Rapid ADA Prototyping:
Principles and example of a complex application

F. Kordon  P. Estrallier  R. Card

Laboratory MASI - CNRS UA-818
University F. et M. Curie
4, place Jussieu
75252 Paris CEDEX 05

ABSTRACT

The automatic prototyping methodology presented here, derived from Petri net theory, has been developed for the PN_TAGADA project (Petri Net Translation, Analysis and Generation of ADA code).

Coloured Petri nets allow concise modelling and verification of distributed systems. Their quantitative analysis provides invariants which are of particular interest for rapid prototyping of parallel applications.

Management of a phone conversation, a real size, complex application, is used as an example to demonstrate the methodology of our ADA code generator. Description of subscriber behaviour and services available at a Private Automatic Branch Exchange (PABX) is presented along with a qualitative analysis of the model/Ada listing abstract of the task associated with a given subscriber is provided. Execution of the code generated for that model presents the process sequence involved in management of different subscribers.

1. INTRODUCTION

Rapid prototyping has become a widely accepted technique for system implementation. This is, in particular, justified for the following purposes:

- The system potential can be presented to customers without having to invest effort on implementing it completely. The rationale is that customers without extensive experience in system design may not be able to interpret written specifications but have to be able to see the behavior of a prototype.
- In order to experiment specific aspects of a system. In this case, the system designer wants to get experience using in a system prototype. He can validate his model in a more realistic situation than in simulation.

In both cases the primary goal is not to generate actual systems from prototypes. Although successive improvements of an initial prototype would make this possible, the approach is usually applied to obtain a preliminary implementation and to modify the model according to the results of an execution.

The aim of this paper is to present a comprehensive study of a real size system which has been prototyped from its Colored Petri net based specifications. In order to illustrate this, we have chosen to apply our methodology to phone conversation management. The final goal of our team is to realize a complete template prototype of any coloured Petri Net specification.

The first step was aimed at designing and validating a new protocol for the management of medium redundancy [Cousin 88b]. This step studies the establishment and release phases of a conversation between two subscribers. The objective is thus to quickly produce a prototype of subscriber behavior.

The paper is structured as follows. Basic concepts of the methodology and prototyping steps are presented in Section 2. Requirements for a PABX are recalled in Section 3. Section 4 describes the global Colored Petri Net model, specifying behavior of a subscriber and facilities provided by a PABX during establishment and release of a communication between two subscribers. Principles underlying code generation are presented in Section 5. Finally, the code generation based on the global model is commented in the last section. The formal Petri net model, the partial listing of a task implementing a subscriber and the execution of complete compiled generated Ada program are given.

2. PROTOTYPING METHODOLOGY

We have chosen Petri nets as a modeling tool because of its ability to yield theoretical results. Section 5 will present relations between linear invariants and task determination. Petri nets allow us to validate the model before generating a code. There are several advantages:

- The model is a formal specification describing the engineer's requirements.
- The model has been verified, no design errors are left, our starting point remains sound. This is important from a scientific and economic point of view.
- In order to define and to optimize objects involved in programming (process, data, resources, ...), we may use the structural and behavioral properties based on the model proof. The main idea is to reuse linear invariants given by the model validation. These invariants allow us to define the optimal number of processes.

Rapid code generation from a Petri net model is a process that requires several steps since the transformation distance (i.e. the change of degree of abstraction and specificity) is large.

This transformation is divided into several steps (Figure 1):

- The modeling step produces a formal model from precise requirements of both application characteristics and constraints. This kind of model is a complete specification of application behaviors.
- Because our goal is to produce a prototype, the designer must apply structural rules.
- The Implementation of an application, e.g. the code generation, is the logical issue of the previous step. Therefore, the model can be used as a kernel (validated at priori) of the specification.
3. REQUIREMENTS OF A MULTISERVICE PABX

3.1 Introduction

We have chosen to illustrate this article with a PABX model because of its size and complexity. We think that very few programmers will be able to implement it safely and quickly. The generation process of that model requires less than three minutes.

