〔公　開〕

ＴＲ－Ｃ－００７４

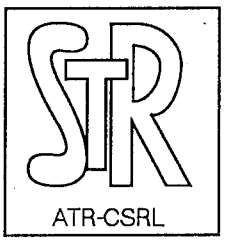
# State Transition Rule (STR) Description

平川　豊
Yutaka HIRAKAWA

１９９２．１．１４

ＡＴＲ通信システム研究所

# State Transition Rule (STR) Description




STR-1.0 January 1992

Yutaka Hirakawa

## Contents

For inquiries concerning these specifications, contact:

Yutaka Hirakawa
ATR Communication Systems Laboratories
Sanpeidani, Inuidani, Seika-cho, Soraku-gun, Kyoto-fu 619-02
Tel: 07749-5-1236 Fax: 07749-5-1208

Version History:

STR-0.0: December 2, 1991 (Hirakawa)

STR-1.0: December 4, 1991 (The following items were added.)

- "m-" description items added
- Format of internal communication signal (internal events) changed

STR-1.0: December 13, 1991 (Hirakawa)

- BNF revision

STR-1.0: December 19, 1991 (Hirakawa)

- Documentation

# 1. Introduction

This document explains a method of describing state transition rules (STR) developed for telecommunication services operations description. STR is a language that has been under development at the ATR Communication Systems Research Laboratories since the spring of 1989. It has been used primarily for research on telecommunication software.

STR has four main aims:

(1) When operations as viewed from outside the system are the same, regardless of the internal structure, it is desirable to first write the descriptions that are independent of internal structure. (It is common for exchanges to have various architectures, but to have completely identical function as seen from the outside.)

(2) At the stage of description from an external point of view, including operations at the time of service composition, it is desirable to define operations formally.

(3) At the stage of operations described from an external point of view, it is desirable to determine the logical conflicts of each service, and support operations design according to the circumstances in which the services are combined.

(4) After specifying the operations which are independent of internal structure, it is desirable to describe the details of operations which do depend on internal structure, with the aid of a computer.

In STR, the telecommunication system in question is regarded in the following way so as to take the internal structure of the network as a black box. "Within a telecommunications system, there are an unspecified number of terminal control processes. These terminal control processes all have identical functions. There are no processes other than terminal processes."

The STR description method specifies the range of influence for when one event occurs, and all of the state transitions among terminals that are included in that range are described.

# 2. Overview of State Transition Rules

State transition rules are written as set of declarations and rules. Each rule consists of three parts: "current state description", "event description", and "next state description". The following are examples of state transition rules.

```
idle(A)          offhook(A):    dial-tone(A).

dial-tone(A),idle(B)  dial(A,B):      ringback(A,B),ringing(B,A).
```

The first of the above rules specifies that if the receiver is lifted off hook in the idle state, the state changes to the receive dial tone state. The second rule specifies that if a terminal in the dial tone receive state dials to a terminal in the idle state, the dialing terminal goes into the receive ringback tone state and the dialed terminal goes into the receive ringing tone state. At this time, the identifier of the other terminal is stored as the second argument of the state description element of each terminal.

This state transition rule defines the operations for a system that comprises an arbitrary number of processes. If a dial event occurs in the system, and if, taking A as the process which generated the event and B as the process of the dialed destination, the current state description of the second rule is satisfied as a condition, then operations are performed according to this second rule.

Here, we call the elements from which the state descriptions are composed, "state primitives". Intuitively, the state primitives represent the resources and memory used by a terminal in that state. Although memory is not easily recognized as a state, as in the state in which a call is in progress and the call transfer service is set, it is an indispensable element for proper specification of state transitions from outside the system.

The system initially consists of processes which are all in the idle state. When an event occurs, the appropriate rule is applied, and the status of each process is rewritten.

## 3. Description Elements

### 3-1 State primitives

In STR description, the state of each terminal process is described as a set of state primitives. Examples of state primitives are given below.

POTS:

idle(A), dial-tone(A), busy(A), busy-dial(A), ringback(A,B),
r-path(A,B), ringing(A,B), path(A.B)

Call-waiting Service:

cw-ringing(A,B), m-cw(A), cw(A), path-passive(A,B)

Call-forwarding Service:

m-cfv(A,B), confirmation(A), pingring(A,B)

Three-way Call Service:

m-3wc(A), 3wc1(A,B), 3wc2(A,B)

The meaning of these state primitives is explained below.

r-path(A,B): Reserves the communication path from A to B

cw-ringing(A,B): Indicates the notification of terminal A by a tone that there is an incoming call from terminal B

pingring(A,B): Indicates that terminal B is set for call-forwarding, and that a tone is sounded to notify that the call from B is forwarded if the terminal is idle

path-passive(A,B): Indicates that the path to terminal B is on hold

confirmation(A): Tone that confirms that the call forwarding service has been

subscribed

m-3wc(A): Means that the three-way call mode is in effect

3wc1(A,B): Indicates that a three-way call is in progress, and that B was the first party called

3wc2(A,B): Indicates that a three-way call is in progress, and that B is the third party called

The states of all terminals are described by these state primitives. An example is given below.

From the call state (path(A,B)), a flash hook will change the state to the receive dial tone state (path-passive(A,B),dial-tone(A),m-3wc(A)) for the purpose of calling the third party. If the dialed party answers, this terminal goes into the (path-passive(A,B),path(A,C),m-3wc(A)) state, and a flash hook will complete the three-way conversation (m-3wc(A),3wc1(A,B),3wc2A,C),path(A,B),path(A,C)).

We will explain a little about the relationship between state primitives and later processing (design refinement stage). Although it is not a requirement for STR description, it is useful to know something about the use of tools for more detailed description following STR description.

(1) State primitives

Ordinary state primitives model the equipment (resources) used by each state. This information is important in design refinement stage. That is, when moving from the state description path(A,B) to path-passive(A,B), for example, the label task suspend(B) can be inserted as detailed description.

(2) "m-" descriptions

Authorizing call waiting "m-cw(A)" is an example. This state primitive can be thought of as a state variable or as control register information. When converted to SDL, this state transition description element is not recognized as a state. In other words, when an incoming call request event is received in the call-in-progress state, a test function determines if there is authority for call waiting, and operations then decide the next state according to that result.

3-2 Types of Declarations

(1) State primitive declarations

It is necessary to declare the state primitives in advance. An example is given below.

