }
                           end;
                        end; { define }
        end; {accept}

endproc Response_Idle



procedure Response_Connection( concern_np : string ) =


        choice

            accept
               Disconnect_Indication
               provided concern_np = Np
               do
                       writeln "  ";
                       return;
                       Accept_Hang_Up( concern_np, "Response");
            end; { accept }
        or
            accept
               Send_M
               provided concern_np = Np
               do
                  define np = valid_np(concern_np)
                  in
                    np!Send_Message(Message);
                    writeln "  " ;
                    return;
                    Response_Connection(concern_np);
                   end;
             end; { accept}

        or
            accept
               Receive_M
               provided concern_np = Np
               do
                    writeln "   " ;
                    writeln Message ;
                    return;
                    Response_Connection(concern_np);

            end ; { accept }

        or
           Accept_Hang_Up(concern_np,"Response");
        end; {choice}

endproc Response_Connection


procedure Accept_Hang_Up( concern_np  : string ;
                          fonction    : string ) =

        accept
                Hang_Up
                provided Np       = concern_np  and
                         Fonction = fonction
                do
                   define np = valid_np( Np )
                   in
                        np!On_Hook;
                        writeln "  " ;
                        writeln " HANG_UP ACCEPTED" ;
                        writeln "  " ;
                   end;
                  return ;
        end; {accept}

endproc Accept_Hang_Up


 procedure Serie_Idle =

        accept
                Do_Serie
```

```
do
   define np = valid_np( Np )
   in
      if np = nil
      then
         writeln "  " ;
         writeln " DO_SERIE INVALID (PORT INVALID) " ;
         return;
         Serie_Idle ;
      else
         define Etat = np!Off_Hook(self)
         in
         if Etat  = "busy"
         then
            writeln "  " ;
            writeln "  DO_SERIE INVALID (PORT IS OCCUPIE
            return;
            Serie_Idle ;
         else
            np!Signal_Pcs_Code;
            define Result_Authentication = np!Signal_Aut
            in
            if Result_Authentication = 1
            then
               writeln "  " ;
               writeln " DO_SERIE INVALID (AUTHENTICATIO
               return;
               Accept_Hang_Up
                    (Np,"Do_Serie");
            else
              if Result_Authentication = 2
              then
                 writeln "  " ;
                 writeln " DO_SERIE INVALID (AUTHENTICAT
                 return;
                 Accept_Hang_Up
                    (Np,"Do_Serie");
              else { Result_Authentication = 0 }
                 define Result_Signal = np!Signal_Start_
                                        (Service_Type)
                 in
                 if Result_Signal <> 0
                 then
                    writeln "  " ;
                    writeln " DO_SERIE INVALID ( SERVICE
                    return;
                    Accept_Hang_Up
                      (Np,"Do_Serie");
                 else
                    writeln " " ;
                    writeln "  DO_SERIE IS ACCEPTED " ;
                    return;
                    parallel
                       Ready_For_Call( Np );
                    and
                       Serie_Idle;
                    end;
                 end; { if Result_Signal <> 0 }
                 end;
              end; { if Result_Authentication = 2 }
            end; { if Result_Authentication = 1 }
            end;
         end; { if Etat }
         end;
      end; { if np = }
   end ; { define }

      end; {accept}

endproc Serie_Idle


procedure Ready_For_Call( concern_np : string ) =

      choice
            accept
               Dial
               provided concern_np = Np
               do
                 define np = valid_np( Np )
                 in
                    define Result_Signal= np!Dial_Serie( Sn )
                    in
                    if Result_Signal = 1
                    then
                       writeln "  " ;
                       writeln " DO_SERIE INVALID ( SN INVALID IN DI
                       return ;
                       Choice_Waiting(concern_np) ;
                    else
                       return;
                       Serie_Waiting_Connection_Response(concern_np)
                    end; { Result_Signal }
                    end;
                 end; { define }
            end; { accept }
      or
            Accept_Hang_Up(concern_np,"Do_Serie");

      end; {choice}

endproc Ready_For_Call


procedure Serie_Waiting_Connection_Response( concern_np : string ) =

      accept
            Connect_Response
            provided concern_np = Np and
                     Response = "connection_rejection"
            do
               writeln "  " ;
               writeln " DO_SERIE INVALID - PORT OCCUPIED";
               return;
               Choice_Waiting( concern_np );
      or

            Connect_Response
            provided concern_np = Np and
                     Response = "connected"
            do
               writeln "  " ;
                  return;
               Serie_Application_Connection_Waiting( concern_np );

      end; {accept}

endproc Serie_Waiting_Connection_Response


procedure Choice_Waiting( concern_np : string ) =
```

```
        choice
            accept
                Signal_Continu
                provided concern_np = Np
                do
                   define np = valid_np( Np )
                   in
                        writeln " " ;
                        writeln "DO_SERIE SIGNAL_CONTINU ACCEPTED ";
                        np!Signal_Continu;
                        return;
                        Ready_For_Call( concern_np );
                   end;
            end; { accept }
        or
            Accept_Hang_Up(concern_np,"Do_Serie");
        end; {choice}

endproc Choice_Waiting



procedure Serie_Application_Connection_Waiting( concern_np : string ) =

        choice
            accept
                Response_Indication
                provided concern_np = Np
                do
                        writeln  "  " ;
                        return;
                        Serie_Connection( concern_np );
             end; { accept }
          or
             accept
                Signal_Continu
                provided concern_np = Np
                do
                   define np = valid_np( Np )
                   in
                        writeln "  " ;
                        writeln " DO_SERIE SIGNAL_CONTINU IS ACCEPTED "
                        np!Signal_Continu;
                        return;
                        Ready_For_Call( concern_np );
                   end;
             end; { accept }

        or
                Accept_Hang_Up(concern_np,"Do_Serie");

        end; {choice}

endproc Serie_Application_Connection_Waiting


procedure Serie_Connection( concern_np : string ) =

        choice
            accept
                Disconnect_Indication
                provided concern_np = Np
                do
                        writeln  "  ";
                        return;
                        Choice_Waiting( concern_np );
             end; { accept }
          or
             accept
                Signal_Continu
                provided concern_np = Np
                do
                   define np = valid_np( Np )
                   in
                        writeln "  " ;
                        writeln "DO_SERIE SIGNAL_CONTINU IS ACCEPTED " ;
                        np!Signal_Continu;
                        return;
                        Ready_For_Call( concern_np );
                   end;
             end; { accept }

          or
             accept
                Send_M
                provided concern_np = Np
                do
                   define np = valid_np(concern_np)
                   in
                     np!Send_Message(Message);
                     writeln "  " ;
                     writeln " MESSAGE SENT    " ;
                     return;
                     Serie_Connection(concern_np);
                    end;
             end; { accept }

          or
             accept
               Receive_M
               provided concern_np = Np
               do
                    writeln "   " ;
                    writeln " MESSAGE RECEIVED  " ;
                    writeln Message ;
                    return;
                    Serie_Connection(concern_np);

             end; { accept }
          or
               Accept_Hang_Up(concern_np,"Do_Serie");

            end; {accept}

       endproc Serie_Connection



endtype USER



type NP = ENTITY with

        value : string;
```

```
hide
     state : var[ string ]; { enumerate: ("busy", "idle") }

operation


            {***** user          :  ******************}
            {***** SnValue       :  ****************}
            {******SnCalling     :  ****************************}
            {***** StValue       :  *************}
            {***** PinValue      :  **************}
            {***** NpValue       :  ************}
            {***** Duration      :  *******}
            {***** Message       :  ***}


      Off_Hook

      ( user        : USER
       ) : string  ;

      On_Hook;


      Dial_Serie

      ( SnValue     : string
       ) : integer ;

      Dial

      ( SnValue     : string ;
        SnCalling   : string ;
        StValue     : string
       ) : integer ;

      Signal_Pcs_Code;

      Signal_Authentication

      ( SnValue     : string ;
        PinValue    : string
       ) : integer ;

      Signal_Start_Session_Outcall

      ( NpValue     : string ;
        StValue     : string ;
        Duration    : integer
       ) : integer ;

      Signal_Terminate_Session_Outcall

      ( NpValue     : string ;
        StValue     : string
       ) : integer ;

      Signal_Terminate_Other_Session_Outcall

      ( StValue     : string
       ) : integer ;

      Signal_Start_Session_Incall

      ( NpValue     : string ;
        StValue     : string ;
        Duration    : integer
       ) : integer ;

      Signal_Terminate_Session_Incall

      ( StValue     : string
       ) : integer ;

      Signal_Terminate_Other_Session_Incall

      ( SnValue     : string ;
        StValue     : string
       ) : integer ;

      Signal_Start_Serie

      ( StValue     : string
       ) : integer ;

      Signal_Continu;


      Send_Message

      ( Message     : string
       ) ;


            {***** Message       : Receive_Message*)
            {***** remote_np     : ******************)
            {***** called_sn     : ********************************)
            {***** calling_sn    : ***************************)
            {***** st            : *************)


      Receive_Message

      ( Message     : string
       ) ;


      Incoming_Call

      ( remote_np   : NP      ;
        called_sn   : SN      ;
        calling_sn  : SN      ;
        st          : ST
       ) : string  ;

      Disconnect_Indication;

      Connect_Indication;

behavior

      parallel
            Incoming_Call_Idle;
      and
            User_Utilisation_Idle;
      end; {parallel}


where
```

```
procedure valid_np( NpValue : string ): NP =

            ifexist np : NP suchthat np.value = NpValue
            then
               return np;
            else
               return nil ;
            end;

endproc valid_np


procedure valid_sn( SnValue : string ): SN =

            ifexist sn : SN suchthat sn.value = SnValue
            then
               return sn;
            else
               return nil;
            end;

endproc valid_sn


procedure valid_st( StId : string ): ST =

            ifexist st : ST suchthat st.id = StId
            then
               return st;
            else
               return nil;
            end;

endproc valid_st


procedure sn_billed( sn : SN ; st : ST ) : SN =

   ifexist out_sn_resp : OUTCALL_SN_RESPONSIBLE
   suchthat out_sn_resp.np = self and out_sn_resp.st = st
                             and out_sn_resp.sn = sn
   then
      return sn;
   else
      ifexist o : OWNER suchthat o.np = self
      then
         return o.sn ;
      else
         return nil ;
      end ; { if exist }
   end;

endproc sn_billed


procedure sn_owner( np : NP ) : SN =

   ifexist o : OWNER suchthat o.np = np then
      return o.sn;
   else
      return nil ;
   end;
```

```
endproc sn_owner


procedure sn_home( sn : SN; st : ST ) : NP =

      ifexist home : HOME_LOCATION
      suchthat home.sn = sn and home.st = st
      then
        return home.np;
      else
        return nil ;
      end;

endproc sn_home


procedure np_owner( sn : SN ) : NP =

   ifexist o : OWNER suchthat o.sn = sn then
      return o.np;
   else
      return nil;
   end;

endproc np_owner


procedure horloge_int  : HORLOGE_INTERNE =

 ifexist hr : HORLOGE_INTERNE
            then
               return hr;
            else
               return nil ;
            end;

endproc horloge_int


procedure User_Utilisation_Idle =

      accept
            Off_Hook
            do
               if state^ = "idle"
               then
                    state := "busy";
                    return "idle";
                    parallel
                       Ready( user );
                    and
                       User_Utilisation_Idle;
                    end;
               else
                    return "busy";
                    User_Utilisation_Idle ;
               end; {if}
       end; {accept}

endproc  User_Utilisation_Idle


procedure Ready( user: USER ) =
```

```
accept
   Dial
   do
    define current_st = valid_st( StValue ),
           sn_called  = valid_sn( SnValue ),
           sn_calling = valid_sn( SnCalling)
    in
     ifexist incall_location : INCALL_LOCATION
     suchthat incall_location.sn = sn_called and
              incall_location.st = current_st and sn_calling <>
     then
       define Result_Incoming =  incall_location.np!Incoming_Cal
                                (self, sn_called, sn_billed( sn_cal
                                 current_st )
       in

       if Result_Incoming = "busy"
       then
         return 0 ;
         user!Connect_Response(value,"connection_rejection","  "
         Accept_On_Hook( nil);
       else
         return 0 ;
         user!Connect_Response(value,"connected",sn_billed(sn_ca
         Utilisation_Waiting_Connection(incall_location.np,user,
           sn_billed( sn_calling,current_st ) );
       end; {if}
       end;

     else {ifexist}
       if (sn_called = nil) or (sn_calling = nil)
       then
         return 1 ;
       else
         return 2 ;
       end; { if sn_called = nil or sn_calling = nil}
       Accept_On_Hook(nil);
     end; {ifexist}
    end; {define}
or
   Signal_Pcs_Code
   do
      return;
      accept
        Signal_Authentication
        do
           ifexist sn : SN
           suchthat sn.value = SnValue
           then
              define Result_Authentication = sn!Authenticate( Pi
              in
              if Result_Authentication = true
              then
                   return 0 ;
                   In_Pcs_Mode( user, sn );
              else
                   return 2 ;
                   Accept_On_Hook(nil);
              end; {if}
              end;
           else {ifexist}

              return 1 ;
              Accept_On_Hook(nil);
           end; {ifexist}
        end; {accept}
```

```
      end; {accept}

endproc Ready



procedure Utilisation_Waiting_Connection( remote_np : NP;
                                          user: USER;
                                          sn_billable : SN ) =

   choice
      accept
        Connect_Indication
        do
                user!Response_Indication(value);
                return;
                Utilisation_Connected( remote_np, user,
                                       sn_billable );

      end; {accept}
    or
      Accept_On_Hook( remote_np);
    end; {choice}

endproc  Utilisation_Waiting_Connection


procedure Utilisation_Connected( remote_np : NP; user: USER;
                                 sn_billable : SN ) =
   choice
      accept
         Disconnect_Indication
         do
            user!Disconnect_Indication(value);
            return;
            Accept_On_Hook(nil );
      end; { accept }
   or
       accept
         Receive_Message
         do
              user!Receive_M(value,Message);
              return ;
              Utilisation_Connected( remote_np, user,sn_billable );
       end; { accept }
    or
       accept
          Send_Message
          do
               remote_np!Receive_Message(Message);
               return ;
               Utilisation_Connected( remote_np, user, sn_billable );
       end; {accept}
    or
       Accept_On_Hook( remote_np);
    end; {choice}


 endproc Utilisation_Connected
```

```
procedure Accept_On_Hook( remote_np : NP ) =

    accept
         On_Hook
         do
           state := "idle";
           if remote_np <> nil
           then
              remote_np!Disconnect_Indication;
           end; {if}
           return;

     end; {accept}

endproc Accept_On_Hook



procedure In_Pcs_Mode( user : USER;
                       sn   : SN ) =

choice
         Start_Session_Outcall(user, sn);
   or
         Terminate_Session_Outcall( user,sn );
   or
         Terminate_Other_Session_Outcall(user,sn);
   or
         Start_Session_Incall( user,sn);
   or
         Terminate_Session_Incall( user,sn);
   or
         Terminate_Other_Session_Incall(user,sn);
   or
         Start_Serie_Idle( user, sn );
end; { choice }

endproc In_Pcs_Mode



procedure Start_Session_Outcall( user : USER;
                                 sn   : SN ) =

accept
   Signal_Start_Session_Outcall
   do
   define current_st = valid_st( StValue ),
          concern_np = valid_np( NpValue )
   in
     if current_st = nil
     then
       return 1 ;
       Accept_On_Hook(nil);
     else
       if concern_np = nil
       then
          return 2 ;
          Accept_On_Hook(nil);
       else
         define Controle = (sn = sn_owner( concern_np ))
         in
         if Controle
         then
           return 3 ;
           Accept_On_Hook(nil) ;
         else
           ifexist outcall_relation : OUTCALL_SN_RESPONSIBLE
           suchthat outcall_relation.np = concern_np and
                    outcall_relation.st = current_st and
                    outcall_relation.sn = sn
           then
             return 4 ;
             Accept_On_Hook(nil);
           else

              return 0 ;
              define outcall =  new OUTCALL_SN_RESPONSIBLE as
                                TIMED_LOCATION_RELATIONSHIP( Duration) as
                                LOCATION_RELATIONSHIP(sn,current_st,conce

              in
                 if Duration > 0
                 then
                    define HI = horloge_int
                    in
                       HI!Repere_Out(user,Duration,outcall);
                    end;
                 end; { if Duration }
              end; {define outcall}
              Accept_On_Hook(nil);
           end; {ifexist }
         end; { if Controle }
         end;
       end; { if concern_np }
     end; { if st = nil }
   end; {define}
 end; { accept }

endproc Start_Session_Outcall



procedure Terminate_Session_Outcall( user : USER ;
                                     sn   : SN ) =

accept
  Signal_Terminate_Session_Outcall
  do
     define current_st = valid_st( StValue ),
            current_np = valid_np( NpValue )
     in
         ifexist outcall_relation : OUTCALL_SN_RESPONSIBLE
         suchthat outcall_relation.np = current_np and
              outcall_relation.st = current_st and
              outcall_relation.sn = sn
         then
           outcall_relation!delete;
           return  0 ;
           Accept_On_Hook(nil) ;
         else
           return  1 ;
           Accept_On_Hook(nil);
         end; { if exist }
     end; {define}
end; { accept }

endproc Terminate_Session_Outcall


procedure Start_Session_Incall( user : USER ;
                                sn   : SN  ) =
```

