〔公　開〕

ＴＲ－Ｃ－００７３

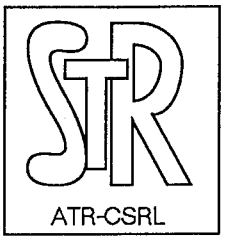
ＳＴＲ（State Transition Rule）
記述仕様書

平川　豊
Yutaka HIRAKAWA

１９９２．１．１４

ＡＴＲ通信システム研究所

STR (State Transition Rule) 記述仕様書



平成３年１２月　STR-1.0

平　川　　豊

目　次

本仕様書に関する問い合わせ先
619-02　京都府相楽郡精華町乾谷・三平谷
ＡＴＲ通信システム研究所　　平川　豊
Tel:　07749-5-1236　　Fax:　07749-5-1208

バージョンアップ記録
STR-0.0：平成３年１２月２日（平川）
STR-1.0：平成３年１２月４日下記項目の追加（平川）
・"m-"記述の項追加
・内部通信信号（内部イベント）の表記法を変更
STR-1.0：平成３年１２月１３日（平川）
・ＢＮＦ修正
STR-1.0：平成３年１２月１９日（平川）
・文章の推考

１、はじめに

本稿は、通信サービスの動作記述のために開発されたＳＴＲ（State Transition Rule）記述手法について概説する。ＳＴＲは、ATR通信システム研究所において、平成元年春より研究が開始された言語であり、主に、通信ソフトウエアの研究用に用いられている。

ＳＴＲでは、以下の点を狙いとしている。

(1)システムの内部構成によらず外側から認識できる動作が同じである時、まず内部構成に依存しない記述で規定したい。（交換機は様々なアーキテクチャがあるが、外側から見た機能は全て同じ場合が多い。）

(2)外側からの記述の段階で、サービス複合時の場合の動作も含め、形式的に動作を規定したい。

(3)外側から記述された動作の段階で、サービス毎の論理的な衝突を検出し、サービスが組み合わされた状況に対する動作設計の支援をしたい。

(4)内部構造に依存しない動作規定を行なった後、計算機支援を受けながら、内部構成に依存したものに詳細化したい。

ＳＴＲ記述手法では、ネットワーク内部の構造をブラックボックスとするため、対象とする通信システムを次のように捉えている。

「通信システム内には、不特定多数個の端末制御プロセスが存在する。これらの端末制御プロセスは、全てが同一の機能を有している。端末プロセス以外のプロセスは存在しない。」

ＳＴＲ記述は、１つのイベントが生起した際に影響を受ける範囲を指定し、その範囲に含まれる端末制御プロセスの全ての動作を記述する手法である。

２、状態遷移規則の概要

状態遷移規則（ＳＴＲ）記述は、いくつかの宣言の記述と、規則の集合で記述される。各規則は、「現状態記述」「イベント記述」「次状態記述」の3つで構成される。動作規則の例を示す。

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

最初の規則は、空状態で受話器上げ(オフフック)すると、ダイヤル音受信状態に遷移することを規定している。2番目の規則は、ダイヤル音受信状態の端末から空状態の端末にダイヤルすると、ダイヤル端末は、リングバック音受信状態、ダイヤルされた端末は、リンギング音受信状態に遷移することを規定している。この時、各端末では、互いに相手の識別子を状態記述要素の第二番目の引き数として記憶する。

この動作規則は、任意個のプロセスで構成されるシステムに対して動作規定を行なう。システム内でダイヤルイベントが生起した場合、そのイベント生起プロセスをA、ダイヤル先のプロセスをＢとしたとき、2番目の規則の現状態記述を条件として満足する場合には、この2番目の規則に従った動作を行なうことを規定している。

ここで、状態記述を構成しているものを状態記述要素と呼ぶ。直感的には、状態記述要素は、その状態で端末が使用しているリソースあるいはメモリなどを表現している。メモ

リとは、通話中でかつ着信転送を設定している状態などのように、状態として陽に認識されるものではないが、システム外部からの動作を正確に規定する上で不可欠の要素である。

対象とするシステムは、最初全て状態が空のプロセスで構成され、イベントが生起すると、適当な規則が適用されて、プロセスの各状態が書き換わると理解することができる。

3、記述要素

3－1、状態記述要素

ＳＴＲ記述では、各端末プロセスの状態は状態記述要素の集合で記述される。状態記述要素の例を以下に示す。

POTS:

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

話中着信:

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

着信転送:

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

3者通話:

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

状態記述要素について説明する。

r-path(A,B): AからBへの通話路を既に予約している。

cw-ringing(A,B): B端末から話中着信していることをA端末に音により通知していることを示す。

pingring(A,B): 端末Aが着信転送を設定しており、空状態の時、B端末から着信した呼びが転送されたことを知らせる音が出ていることを示す。

path-passive(A,B): 端末Ｂへの通話路を保留していることを示す。

confirmation(A): 着信転送を受け付けたことを意味する確認音。

m-3wc(A): 3者通話モードに入っていることを意味する。

3wc1(A,B): 3者通話を行なっている状態で、Ｂは最初に通話していた相手であることを示す。

3wc2(A,B): 3者通話を行なっている状態で、Ｂは第三者として呼び出された相手であることを意味する。

これらの状態記述要素により、各端末の状態が表現される。例を以下に示す。
通話状態(path(A,B))でフラッシュフックにより、第三者呼び出しのためのダイヤル音受信状態(path-passive(A,B),dial-tone(A),m-3wc(A))となる。ダイヤルした相手が応答すれば、この端末は、通話状態(path-passive(A,B),path(A,C),m-3wc(A))となり、フラッシュフックにより、3者通話が完成される(m-3wc(A),3wc1(A,B), 3wc2(A,C),path(A,B),path(A,C))。

状態記述要素とその後の取り扱いとの関係について若干の説明を行っておく。これは、ＳＴＲを記述する上で必須なものではないが、その後の詳細化ツールを用いる上では認識しておく必要がある。

（１）状態記述要素

通常の状態記述要素は、各状態で用いられる装置（リソース）をモデル化している。この情報は、詳細化を行う上では重要な情報となる。即ち、path(A,B)なる状態記述からpath-passive(A,B)への遷移の際には、suspend(B)というラベルのタスクを挿入するなどといった詳細化が可能となる。

（２）"m-"の記述

話中着信の権利"m-cw(A)"などが例である。この状態記述要素は、状態変数あるいは、制御用のレジスタ情報と考えることができる。この状態記述要素は、ＳＤＬに変換した際、状態として識別されない。即ち、通話状態で着信要求イベントを受信したときには、テスト関数により話中着信の権利の有無を判定し、その有無により次状態が決定される動作となる。

３－２　各種の宣言

(1)状態記述要素の宣言

状態記述要素は、必ず事前に宣言をする必要がある。記述例を次に示す。

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

(2)イベントの宣言

ＳＴＲ記述で用いられるイベントは、事前に宣言する必要がある。

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

状態記述要素とイベントの宣言は、記述の誤りチェックの為であり、ＳＴＲ記述にとって本質的なものではない。

(3)時間制限付きの状態記述要素の宣言

dial-tone(A)などの状態は、ユーザがダイヤルせず待機していると、自動的にビジー音接続となる。このような動作を記述することを目的として、時間制限付きの状態記述要素が導入されている。

ただし、時間制限を状態記述要素(あるいは状態記述要素の並び)に付加するという概念により、全ての動作記述が可能であるという保証ない。現時点で解かっているのは、いくつかの電話サービスを記述する上では、この概念だけで十分対処できたという経験的な事実のみである。(現在把握できている詳細化のレベルにも依存していると思われる。言い換えると、目的システムの構造や制限によっては、この概念だけでは対処できない場合が

ありえると考えられる。)

記述例を以下に示す。

```
Limited-Time-Primitive:
        busy(A)                          30sec.
        busy(A),m-3wc(A),3wc2(A,B)       30sec.
```

(4)内部イベントを分配することをデフォルトとすることの宣言

オンフック動作の場合には、広域的に記述しようとすると、規則数が関連端末の状態数の乗算的に増えてしまい、記述が大変である。そのため、自由に内部通信信号を定義し、使用することを可能化した。この際、オンフックした際の動作は、比較的定形的に扱うことができ、信号の分配をデフォルトとすると記述が容易になる。そのために、以下のような宣言を導入した。

```
Internal-Siganl-Deliveration:
        onhook:  -->  sig-onhook
```

この記述により、特に指定の無い場合には、任意の端末がオンフックした場合には、その時に知っている他端末の全てに信号sig-onhookを送信することを前提とする。このため、オンフック動作の記述は以下のように簡便化される。

規則１）path(A,B)　onhook(A):　idle(A).

規則２）path(A,B)　sig-onhook(B,A):　busy(A).