Primitives:
idle(A),ringback(A,B),
ringing(A,B)

(2) Event declarations

It is necessary to declare the events used in STR description in advance.

Events:
dial(A,B),flash(A)

State primitive and event declarations are for description error checking, and are not an essential part of STR description itself.

(3) Declaration of state primitives that have time restrictions

If in the dial-tone(A) state, for example, the user simply waits without dialing, he receives a busy tone. To describe this type of operation, state primitives that have time restrictions attached are introduced.

However, the concept of attaching time restrictions to state primitives (or a series of state primitives) means that there is no guarantee that description of all operations is possible. What is understood at the present time is only the empirical fact that this concept alone was sufficient for the description of a number of telephone services. (This is also considered to depend on the level of detailed description currently available. Put another way, there are probably times when this concept alone is insufficient, depending on the structure and constraints of the system in question.)

A description example is given below.

Limited-Time-Primitive:

| | |
|---|---|
| busy(A) | 30sec |
| busy(A),m-3wc(A),3wc2(A,B) | 30sec |

(4) Declaration of internal event delivery as default

In on-hook operations, for a broad-area description, the number of rules increases exponentially with the number of the states of the connected terminals, and description becomes very difficult. For that reason, definition and use of an internal communication signal was made possible. When this is done, on-hook operations can be handled in a relatively fixed form, and if the signal delivery is made the default situation, description becomes easy. Thus, the following declarations were introduced.

Internal-Signal-Delivery:
onhook: --> sig-onhook

By means of this description, when there is no particular specification, a sig-onhook signal is assumed to be sent to all other recognized terminals when any terminal is in the on-hook state. Because of this, the description of on-hook operations is simplified as shown below.

| | | |
|---|---|---|
| Rule 1) | path(A,B) onhook(A): | idle(A) |
| Rule 2) | path(A,B) sig-onhook(B,A): | busy(A) |

Here, sig-onhook(B,A) means that A receives the internal communication signal sig-onhook from B.

Also, by introduction of an internal signal, description of the operation of sending an internal signal is also introduced. For example we have the following type of description.

Rule 3) 3wc(A),3wc1(A,B),3wc2(A,C),path(A,B),path(A,C)
flash(A): path(A,B),>sig-onhook(A,C)

The above rule describes the operation in which a flash during a three-way conversation cuts off the third party and returns to a two-party call.

When it is desired to control the destination of the internal signal sent during the on-hook state, the following declaration is used.

Signal-Delivery:
range(sig-onhook,path-passive(A,B))={}

By this declaration, the sig-onhook signal will not be sent to a terminal from which there is an on-hold call even if the called terminal is on-hook.

(5) Internal event declarations

If the above described default signal delivery is not used but the exchange of internal signals is used as an interface, those signals are declared here.

Internal-Events:
invoke(A,B), request(A,B), ack(A,B),nack(A,B)

Using the declared internal events, sending or receiving operations are specified. For example, if ">invoke(A,B)" is written in the next state description, it indicates that the invoke signal is sent from process A to process B. Also, if "invoke(A,B):" is written in the event description, it means that the invoke signal sent from process A is received by process B.

(6) Definition of macro-descriptions

Description macros are introduced for sets of state descriptions which normally appear only in groups and for sets of descriptions which are conceptually clear-cut as a group. The following are examples.

Macro-Primitives:
Calling(A,B) = {ringback(A,B),r-path(A,B),ringing(B,A)}

Rule 1) dial-tone(A),idle(B) dial(A,B): Calling(A,B).

(7) Inhibit-sets declarations

The following is an example of an inhibit-sets description.

Inhibit-Primitive-Sets:
{cw-ringing(A,B),cw-ringing(A,C)}(busy)

This declaration inhibits the inclusion of the above description set in the state description of one process. For example, consider the following description.

Rule 1) dial-tone(A),not[idle(b)],cond:m-cw(B),path(B,C)
dial(A,B): ringback(A,B),path(B,C),cw-ringing(B,A).

If rule 1 were applied alone, its condition would be satisfied if another call arrived

during a call-waiting state for terminal A, and thus any number of waiting calls could result. The purpose of the inhibit-set description is to prevent this.

If rule 1 is applied when a terminal that is in the call-waiting state is dialed by another terminal, a terminal possessing the above inhibited set might appear. In that case, this rule is not applied and the process generating the event is connected to a busy signal. The next state of the event generating process consists of the following procedure.

[next state] (busy): First, from the current state, the state primitive that would be removed if the rule is applied is removed. Then "busy(process-id)" is added.

If "(busy)" is not clearly specified by the inhibit state, the next state of the problem event generating process is created by the following procedure.

No [next state] (busy): From the current state, the state primitive that would be removed if the rule is applied is removed.

Checking for this inhibited state is done after the normal rule application decision. Thus, the applied rule is not changed because of conflict with the inhibit state set.

## 3-3 Event Descriptions

With STR, events from outside the system can be described freely. However, it is necessary to declare the events to be used in advance.

It is possible to describe events on various levels of abstraction. In STR, the following four types of events can be used.

1) User events

These are user-generated events, such as dial(A,B):, flash(A):, etc.

2) Time-over events

These are used when describing operations which have time restrictions. State primitives follow after "timeover(", as in "timeover(dial-tone(A)):", and so on. The timers are recognized by this state primitive. (Refer to the section on description of operations that have time restrictions.)

3) Pseudo-events

For CCBS operations, the following type of description is used.

Rule 1) m-CCBS(A,B),confirmation(A),m-CCBSed(B,A)
[idle(B)]: ringback(A,B),ringing(B,A).

This specifies the operation that when A dials to B and the party is busy, CCBS is requested and if the caller stands by with the receiver off hook, the call is placed as soon as the other party's terminal goes into the idle state. In this case, the state transition must be carried out on the opportunity of the other party's terminal going into the idle state. The pseudo-event is introduced as a method of describing this kind of operation.

4) Internal events

Internal communication signals specified by the :Internal-Signal-Delivery:" declaration and signals specified by the "Internal-Events:" declaration can be described as internal event operations. The following is an example.

Rule 1) path(A,B) sig-onhook(B,A): busy(A).

Internal event receiving operations must not be single terminal operations. That is, the first argument of the state primitive used in the current state description must be consistent with the second argument of the internal event.

3-4 Other Descriptions

(1) "cond" descriptions

The following rule 1 is the same as rule 2.