PABX is traditionally intended for telephone applications. We study a third-generation analog multi-service PABX characterized by circuit switching and by different services such as speech, image and data transmission. Relying on existing standards in communications, this PABX represents a first step toward an Integrated Service Digital Network.

Our study can be considered as a generalization of those proposed for both first and second generation automatic exchanges [Jensen 81]. It allows an arbitrary number of sites and any type of transmission, while respecting the layered architecture tailored to the services provided by third generation PABXs. Moreover, we take into account the handling of all exceptions (Hanging up, call-error, routing aborted, callee’s party busy, callee’s party out...).

3.2 Services provided by a PABX

PABX establishes a communication between any interconnected telephone subscribers. PABX handles and supervises a communication until it is finished. Both establishment and release of a communication are handled at the distribution and commutation level.

- Distribution handles the set of elements situated between a telephone subscriber and its PABX (receiver, linking cables, distributor).
- Commutation consists in establishing a temporary circuit between the lines of both telephone subscribers who take part in a communication.

A set of signaling rules is defined in order to regulate communication between PABX and telephone subscribers (caller and callee). Different signals are itemized in Figure 2.

3.3 A communication in progress

We now describe the actions generated when a communication is established and terminated (Figure 3). This section deals with the general behavior of the system. More particular aspects such as exception handling will be described in §3.4.

Communication treatment involves different steps:

- Preselection A subscriber's line is constantly supplied with direct current by the PABX it depends upon. By picking up the receiver, a caller causes a current variation detected by PABX. After allocating all required resources to handle communication, PABX emits a dial tone and sends it to the caller site.
- The Routing number dialed by the caller is analyzed by PABX. In this way, the callee is identified. A routing signal is then sent to the caller; it signals an attempt to establish a communication with the callee on going. The callee's line becomes busy and a ringing signal informs him of an incoming call. A ringback tone informs the caller that his request for communication has been taken into account.
- Supervision PABX waits for two kinds of events:
  - The callee picks up the receiver thus ending transmission of the ringing signal.
  - The callee hangs up the receiver thus ending the ringing signal. The callee's and the caller's lines become free again.
- Conversation PABX ensures transmission of vocal signals between Caller and Callee.
Figure 3: Description of a communication in progress

Figure 4: The Petri net model
3.4 Exception handling

Some requirements have to be achieved in order to account for particular cases

Preselection A particular situation occurs if a subscriber picks up the receiver while PABX is sending him simultaneously a ringing signal coming from an other subscriber. The first subscriber behaves as if he was called by the PABX. Therefore, this subscriber will be amazed to take part in a communication without having heard his phone ringing.

Routing If the callee's line is busy, a call-error signal is sent back to the caller who has no other choice but to hang up. If the callee is absent, the caller, weary of hearing the ringing tone, hangs up the receiver.

Release Caller and callee hanging up the receiver are not treated equally by the PABX. In fact, the caller is the master of the connection. The callee hangs up the receiver but this event does not release his line. After a waiting time, a call-error signal is sent to the caller. The callee's line becomes free only when the caller hangs up the receiver. In this way, if the callee picks up the receiver before the call-error signal is transmitted, the conversation can be resumed.

4. THE MODEL

In order to specify the behavior of a subscriber (including all exceptions handling), we need to define its environment and consequently to construct a complete (and unfortunately complex!!) model of the service provided by a PABX. Because of the great complexity of the PABX, it would not be realistic to include a model of the realization of our PABX.

Having outlined the context of our study, we propose a submodel for the behavior of telephone subscribers. This submodel is then added to the submodel of the service provided by a PABX (or a network of PABX) for communication management.

The final model allows us to both analyze communication establishment and evaluate its performance. It will be used as a kernel (validated a priori) for the implementation step.