```
accept
  Signal_Start_Session_Incall
  do
     define current_st = valid_st( StValue ),
            concern_np = valid_np( NpValue )
     in
       if current_st = nil
       then
          return 1;
          Accept_On_Hook(nil);
       else
         if concern_np = nil
         then
            return 2 ;
            Accept_On_Hook(nil);
         else
            define Controle = (concern_np = sn_home( sn, current_st))
            in
            if Controle
            then
                return 3 ;
                Accept_On_Hook(nil);
            else
                ifexist incall_location : INCALL_LOCATION
                suchthat incall_location.sn = sn and
                         incall_location.st = current_st
                then
                  if incall_location.np = concern_np
                  then
                    return  4;
                    Accept_On_Hook(nil);
                  else
                    incall_location!delete ;
                    return 0 ;

                    define incall =  new INCALL_LOCATION as
                                     TIMED_LOCATION_RELATIONSHIP( Durati
                                     LOCATION_RELATIONSHIP(sn,current_st

                    in
                      if Duration > 0
                      then
                         define HI = horloge_int
                         in
                           HI!Repere_In(user,Duration,incall);
                         end;
                      end; { if Duration }
                    end; {define incall}
                    Accept_On_Hook(nil);
                  end; { if incall = }
                end; { if exist }
             end; { Controle }
             end;
          end; { if concern_np = nil }
        end; { if current_st = nil }
      end; {define}
  end; { accept }
endproc Start_Session_Incall



procedure Terminate_Session_Incall( user : USER ;
                                    sn   : SN    ) =

accept
  Signal_Terminate_Session_Incall
  do
     define current_st = valid_st( StValue )
     in
        ifexist incall_relation : INCALL_LOCATION
        suchthat incall_relation.sn = sn and
            incall_relation.st = current_st
        then
           define Controle =  (incall_relation.np = sn_home(sn,current_s
           in
           if Controle
           then
              return 1 ;
              Accept_On_Hook(nil);
           else
              return 0 ;
              incall_relation!delete ;
              eval new INCALL_LOCATION as
                 TIMED_LOCATION_RELATIONSHIP( 0 ) as
                 LOCATION_RELATIONSHIP( sn, current_st,
                     sn_home( sn, current_st ) );
              Accept_On_Hook(nil);
           end; { if incall_relation }
           end;
        end; { if exist }
     end; {define }
end; {accept}

endproc Terminate_Session_Incall



procedure Terminate_Other_Session_Outcall ( user  : USER ;
                                            sn    : SN    ) =

accept
  Signal_Terminate_Other_Session_Outcall
  do
     define  current_np = np_owner( sn ),
             current_st = valid_st( StValue )
     in
       if current_st = nil
       then
           return 1 ;
           Accept_On_Hook(nil);
       else
          if current_np = nil
          then
             return 2 ;
             Accept_On_Hook(nil);
          else
             forall  outcall_relation : OUTCALL_SN_RESPONSIBLE
                  suchthat outcall_relation.np = current_np and
                  outcall_relation.st = current_st
             do
                outcall_relation!delete;
             end; { for all}
             return 0 ;
             eval new OUTCALL_SN_RESPONSIBLE as
                     TIMED_LOCATION_RELATIONSHIP( 0 ) as
                     LOCATION_RELATIONSHIP(sn,current_st,current_np);
             Accept_On_Hook(nil);
          end; { if current_np = nil }
       end; { if current_st = nil }
     end; { define}
end; {Accept}

endproc Terminate_Other_Session_Outcall
```

```
procedure Terminate_Other_Session_Incall ( user : USER ;
                                           sn   : SN  ) =

  accept
    Signal_Terminate_Other_Session_Incall
    do
        define current_st = valid_st( StValue ),
               current_np = np_owner( sn ),
               concern_sn = valid_sn(SnValue)
        in
            if current_st = nil
            then
                return 1 ;
                Accept_On_Hook(nil);
            else
               if current_np = nil
               then
                  return 2 ;
                  Accept_On_Hook(nil);
               else
                  return 0 ;
                  if concern_sn = nil
                  then
                     forall incall_relation : INCALL_LOCATION
                     suchthat incall_relation.np=current_np and
                     incall_relation.st=current_st
                     do
                         eval new INCALL_LOCATION as
                         TIMED_LOCATION_RELATIONSHIP( 0 ) as
                         LOCATION_RELATIONSHIP(incall_relation.sn,curren
                          sn_home( incall_relation.sn, current_st ) );
                         incall_relation!delete ;
                     end; { for all}
                  else
                    ifexist incall_relation : INCALL_LOCATION
                    suchthat incall_relation.np = current_np and
                             incall_relation.st = current_st and
                             incall_relation.sn = concern_sn
                    then
                       eval new  INCALL_LOCATION as
                       TIMED_LOCATION_RELATIONSHIP( 0 ) as
                       LOCATION_RELATIONSHIP(incall_relation.sn,current_
                            sn_home( incall_relation.sn, current_st ) );
                       incall_relation!delete;
                    end; { if exist }
                  end; { if concern_sn = nil }
                  Accept_On_Hook(nil);
               end; { if current_np = nil }
            end; { if current_st = nil }
      end; {define}

  end; {accept}

endproc Terminate_Other_Session_Incall


procedure Start_Serie_Idle( user : USER;
                            sn   : SN) =
    accept
      Signal_Start_Serie
      do
      define current_st = valid_st(StValue)
      in
          if current_st = nil
          then
             return 1 ;
             Accept_On_Hook(nil);
          else
             return 0 ;
             Start_Serie(user,sn,current_st);
          end; { if current_st = nil }
       end; { define }
    end; { accept }

endproc Start_Serie_Idle


procedure Start_Serie ( user       : USER;
                        sn         : SN  ;
                        current_st : ST) =

            choice
                accept
                 Dial_Serie
                 do
                 define current_sn = valid_sn(SnValue)
                 in
                    if current_sn = nil
                    then
                       return 1 ;
                       Accept_On_Hook(nil);
                    else
                       return 0 ;
                       ifexist incall_location: INCALL_LOCATION
                           suchthat incall_location.sn= current_sn and
                           incall_location.st = current_st
                       then
                          define Result_Incoming = incall_location.np!I
                                                   (self, current_sn, sn, cur
                          in

                          if  Result_Incoming = "busy"
                          then
                             user!Connect_Response(value,"connection_re
                             Choice_Serie( user, sn, current_st );
                          else
                             user!Connect_Response(value,"connected",sn
                             Serie_Waiting_Connection(incall_location.n
                          end; {if Result_Incoming }
                          end;
                       else {ifexist}
                          user!Connect_Response(value,"connection_rejec
                          Choice_Serie( user, sn, current_st );
                       end; {ifexist}
                    end; { if current_sn = nil }
                 end; { define current_sn }
                end; { accept }
            or
                Accept_On_Hook( nil);
            end; {choice}

endproc Start_Serie

procedure Serie_Waiting_Connection( remote_np   : NP  ;
                                    user        : USER;
                                    sn          : SN  ;
                                    st          : ST          ) =

   choice
      accept
```

```
        Connect_Indication
        do
                user!Response_Indication( value );
                return;
                Serie_Connected( remote_np, user, sn, st );
      end; {accept}
   or
      accept
        Signal_Continu
        do
              remote_np!Disconnect_Indication;
              return;
              Start_Serie( user, sn, st );
        end; {accept}
   or
         Accept_On_Hook( remote_np);

   end; {choice}

endproc  Serie_Waiting_Connection


procedure Serie_Connected ( remote_np : NP;
                            user      : USER;
                            sn        : SN;
                            st        : ST  ) =
   choice
      accept
         Disconnect_Indication
         do
            user!Disconnect_Indication( value );
            return;
            Choice_Serie( user, sn, st );
      end; { accept }
   or
      accept
         Receive_Message
         do
              user!Receive_M(value,Message);
              return ;
              Serie_Connected( remote_np, user,sn,st );
      end; { accept }
     or
      accept
         Send_Message
         do
              remote_np!Receive_Message(Message);
              return;
              Serie_Connected( remote_np, user, sn,st);

      end; {accept}
   or
      Accept_On_Hook( remote_np);
   or
      accept
         Signal_Continu
         do
            remote_np!Disconnect_Indication;
            return;
            Start_Serie( user, sn, st );
      end; {accept}
   end; {choice}

endproc Serie_Connected
```

```
procedure Choice_Serie ( user : USER;
                         sn   : SN;
                         st   : ST       ) =

  choice
        accept
          Signal_Continu
          do
             return;
             Start_Serie( user, sn, st );
        end; {accept}
       or
         Accept_On_Hook( nil);
  end; {choice}

endproc Choice_Serie


procedure Incoming_Call_Idle =

   accept
      Incoming_Call
      do
        if state^ = "busy"
        then
           return "busy";
           Incoming_Call_Idle;
        else
           return "idle";
           parallel
                Incoming_Call_Waiting_Connection( remote_np, st );
           and
                Incoming_Call_Idle;
           end;
        end; {if}
    end; {accept}

endproc  Incoming_Call_Idle


procedure Incoming_Call_Waiting_Connection( remote_np : NP;
                                            st : ST  ) =
   choice
      accept
        Off_Hook
        do
                return state^ ;
                state := "busy";
                remote_np!Connect_Indication;
                Incoming_Call_Connected( user,  remote_np );
      end; {accept}
     or
      accept
        Disconnect_Indication
        do
          return;
      end; {accept}
   end; { choice }
endproc  Incoming_Call_Waiting_Connection


procedure Incoming_Call_Connected( user: USER; remote_np : NP ) =

        choice
```

```
                accept
                   Disconnect_Indication
                   do
                     user!Disconnect_Indication( value );
                     return;
                     Accept_On_Hook( nil);
                 end; { accept }
                or
                  accept
                   Receive_Message
                   do
                   user!Receive_M(value,Message);
                   return ;
                   Incoming_Call_Connected( user,remote_np);
                  end; { accept }
                or
                  accept
                   Send_Message
                   do
                   remote_np!Receive_Message(Message);
                   return ;
                   Incoming_Call_Connected( user,remote_np);

                 end; {accept}
              or
                 Accept_On_Hook( remote_np);
              end; {choice}

      endproc  Incoming_Call_Connected



endtype NP



type ST = ENTITY with

        id : string;

endtype ST


type SUBSCRIBER = USER with

        Adresse : string;

endtype SUBSCRIBER



type PIN = ENTITY with

   hide
        value : var[string];


   operation


    validate
    ( parameter  :  string
     ) : boolean ;

   behavior

     comparer ;

   where

      procedure comparer  =
         accept
            validate
            do
             if value^ = parameter
             then
                return true ;
             else
                return false ;
             end; { if }
             comparer ;
        end; { accept }
     endproc comparer

endtype PIN



type SN = ENTITY with

        value : string;


  operation


   Authenticate

   ( parameter : string
    ) : boolean ;

  behavior

    verifier ;

  where

    procedure verifier =
           accept
                Authenticate
                do
                   ifexist a:AUTHENTICATION
                   suchthat a.sn = self
                   then
                        return a.pin!validate( parameter );
                   else
                        return false ;
                   end; {ifexist}
                   verifier ;
           end; {accept}
    endproc verifier

endtype SN
```

```
type AUTHENTICATION = RELATIONSHIP with

        subscriber : SUBSCRIBER;
        sn : SN;
        pin : PIN;

endtype AUTHENTICATION



type LOCATION_RELATIONSHIP = RELATIONSHIP with

        sn : SN;
        st : ST;
        np : NP;

endtype LOCATION_RELATIONSHIP


{************* RELATION HOME_LOCATION*******************}
{*******************************************************}

type HOME_LOCATION = LOCATION_RELATIONSHIP

endtype HOME_LOCATION


{************* RELATION TIMED_LOCATION_RELATIONSHIP******}
{*******************************************************}

type TIMED_LOCATION_RELATIONSHIP = LOCATION_RELATIONSHIP with

     duration : integer;

endtype TIMED_LOCATION_RELATIONSHIP


{************* RELATION OUTCALL_SN_RESPONSIBLE**********}
{*******************************************************}
type  OUTCALL_SN_RESPONSIBLE = TIMED_LOCATION_RELATIONSHIP

endtype  OUTCALL_SN_RESPONSIBLE


{************* RELATION INCALL_LOCATION*****************}
{*******************************************************}

type  INCALL_LOCATION =  TIMED_LOCATION_RELATIONSHIP

endtype  INCALL_LOCATION


{************* RELATION OWNER***************************}
{*******************************************************}
type OWNER = RELATIONSHIP with
        sn : SN;
        np : NP;

endtype OWNER


type HORLOGE_INTERNE = ENTITY with



  behavior

    parallel
      rafraichir;
    and
      Repere ;
    end; ( parallel )

  where

      procedure sn_home( sn : SN; st : ST ) : NP =

              ifexist home : HOME_LOCATION
              suchthat home.sn = sn and home.st = st
              then
                return home.np;
              else
                return nil ;
              end;

      endproc sn_home


    procedure rafraichir  =
        accept
           Impulsion
           do
              return;
              Temps := heure + Temps^ ;
              rafraichir ;
        end; ( accept )
    endproc rafraichir

    procedure Repere =
          accept
            Repere_Out
            do
               return ;
               Nbr_Relation := Nbr_Relation^ + 1 ;
               parallel
                 Verdict_Out(user,Duree,Relation,Temps^);
               and
                 Repere ;
               end;

           or
            Repere_In
            do
               return;
               Nbr_Relation := Nbr_Relation^ + 1 ;
```