Rule 1) dial-tone(A),cond:path(B,C),cond:m-cw(B)
dial(A,B): CW-ringing(A,B).
Rule 2) dial-tone(A),path(B,C),m-cw(B)
dial(A,B): CW-ringing(A,B),path(B,C),m-cw(B).

In this way, at state transition, although reference is made, when an object description element is not replaced, it can be written in the current state description and again in the next state description as in rule 2, or it can be written in an abbreviated fashion as in rule 1. The "cond" description can only be used in the current state description.

(2) "not" descriptions

These are used in a manner such as "not[idle(B)]". They can only be used in current state descriptions. In the same way as for "cond" descriptions, they are descriptions for testing, as a rule application condition, whether the state primitives specified in the state description of the process are included.

The "not" description is convenient in terms of description, but there are drawbacks to using it. For example, consider the following two rules.

Rule 1) dial-tone(A),not[idle(B)] dial(A,B): busy(A).
Rule 2) dial-tone(A),path(B,C),m-cw(B)
dial(A,B): ringback(A,B),cw-ringing(B,A),path(B,C),m-cw(B).

When these two rules co-exist, and rule 2 is applicable, then rule 1 is also necessarily applicable. However, this fact is difficult to judge mechanically. Accordingly, it is a major problem when considering interpreted systems, verification, and so on. (In mechanical judgement, the possibility of path(B,C) and idle(B) co-existing cannot be excluded.)

Thus, at this time, so as to clearly understand mutually exclusive items, when investigating the inclusive relationship between two rules, we introduce the constraint that if "not[idle(A)]" exists in one rule, then either "idle(A)" or "not[idle(A)]" must necessarily exist in the other.

(3) "or" descriptions

Rule 1) path(A,B) onhook(A): idle(A).
Rule 2) busy(A) onhook(A): idle(A).

The above two rules can be written together in the following way.

Rule 3) (path(A,B) | busy(A)) onhook(A): idle(A).

## 4. Operations Specified by STR Description

### 4-1 Basic rule application

Consider the following situation, where p, q, r, and s are terminal identifiers.

Terminal p) dial-tone(p)
Terminal q) dial-tone(q)
Terminal r) idle(r)
Terminal s) idle(s)

Consider the following rule.

Rule 1) dial-tone(A),idle(B) dial(A,B): ringback(A,B),ringing(B,A).

If, under the above circumstances, terminal p dials to terminal r, rule 1 applies with A = p and B = r, and the state changes to terminal p calling terminal r. Further, if terminal r goes on to dial terminal s, the same rule applies with A = q and B = s and the state changes to terminal q calling terminal s.

In this way, variables used in rules restrict the opportunities for events to occur, and when the descriptions specified in the current state description are completely satisfied, that rule is applied. However, the different variables used in the rules must correspond to their various respective identifiers. For example, consider the following situation.

Terminal p) path(p,q),m-cfv(p,q)
Terminal q) path(q,r)
Terminal r) dial-tone(r)

Consider the following rule.

Rule 2) dial-tone(A),cond:m-cfv(B,C),cond:path(B,D),not[idle(C)] dial(A,B): busy(A).

The above situation can be considered the case in which terminal r has dialed to terminal p. Looking at the possibility of the rule applying, we see that the correspondence A = r, B = p, C = q, and D = q is possible. However, here, the different variables C and D correspond to the same identifier, q, therefore rule 2 cannot be applied.

### 4-2 Rule application priorities

First, as basic service operations, the following rule is described.

Rule 1) dial-tone(A),idle(B) dial(A,B): ringback(A,B),ringing(B,A).

Next, if call transfer is considered, it is necessary to transfer the call even if the called terminal is idle. This is described in STR as follows.

Rule 2) dial-tone(A),idle(B),cond:m-cfv(B,C),idle(C)
dial(A,B): ringback(A,C),pingring(B,A),ringing(C,A).

Here, the following situation is considered.

Terminal p) dial-tone(p)
Terminal q) idle(q),m-cfv(q,r)
Terminal r) idle(r)

In the above situation, application is possible for both rule 1 and rule 2, and they specify respectively different operations. However, after basic operations are specified, as by rule 1, and as design proceeds, attention is often given to the specification of exceptional operations such as by rule 2. By introducing rule 2, we hope to avoid frequent modification of existing rules.

To make this possible, the following priority order among rules is introduced in STR.

[Rule priorities]

When, as with the above two rules, the application conditions of rule 1 are entirely included by the application conditions of rule 2, rule 2 is applied with priority in situations where either rule can be applied.

By introducing this kind of priority among rules, existing rules need no modification at all when it is necessary to specify exceptional operations one-by-one according to design development, and operation can be specified by simply adding rules.

## 5. Constraints on Descriptions

The constraints which must be observed in STR description are described below.

(Constraint 1)

Process identifiers used in the next state description of STR rules cannot be the same as those used in the current state description or event description of the same rule.

(Constraint 2)

In operation description for when a user event occurs, the state description of the event generating process must be included in the current state description.

(Constraint 3)

In operation description for when a user event occurs, if the event descriptions and current state descriptions are represented as a graph, with the event-generating process as a root, it is necessary that every point be reachable.

(Constraint 4)

In operation description for when an internal event occurs, the specification should be for a single process.

STR descriptions are rules replacing sets of terminal state descriptions. Thus, descriptions that cannot be interpreted by semantic substitution are not acceptable. Also, it is necessary to confirm the existence of the states specified by STR rules through complete intercommunication. For that purpose, there must be a means of directly or indirectly accessing all of the processes specified from the event generating process.

## 6. Related Tools

The following are tools which have been developed as of this time (1991).

(1) Interpreter system

Operating requirements:
LISP machine

Description:
Reads STR description (old version assumed; notation is somewhat different from present version) and displays operations situation by means of animation on the screen according to events generated using the mouse.

Benefit:
Allows rapid prototyping during STR description development.

Comments:
Completion by the end of 1992 of a new system which will run on the SUN workstation is planned.

(2) SDL conversion system (for demonstration)

Operating requirements:
LISP machine

Description:
Reads STR description and outputs the SDL description for control of each terminal.

Use:
Demonstration

Comments:
Completion by the end of 1992 of a new system which will run on the SUN workstation is planned.

(3) SDL detailing system

Operating requirements:
LISP machine

Description:
Reads rough SDL and detailed description knowledge files and outputs detailed SDL.