<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-T0</td>
<td>Receiving caller's off-hook signal and emitting dial tone. Callers line switches to the busy state.</td>
</tr>
<tr>
<td>X-T1</td>
<td>Receiving the number dialed by the caller and acknowledging it with a routing signal.</td>
</tr>
<tr>
<td>X-T2a</td>
<td>Sending a call-error signal to the caller when the callers line is already busy.</td>
</tr>
<tr>
<td>X-T2b</td>
<td>Sending a ringback tone to the caller when the callee's line is free.</td>
</tr>
<tr>
<td>X-T3a</td>
<td>Supplying d.c. current to the call's line. This subscriber may pick up the receiver again, either as a caller or as a receiver.</td>
</tr>
<tr>
<td>X-T3b</td>
<td>Connecting the callee with the caller after the callee has picked up the receiver.</td>
</tr>
<tr>
<td>X-T3c</td>
<td>Detecting the on-hook signal (hanging up) of the callee and supplying d.c. current to his line. The line remains busy since the callee has not hung up yet.</td>
</tr>
<tr>
<td>X-T4a</td>
<td>Detecting the on-hook signal of the caller and supplying d.c. current to his line. The line becomes free.</td>
</tr>
<tr>
<td>X-T4b</td>
<td>Detecting the on-hook signal of the caller and supplying d.c. current to his line. The line becomes free.</td>
</tr>
<tr>
<td>X-T5a</td>
<td>Connecting the callee with the caller. The callee answers the telephone and the caller has not hung up yet.</td>
</tr>
<tr>
<td>X-T5b</td>
<td>Sending a call-error signal to the callee, the callee has hung up.</td>
</tr>
<tr>
<td>X-T5c</td>
<td>Supplying d.c. current to the callee's line which switches to the free state once the callee has hung up.</td>
</tr>
<tr>
<td>X-T6a</td>
<td>Switching the callee's line to the free state once the callee has hung up.</td>
</tr>
<tr>
<td>X-T6b</td>
<td>Detecting the on-hook signal of the callee and supplying d.c. current to his line. The line becomes free.</td>
</tr>
<tr>
<td>X-T7a</td>
<td>Stopping the transmission of the ring signal to the callee when the caller has hung up.</td>
</tr>
<tr>
<td>X-T7b</td>
<td>Sending a call-error signal to the callee if he hangs up during the conversation.</td>
</tr>
<tr>
<td>X-T8</td>
<td>Releasing the callee's line when the caller hangs up.</td>
</tr>
<tr>
<td>X-T9</td>
<td>Releasing the call and the callee's line when the caller hangs up.</td>
</tr>
<tr>
<td>X-T10</td>
<td>Supplying d.c. current to the callee's line D.c. which becomes free when the callee hangs up.</td>
</tr>
</tbody>
</table>

**PABX**

<table>
<thead>
<tr>
<th>SUBSCRIBER</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-T0a</td>
<td>Ringing telephone (callee's party only).</td>
</tr>
<tr>
<td>S-T0b</td>
<td>Picking up the receiver (callers party only). This action takes place only if the callee is not already being called. This implies that the line is idle.</td>
</tr>
<tr>
<td>S-T0c</td>
<td>Answering the telephone. This action interrupts ringing.</td>
</tr>
<tr>
<td>S-T1a</td>
<td>Receiving a call-error tone. The subscriber picks up the receiver and hears it. He has no choice but to hang up.</td>
</tr>
<tr>
<td>S-T1b</td>
<td>Hearing dial tone and dialing (callers party only).</td>
</tr>
<tr>
<td>S-T1c</td>
<td>Connection with the caller (callers party only).</td>
</tr>
<tr>
<td>S-T2a</td>
<td>Receiving the routing signal.</td>
</tr>
<tr>
<td>S-T2b</td>
<td>Receiving the call-error signal (busy).</td>
</tr>
<tr>
<td>S-T2c</td>
<td>Receiving the ringback tone.</td>
</tr>
<tr>
<td>S-T2d</td>
<td>Hanging up after a subscriber is found unavailable.</td>
</tr>
<tr>
<td>S-T3a</td>
<td>Establishing the connection with the callee (callers party only).</td>
</tr>
<tr>
<td>S-T3b</td>
<td>Hanging up the receiver after the conversation (this action can be performed either by the caller or the callee).</td>
</tr>
<tr>
<td>S-T4</td>
<td>Hanging up after a call-error (hanging up after the other subscriber has already hung up).</td>
</tr>
</tbody>
</table>