ここで、sig-onhook(B,A)は、内部通信信号sig-onhookをAがBから受信したことを表現する。

また、内部信号の導入により、通常の状態遷移時に、内部信号を送信する動作の記述も導入される。例えば、次のような記述である。

規則３）3wc(A),3wc1(A,B),3wc2(A,C),path(A,B),path(A,C)

```
        flash(A):    path(A,B),>sig-onhook(A,C).
```

上記動作は、三者通話中にフラッシュにより、第三者として呼び出した相手との通話を切断し、元の二者通話に戻る際の動作記述である。

また、オンフック時に内部信号の送信先を制御したい場合には、次の宣言を用いる。

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

この宣言により、保留中の呼びが存在する場合には、オンフックしてもその相手にはsig-onhook信号が送られない。

(5)内部イベントの宣言

前記のデフォルトの信号分配は用いないが、内部信号の授受を動作の切り口として用いる場合、ここで宣言する。

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

宣言された内部イベントを用いて、受信時の動作、あるいは送信動作を規定する。例えば、次状態に">invoke(A,B)"と記述した場合には、invoke信号をプロセスAからプロセスB

に送信することを表す。また、イベント記述に"invoke(A,B):"と記述した場合には、プロセスAからのinvoke信号をプロセスBが受信することを意味する。

(6)マクロ記述の定義

常に組み合わせでしか出現しない状態記述の組み合わせ、あるいは、組み合わせたほうが概念的にスッキリするものについてマクロ記述を導入している。以下に例を示す。

Macro-Primitives:

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

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

(7)禁止状態集合の宣言

禁止状態の記述例を以下に示す。

Inhibit-Primitive-Sets:

{cw-ringing(A,B),cw-ringing(A,C)}(busy)

これは、1つのプロセスの状態記述に上記の集合が含まれることを禁止するものである。

例えば、以下の記述を考えよう。

規則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).

規則１を適用しただけでは、端末Aが話中着信を受けている状況で、更に他の着信があったとき、規則の条件が満足されるため、いくらでも呼びが着信してしまう。これを防ぐための記述が禁止集合である。

いったん話中着信を受け付けている状態で他の端末からダイヤルされた場合に、規則１を適用しようとすると、上記の禁止集合を持つ端末が現われてしまう。その場合、この規則の適用を行なわず、その原因となったイベント生起プロセスをビジー音接続とする。この際のイベント生起プロセスの次状態構成手順は次のとおり。

[次状態] (busy)：まず、現在の状態から、規則が適用される場合に取り除かれる状態記述要素を取り除く。その後「busy(process-id)」を付加する。

禁止状態で「(busy)」の明記されていない場合には、原因となったイベント生起プロセスの次状態は次の手順で作られる。

[次状態](busy)なし：現在の状態から、規則が適用される場合に取り除かれる状態記述要素を取り除く。

この禁止状態に関するチェックは、通常の適用規則決定の後に行なわれる。従って、禁止状態集合に抵触したために適用規則が変更されることはない。

３－２　イベント記述

ＳＴＲでは、外部からのイベントは自由に記述することができる。ただし、事前に使用するイベントを宣言する必要がある。

イベントは様々な抽象レベルで記述することができる。

ＳＴＲ記述では、次の4種類のイベントを用いることが出来る。

１）ユーザイベント

ユーザから投入されるイベントを記述したもので、dial(A,B):、flash(A):などの例を挙げることができる。

２）タイムオーバイベント

時間制限のある動作を記述する際に用いられる。timeover(dial-tone(A)):などのように、"timeover("の後に、状態記述要素の並びが来る。タイマーは、この状態記述要素により識別される。(時間制限のある動作の記述の項参照)

３）疑似イベント

CCBS動作では、次のような記述を用いている。

規則１） m-CCBS(A,B),confirmation(A),m-CCBSed(B,A)

[idle(B)]: ringback(A,B), ringing(B,A).

これは、AがＢにダイヤルし、相手ビジーに遭遇した場合、CCBSを要求し、受話器を上げたまま待機していると、相手がアイドル状態となると直ちに相手呼び出しの状態となる動作を規定している。この際には、相手がアイドル状態となることを契機として状態遷移が行われなくてはならない。このような動作を記述するために導入された記法である。

４）内部イベント

Internal-Signal-Delivery:宣言により指定された内部通信信号、あるいは、Internal-Events:宣言により指定されている信号は、これを内部イベントとする動作の記述が可能である。例を以下に示す。

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

内部イベント受信動作は、必ず1つの端末の動作でなくてはならない。即ち、現状態記述に用いられる状態記述要素の第一引数は、内部イベントの第二引数に一致していなくてはならない。

３－３、その他の記述

(1)cond記述

以下に示す規則１は規則２と同じである。

規則１)dial-tone(A),cond:path(B,C),cond:m-cw(B)

dial(A,B): CW-ringing(A,B).