```
            parallel
              Verdict_In(user,Duree,Relation,Temps^);
            and
              Repere ;
            end;
        end; { accept }
    endproc Repere

    procedure Verdict_Out (user     : USER ;
                           Duree    : integer ;
                           Relation : OUTCALL_SN_RESPONSIBLE ;
                           Temps_D  : integer      ) =
      if Temps^ > Temps_D
      then
         if ( Duree -Temps^) > 0
         then
            return ;
            Verdict_Out (user,Duree,Relation,Temps^);
         else
          ( ****** suppression de la relation  nbr-relation - 1 ******)
          Nbr_Relation := Nbr_Relation^ - 1 ;
          return ;
          user!Outcall_Session_Time_Out(Relation.np.value,Relation.sn.value,Rel
          Relation!delete;
         end; { if duree -temps }
      else
            Verdict_Out (user,Duree,Relation,Temps_D);
      end; { if Temps > Temps_D }

    endproc Verdict_Out


    procedure Verdict_In  (user     : USER ;
                           Duree    : integer ;
                           Relation : INCALL_LOCATION ;
                           Temps_D  : integer      ) =
      if Temps^ > Temps_D
      then
         if ( Duree -Temps^) > 0
         then
            Verdict_In (user,Duree,Relation,Temps^);
         else
          ( ****** suppression de la relation  nbr-relation - 1 ******)
           Nbr_Relation := Nbr_Relation^ - 1 ;
           return;
           user!Incall_Session_Time_Out(Relation.sn.value,Relation.st.id);
           eval new INCALL_LOCATION as
                        TIMED_LOCATION_RELATIONSHIP( 0 ) as
                        LOCATION_RELATIONSHIP( Relation.sn, Relation.st,
                            sn_home( Relation.sn, Relation.st ) );
           Relation!delete;

         end; { if duree -temps }
       else
            Verdict_In (user,Duree,Relation,Temps_D);
       end; { if Temps > Temps_D }

    endproc Verdict_In


endtype



type HORLOGE_EXTERNAL = ENTITY with
```

```
  operation


     procedure avance_temps  =
        accept
           Temps
           do
            define hrl = horloge_int
            in
              hrl!Impulsion(heure);
            end; { define }
            return;
            avance_temps ;
        end; { accept }
     endproc avance_temps

endtype HORLOGE_EXTERNAL




type SIMULATION_TISER = actor with

private

   var_str : var[string];
   var_int : var[integer];

behavior

   define var_str = new var[string];
   define var_int = new var[integer];

   print_menu;
   simulation ;

where

  procedure simulation  =

  try
    define action = read_integer( "WHICH COMMAND ? : " )
    in

     if action = 0
     then
       writeln "Bye";

     else



       if action = 211
       then
         define
            USER1 = select_user("user1"),
            USER2 = select_user("user2")
         in

            ( ******* START_OUTCALL_SESSION **************)
```

```
{port invalide }
USER1!Start_Outcall_Session( "np6","sn1","pin1","np2","image", 0);

{authentication invalide sn invalide}
USER1!Start_Outcall_Session( "np1","sn6","pin1","np2","image", 0);
USER1!Hang_Up("np1","Start_Outcall_Session");

{authentication invalide pin invalide}
USER1!Start_Outcall_Session( "np1","sn1","pin6","np2","image", 0);
USER1!Hang_Up("np1","Start_Outcall_Session");

{signal_start_outcall_Session invalide Action_Np invalide}
USER1!Start_Outcall_Session( "np1","sn1","pin1","np7","image", 0);
USER1!Hang_Up("np1","Start_Outcall_Session");

{service invalide}
USER1!Start_Outcall_Session( "np1","sn1","pin1","np2","abrakadabra",
USER1!Hang_Up("np1","Start_Outcall_Session");

{usager owner du port}
USER1!Start_Outcall_Session("np1","sn1","pin1","np1","image",0);
USER1!Hang_Up("np1","Start_Outcall_Session");

{start_outcall accepte}
USER1!Start_Outcall_Session( "np1","sn1","pin1","np2","image", 0);
USER1!Hang_Up("np1","Start_Outcall_Session");

{ ********** TERMINATE_OUTCALL_SESSION****************** }

{port invalide}
USER1!Terminate_Outcall_Session( "np6","sn1","pin1","np2","image" );

{authentication invalide pin invalide}
USER1!Terminate_Outcall_Session( "np1","sn1","pin6","np2","image" );
USER1!Hang_Up("np1","Terminate_Outcall_Session");

{terminate_outcall_session accepte}
USER1!Terminate_Outcall_Session( "np1","sn1","pin1","np2","image" );
USER1!Hang_Up("np1","Terminate_Outcall_Session");

{********* TERMINATE_OTHER_OUTCALL_SESSION ********************}

{preparation d'une outcall_session}
USER2!Start_Outcall_Session( "np2","sn2","pin2","np1","voix", 0);
USER2!Hang_Up("np2","Start_Outcall_Session");

{port invalide}
USER1!Terminate_Other_Outcall_Session( "np6","sn1","pin1","voix");

{authentication invalide  sn invalide}
USER1!Terminate_Other_Outcall_Session( "np1","sn6","pin1","voix");
USER1!Hang_Up("np1","Terminate_Other_Outcall_Session");

{authentication invalide  pin invalide}
USER1!Terminate_Other_Outcall_Session( "np1","sn1","pin6","voix");
USER1!Hang_Up("np1","Terminate_Other_Outcall_Session");

{service invalide}
USER1!Terminate_Other_Outcall_Session( "np1","sn1","pin1","abrakadab
USER1!Hang_Up("np1","Terminate_Other_Outcall_Session");

{terminate_other_outcall_session accepte}
USER1!Terminate_Other_Outcall_Session( "np1","sn1","pin1","voix");
USER1!Hang_Up("np1","Terminate_Other_Outcall_Session");

    {terminer l'automate de outcall_session redandance de l'automate}
    USER2!Terminate_Outcall_Session( "np2","sn2","pin2","np1","voix");
    USER2!Hang_Up("np2","Terminate_Outcall_Session");

  end;   ( define )
  print_menu;
  simulation ;
end; {if action =211 }



if action = 212
then
  define
     USER1 = select_user("user1"),
     USER2 = select_user("user2")
  in

     { ******* START_INCALL_SESSION **************}

     {*****  port invalide
     USER1!Start_Incall_Session( "np6","sn1","pin1","np2","image", 0);

     authentication invalide sn invalide
     USER1!Start_Incall_Session( "np1","sn6","pin1","np2","image", 0);
     USER1!Hang_Up("np1","Start_Incall_Session");

      authentication invalide pin invalide
     USER1!Start_Incall_Session( "np1","sn1","pin6","np2","image", 0);
     USER1!Hang_Up("np1","Start_Incall_Session");

      signal_start_incall_Session invalide Action_Np invalide
     USER1!Start_Incall_Session( "np1","sn1","pin1","np7","image", 0);
     USER1!Hang_Up("np1","Start_Incall_Session");

     service invalide
     USER1!Start_Incall_Session( "np1","sn1","pin1","np2","abrakadabra",
     USER1!Hang_Up("np1","Start_Incall_Session");

      preparation pour np occupe
     USER2!Start_Incall_Session( "np1","sn1","pin6","np2","image", 0);
      np occupe
     USER1!Start_Incall_Session( "np1","sn1","pin1","np2","image", 0);
      fin preparation
     USER2!Hang_Up("np1","Start_Incall_Session");

      usager owner du port
     USER1!Start_Incall_Session("np1","sn1","pin1","np1","image",0);
     USER1!Hang_Up("np1","Start_Incall_Session");

       start_incall accepte  ****}
     USER1!Start_Incall_Session( "np1","sn1","pin1","np2","image", 0);
     USER1!Hang_Up("np1","Start_Incall_Session");

     {********** TERMINATE_INCALL_SESSION******************}

     {**** port invalide
     USER1!Terminate_Incall_Session( "np6","sn1","pin1","image" );

      authentication invalide sn invalide
     USER1!Terminate_Incall_Session( "np1","sn6","pin1","image" );
     USER1!Hang_Up("np1","Terminate_Incall_Session");

      authentication invalide pin invalide
```

```
USER1!Terminate_Incall_Session( "np1","sn1","pin6","image" );
USER1!Hang_Up("np1","Terminate_Incall_Session");

 service invalide
USER1!Terminate_Incall_Session( "np1","sn1","pin1","abrakadabra" );
USER1!Hang_Up("np1","Terminate_Incall_Session");

 preparation pour np occupe
USER2!Start_Incall_Session( "np1","sn1","pin6","np2","image", 0);
 np occupe
USER1!Terminate_Incall_Session( "np1","sn1","pin1","image" );
 fin preparation
USER2!Hang_Up("np1","Start_Incall_Session");

**** a ajouter  usager owner pour terminate Incall session**********


   terminate_intcall_session accepte  ********   )
USER1!Terminate_Incall_Session( "np1","sn1","pin1","image" );
USER1!Hang_Up("np1","Terminate_Incall_Session");

{********* TERMINATE_OTHER_INCALL_SESSION ********************}

{    preparation d'une incall_session     }
USER2!Start_Incall_Session( "np2","sn2","pin2","np1","voix", 0);
USER2!Hang_Up("np2","Start_Incall_Session");

{***** port invalide
USER1!Terminate_Other_Incall_Session( "np6","sn1","pin1","sn2","voix

 authentication invalide  sn invalide
USER1!Terminate_Other_Incall_Session( "np1","sn6","pin1","sn2","voix
USER1!Hang_Up("np1","Terminate_Other_Incall_Session");

 authentication invalide  pin invalide
USER1!Terminate_Other_Incall_Session( "np1","sn1","pin6","sn2","voix
USER1!Hang_Up("np1","Terminate_Other_Incall_Session");

 service invalide
USER1!Terminate_Other_Incall_Session( "np1","sn1","pin1","sn2","abra
USER1!Hang_Up("np1","Terminate_Other_Incall_Session");

 preparation pour np occupe
USER2!Start_Incall_Session( "np1","sn2","pin6","np1","voix", 0);
 np occupe
USER1!Terminate_Other_Incall_Session( "np1","sn1","pin1","sn2","voix
 fin preparation
USER2!Hang_Up("np1","Start_Incall_Session");

USER2!Terminate_Other_Incall_Session("np1","sn2","pin2","sn2","voix"
USER2!Hang_Up("np1","Terminate_Other_Incall_Session");

 terminate_other_incall_session accepte  ****}
USER1!Terminate_Other_Incall_Session( "np1","sn1","pin1","sn2","voix
USER1!Hang_Up("np1","Terminate_Other_Incall_Session");

USER2!Terminate_Incall_Session( "np2","sn2","pin2","voix");
USER2!Hang_Up("np2","Terminate_Incall_Session");

  end;   { define }
  print_menu;
  simulation ;
end; {if action =212 }
```

```
if action = 213
then
  define
    USER1 = select_user("user1"),
    USER2 = select_user("user2")
  in

    {******* CALL *****************}

   {***************
    np invalide
    USER1!Call("np6","sn1","sn2","voix");

    sn invalide
    USER1!Call("np1","sn1","sn6","voix");
    USER1!Hang_Up("np1", "Call");

    service invalide
    USER1!Call("np1","sn1","sn2","abrakadabra");
    USER1!Hang_Up("np1", "Call");

    preparation port occupe
    USER2!Start_Incall_Session( "np1","sn2","pin6","np1","voix", 0);
    np occupe
    USER1!Call("np1","sn1","sn2","voix");
    USER1!Hang_Up("np1", "Call");
    fin preparation
    USER2!Hang_Up("np1","Start_Incall_Session");

    annuller l'appel
    USER1!Call("np1","sn1","sn2","voix");
    USER1!Hang_Up("np1", "Call");

    preparation appellant occupe
    USER2!Call("np2","sn2","sn3","voix");
    appellant occupe
    USER1!Call("np1","sn1","sn2","voix");
    USER1!Hang_Up("np1", "Call");
    fin preparation
    USER2!Hang_Up("np2", "Call");
    *********}

    USER1!Call("np1","sn1","sn2","voix");

    { ****************** RESPOND *****************}

    {********
    port invalide
    USER2!Respond("np6");

    **********}
    { respond accepte }
    USER2!Respond( "np2" );

    USER1!Hang_Up("np1", "Call");
    USER2!Hang_Up("np2", "Respond");

  end;   { define }
  print_menu;
  simulation ;
end; {if action = 213 }



if action = 214
then
```

```
      define
        USER1 = select_user("user1"),
        USER2 = select_user("user2"),
        USER3 = select_user("user3")
      in
        (************** DO_SERIE *********************)

        USER1!Do_Serie( "np1","sn1","pin1","voix");
        USER1!Dial( "np1","sn3");
        USER3!Respond( "np3");
        USER1!Hang_Up("np1","Do_Serie");
        USER3!Hang_Up("np3","Respond");
        USER1!Do_Serie( "np1","sn1","pin1","voix");
        USER1!Hang_Up("np1","Do_Serie");
        USER1!Do_Serie( "np1","sn1","pin1","voix");
        USER1!Dial( "np1","sn2");
        USER2!Respond( "np2");
        USER2!Hang_Up("np2","Respond");
        USER1!Signal_Continu("np1");
        USER1!Dial( "np1","sn3");
        USER3!Respond( "np3");
        USER3!Hang_Up("np3","Respond");
        USER1!Hang_Up("np1","Do_Serie");
        USER1!Do_Serie( "np1","sn1","pin1","voix");
        USER1!Dial( "np1","sn2");
        USER1!Hang_Up("np1","Do_Serie");
        USER1!Do_Serie( "np1","sn1","pin1","voix");
        USER1!Dial( "np1","sn7");
        USER1!Hang_Up("np1","Do_Serie");
        USER1!Do_Serie( "np1","sn1","pin1","abrakadabra");
        USER1!Dial( "np1","sn2");
        USER1!Hang_Up("np1","Do_Serie");
        USER2!Call("np2","sn2","sn3","voix");
        USER3!Respond( "np3");
        USER1!Do_Serie( "np1","sn1","pin1","voix");
        USER1!Dial( "np1","sn2");
        USER1!Hang_Up("np1","Do_Serie");
        USER2!Hang_Up("np2","Call");
        USER3!Hang_Up("np3","Respond");
        USER1!Do_Serie( "np1","sn1","pin1","voix");
        USER1!Dial( "np1","sn2");
        USER1!Signal_Continu("np1");
        USER1!Dial( "np1","sn3");
        USER3!Respond( "np3");
        USER1!Signal_Continu("np1");
        USER3!Hang_Up("np3","Respond");
        USER1!Hang_Up("np1","Do_Serie");
      end;   ( define )
      print_menu;
      simulation ;
    end; {if action = 214 }




  end; {if ACTION = 0  }

end; {define  ACTION}

except of
  type aborted => simulation ;
end;