*Figure 3: Decomposition of operations in the PABX model.*
Our conceptual model (Figure 4) is broken down into two submodels:

- The telephone subscriber (either caller or callee)
- The service provided by a PABX

Operations of our model are detailed in Figure 5. Modeling has been designed in order to attribute Subscriber processes to be either caller or callee.

In order to achieve easy modeling, validation and evaluation of obtained models, we have designed a software environment: AMI [Bernard 88]. AMI provides application management and communication with MACAO, a graphic software intended for graph conception (Petri Net for example).

The complete prototyping operation will then be:

- designing the model using MACAO
- using other tools provided by AMI workshop (reduction, semi-flow calculation...) for qualitative validation
- generating the prototype using TAGADA and using it for real simulation.

If the code generator needs some theoretical results which are not already processed (flow calculation for example), it can dialog with others applications supported by AMI.

5. PROTOTYPING

5.1 Introduction

The objectives of the PN-TAGADA project (Petri Net-Translation, Analysis and Generation of ADA code) are to design and implement tools performing code generation in a distributed architecture context. We thus need to design and implement a system for semi-automatically producing a reliable and efficient distributed code from a validated Petri net model.

Imperative programming is chosen most of the time as: ADA [Colom 86], CHILL, OCCAM [Steinmetz 86] or others [Nelson 83]. For particularly tailored net classes, object-oriented programming paradigms are being explored for Petri net model prototype implementation [Bruno 86]. Among high-level concurrent languages, ADA has been chosen because it provides advanced tasking and structuring mechanisms. We have also studied generation of an OCCAM code for a multi-transputer environment [Braant 89].

5.2 Methodology of implementation

The basic principle of this implementation is to split the Petri net model into a set of cooperating sequential tasks, possibly synchronized by means of rendez-vous and/or messages.

We define three steps (see Figure 1) in order to progressively integrate constraints (physical system, architecture, decision for implementation by the designer, programming language...) and objectives (quality and performance criteria).

Each step produces a set of objects associating Petri net elements to each of these constraints.

5.2.1 Translation and analysis

The first step is concerned with model semantics and internal synchronizations. It performs the Analysis of the validated net model and the Translation of each component (place or transition) into software objects needed for the next step.

The control structure analysis of the net based models permits to split a set of cooperating sequential processes. Each one is implemented as a sequential task. We use the linear invariants obtained from the validation phase. An invariant is a conservative flow of the model which may be interpreted as process sequential states. These invariants allow us to define the process distribution over the model, and then an optimal process number. Problems concerning conflicts need careful consideration (deadlock, management of shared resources...).

5.2.2 Object location

According to initial marking and colors associated with the objects produced by the previous step, the second step defines the software object location for a given distributed architecture. It manages hardware and software resource allocation. Environment related constraints are integrated. The objects produced are specific according to the system architecture (multi-processor, shared memory,...).

5.2.3 Code generation

The third step generates the code. Semantic rules of Colored Petri net behavior have to be expressed using the programming language. Moreover, new constraints may be added by the language itself. Software objects delivered by the location step are defined in the next paragraph. Principles used for code generation are described under §5.4.