Comments:
Currently (December 1991) is incorporated in the SDL conversion system described above. Completion of conversion to an independent system which will run on the SUN workstation by the end of 1991 is planned.

(4) SDL-to-CLS conversion system

Operating requirements:
LISP machine

Description:
Converts detailed SDL to CSL so that it can run on a network simulator.

(5) Generated software operating environment

Operating requirements:
LISP machine

Description:
Network simulators vary only in the version of the ?system of expression?. Creating new version of the ?system of expression? makes it possible for the network simulator to run, overwriting the CSL description for each machine with automatically generated description. This operation makes the running of automatically generated CSL truly possible.

(6) Validation support system

Operating requirements:
LISP machine

Description:
Reads old-version STR description and 1) acquires the state sets which are available to the terminal, 2) generates a state event table, and 3) detects groups of rules which have some possibility of conflict and displays them together with the conflicting situation as an animation.

Comments:
Completion of porting to the SUN workstation by the end of 1992 is planned.

(7) Other systems under development

Visual STR rule input system (linking with the interpretation system is planned)

High-speed validation system

Attainability decision system

## 7. List of Related English Articles

[1] Hirakawa, Harada, Takenaka,
"A Description Method for Advanced Telecommunication Services,"
IJECE Technical Report, SSE89-87, pp.37-42, October 1989

[2] Shibata, Hirakawa, Takenaka,
"Reachability Analysis for a Behavior Description Independent of the Number of Processes," IJECE Technical Repot, JWCC'90, pp. 85-93, July 1990

[3]Hirakawa, Harada, Takenaka
"Behavior Description for a System which consists of an Infinite Number of Processes," Proc. 1990 Bilkent International Conference on New Trends in Communications, Control, and Signal Processing, Ankara, Turkey, pp. 85 - 93, July 1990.

[4]Hirakawa, Takenaka, "Telecommunication Service Description Using State Transitrion Rules," Int. Workshop on Software Specification and Design, pp. 140 - 147, Oct., 1991

[5]Harada, Hirakawa, Takenaka, "A Design Support Method for Service Interactions", Globecom'91, 46 - 3, Dec., 1991

[6]Hirakawa, Kawata, Takenaka, "Rule Descriptions for Telecommunication Serbvices and Its Transformation into Standardized Specification Descriptions," SETSS'92, March 1992

## Appendix 1: BNF for STR Description

```
################################################
  BNF Description for STR-V1 (dated Nov. 1991)
        updated by Hirk,  4 Dec., 1991
################################################

STR-descriptions
 := {Primitive-decl}
   {Event-decl}
   {Other-decl}
   Rules
   STR-descriptions

Other-decl
   :={Time-desc | Signal-delibery | Macros | Inhibits-sets |
        Delibery-range | Internal-events} {Other-decl}
Primitive-decl
   := "Primitives:" primitive-string
primitive-string
   := name-string "(" Var {"," Var} ")" {, primitive-string}
Event-decl
   := "Events:" event-string
Internal-events
   := "Internal-Events:"  primitive-string
event-string
   := name-string "(" Var {"," Var} ")"
    | name-string "(" Var {"," Var} ")"
    | "timeover(" primitive-string ")"
    | "[" primitive-string "]"
Time-desc
    := "Limited-Time-Primitives:" time-string
time-string
   := primitive-string  number"sec" {time-string}
Signal-delivery
   := "Internal-Signal-Deliveration:"
            event-string "-->" internal-signal-name
Macros
   := "Macro-Primitives:" macro-string
macro-string
   := Capital-starting-name "(" Var {"," Var} ") = {" primitive-string "}"
    | Capital-starting-name "(" Var {"," Var} ") = {" primitive-string "}"
        macro-string
Inhibits-sets
   := "Inhibited-Primitive-Sets:" inhibit-string
inhibit-string
   := "{" primitive-string "}" { "(busy)" } | "{" primitive-string "}" { "(busy)" }
       inhibit-string
```

```
Delibery-range
   := "Delibery-Range:" delibery-string
delibery-string
   := "range(" internal-signal-name":" primitive-string")
     = {" Var-list "}" | "range(" internal-signal-name":" primitive-string") = {"
         Var-list "}" delibery-string
Var-list
   := Var | Var {"," Var-list}
Rules
   := "Rules:" rule-string
rule-string
   := rule-name")" p-string event-string":" next-string"."
   | rule-name")" p-string event-string":" next-string"." rule-string
p-string
   := name-string "(" Var {"," Var} ")" {"," p-string }
   | "not[" string "(" Var {"," Var} ")]" {"," p-string }
   | "cond:" string "(" Var {"," Var} ")" {"," p-string }
   | "cond:(" p-string ")" {"," p-string}
   | "(" p-string "|" p-string ")" {"," p-string }
next-string
   := name-string "(" Var {"," Var} ")" {, next-string}
   | ">"internal-signal-name"(" Var "," Var ")"{"," next-string} | "empty"
string          := ( number | char | "." | "-" ) { string }
char   := "a" | "b" | .. | "z" | "-"
Var    := "A" | "B" | .. | "Z"
Capital-starting-name   := ( "A" | "B" | .. | "Z" ) ( string | Capital-starting-name )
number        := ( "0" | .. | "9" ) { number }
internal-signal-name        := string
rule-name             := string
name-string  := ch-num-string
ch-num-string         := ( number | char | - ) ch-num-string

Comment-line        := "#" string
```