endproc simulation
```

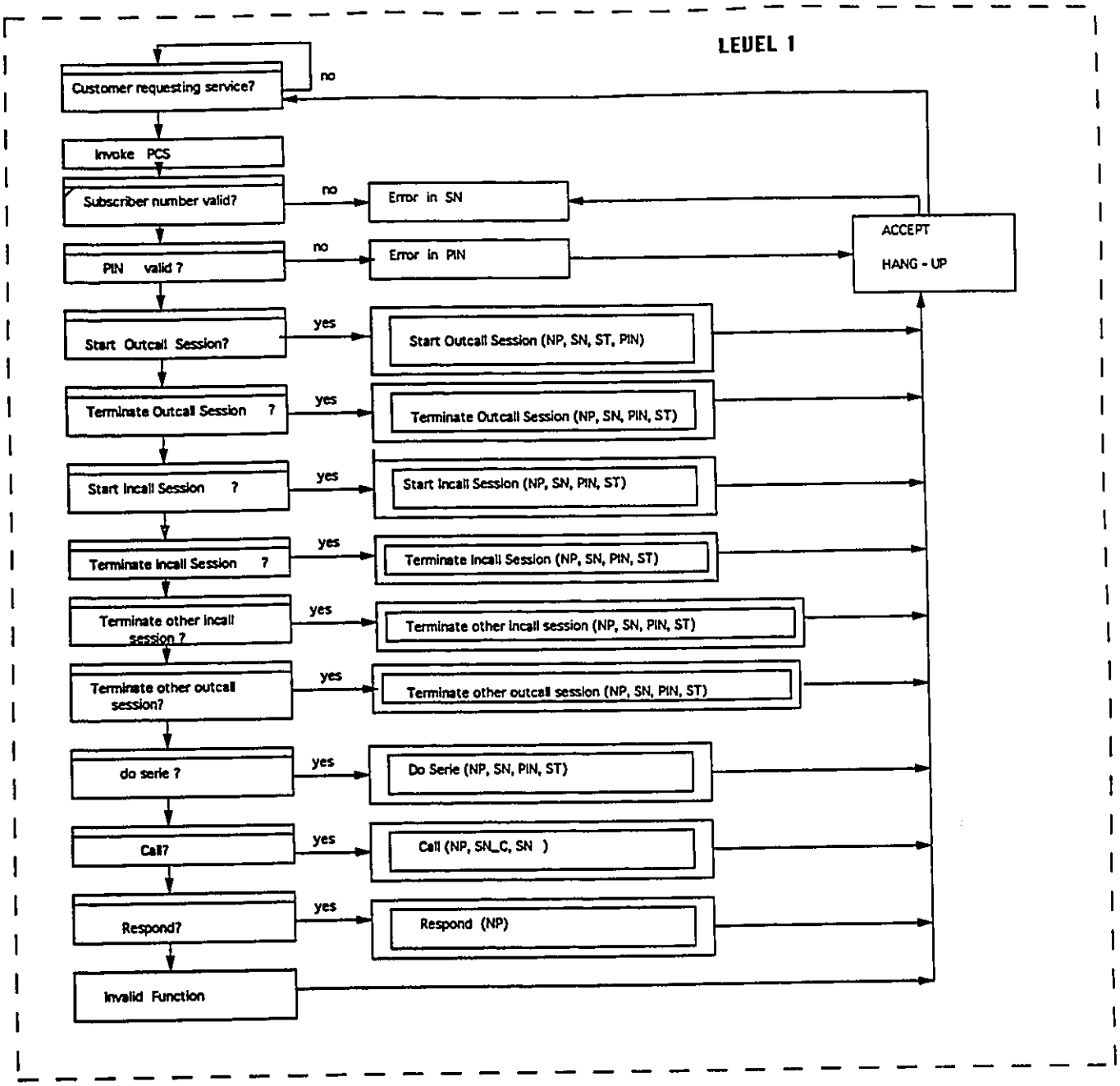
```
procedure select_user( UserValue : string ) : USER =

  ifexist u : USER
     suchthat u.name = UserValue
    then
      return u;
    else
      writeln " IL N'YA PAS D'USAGER AVEC CE NOM " ;
      return nil;
  end;

endproc select_user


procedure  read_integer( prompt : string ) : integer =

  write prompt;
  return (var_int.get)^;

endproc read_integer


endtype


behavior

   define
     pin1 = new PIN( assign_initial_value( "pin1" ) ),
     pin2 = new PIN( assign_initial_value( "pin2" ) ),
     pin3 = new PIN( assign_initial_value( "pin3" ) ),

     sn1 = new SN( "sn1" ),
     sn2 = new SN( "sn2" ),
     sn3 = new SN( "sn3" ),


     subs1 = new SUBSCRIBER( "ADDRESS 1111111") as
                USER( "sub1" ),

     subs2 = new SUBSCRIBER( "ADDRESS 2222222") as
                USER( "sub2"),

     subs3 = new SUBSCRIBER( "ADRESSE 3333333") as
                USER( "sub3"),

     np1 = new NP( "np1", assign_initial_value( "idle" ) ),
     np2 = new NP( "np2", assign_initial_value( "idle" ) ),
     np3 = new NP( "np3", assign_initial_value( "idle" ) ),


     in
        eval new AUTHENTICATION( subs1, sn1, pin1 );
        eval new AUTHENTICATION( subs2, sn2, pin2 );
        eval new AUTHENTICATION( subs3, sn3, pin3 );

        eval new  HOME_LOCATION as
                  LOCATION_RELATIONSHIP(sn1,st1,np1);
        eval new  HOME_LOCATION as
                  LOCATION_RELATIONSHIP(sn1,st2,np1);
        eval new  HOME_LOCATION as
                  LOCATION_RELATIONSHIP(sn2,st1,np2);
        eval new  HOME_LOCATION as
```

```
                LOCATION_RELATIONSHIP(sn2,st2,np2);
        eval new  HOME_LOCATION as
                LOCATION_RELATIONSHIP(sn3,st1,np3);
        eval new  HOME_LOCATION as
                LOCATION_RELATIONSHIP(sn3,st2,np3);

        eval new INCALL_LOCATION as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn1,st1,np1);
        eval new INCALL_LOCATION as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn1,st2,np1);
        eval new INCALL_LOCATION as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn2,st1,np2);
        eval new INCALL_LOCATION as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn2,st2,np2);
        eval new INCALL_LOCATION as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn3,st1,np3);
        eval new INCALL_LOCATION as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn3,st2,np3);

        eval new OWNER( sn1, np1 );
        eval new  OWNER( sn2, np2 );
        eval new  OWNER( sn3, np3 );

        eval new OUTCALL_SN_RESPONSIBLE as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn1,st1,np1);
        eval new  OUTCALL_SN_RESPONSIBLE as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn1,st2,np1);
        eval new  OUTCALL_SN_RESPONSIBLE as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn2,st1,np2);
        eval new  OUTCALL_SN_RESPONSIBLE as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn2,st2,np2);
        eval new  OUTCALL_SN_RESPONSIBLE as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn3,st1,np3);
        eval new  OUTCALL_SN_RESPONSIBLE as
                TIMED_LOCATION_RELATIONSHIP( 0 ) as
                LOCATION_RELATIONSHIP(sn3,st2,np3);

        eval new USER( "user1" );
        eval new USER( "user2" );
        eval new USER( "user3" );
        eval new HORLOGE_EXTERNAL ;
        eval new HORLOGE_INTERNAL
( assign_initial_valuei(0), assign_initial_valuei(0)) ;
        eval new SIMULATION_TESTER;

   end; (define)

where

   procedure assign_initial_value( s : string ) : var[string] =

     define v = new var[string]
     in
       v := s;
       return v;
     end;
   endproc assign_initial_value


   procedure assign_initial_valuei( s : integer ) : var[integer] =

     define v = new var[integer]
     in
       v := s;
       return v;
     end;
   endproc assign_initial_valuei