5.3 Software objects for code generation

Let us now determine the four types of software objects involved in the code generation:

- **Process**: (Issued from flows in the model); characterizes a possible concurrent program unit
- **State Process**: (Issued from places of the model); characterizes a possible process state.
  - **Simple**: only one action is possible after this state
  - **Alternative**: many actions are possible after this state
- **Resources**: (Issued from places of the model which cannot be considered as state_process); define a resource (data, message, record...)
  - **Private**: exclusive access for one process
  - **Shared**: concurrent access by several processes
- **Actions**: (Derived from transitions of the model); express treatments to be performed
  - **Process**: models an action to be performed by a single process.
  - **Synchronization**: models an action which must be performed by many cooperating processes in a synchronous way (rendez-vous).

An action may be either guarded (a condition on resource may prevent the firing of the transition) or simple.

5.4 Code generation principles

In order to be efficient, code generation must integrate target language constraints and take into account the specificity of each object type.

The version v0.2 of our code generator is written in ADA and makes intensive use of packages to facilitate generic production of target code. This version is able to generate ADA code from...
Binary P/T Nets. We are developing a new version which takes into account Colored Petri Nets. Its present state of development allows to generate code from the colored model presented in Appendix I.

Let us just briefly introduce some principles used for ADA code generation.

5.4.1 Generation of object type : State_Process

Our code generator identifies each process with an associated set of State_processes. The number of processes depends upon the cardinality of initial marking. Each process location is determined by the location step (see §5.2.2). A homogenous status word characterizes each process state. It may contain more than one component (it is thus derived from a place which contains compound tokens). Its value is tested during the Precondition evaluation of an action and modified if necessary by its postcondition (see §5.4.3).

According to the nature of a State_Process object, many cases may occur:
- If the state is alternative, we associate a variable which contains the status word associated to the process. We have to generate a code which chooses an action and can perform back-tracking (a transition is not firable).
- If the state is simple, no particular information appears in the generated code.

5.4.2 Generation of object type : Resource

A variable is associated to each Resource type object. Its initial value depends upon initial marking. The number and location of these variables are set by the location step (see §5.2.2).

Management of a shared resource needs the implementation of mechanisms derived from distributed systems. Naturally, the choice of the algorithm depends on constraints of the system architecture.

5.4.3 Generation of object type : Action

An action is split into three operations:
- The precondition requires all processes and resources needed to execute the action.
- The treatment associated with the action processes the resources. The body of the treatment is an external procedure defined by the model conceptor.
- The postcondition releases all processes and resources used during treatment.

When the type of the action is synchronization, we associate an ADA task (server) for Rendez-vous management [Cousin 88a].

5.4.4 Generation of object type : Process

ADA implementation intensively uses tasking. We must thus divide the Petri net into a set of processes. Each process in the Petri net is implemented as an ADA task.

Knowledge of linear invariants (like flows) is very important for clear determination of each process. At present, process determination is only semi-automatic.

On the PABX model, we use two interesting flows:
- \( S_{P0}, S_{P1}, S_{P2}, S_{P3}, S_{P4}, v \times e \) considered domain (set of connected subscribers) is related to a subscriber. It allows us to determine a task which models subscriber behaviours.
- \( X_{P0}, X_{P1}, X_{P2}, X_{P3}, X_{P4}, X_{P5}, X_{P6}, X_{P7}, X_{P8}, X_{P9}, X_{P10} \) \( v \times e \) considered domain (set of connected PABX) is related to a PABX. It allows to determine a task which models PABX behaviours.

Those flows are conservative (marking of their places is constant all over the execution). This is their only required characteristics.

Based on that determination, we can define two classes of tasks in this model. The first one is related to each subscriber, the second to each PABX available (having N PABX means that N conversations can occur at the same time).

Other flow combinations, using message places, are not considered as interesting in this model. Interesting flows are called process-flows.

6. APPLICATION TO OUR PHONE CONVERSATION MODEL

The model introduced in section 4 does not integrate all types of defined objects. In [Cousin 88a], we have presented a model especially built in order to cover technical problems and to include all object types.

For each type, we have defined a set of objects used for code generation. They are identified by the name of the element (place or transition) from which they are derived. These sets have been constructed (semi-automatically for this application) by using results of the qualitative analysis and according to initial marking.