規則２)dial-tone(A),path(B,C),m-cw(B)

dial(A,B): CW-ringing(A,B),path(B,C),m-cw(B).

このように、状態遷移に際して、参照はするが、置き換えの対象でない記述要素については、規則２のように現状態記述と次状態記述に重複して書いてもよいし、また、規則１のように略記することができる。cond記述は、現状態記述以外では用いてはならない。

(2)not記述

not[idle(B)]などのように使用する。この記述は、現状態記述以外では用いてはならない。cond記述と同じように、規則適用の条件として、該当プロセスの状態記述に指定され

た状態記述要素が含まれないことをテストする記述である。

not記述は、記述する側からすれば便利な記法であるが、問題点も含んでいる。例えば、次の2つの規則を考えてみよう。

規則１）dial-tone(A),not[idle(B)] dial(A,B): busy(A).

規則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).

この2つの規則が同時に存在した場合には、規則２が適用可能な状況では、必ず規則１も適用可能となる。しかし、このことを機械的に判断することは困難である。従って、解釈実行システム、検証などを考える際に大きな問題となる。

（機械が判断する際には、path(B,C)とidle(B)の共存の可能性を排除できないためである。）

そのため現状では、互いに排他的であることが明確に解かるように、2つの規則間の包含関係を調べる際、一方の規則にnot[idle(A)]が存在する場合には、他方には、必ずidle(A)かnot[idle(A)]のどちらかが存在していないといけないという制約条件を導入している。

(3)or記述

規則１）path(A,B) onhook(A): idle(A).

規則２）busy(A) onhook(A): idle(A).

上記2つの規則は次のようにまとめて書くことができる。

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

４、ＳＴＲが規定する動作

４－１規則適用の基本

次のような状況を考えてみよう。ここで、ｐ、ｑ、ｒ、ｓはそれぞれ各端末の識別子である。

端末ｐ）dial-tone(ｐ)

端末ｑ）dial-tone(ｑ)

端末ｒ）idle(ｒ)

端末ｓ）idle(ｓ)

次の規則を考えてみる。

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

上記の状況で、端末ｐが端末ｒにダイヤルした場合には、規則１がA=ｐ、Ｂ=ｒとして規則が適用され、端末ｐが端末ｒを呼び出している状態に遷移する。また、続いて端末ｒが端末ｓにダイヤルすると、同じ規則がA=ｑ、Ｂ＝ｓとして適用され、端末ｑが端末ｓを呼びだしている状態に遷移する。

このように、規則で用いられている変数は、イベントが生起したことを契機として束縛され、現状態記述で指定された記述が全て満足されたとき、その規則が適用される。

ただし、規則で用いられている異なる変数にはそれぞれ異なる識別子が対応しなくてはならない。例えば、次の状況を考えよう。

端末ｐ）path(ｐ,ｑ),m-cfv(ｐ,ｑ)

端末ｑ） path(ｑ,ｒ)
端末ｒ） dial-tone(ｒ)
　次の規則を考える。
規則２） dial-tone(A),cond:m-cfv(B,C),cond:path(B,D),not[idle(C)]
　　dial(A,B):　busy(A).
　上記の状況で、端末ｒが端末ｐにダイヤルした場合を考える。規則の適用可能性を調べると、A=ｒ、B=ｐ、C=ｑ、D=ｑと対応付けが可能である。しかし、ここで、CとDという異なる変数に同じ識別子ｑが対応するため、規則２は適用されない。

4－２、規則適用の優先順位
　まず、基本サービスの動作として次の規則を記述したとしよう。
規則１） dial-tone(A),idle(B)　dial(A,B): ringback(A,B),ringing(B,A).
　次に、着信転送を考えた場合、ダイヤルさきが空状態でも呼びは転送される必要がある。これをＳＴＲで記述すると、次のようになる。
規則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).
　ここで、以下の状況を考える。
端末ｐ） dial-tone(ｐ)
端末ｑ） idle(ｑ),m-cfv(ｑ,ｒ)
端末ｒ） idle(ｒ)
　上記の状況では、規則１規則２が共に適用可能であり、それぞれ別の動作を規定している。しかし、規則１のような基本動作を規定した後、設計が進むに連れて、規則２のような例外的な動作の規定に気がつくことはよくあることである。規則２を導入することで、既存の規則をしばしば修正することは避けたい。
　これを可能とするため、ＳＴＲでは、次のような規則間の優先順位を導入している。

[規則間の優先順位]
　上記2つの規則のように、規則１の適用条件が規則２の適用条件に完全に包含されている場合、両規則が適用可能な状況では、優先的に規則２を適用する。