endunit spec
```

APPENDIX B :

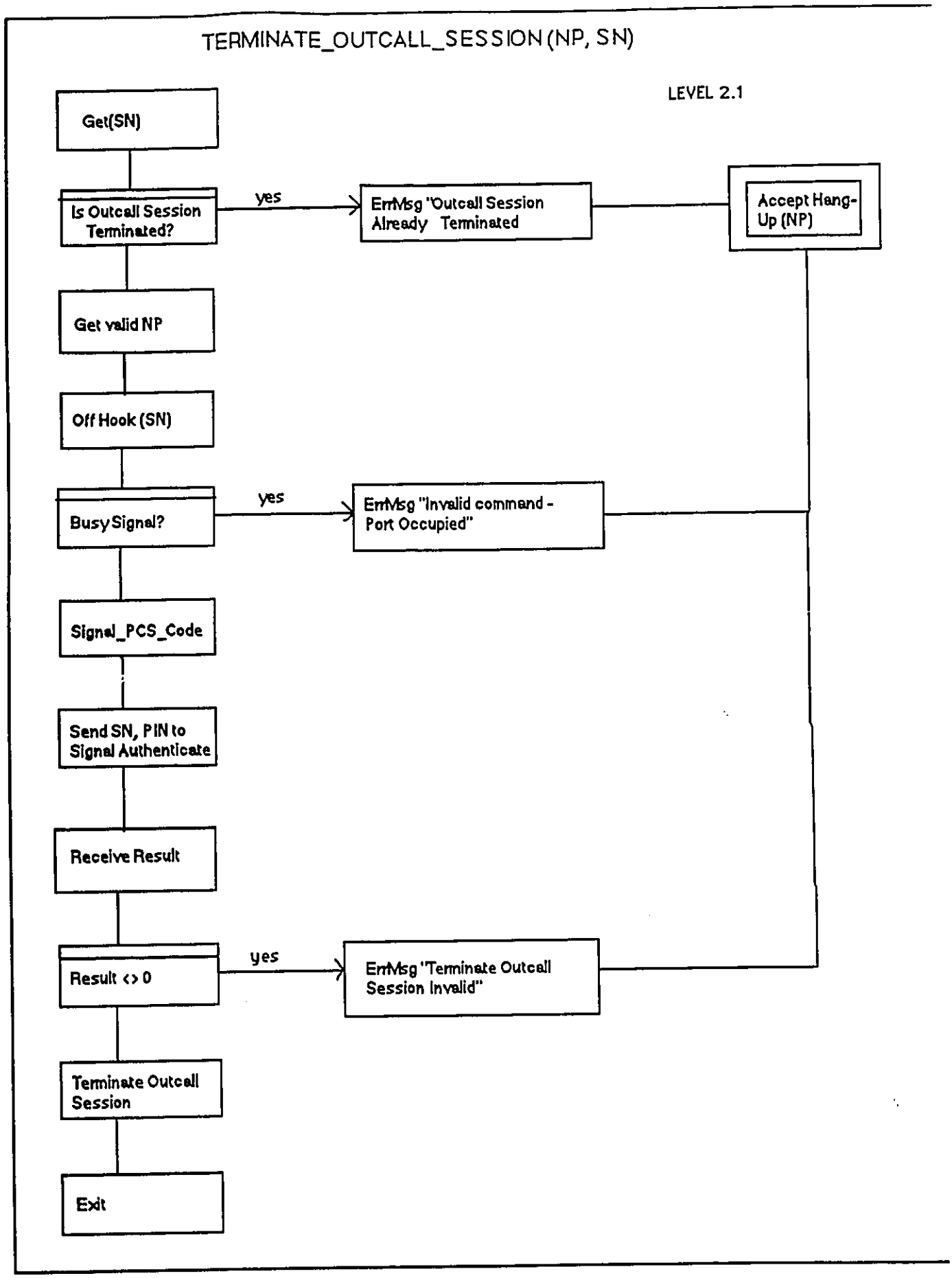
MAPPING TO SDM's

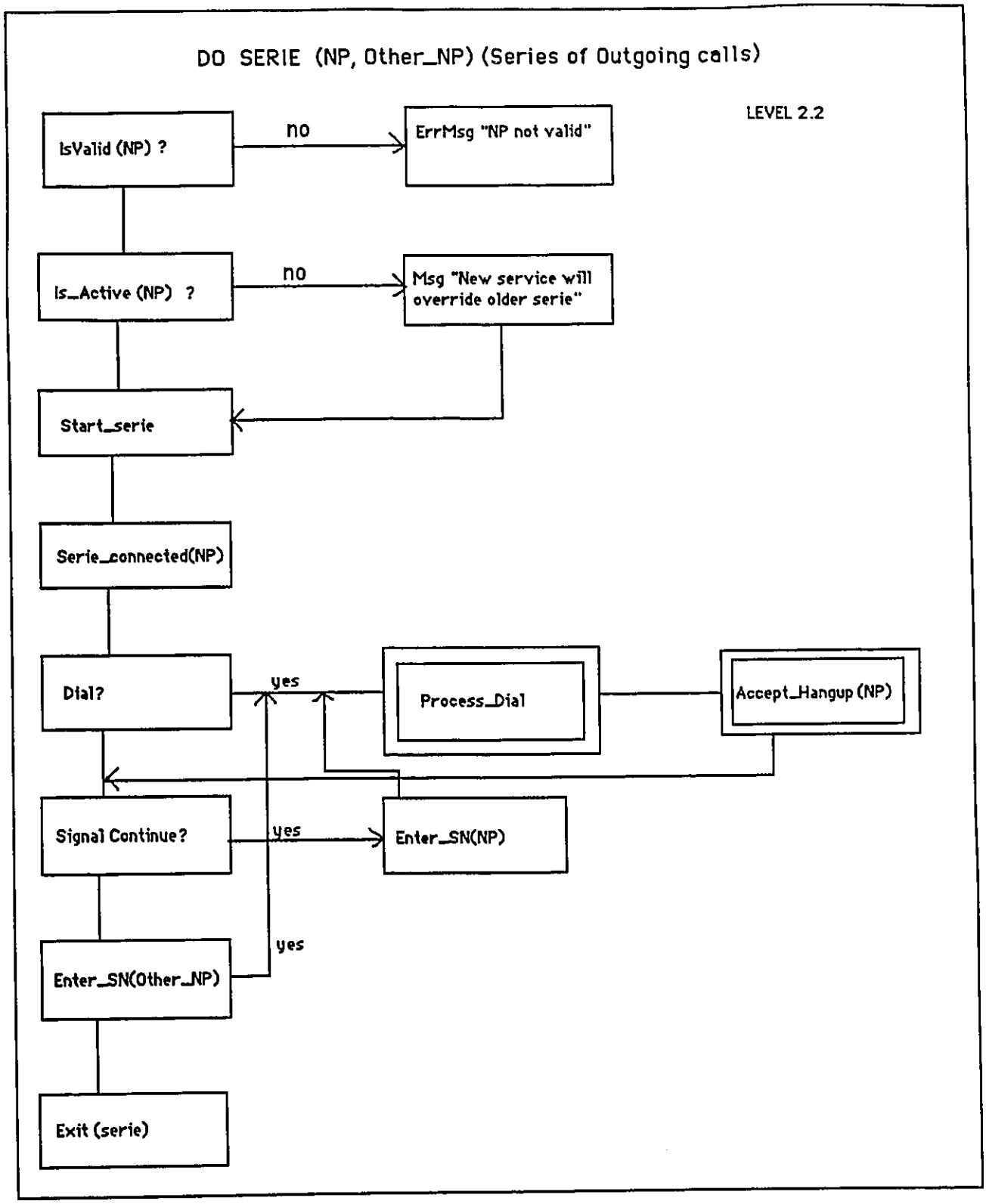
The mapping process of the PCS Specification to SDM's has a one-to-one mapping.The highest level is Level 1.The customer requests service, invokes PCS, once the SN and PIN are valid.The customer then has the option of selecting one of nine functions.Level 2 would then expand each of these nine functions separately to a higher sublevel. This mapping to SDM's follows from the PCS Specification in the previous appendix.



HIGH LEVEL SDM REPRESENTATION

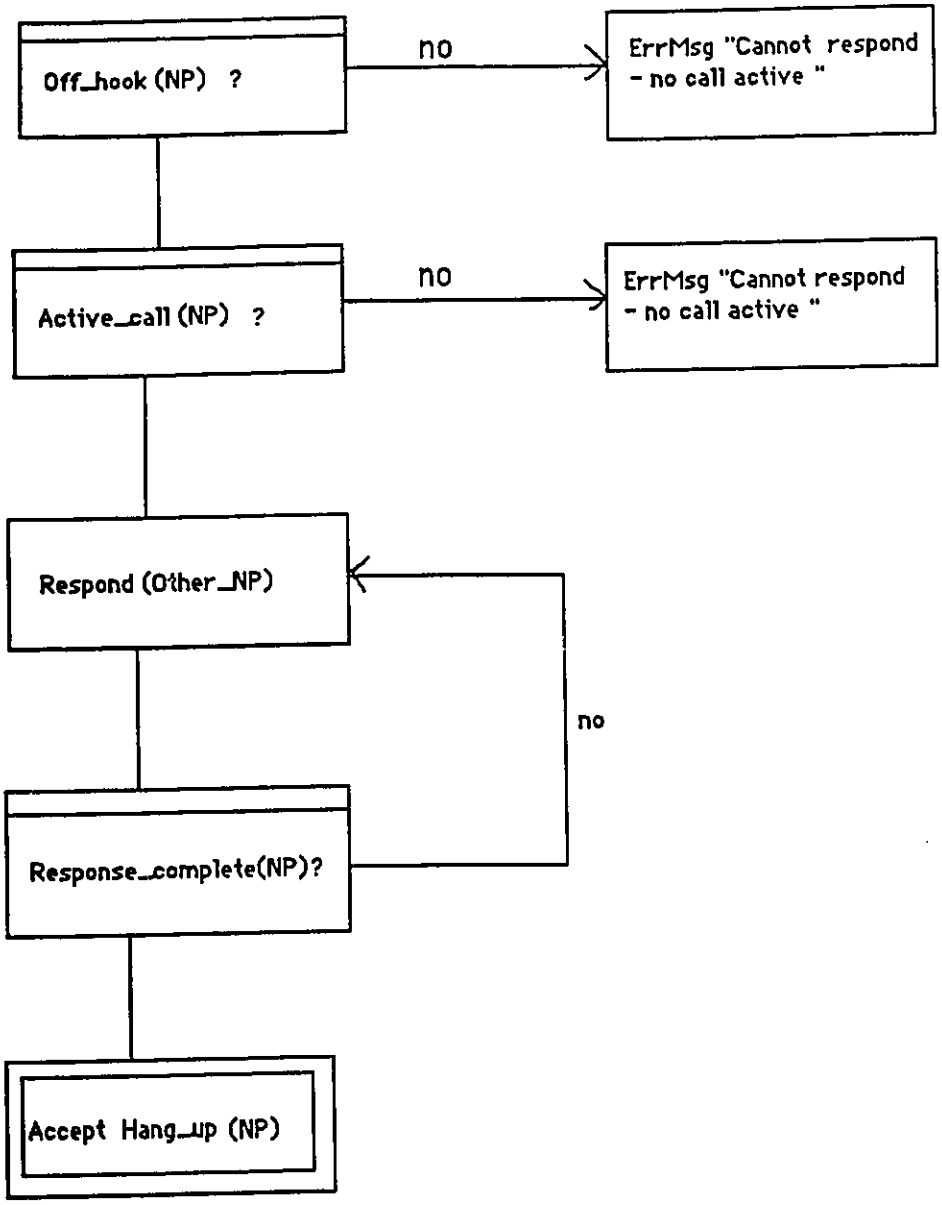
FOR PERSONAL COMMUNICATION SYSTEMS

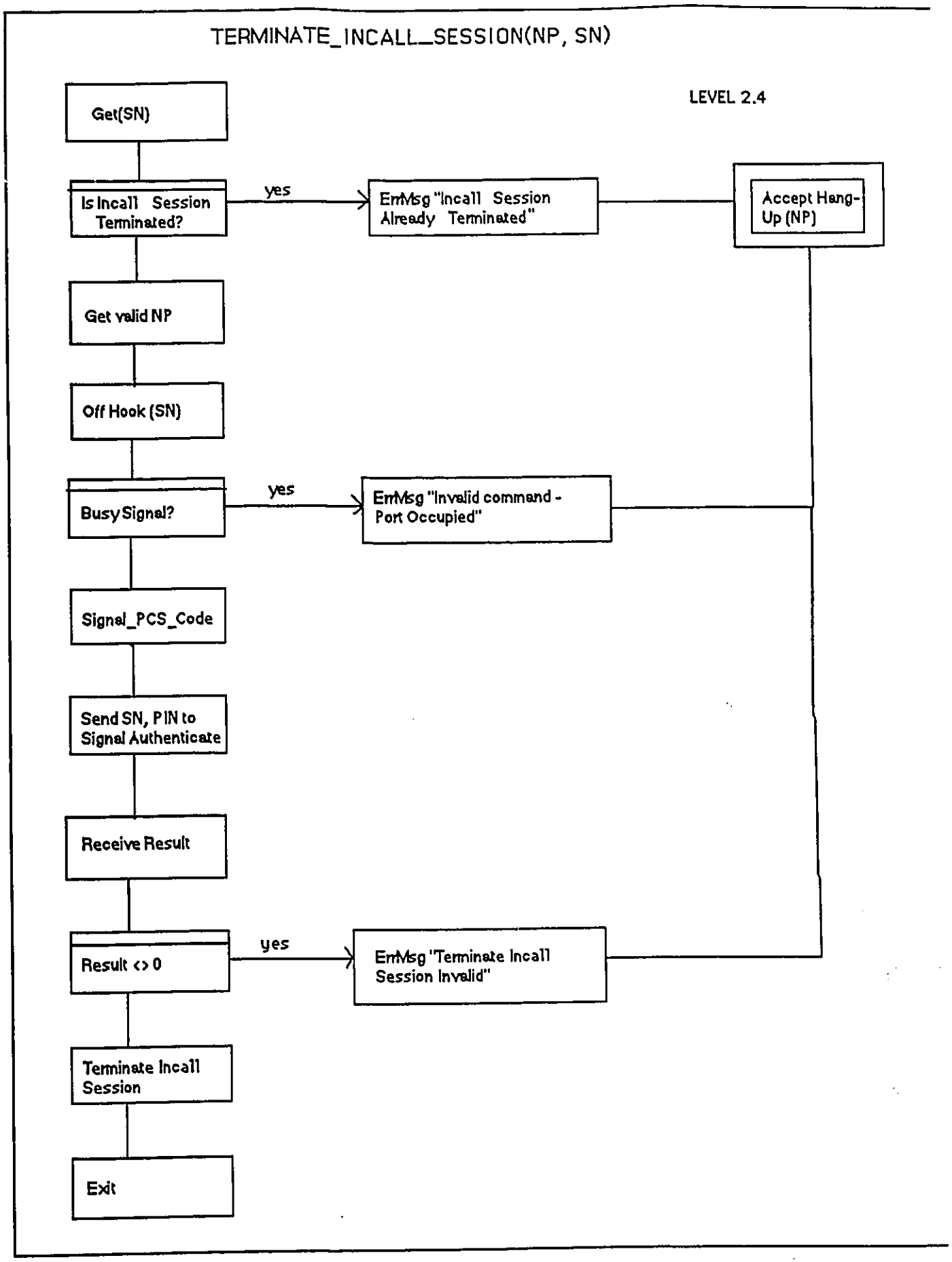

TERMINATE_OUTCALL_SESSION (NP, SN)
LEVEL 2.1
Get(SN)
Is Outcall Session Terminated?
yes
ErrMsg "Outcall Session Already Terminated
Accept Hang-Up (NP)
Get valid NP
Off Hook (SN)
Busy Signal?
yes
ErrMsg "Invalid command - Port Occupied"
Signal_PCS_Code
Send SN, PIN to Signal Authenticate
Receive Result
Result <> 0
yes
ErrMsg "Terminate Outcall Session Invalid"
Terminate Outcall Session
Exit

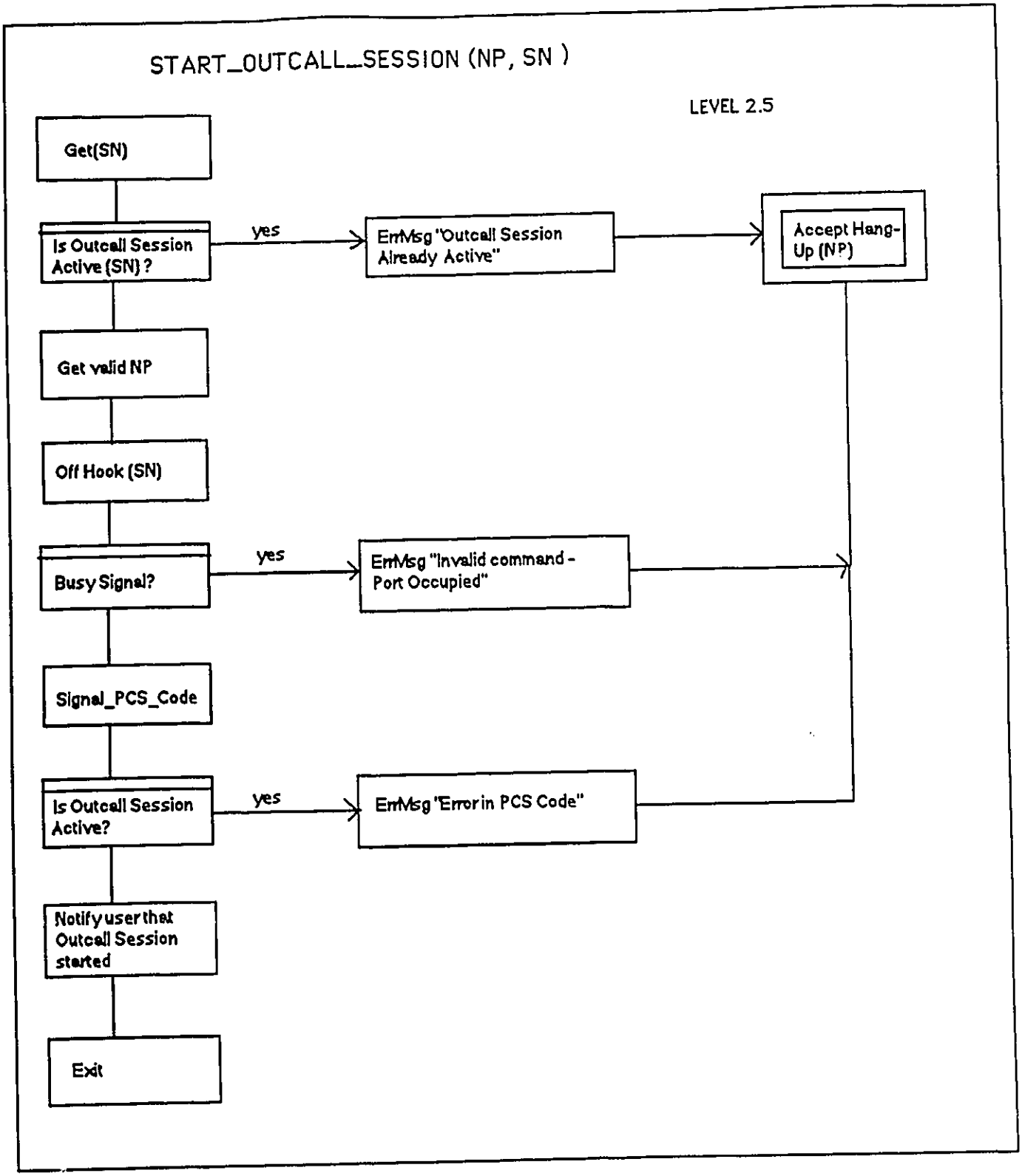

DO SERIE (NP, Other_NP) (Series of Outgoing calls)
LEVEL 2.2
IsValid (NP) ?
no
ErrMsg "NP not valid"
Is_Active (NP) ?
no
Msg "New service will override older serie"
Start_serie
Serie_connected(NP)
Dial?
yes
Process_Dial
Accept_Hangup (NP)
Signal Continue?
yes
Enter_SN(NP)
Enter_SN(Other_NP)
yes
Exit (serie)

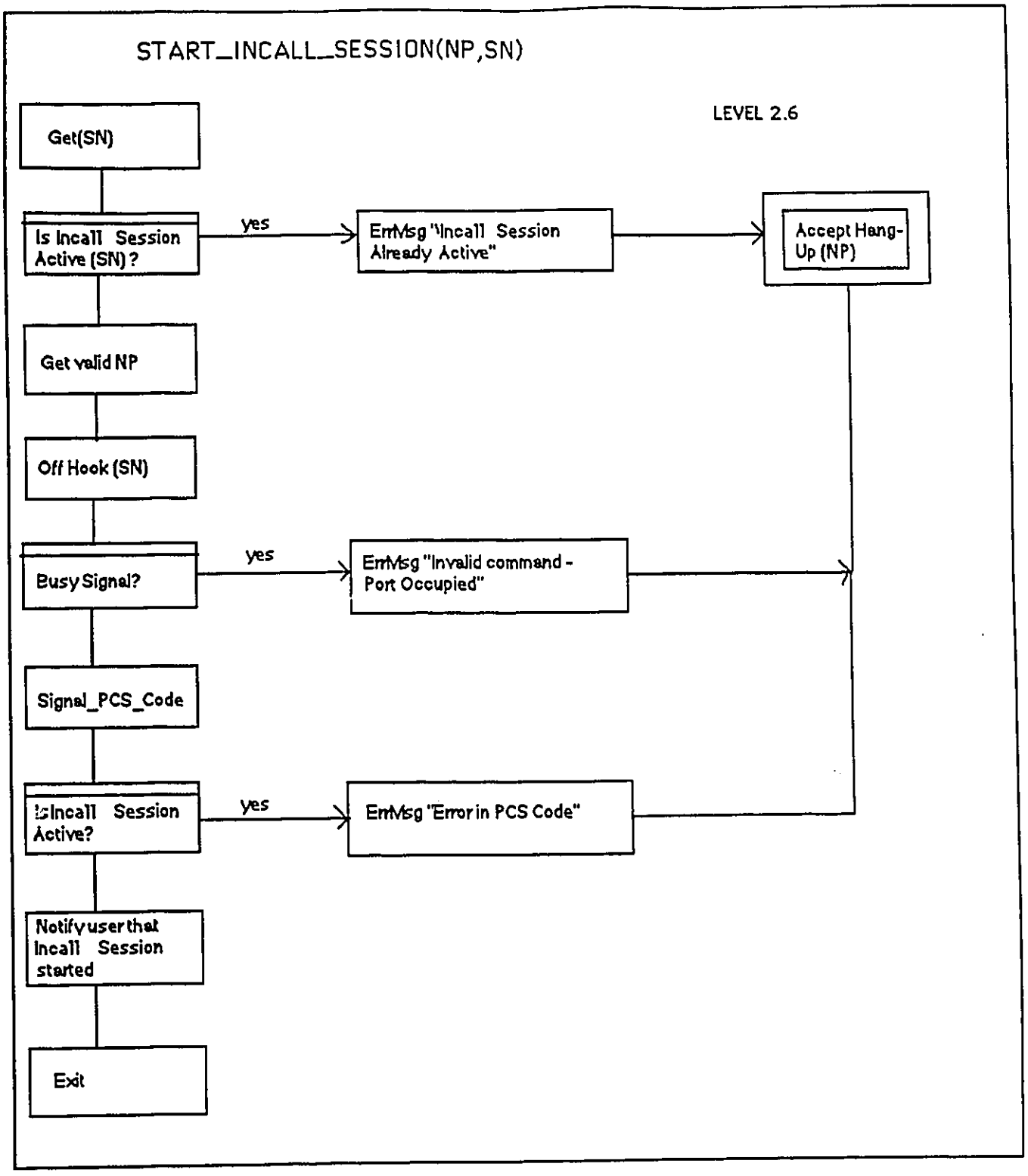
RESPONSE (NP, OTHER_NP)

LEVEL 2.3



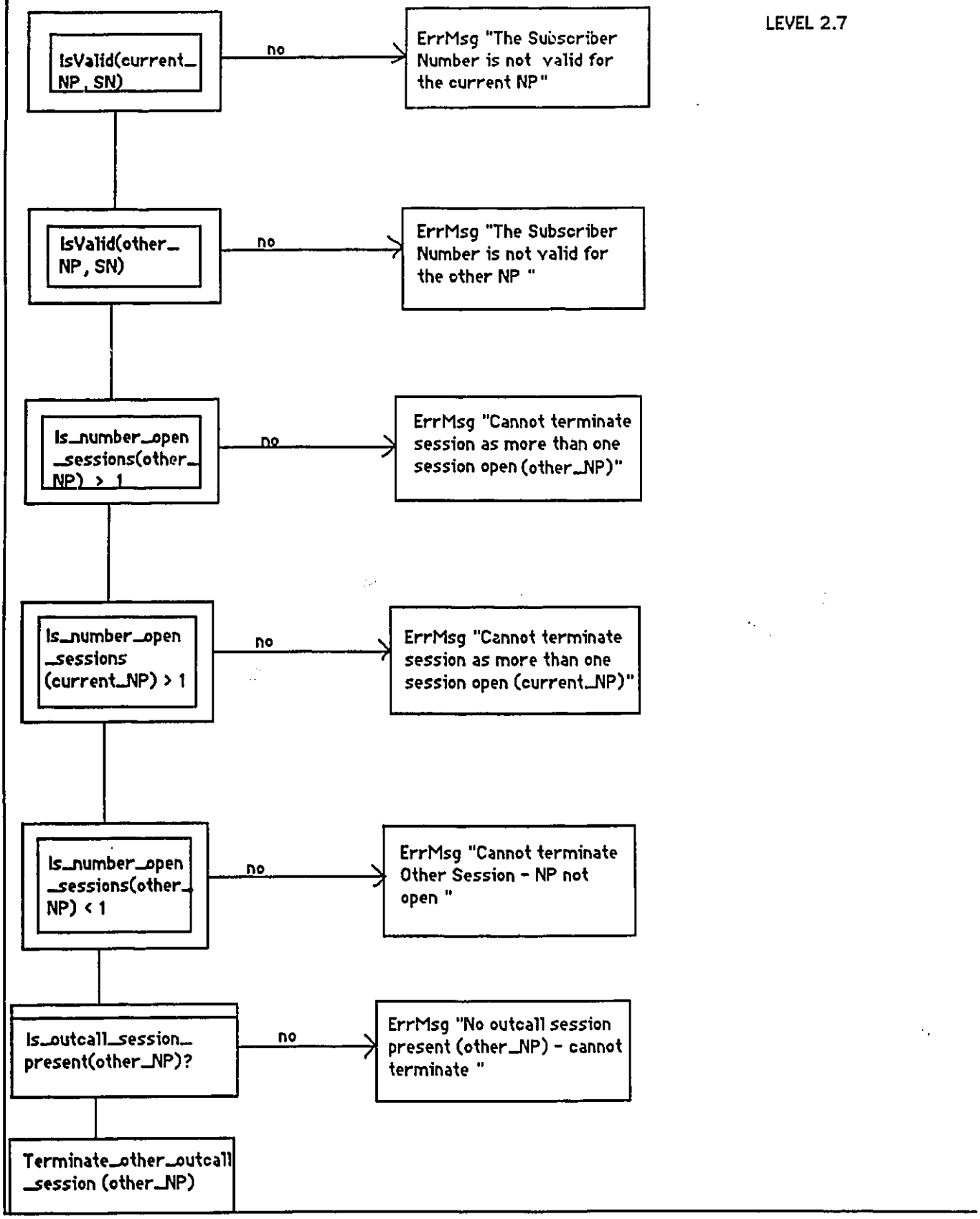

TERMINATE_INCALL_SESSION(NP, SN)
LEVEL 2.4
Get(SN)
Is Incall Session Terminated?
yes
ErrMsg "Incall Session Already Terminated"
Accept Hang-Up (NP)
Get valid NP
Off Hook (SN)
Busy Signal?
yes
ErrMsg "Invalid command - Port Occupied"