This example contains three subscribers \( (U_1, U_2, U_3) \) which are managed by two available PABX processes (PABX1 and PABX2). In that way, we generate three "clones" of a Subscriber process and two of the PABX process. In code generation, clones have identical codes but are identified by their names. Local resources are then considered as global ones because of possible concurrent access by all clone processes.

In the proposed version, object location (second phase of the methodology) involves only one station. All concurrent processes run on the same machine (PC-AT for this example). We plan to apply this example to a real distributed system (SUN workstations connected with Ethernet). Therefore, each process will run on its own machine; shared resource and communication management will be implemented by using UNIX sockets.

In appendix I, we present an abstract from a listing of subscriber task, An example of execution of the generated code (appendix II) shows a sequence of actions where the PABX manages three subscribers.

These documents have been automatically produced by our code generator (we have just modified the presentation of the listing with a text editor in order to avoid an excessive number of pages).

Generated ADA code corresponds to the first level of results that we are now able to produce from a Colored Petri Net model. In particular, in this version the PAC place has derived as many objects as the cardinality of the set of transmitted signals (places PAC0 to PAC10).
7. CONCLUSION

PN_TAGADA project objectives are to study specific problems caused by code generation from Colored Petri Net models and to design an automatic (or semi-automatic) code generator.

Our approach permitted us to realize a complete automatic prototype translation of a formal model implementing a real problem. Although generated code cannot be as optimized as that of a human programmer, it allows to:

- give a quantitative validation of the model. Conceptors can study its behaviour through a simulation in a real environment. Low cost of such a prototype allows a backtrack to the formal model.
- Check all important specifications. Choice impact on model behaviours can be easily studied.

Our work has just begun (in 1988) but we have already obtained interesting results on a real and complex application. Naturally, many theoretical problems, close to classic distributed programming problems, still need to be solved (net decomposition techniques into subnets of sequential tasks, coupled conflict problems, ...).

We realized a first version running on a PC AT (the generated code we presented in APPENDIX I was obtained from it). This version was adapted and became part of the AMI project.

Color management of our previous version is still very poor. We plan to extend it to all kinds of colored Petri nets. Some code optimizations (use of ADA types, increased use of genericity...) are under way.

Our aim is to achieve high performance modeling and verification procedures. The final version of our code generator will complete the AMI environment [Bernard 88] by allowing easy prototyping processes to be generated from a formal modelization.

REFERENCES


[Cousin 88b] B.Cousin, J.M. Couvreur, C. Dutheillet, P.Estrailliier Validation of a protocol managing a multi-token architecture 9th European Workshop on Application and theory of Petri Nets, Venice-Italy (June 88)


APPENDIX I

Abstract from listing of a task SUBSCRIBER

-- Task specification

task SUBSCRIBER is
  entry marq init (init in natural);
end SUBSCRIBER;

-- Task body

task body SUBSCRIBER is
  -- declaration of possible choices at P0, P1, P2 and P3
  chx_S_P0: tab_expr(1..3) := (1 => SUBSCRIBER_S_T0a,
    2 => SUBSCRIBER_S_T0b,
    3 => SUBSCRIBER_S_T0c);
  chx_S_P1: tab_expr(1..3) := (1 => SUBSCRIBER_S_T1a,
    2 => SUBSCRIBER_S_T1b,
    3 => SUBSCRIBER_S_T1c);
  chx_S_P2: tab_expr(1..5) := (1 => SUBSCRIBER_S_T2a,
    2 => SUBSCRIBER_S_T2b,
    3 => SUBSCRIBER_S_T2c,
    4 => SUBSCRIBER_S_T2d,
    5 => SUBSCRIBER_S_T2e);
  chx_S_P3: tab_expr(1..2) := (1 => SUBSCRIBER_S_T3a,
    2 => SUBSCRIBER_S_T3b);