## Appendix 2: Examples of STR Description for Telecommunication Services

File: POTS.text

```
########################################
###    POTS Service                  ###
###  Updated by Hirk, 4. Dec, 1991   ###
########################################

Primitives:
   idle(A),dial-tone(A),busy(A),busy-dial(A,B),
   path(A,B),hangup(A),r-path(A,B),ringing(A,B),
   ringback(A,B),path-passive(A,B)
Events:
   offhook(A), onhook(A), dial(A,B)
Limited-Time-Primitives:
   dial-tone(A)                                    30sec
   busy(A)                                         30sec
   busy-dial(A,B)                                  30sec
Internal-Signal-Deliveration:
   onhook --> sig-onhook
Macro-Primitives:
   Calling(A,B)   = {ringback(A,B),r-path(A,B),ringing(B,A)}
   Talk(A,B)      = {path(A,B),path(B,A)}
   Busy(A,B)      = {(busy(A)|busy-dial(A,B))}
Inhibited-Primitive-Sets:
   {path-passive(A,B),busy-dial(A,B)} (busy)
Delivery-Range:
   range(sig-onhook: busy-dial(A,B))   = {}
   range(sig-onhook: path-passive(A,B)) = {}

Rules:
pots-1)idle(A)                  offhook(A):   dial-tone(A).
pots-2)dial-tone(A),idle(B)          dial(A,B):     Calling(A,B).
pots-3)dial-tone(A)             dial(A,A):     busy(A).
pots-4)dial-tone(A),not[idle(B)]    dial(A,B):     busy-dial(A,B).
pots-5)Calling(A,B)             offhook(B):   Talk(A,B).
pots-6)path(A,B)     onhook(A):   idle(A).
##pots-7)path(A,B),not[ringing(A,B)] sig-onhook(B): busy(A).
pots-7)path(A,B)     sig-onhook(B,A):     busy(A).
pots-8)Busy(A,B)     onhook(A):   idle(A).
pots-9)dial-tone(A)             onhook(A):   idle(A).
##pots-10)ringing(A,B)      sig-onhook(B):       idle(A).
pots-10)ringing(A,B)            sig-onhook(B,A):     idle(A).
pots-11)ringack(A,B),r-path(A,B)  onhook(A):    idle(A).
pots-12)hangup(A)               onhook(A):   idle(A).
pots-t-1)dial-tone(A)           timeover(dial-tone(A)):      busy(A).
pots-t-2)busy(A)     timeover(busy(A)):  hangup(A).
pots-t-3)busy-dial(A,B) timeover(busy-dial(A,B)): hangup(A).
```

File: CW.text

```
#########################
###  CW Service     ###
#########################

Primitives:
   cw(A),m-cw(A),cw-ringing(A,B)
Events:
   cw(A), flash(A)
Limited-Time-Primitives:
   busy(A),cw-ringing(A,B)                                    30sec
Macro-Primitives:
   CW-calling(A,B) = {ringback(A,B),cw-ringing(B,A),
                              r-path(A,B)}
Inhibited-Primitive-Sets:
   {cw-ringing(A,B),cw(A)} (busy)
   {cw-ringing(A,B),cw-ringing(A,C)} (busy)
Delivery-Range:
   range(sig-onhook: cw-ringing(A,B))   = {}

Rules:
cw-1)cond:path(A,B),dial-tone(C),cond:m-cw(A),not[idle(A)]
   dial(C,A):          CW-calling(C,A).
cw-2-1)cw-ringing(A,C),path(A,B) onhook(A): ringing(A,C).
cw-3)CW-calling(C,A),path(A,B)
   flash(A):   cw(A),Talk(A,C),path-passive(A,B).
cw-4)path(A,C),path-passive(A,B),cond:cw(A)
   flash(A):   path(A,B),path-passive(A,C).
cw-5)cond:cw(A),path(A,C),path-passive(A,B)
   onhook(A):          ringing(A,B),path-passive(A,B).
##cw-6)path-passive(A,B),cw(A),cond:path(A,C)
##     sig-onhook(B):       empty.
cw-6)path-passive(A,B),cw(A),cond:path(A,C)
   sig-onhook(B,A):                  empty.
##cw-7)path(A,C),path-passive(A,B),cw(A)
##     sig-onhook(C):       path(A,B).
cw-7)path(A,C),path-passive(A,B),cw(A)
   sig-onhook(C,A):          path(A,B).
cw-8)idle(A)          cw(A):                  idle(A),m-cw(A).
cw-9)idle(A),m-cw(A)       cw(A):                  idle(A).
##cw-10)cw-ringing(A,B)  sig-onhook(B):       empty.
w-10)cw-ringing(A,B)       sig-onhook(B,A):    empty.
cw-11)ringing(A,B),path-passive(A,B),cw(A)
   offhook(A):         path(A,B).
##cw-12)ringing(A,B),path-passive(A,B),cw(A)
##     sig-onhook(B):       idle(A).
cw-12)ringing(A,B),path-passive(A,B),cw(A)
   sig-onhook(B,A):                  idle(A).
cw-13)busy(A),cw-ringing(A,B)    onhook(A):   ringing(A,B).
```

```
cw-14)busy(A),cw-ringing(A,B)    flash(A):       path(A,B).
cw-15-mod)cw-ringing(A,B),hangup(A)          onhook(A):    ringing(A,B).
cw-t-1)busy(A),CW-calling(B,A)
   timeover(busy(A),cw-ringing(A,B)):    Talk(A,B).

File: 3wc.text
########################################
###    3WC Service                ###
########################################

Primitives:
   m-3wc(A),3wc1(A,B),3wc2(A,B)
Events:
   flash(A)
#III-Fork:
#  {3wc1(A,B),3wc2(A,C),not[m-3wc(A)]}
#  {path-passive(A,B),not[cw(A)],not[m-3wc(A)]}
Limited-Time-Primitives:
   path-passive(A,B),busy(A),m-3wc(A)                30sec
   path-passive(A,B),busy-dial(A,C),m-3wc(A)         30sec
Inhibited-Primitive-Sets:
   {m-3wc(A),m-3wc(A)}
   {path-passive(A,B),busy-dial(A,B)} (busy)
Delivery-Range:
   range(sig-onhook: path-passive(A,B)) = {}
   range(sig-onhook: 3wc1(A,B))       = {}
   range(sig-onhook: 3wc2(A,B))       = {}

Rules:
3wc-1)path(A,B)
   flash(A):   path-passive(A,B),dial-tone(A),m-3wc(A).
3wc-2)path-passive(A,B),ringback(A,C),cond:m-3wc(A)
   flash(A):   3wc1(A,B),3wc2(A,C),ringback(A,C),path(A,B).
3wc-3)path-passive(A,B),path(A,C),cond:m-3wc(A)
   flash(A):   3wc1(A,B),3wc2(A,C),path(A,B),path(A,C).
3wc-4)path-passive(A,B),busy-dial(A,C),cond:m-3wc(A)
   flash(A):   3wc1(A,B),3wc2(A,C),path(A,B),busy-dial(A,C).
3wc-5)3wc1(A,B),3wc2(A,C),path(A,B),busy-dial(A,C),m-3wc(A)
   flash(A):   path(A,B).
##3wc-6)3wc1(A,B),3wc2(A,C),path(A,B),path(A,C),m-3wc(A)
##     flash(A):      path(A,B),>sig-onhook:C.
3wc-6)3wc1(A,B),3wc2(A,C),path(A,B),path(A,C),m-3wc(A)
   flash(A):   path(A,B),>sig-onhook(A,C).
3wc-7)path-passive(A,B),busy(A),m-3wc(A)    lash(A):       path(A,B).
3wc-8)path-passive(A,B),ringing(A,B),m-3wc(A)
   offhook(A):         path(A,B).
3wc-9)3wc1(A,B),3wc2(A,C),path(A,B),Calling(A,C),m-3wc(A)
   flash(A):   path(A,B),idle(C).
3wc-10)path-passive(A,B),(path(A,C)|ringback(A,C),r-path(A,C)|Busy(A,C)|
   dial-tone(A)|hangup(A)),cond:m-3wc(A)
```

```
   onhook(A):        path-passive(A,B),ringing(A,B).
##3wc-11)path-passive(A,B),m-3wc(A),cond:not[ringing(A,B)]
##     sig-onhook(B):       empty.
3wc-11)path-passive(A,B),m-3wc(A)    sig-onhook(B,A):    empty.
##3wc-12)(3wc1(A,B),3wc2(A,C)|3wc1(A,C),3wc2(A,B)),
##path(A,B),m-3wc(A),not[ringing(A,B)]
##     sig-onhook(B):       empty.
3wc-12)(3wc1(A,B),3wc2(A,C)|3wc1(A,C),3wc2(A,B)),path(A,B),
              m-3wc(A)
              sig-onhook(B,A):          empty.
3wc-13)3wc1(A,B),3wc2(A,C),(path(A,C)|ringback(A,C),
       r-path(A,C)|Busy(A,C)|hangup(A)),path(A,B),m-3wc(A)
       onhook(A):   idle(A).
##3wc-14)path-passive(A,B),ringing(A,B),m-3wc(A)
##     sig-onhook(B):       idle(A).
3wc-14)path-passive(A,B),ringing(A,B),m-3wc(A)
   sig-onhook(B,A):              idle(A).
3wc-t-1)path-passive(A,B),busy-dial(A,C),m-3wc(A)
       timeover(path-passive(A,B),busy-dial(A,C),m-3wc(A)):
       path(A,B).
3wc-t-2)path-passive(A,B),busy(A),m-3wc(A)
       timeover(path-passive(A,B),busy(A),m-3wc(A)):
       path(A,B).

File: CCBS.text
########################################
###          CCBS                    ####
########################################

Primitives:
   m-CCBS(A,B),m-CCBSed(A,B),acceptCCBS(A)
Events:
   CCBS(A), [idle(A)]
Macro-Primitives:
   M-CCBS(A,B) = {m-CCBS(A,B),m-CCBSed(B,A)}

Rules:
ccbs-1)busy-dial(A,B),not[idle(B)]
   CCBS(A):        acceptCCBS(A),M-CCBS(A,B).
ccbs-2)acceptCCBS(A),M-CCBS(A,B)  [idle(B)]:     Calling(A,B).
ccbs-3)acceptCCBS(A),m-CCBS(A,B)  onhook(A):  idle(A).
##ccbs-4)m-CCBSed(A,B)        sig-onhook(B):      empty.
ccbs-4)m-CCBSed(A,B)   sig-onhook(B,A):           empty.
```