Signal_PCS_Code
Send SN, PIN to Signal Authenticate
Receive Result
Result <> 0
yes
ErrMsg "Terminate Incall Session Invalid"
Terminate Incall Session
Exit

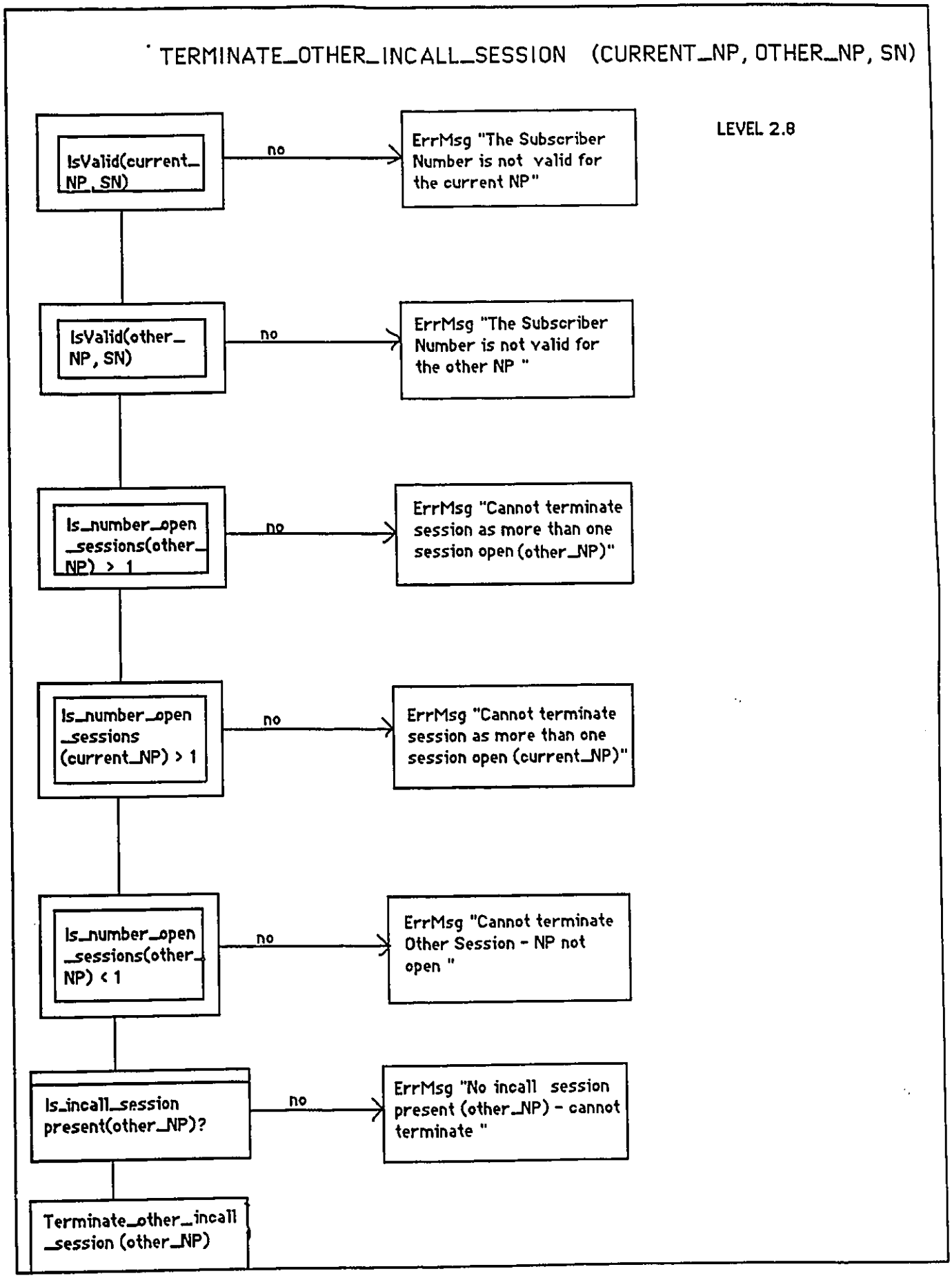

START_OUTCALL_SESSION (NP, SN)
LEVEL 2.5
Get(SN)
Is Outcall Session Active (SN) ?
yes
ErrMsg "Outcall Session Already Active"
Accept Hang-Up (NP)
Get valid NP
Off Hook (SN)
Busy Signal?
yes
ErrMsg "Invalid command - Port Occupied"
Signal_PCS_Code
Is Outcall Session Active?
yes
ErrMsg "Error in PCS Code"
Notify user that Outcall Session started
Exit

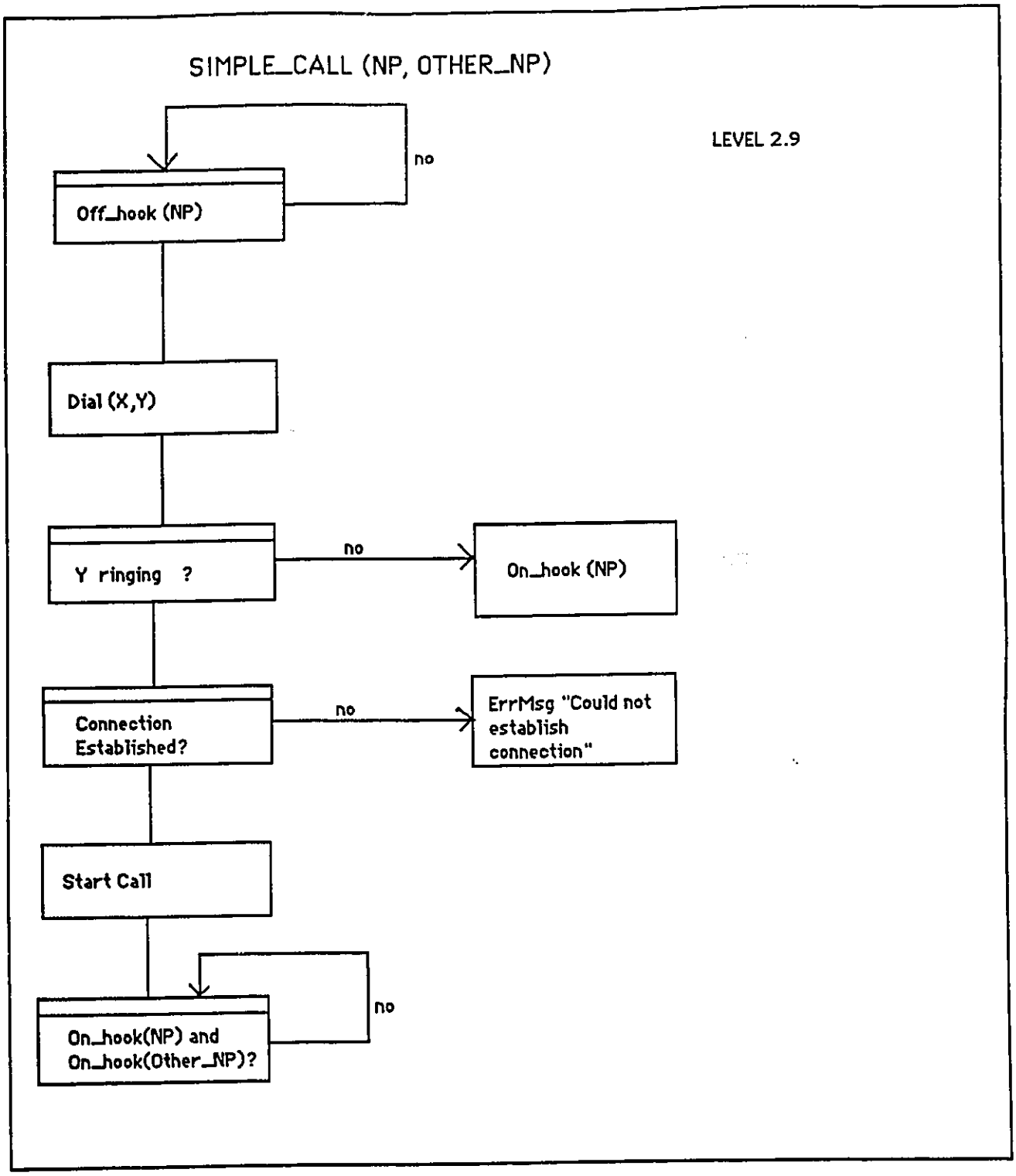

START_INCALL_SESSION(NP,SN)
LEVEL 2.6
Get(SN)
Is Incall Session Active (SN)?
yes
ErrMsg "Incall Session Already Active"
Accept Hang-Up (NP)
Get valid NP
Off Hook (SN)
Busy Signal?
yes
ErrMsg "Invalid command - Port Occupied"
Signal_PCS_Code
Is Incall Session Active?
yes
ErrMsg "Error in PCS Code"
Notify user that Incall Session started
Exit

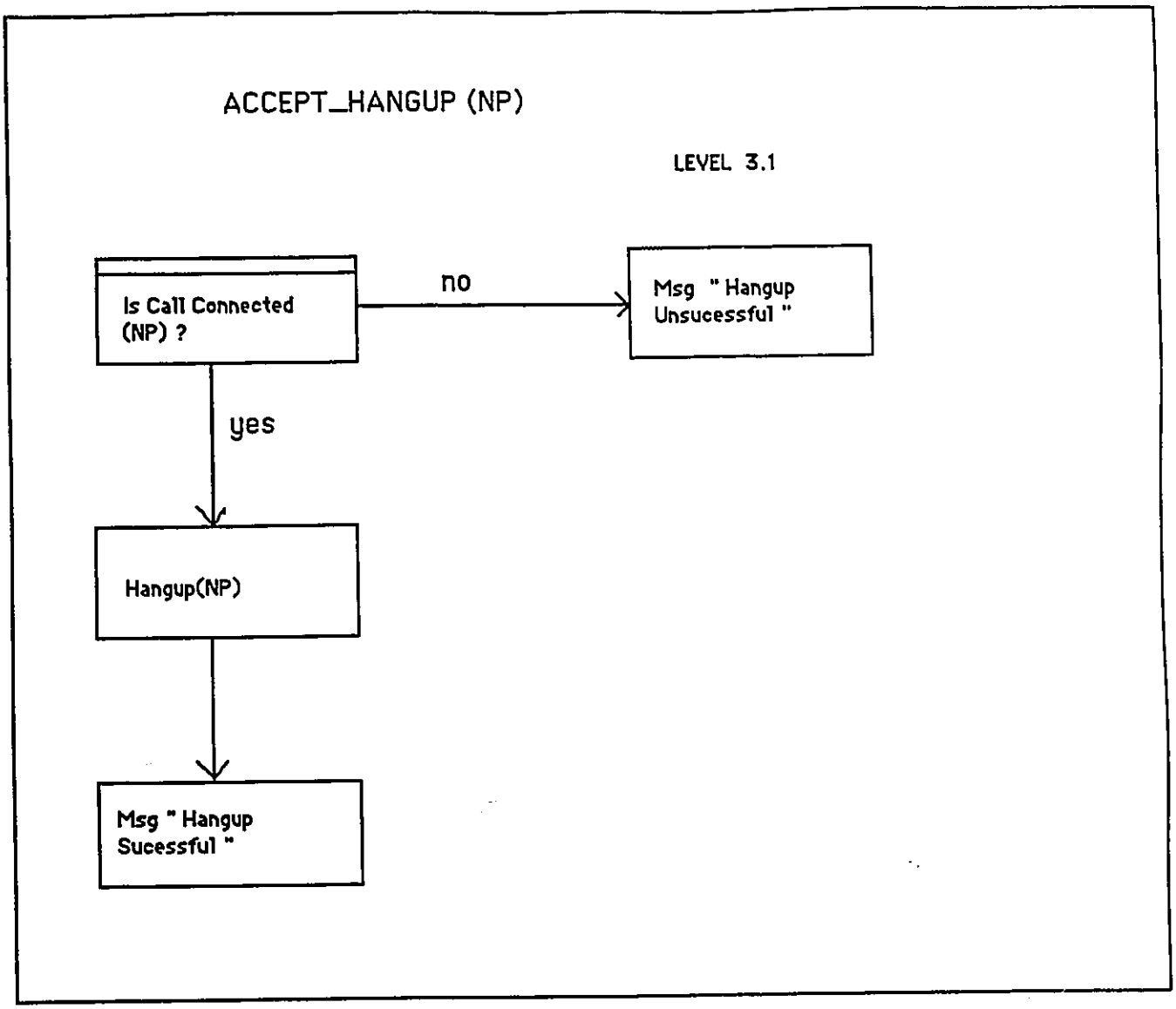
TERMINATE_OTHER_OUTCALL_SESSION (CURRENT_NP, OTHER_NP, SN)

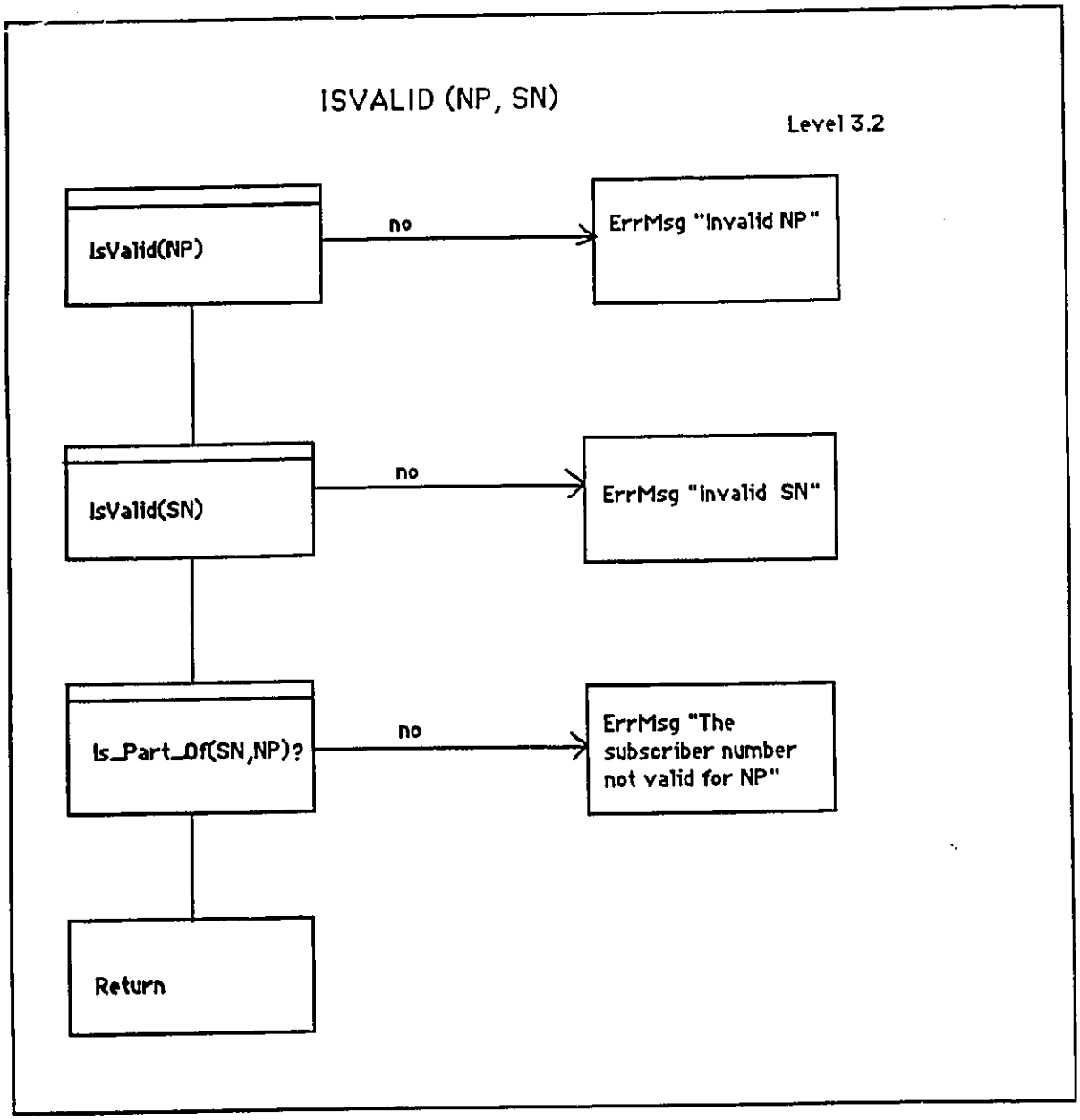
LEVEL 2.7

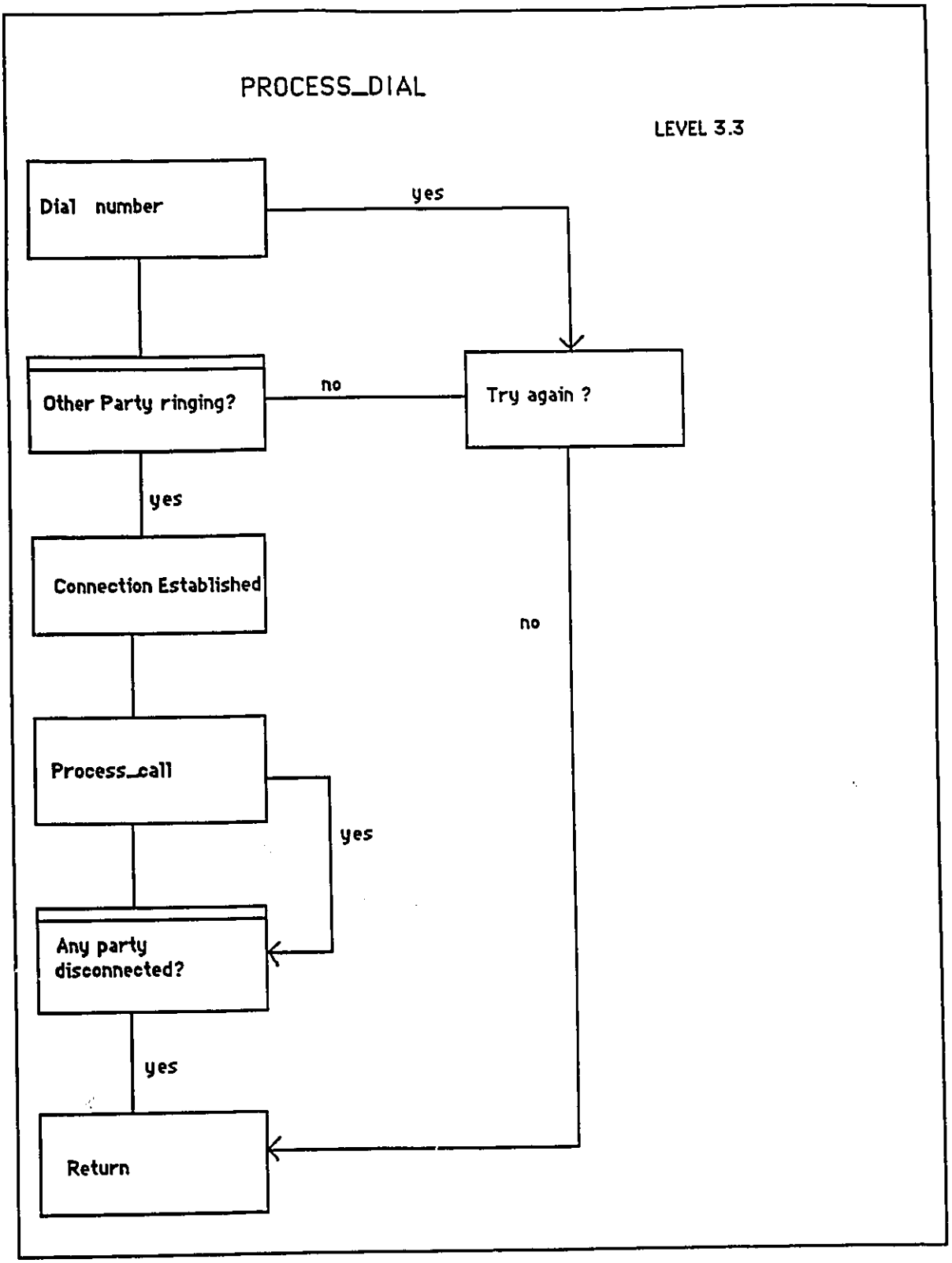


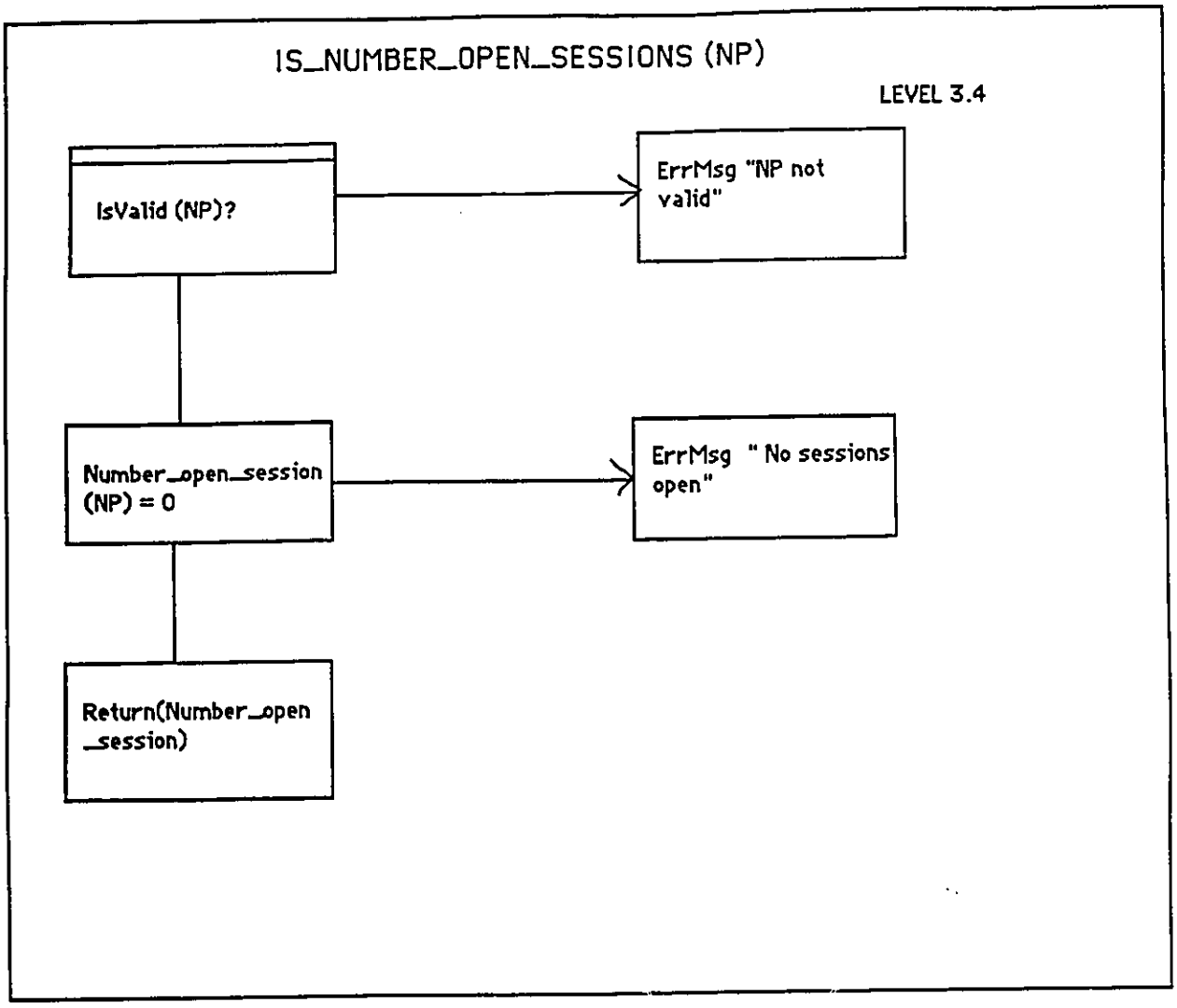

TERMINATE_OTHER_INCALL_SESSION (CURRENT_NP, OTHER_NP, SN)
LEVEL 2.8
IsValid(current_NP, SN)
no
ErrMsg "The Subscriber Number is not valid for the current NP"
IsValid(other_NP, SN)
no
ErrMsg "The Subscriber Number is not valid for the other NP "
Is_number_open_sessions(other_NP) > 1
no
ErrMsg "Cannot terminate session as more than one session open (other_NP)"
Is_number_open_sessions (current_NP) > 1
no
ErrMsg "Cannot terminate session as more than one session open (current_NP)"
Is_number_open_sessions(other_NP) < 1
no
ErrMsg "Cannot terminate Other Session - NP not open "
Is_incall_session present(other_NP)?