-- Others variables

state : natural;
continue : boolean := true;
precondition : boolean;
response : boolean;
bidon : boolean;
cpt : natural;
g_lst_loc : pr rg pt el;
p_lst_loc : pr dom pt el;
begin
-- Dynamic initialisation of PN process
accept mxq-init (init :in natural) do
  state := init;
end mxq-init;
-- Running
while continue loop
  case state is
    -- S_T4 (Type : Action/Process/Guarded)
    when SUBSCRIBER_S_T4 =>
      precondition := true;
      -- precondition evaluation
      if precondition then
        g_list := NULL;
        gc_list := NULL;
        if gconsume (PAC_7, U1) then
          g_list := ajouter (PAC_7, g_list);
          gc_list := ajouter (U1, gc_list);
        else
          precondition := false;
        end if;
      else
        precondition evaluation failed, we must restore previous resources
        state := SUBSCRIBER_S_T4;
        while glst-loc /= NULL loop
          if not precondition then
            gproduce (content(glst-loc), content(gcst-loc));
          end if;
          glst-loc := suppress (glst-loc);
          gcst-loc := suppress (gcst-loc);
        end loop;
        if precondition then
          -- Possible activation and postcondition
          proc_S_T4:
            gproduce (PAC_8, U1);
            state := SUBSCRIBER_S_P0;
        else
          end if;
        end if;
        delay 0.0; -- pour Euratec
      end if;
      -- Others actions
    end case;
  end loop;
end SUBSCRIBER;

APPENDIX II

An execution of the compiled ADA program

Scenario: Sub#1 call Sub#2 via PABX1. during the conversation, Sub#3 tries to call Sub#2 via PABX2 and get a call-error signal.

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>ACTION</th>
<th>Sub#1</th>
<th>Sub#2</th>
<th>Sub#3</th>
<th>PABX1</th>
<th>PABX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscriber #1 : Picking up telephone.</td>
<td>S_T6b</td>
<td>X_T0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX1 : Receiving callers off-hook, emitting dial tone.</td>
<td>S_T1b</td>
<td>X_T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #1 : Hearing dial tone and dialing.</td>
<td>S_T1a</td>
<td>S_T2a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX1 : Receiving the number dialled by the caller.</td>
<td>S_T2b</td>
<td>X_T2b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX1 : Receiving the routing signal.</td>
<td>S_T2a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #1 : Receiving ringback tone.</td>
<td>S_T2c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX1 : Supplying d.c. current to the callee’s line.</td>
<td>S_T2e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #2 : Ringing telephone.</td>
<td>S_T3a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #3 : Picking up telephone.</td>
<td>S_T3b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #2 : Receiving ringback tone.</td>
<td>S_T3c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #2 : Answering the telephone.</td>
<td>S_T3d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #2 : Ringing telephone.</td>
<td>S_T3e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #1 : Receiving ringback tone.</td>
<td>S_T3f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX2 : Receiving callers off-hook, emitting dial tone.</td>
<td>S_T5b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #3 : Hearing dial tone and dialing.</td>
<td>S_T5c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #2 : Connection with the caller.</td>
<td>S_T5d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #1 : Establishing the connection with the callee.</td>
<td>S_T5e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX2 : Sending a call-error signal to the callee.</td>
<td>S_T5f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #3 : Receiving call-error signal.</td>
<td>S_T5g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #2 : Hanging up after the conversation.</td>
<td>S_T5h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX1 : Detecting the on hook signal of the caller.</td>
<td>S_T5i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #3 : Hanging up after a call-error.</td>
<td>S_T5j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX2 : Reloading the callee’s line.</td>
<td>S_T5k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX1 : Sending a call-error signal to the caller.</td>
<td>S_T5l</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #1 : Processing an hang up.</td>
<td>S_T5m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriber #3 : Hanging up after a call-error.</td>
<td>S_T5n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PABX1 : Reloading the callee and the caller’s line d.c.</td>
<td>S_T5o</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

460