# Appendix 3: Introduction to STR

## Introduction to STR

Yutaka Hirakawa
ATR Communication Systems Research Laboratories

## 1. Description of telecommunication services

This paper explains the nature and significance of the STR description method.

In the development of software, first what is to be accomplished by the software is described as a set of specifications. This is an important stage in software development because by describing the functions to be realized by the software in advance, problems in design can be identified early in the process.

First we use the example of telephone exchange system software to show the problems with conventional specification description methods.

## 2. Special features of telephone services

Three-way calling and call waiting are concrete examples of telephone services.

(1) Three-way calls

When the user performs a flash hook at a terminal during a call, the other party is put on hold and the terminal goes into the receive dial tone state, in which the third party can be called. Another flash hook brings the party on hold back on line, making three-way conversation possible. The terminal which requested and established the three-way call is called the controller of the three-way call.

(2) Call waiting

Consider the situation in which a terminal that has a call in progress is dialed by another party. If the dialed terminal subscribes to the call waiting service, the terminal of the dialing party goes into the receive ringback tone state; at the call destination, the called party is notified of a call waiting by a tone signal. In this state, the called party can switch between other parties by a flash hook.

These services are offered at the same time and there is an unspecified number of terminals in the system; these have mutual effects on each other. For example, a terminal which is the controller of a three-way call cannot request a new three-way call until the existing three-way call service has ended. However, a participant in a three-way call

which is not the controller can request a new three-way call. This can be considered an example of an established three-way call being affected by the operations of a new three-way call.

Furthermore, consider the case in which all of the parties in a three-way conversation are subscribers to the call waiting service. When some terminal dials to one of the terminals and that terminal is busy, and even if that party is a subscriber to the call waiting service, that service is not available to the terminal if it is the controller of the three-way call. On the other hand, if the other party is in a three-way call but is not the controller, the call waiting service is available. This can be considered an example of an established three-way call service state having an effect on the operations of a different service (call waiting).

In this way, we see that the state of an existing service can affect the operations of other services. This is usually called service interaction. In most service provision systems, this presents a severe problem in software design because it is necessary to consider the interaction with all of the existing services when a new service is added to the system.

In the next section, we consider the conventional method of specification description from the point of view of such complex interaction among multiple services.

## 3. Conventional specification description methods

Most of the specification description languages currently in use employ procedural descriptions, where the term "procedural" refers to description of a chronological flow of processing. Usually, the architecture (what kinds of processes there are, which processes have communication channels with which processes, etc.) of the target system is first defined. Then what kind of transactions the various processes in one system have with peripheral processes, and what kind of processing is done by each process is described.

This method has the following two points of difficulty.

(1) Concise description of the operations for multiple interacting services is difficult

In conventional specification description, even operations such as the exchange of communication signals between processes and so on that cannot be recognized from outside the system are specified rigorously. Thus description is affected by factors such as the data structure of the communication signals and the implementation method.

What is actually done at present is that in the initial stage of the design process, only typical operations for a single service are described, and operations for multiple interacting services are specified in natural language.