no
ErrMsg "No incall session present (other_NP) - cannot terminate "
Terminate_other_incall_session (other_NP)


SIMPLE_CALL (NP, OTHER_NP)
LEVEL 2.9
no
Off_hook (NP)
Dial (X,Y)
Y ringing ?
no
On_hook (NP)
Connection Established?
no
ErrMsg "Could not establish connection"
Start Call
no
On_hook(NP) and On_hook(Other_NP)?


ACCEPT_HANGUP (NP)
LEVEL 3.1
Is Call Connected (NP) ?
no
Msg " Hangup Unsucessful "
yes
Hangup(NP)
Msg " Hangup Sucessful "


ISVALID (NP, SN)
Level 3.2
IsValid(NP)
no
ErrMsg "Invalid NP"
IsValid(SN)
no
ErrMsg "Invalid SN"
Is_Part_Of(SN,NP)?
no
ErrMsg "The subscriber number not valid for NP"
Return


PROCESS_DIAL
LEVEL 3.3
Dial number
yes
Other Party ringing?
no
Try again ?
yes
Connection Established
no
Process_call
yes
Any party disconnected?
yes
Return


IS_NUMBER_OPEN_SESSIONS (NP)
LEVEL 3.4
IsValid (NP)?
ErrMsg "NP not valid"
Number_open_session (NP) = 0
ErrMsg " No sessions open"
Return(Number_open_session)

APPENDIX C :

OUTPUT RUNS

FOR TOOL

This Appendix is explained in sections 5.2 and 5.3

This Appendix shows the Output run of a Symbolic Scenario Prototype. The prototype is not complete by any means, it is used to show the feasibility of implementing such a product.

The following options are presented:
1.Create SDM
2.Execute SDM
3.Generate Traces
4.Show Coverage
5.Quit

1. The Create SDM options takes input from a file that contains data in a predefined format. The prototype takes that information and stores it in tables.

The code for the prototype has been written in such a way as to retrieve SDM information from a file instead of from the graphical editor.
Once trace information is requested from the prototype, the user is asked to input the filename. The file is then executed and traces are generated.

e.g.. of the file :