Although this means that the operation specification for the case of multiple interacting services is not finished until the later stages of the design process, the result includes the operations of all services for one terminal and is extremely complicated algorithm description that is nearly as detailed as programmed operations.

(2) A service requires different descriptions for different structures

There are various exchange systems having various structures. However, the system structure is irrelevant from the users point of view, and the operations of the services realized on the different systems are mostly the same. Present specification description

methods require that new specification description be written for a different system structure, even for the same service.

Techniques for solving these problems are discussed in the following sections.

## 4. Service description method

In this section, the STR (State Transition Rule) description method is introduced. This method was developed as a part of research on solutions to the problems described in the previous section. STR description employs a declarative description approach rather than the procedural description that has been used until now. Also, the internal structure of the system in question becomes irrelevant to the description and the internal system structure is treated as a black box. Service operations are specified on the basis of signal inputs (events) from the user and terminal states that can be recognized by the user.

(1) State descriptions

In the STR method, states are modelled and represented as shown by the examples below.

State description examples:

Description of the idle state of terminal A {idle(A)}

Description of terminal A while receiving a dial tone {dial-tone(A)}

Description of the state of terminal A during a call with terminal B {path(A,B)}

Description of the state of terminal A while terminal B is on hold during a call with terminal C {path-passive(A,B),path(A,C)}

The terms such as "idle(A)" used in the state descriptions are called state primitives. These state primitives are models of the equipment resources used by the terminal in that state. Terms such as "path(A,B)" represent that there is a communication path between terminal A and terminal B.

These state descriptions can be used to represent partial situations within a system that comprises multiple terminals, such as in the example below.

{path(A,B),path(B,A),ringback(C,D),ringing(D,C)}

The above example describes a situation involving four terminals in which terminal A and terminal B are talking to each other, and terminal C is calling terminal D.

(2) Operation rule descriptions

Service operations are specified by STR description as sets of state transition rules. Each rule consists of the following three elements.

"current state description"
"event description"
"next state description"

In the current state description and next state description, the partial situation description mentioned above (normally consisting of the state descriptions of multiple terminals) can be used. Examples of rule description are given below. (This is the same as displayed in Fig. 1.)

Rule 1) idle(A) offhook(A): dial-tone(A).
Rule 2) dial-tone(A),idle(B)
dial(A,B): ringback(A,B),ringing(B,A).
Rule 3) dial-tone(A),not[idle(B)] dial(A,B): busy(A).
Rule 4) ringback(A,B),ringing(B,A)
offhook(B): path(A,B),path(B,A).

These rules specify the following respective operations.

Rule 1) If the receiver is lifted off hook while the telephone set is not being used (called the idle state), the state changes to the receive dial tone state.

Rule 2) If terminal A dials to terminal B while terminal A is in the receive dial tone state and terminal B is idle, then terminal A will change to the receive ringback tone state and terminal B will change to the ringing state.

Rule 3) If terminal A dials to terminal B while terminal A is in the receive dial tone state and terminal B is not in the idle state, terminal A changes to the receive busy signal state and the state of terminal B does not change.

Rule 4) If terminal B answers (receiver is lifted off hook) while terminal A is in the receive ringback tone state and terminal B is in the ringing state, then both terminals change to the call-in-progress state.

For comparison, the conventional specification descriptions (SDL) corresponding to the above four rules are shown in Fig. 2. The descriptions in Fig. 1 and Fig. 2 are equivalent. The operations corresponding to rule 2 and rule 3 of Fig. 1 are described as the following type of procedure in Fig. 2.

When terminal A is in the receive dial tone state and a dial specifying terminal x is received, it sends a signal 1 (incoming request) to terminal x. If terminal x receives signal 1 while in the idle state, it returns signal 2 (incoming answer), and changes to the receive ringing tone state. Terminal A changes to the receive ringback tone state. If terminal x receive signal 1 in a state other than the idle state (receive dial tone or other such state corresponding to "not[idle(B)]"), then it returns signal 3 (busy) and terminal A changes to the receive busy signal state while no state transition occurs in terminal x.

In the conventional design method, the designer, understanding what kind of operations are needed for each situation (such as described in Fig. 1), creates the control procedures that will implement them in the form shown in Fig. 2. It is no easy matter to create procedures that do not contradict any of the situations in Fig. 1 and also omit nothing. In contrast with this, the STR method each situation is described separately, so description becomes something that humans can understand and feel comfortable with.

## 5. Description according to design development

Proceeding with design, the following type of process is sometimes experienced.

Step 1) In basic telephone service, a busy tone is received when the dialed party is in a call. The operations for achieving this are designed.

Step 2) Design proceeds, and incorporation of a call waiting service is considered. In the call waiting service, the call is completed even when the dialed party is in a call if that party subscribes to the call waiting service (m-cw(B)). Thus, the basic telephone service designed earlier must be modified.

Generally, in this kind of situation, it is necessary to redo parts of the design as the design process proceeds.

Before continuing on to our discussion on description of operations specification by STR in this step, we explain the stipulations for rule application introduced by STR.

Rule 1) dial-tone(A),idle(B) dial(A,B): ringback(A,B),ringing(B,A).
Rule 2) dial-tone(A),idle(B),m-cfv(B,C),idle(C)
dial(A,B): ringback(A,C),ringing(C,A),pingring(B).

Rule 1 specifies ringing operations for the basic telephone service and rule 2 specifies operations for the incoming call transfer service. In the call transfer service, terminal B invokes the all transfer service and terminal C is specified as the destination for the transfer (m-cfv(B,C)). If terminal C is idle when terminal A dials to terminal B, that call is transferred and terminal B is informed or the transfer by a special ringing tone of very short duration (pingring).

Here, we consider the following situation.

Terminal A: dial-tone(A)
Terminal B: idle(B),m-cfv(B,C)
Terminal C: idle(C)

If in this situation terminal A dials to terminal B, both of the above rules can be applied. In such a case, the following stipulations are abided by.

Rule application stipulations:
In cases where two rules (rule 1 and rule 2) can both be applied, rule 2 is applied if the application conditions of rule 1 are completed included in the application conditions of rule 2.

Having completed the preparation, the two steps mentioned above as described by STR are shown below.

Step 1) dial-tone(A),path(B,C) dial(A,B): busy(A),path(B,C).