```
I1 Start_Outcall_Session  2 3
2 NP_Active 7  9
```

When broken down, the first line means
State I1 is Start_Outcall_Session.
I denotes the node to be an invocation node.
If the user wants to start the outcall session , then we go to state 2 else to state 3.

If the node is not an invocation node the both the yes and no branches have the same value.
e.g..

```
I1 Accept_hangup 3 3
```

Traces are generated by executing this file, line by line. Each line is executed starting from the first. Number of lines executed / total number of lines in the file
* 100 gives the trace percentage.

```
Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit

Input number 1-5 :


Creating SDM .....

input filename to retrieve data :

opening sss.data
finished creating !

Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit

Input number 1-5 :


Executing SDM .....

Answer y/n to the following questions. Type * to quit anytime.

Customer requesting service (y/n/*)?

Invoke PCS

Subscriber Number Valid?

PIN valid?

Start Outcall Session?

Terminate Outcall Session?

Start Incall Session?

Terminate Incall Session?

Terminate Other Incall Session?

Terminate Other Outcall Session?

Do serie?

Call?

Respond?

Invalid Function

End of Execute ....
Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit
```

```
Input number 1-5 :

Showing coverage .....

Level 1 was executed - 54 % of the states were covered
Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit

Input number 1-5 :

Generating Traces ....
Options :
1. Generate Traces for specific SDM number
2. Generate Traces for Entire SDM
3. Quit

Input number 1-3 :


Input SDM number

SDM - Terminate_Outcall_Session
Number - 2.1

Generating Traces ....

Trace #1
Get(SN)
Outcall_session_terminated
Error - Outcall Session Already Terminated
Accept Hangup (NP)

Trace #2
Get(SN)
Not Outcall_session_terminated
Get Valid NP
Off Hook (SN)
Busy Signal
Error - Invalid Command - Port Occupied
Accept Hangup (NP)

Trace #3
Get(SN)
Not Outcall_session_terminated
Get Valid NP
Off Hook (SN)
Not Busy Signal
Signal PCS code
Send SN, PIN to Signal Authenticate
Receive result
Result <> 0
Error - Terminate Outcall Session Invalid
Accept Hangup (NP)

Trace #4
Get(SN)
Not Outcall_session_terminated
Get Valid NP
Off Hook (SN)
Not Busy Signal
Signal PCS code
```

```
Send SN, PIN to Signal Authenticate
Receive result
Not Result <> 0
Terminate Outcall Session
Exit

Finished Tracing for level 2.1

Options :
1. Generate Traces for specific SDM number
2. Generate Traces for Entire SDM
3. Quit

Input number 1-3 :


Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit

Input number 1-5 :

Showing coverage .....

Level 2.1 was executed - 100 % of the states were executed at least once
Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit

Input number 1-5 :

Options :
1. Generate Traces for specific SDM number
2. Generate Traces for Entire SDM
3. Quit

Input number 1-3 :


Input SDM number

SDM - Do Serie
Number - 2.2

Generating Traces ....

Trace #1
Not Isvalid(NP)
Error - NP not valid

Trace #2
Isvalid(NP)
Not Isactive(NP)
Message - New service will override older serie
Start serie
Serie connected(NP)
Dial
Process_Dial
Accept Hangup
Not Signal Continu
```

```
Not Enter SN
Exit (serie)

Trace #3
Isvalid(NP)
Isactive(NP)
Start serie
Serie connected(NP)
Not Dial
Signal Continu
Enter SN (NP)
Dial
Not Signal Continu
Enter SN (other_NP)
Dial
Not Signal Continu
Not Enter SN (other_NP)
Exit

Finished Tracing for level 2.2

Options :
1. Generate Traces for specific SDM number
2. Generate Traces for Entire SDM
3. Quit

Input number 1-3 :


Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit

Input number 1-5 :

Showing coverage .....

Level 2.2 was executed - 100 % of the states were executed at least once

Options :
1. Create SDM
2. Execute SDM
3. Generate Traces
4. Show Coverage
5. Quit

Input number 1-5 :

Quitting .....
```

APPENDIX D:

CODE LISTINGS

FOR SSS TOOL

```
/*****************************************************************************

Code for the Symbolic Scenario Selector

Code to create the GUI

*****************************************************************************/
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/cms.h>
#include <xview/xv_xrect.h>

#define WHITE  0
#define RED    1
#define GREEN  2
#define BLUE   3
#define ORANGE 4
#define AQUA   5
#define PINK   6
#define BLACK  7


GC gc;
unsigned long *colors;


main(argc, argv)
int argc;
char *argv[];
{
    static char stipple_bits[] = {0xAA, 0xAA, 0x55, 0x55};
    static Xv_singlecolor cms_colors[]  = {
          { 255, 255, 255 },
          { 255, 0, 0 },
          { 0, 255, 0 },
          { 0, 0, 255 },
          { 250, 130, 80 },
          { 30, 230, 250 },
          { 230, 30, 250 },};

Cms  cms;
Frame  frame;
Canvas  canvas;
XFontStruct  *font;
Display  *display;
XGCValues  gc_val;
XID  xid;
void   canvas_repaint();
Xv_cmsdata  cms_data;
int use_dynamic = FALSE;
xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
if (*++argv && !strcmp(*argv, "-dynamic"))
   use_dynamic = TRUE;

frame = xv_create(NULL, FRAME,
   FRAME_LABEL,   "xv_canvas_x_draw",
   XV_WIDTH,      400,
   XV_HEIGHT,     300,
   NULL);

cms = xv_create(NULL, CMS,
      CMS_SIZE,      7,
      CMS_TYPE,      use_dynamic? XV_DYNAMIC_CMS : XV_STATIC_CMS,
      CMS_COLORS,    cms_colors,
      NULL);
```

```
canvas = xv_create(frame, CANVAS,
      CANVAS_REPAINT_PROC, canvas_repaint,
      CANVAS_X_PAINT_WINDOW, TRUE,
      XV_VISUAL_CLASS, PseudoColor,
      WIN_CMS,       cms,
      NULL);

  display = (Display *)xv_get(frame, XV_DISPLAY);
  xid = (XID)xv_get(canvas_paint_window(canvas), XV_XID);

  if (!(font = XLoadQueryFont(display, "fixed"))) {
      puts("cannot load fixed font");
      exit (1);
 }
 gc_val.font = font ->fid;
  gc_val.stipple =
           XCreateBitmapFromData (display, xid, stipple_bits, 16, 2);
   gc = XCreateGC(display, xid, GCFont | GCStipple, &gc_val);

   colors = (unsigned long *)xv_get(canvas, WIN_X_COLOR_INDICES);

   xv_main_loop(frame);

}



void
canvas_repaint(canvas, pw, display, xid, xrects)
Canvas     canvas;
Xv_window   pw;
Display     *display;
Window      xid;
Xv_xrectlist   *xrects;
{
   static XPoint box[] = {
        {0,0}, {100, 100}, {0, -100}, {-100, 100}, {0, -100}
          };

   static XPoint points[] = {
        {0,0},
        {25, 0}, {25, 0}, {25, 0}, {25, 0}, {-100, 25},
        {25, 0}, {25, 0}, {25, 0}, {25, 0}, {-100, 25},
        {25, 0}, {25, 0}, {25, 0}, {25, 0}, {-100, 25},
        {25, 0}, {25, 0}, {25, 0}, {25, 0}, {-100, 25},
        {25, 0}, {25, 0}, {25, 0}, {25, 0}, {-100, 25},
       };

   XSetForeground(display, gc, colors[RED]);
   XDrawString(display, xid, gc, 30, 20, "XFillRectangle", 14);
   XFillRectangle(display, xid, gc, 25, 25, 100, 100);
   XSetFunction(display, gc, GXinvert);
   XFillRectangle(display, xid, gc, 50, 50, 50, 50);
   XSetFunction(display, gc, GXcopy);
  XSetForeground(display, gc, colors[BLACK]);
   XDrawString(display, xid, gc, 155, 20, "XFillRect - stipple", 19);
   XFillStyle(display, gc, FillStippled)
   XFillRectangle(display, xid, gc, 150, 25, 100, 100);
   XSetFillStyle(display, gc, FillSolid);

   XSetForeground(display, gc, colors[BLUE]);
   XDrawString(display, xid, gc, 280, 20, "XDrawPoints", 11);
   points[0].x = 275; points[0].y = 25;
   XDrawPoints(display, xid, gc, points,
         sizeof(points)/sizeof(XPoint), CoordModePrevious);
```

```
   XSetForeground(display, gc, colors[ORANGE]);
   XDrawString(display, xid, gc, 30, 145, "XDrawLine - solid", 17);
   XDrawLine(display, xid, gc, 25, 150, 125, 250);
   XDrawLine(display, xid, gc, 25, 250, 125, 150);

   XSetForeground(display, gc, colors[AQUA]);
   XDrawString(display, xid, gc, 155, 145, "XDrawLine - dashed", 18);
   XSetLineAttributes(display, gc, 5,
              LineDoubleDash, CapButt, JoinMiter);
   XDrawLine(display, xid, gc, 150, 150, 250, 250);
   XDrawLine(display, xid, gc, 150, 250, 250, 150);
   XSetLineAttributes(display, gc, 0, LineSolid, CapButt, JoinMiter);

   XSetForeground(display, gc, colors[PINK]);
   XDrawString(display, xid, gc, 280, 145, "XDrawLines", 10);
   box[0].x = 275; box[0].y = 150;
   XDrawLines(display, xid, gc, box, 5, CoordModePrevious);

   XSetForeground(display, gc, colors[GREEN]);
   XDrawRectangle(display, xid, gc,
          5, 5, xv_get(pw, XV_WIDTH)-10, xv_get(pw, XV_HEIGHT)-10);
   XDrawRectangle(display, xid, gc,
         7, 7, xv_get(pw, XV_WIDTH)-14, xv_get(pw, XV_HEIGHT)-14);

}
/*****************************************************************************/
Frame frame;

destroy (argc,argv)
int argc;
char *argv[];
{

 Panel panel;
 void quit();

 xv_init (XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

  frame = (Frame)xv_create (NULL, FRAME,
      FRAME_LABEL,  argv[0],
      XV_WIDTH, 200,
      XV_HEIGHT, 100,
      NULL);

  panel = (Panel)xv_create (frame, PANEL, NULL);

  (void) xv_create (panel, PANEL_BUTTON,
            PANEL_LABEL_STRING,   "Quit",
            PANEL_NOTIFY_PROC,   quit,
            NULL);

   xv_main_loop (frame);
   exit (0);

}

void
quit()
{
   xv_destroy_safe(frame);
}

/*****************************************************************************/

   SDM interpreter code
```

```
*****************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

int flag;
int choice;
char filename[50];
FILE *fp;
char *eng_st_des[100];
int state_id[100]; nsify[100]; nsifn[100];
int p=0;
int id=0;
char inpar[256]=0;
char buf[256]=0;
char outa[256]=0;
char ala[256]=0;
char s[];

#define MAXLINE  100



void getl(line, max)
char line[];
int max;

{
int pq = 0;

while (fgets(line, max, fp) != NULL)
 break_l(line);

return;
}



break_l(sline)
char *sline;
{
char *db;
int i=0;

db = NULL;

while ((db = strtok(sline, " ")) != NULL)
{
i++;

assig_arr(i,db);
sline = NULL;
}
id++;
return;

}

assig_arr(count, lm)
int count;
char *lm;

{
 if (count != 2)
```

```
   p = atoi(lm);
     switch (count)
     {

      case 1:{ state_id[id] = p; break;}

      case 2:{ eng_st_des[id] = strdup(lm);  break;}

      case 3: {nsify[id] = p;  break;}

      case 4: {nsifn[id] = p;   break;}

      default:{ printf("error in assig"); break;}

      }

     return;
}



void Create_SDM()
{
 printf("Creating SDM .....\n");
 printf("input filename to retrieve data :\n");
 scanf(filename);
 fp= fopen(filename, "r");
 if ( fp  == NULL)
  {
   printf("can't open \n");
   return;
  }
  else
  {
   getl(s, MAXLINE);
  }
}

execute_dm()
{
    char  ans[256];
    int bid=0;


    while (bid != -1)
    {
        printf("\n\n");
        printf("%s\n",eng_st_des[bid]);
        printf("\n");
       if (nsify[bid] == nsifn[bid])
        { bid = nsify[bid]-1;
          printf(" do you want to change existing data (Y/N)?\n");
          scanf("%s",ans);
          if (!strcmp(ans,"y") || !strcmp(ans,"Y"))
           {printf("\n\n");fgets(ans,256,stdin);
          display();}
        }
       else
           {
             printf(" input Y/N\n");

             scanf("%s",ans);
           /*  printf("ans = %s\n",ans);*/
             if (!strcmp(ans,"y") || !strcmp(ans,"Y"))
             bid = nsify[bid]-1;
               else
```

```
                    if (!strcmp(ans,"n") || !strcmp(ans,"N"))
                      bid = nsifn[bid]-1;
                     else
                       {
                       printf("error in execute\n");
                       exit (1);
                       }
            }
      }

printf("stop");
return;
}



display()
{
 printf("\n\n");
  printf(" YOUR DATA SO FAR IS\n\n");
  printf(" alarm =%s\n",ala);
  printf("input area =%s\n",inpar);
  printf("buffer =%s\n",buf);
  printf("output area =%s\n\n\n",outa);

  printf(" PLEASE MODIFY YOUR DATA (* for blank)\n\n");

  printf("new alarm =");
   if ( fgets(ala,256,stdin) != NULL)
  printf("\n");
  printf("new input area = ");
   if ( fgets(inpar,256,stdin) != NULL)

  printf("\n");
  printf("new buffer is =");
  if ( fgets(buf,256,stdin) != NULL)
  printf("\n");
  printf("new output area =");
  if ( fgets(outa,256,stdin) != NULL)
  printf("\n");
  return;
}

void printoptions()
{
 printf("1. Create SDM \n");
 printf("2. Execute SDM \n");
 printf("3. Generate Traces \n");
 printf("4. Show Coverage \n");
 printf("5. Quit \n\n");
 flag = 0;
 while (flag = 0){
  printf("Input number 1-5 :\n");
  scanf(choice);
  if (choice < 1 || choice > 5)
    printf("The input value should be between 1 and 5");
  else
    flag = 1;
 }

 switch(choice)
 {
  case '1' : Create_SDM;
             break;
  case '2' : Execute_SDM;
             break;
```

```
  case '3' : Gen_Trace;
             break;
  case '4' : Show_cov;
             break;
  case '5' : Exit;
             break;

  default : printf("Error in choice");
 }

}



main()
{
 printoptions;
}
```