Step 2) dial-tone(A),path(B,C) dial(A,B): busy(A),path(B,C).
dial-tone(A),path(B,C),m-cw(B)
dial(A,B): ringback(A,B),path(B,C),cw-ringing(B,A).

In this way, the operations for step 1 can be specified by one rule, and the operations for step 2 can be specified by two rules. What should be noted here is that in the development of design from step 1 to step 2 the STR description is accomplished by simply adding a new rule.

If, as design proceeds, something is overlooked, redesign is inescapable in the case of

procedural description, but with STR description redesign is most often not required because of "rule application stipulations". This is an extremely important quality during design evolution.

## 6. Example of description of operations for multiple interacting services

Here, we carry out a concrete description for the two examples of service interaction described at the beginning of this paper.

Example 1: Description of interaction between three-way calls

The description of operations for when a three-way call is requested is shown below.

Rule 1) path(A,B) flash(A): m-3wc(A),path-passive(A,B), dial-tone(A).

For example, designing the three-way call controller state as "m-3wc(A),3wc2(A,C),path(A,B),path(A,C)", the states of the other subscribers are "path(B,C)" and "path(C,A)". The above rule can be applied for all three-way call subscribers. To specify that the rule is inapplicable only for the call controller, the following rule is introduced.

Rule 2) path(A,B),m-3wc(A), flash(A): path(A,B),m-3wc(A).

Example 2: Call waiting for a three-way call

The following rule implements the call waiting service.

Rule 1) dial-tone(A),path(B,C),m-cw(B)
dial(A,B): ringback(A,B),path(B,C), m-cw(B),cw-ringing(B,A).

Assuming the state for each terminal of the three-way call described in example 1, if each terminal subscribes to the call waiting service, the above rule makes call waiting available for all of the terminals. Inhibiting the call waiting service for the call controller only is specified by the following rule.

Rule 2) dial-tone(A),path(B,C),m-cw(B),m-3wc(B)
dial(A,B): busy(A),path(B,C), m-cw(B),m-3wc(B).

In this way, various forms of service interaction can be precisely specified by using STR description.

---

This article briefly describes backgrounds of STR method and its merits.This is the first delivery of STR description specifications. This article aims to inform STR description specification for colleagues in ATR communication systems research labs. and related organizations which has an interest in STR method.

Any comments and suggestions are welcome.

```
rule 1) idle(A)   offhook(A):      dial-tone(A).
rule 2) dial-tone(A), idle(B)
            dial(A,B):   ringback(A,B),ringing(B,A).
rule 3) dial-tone(A),not[idle(B)]
            dial(A,B):   busy(A).
rule 4) ringback(A,B),ringing(B,A),
            offhook(B):   path(A,B),path(B,A).
```

Fig. 1 STR description examples

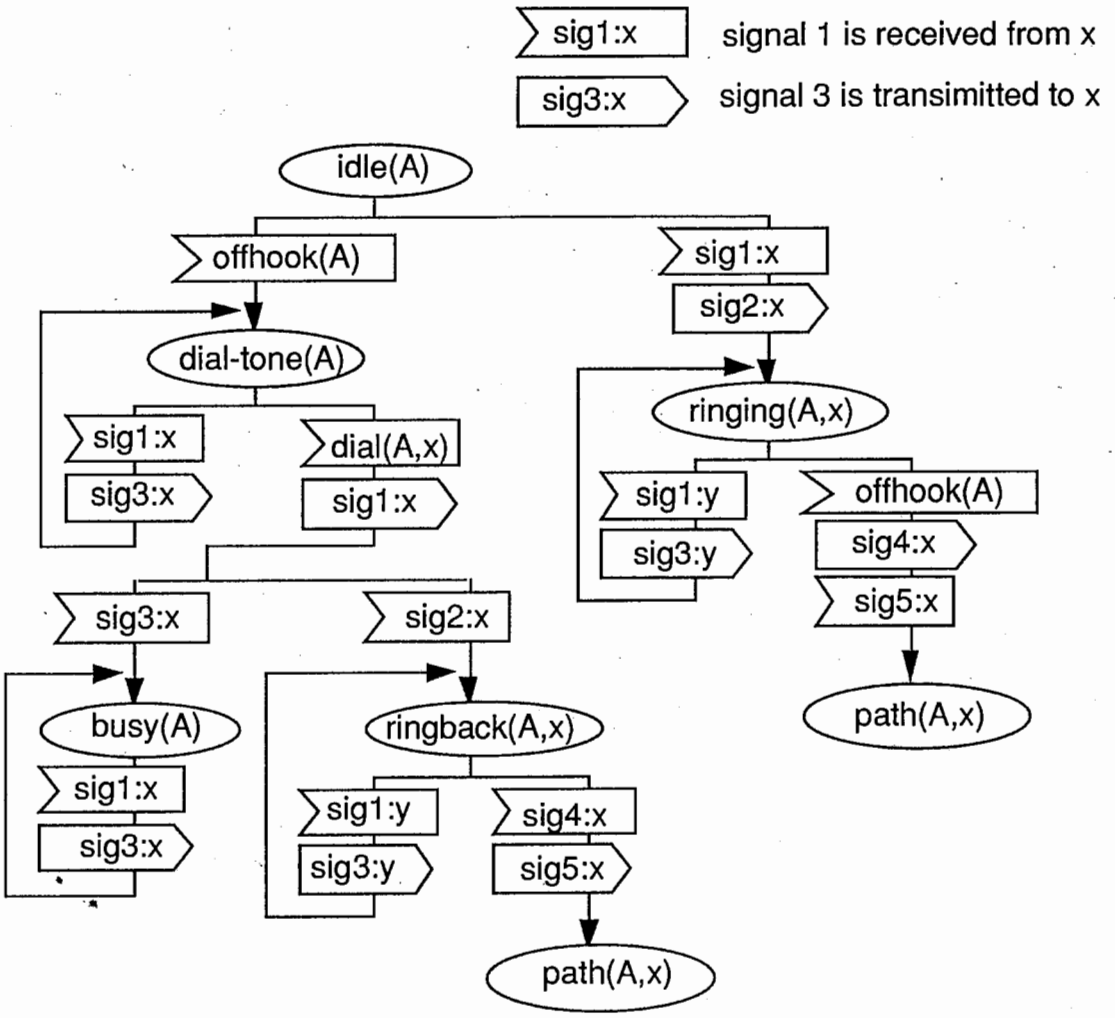



Fig. 2 Conventional specification description