　このような規則間に優先順位を導入することにより、設計の進展に応じて次々に例外的な動作の規定が必要な場合、既存の規則には何の修正も必要なく、規則を追加するのみで動作規定が可能となる。

５、記述上の制約
　ＳＴＲを記述するうえで守らなくてはいけない制約を以下に記述する。
（制約１）
　ＳＴＲ規則の次状態記述で用いられているプロセスの識別子は、同じ規則の現状態記述、あるいはイベント記述中で用いらているものでなくてはならない。

（制約２）

ユーザイベントが生起した場合の動作記述では、イベント生起プロセスの状態記述が現状態記述に含まれていなくてはならない。

（制約３）

ユーザイベントが生起した場合の動作記述では、イベント記述と現状態記述をグラフ表現した場合、イベント生起プロセスを根として、全ての点に到達可能である必要がある。

（制約４）

内部イベントが生起した場合の動作記述は、1つのプロセスに対する動作規定でなくてはならない。

ＳＴＲ記述は、各端末の状態記述集合を置き換える規則である。従って、意味的に置き換えと解釈できない記述は受け入れられない。また、ＳＴＲ規則で指定された状態は全て互いの通信により、その存在を確認する必要がある。そのためには、イベント生起プロセスから指定された全てのプロセスに直接あるいは間接的にアクセスする手段がなくてはいけない。

６、関連するツール

これまで（平成3年度まで）に開発しているツールを以下に列挙する。

(1)解釈実行システム

稼働条件: リスプマシン

概要: ＳＴＲ記述（旧バージョンが前提、記法が若干現状と異なる）を読み込み画面上でのマウスによるイベント投入に従って、動作状況をアニメーション表示するシステムである。

使い道: ＳＴＲ記述を開発する際に、ラピッドプロタイピングを可能とする。

参考事項： 平成4年度内にはSUN上で稼働する新たなシステムが完成予定

(2)ＳＤＬ変換システム(デモ用)

稼働条件: リスプマシン

概要： ＳＴＲ記述を読み込み、各端末を制御するＳＤＬ記述を出力する。

使い道： デモンストレーション

参考事項： 平成4年度内にはSUN上で稼働する新規システムが完成予定

(3)ＳＤＬ詳細化システム

稼働条件： リスプマシン

概要： 概要ＳＤＬと詳細化知識ファイルを読み込み、詳細化ＳＤＬを出力する。

参考事項： 平成3年12月現在は、上記ＳＤＬ変換システムに組み込み。平成3年度末までには、SUN上での独自システムとして移植終了予定

(4)ＳＤＬからＣＳＬへの変換システム

稼働条件： リスプマシン

概要： 詳細化されたＳＤＬをネットワークシュミレータ上で稼働可能とするためにＣＳＬに変換する。

(5)生成ソフトウエア稼働環境

稼働条件：　リスプマシン

概要：　ネットワークシュミレータとは、表示系のバージョンのみが異なる。一度、ネットワークシュミレータを表示系のみ新しいバージョンで立ち上げ、稼働可能な状態とし、各マシンごとのＣＳＬ記述を自動生成したもので上書きする。この操作により、自動生成されたＣＳＬを実稼働可能となる。

(6)検証支援システム

稼働条件：　リスプマシン

概要：　旧バージョンのＳＴＲ記述を読み込み、1)端末の取りえる状態集合の獲得、2)状態イベントテーブルの生成、3)競合する可能性のある規則の組を検出し、競合する状況と共にアニメーション表示する。

参考事項：　平成４年度中にはSUN上に移植完了予定。

(7)その他、開発中のものを以下の示す。

ＳＴＲ規則のビジュアル入力システム
(解釈実行システムと結合予定)
高速検証システム
到達可能性判定システム

７、ＳＴＲに関連する文献リスト

（平成3年12月現在、投稿中のものを除く）

[1]原田,平川,竹中,門田,「サービス仕様の自動生成機構に関する考察」
情報処理学会全国大会, 5S-5, 平成元年１０月
[2]原田,平川,竹中,「サービス仕様の自動生成に関する考察--自動生成機構の構想」
情報処理学会全国大会, 3R-2, 平成２年３月
[3]原田,平川,竹中,「サービス記述検証支援システムの構想」
情報処理学会全国大会, 4H-9, 平成２年９月
[4]河田,平川,竹中,「分散システムの記述とプロセス動作」
情報処理学会全国大会, 4Q-5, 平成２年９月
[5]柴田,平川,竹中,「プロセス数に依存しない動作記述における状態の到達可能性解析」情報処理学会全国大会, 4Q-4, 平成２年９月
[6]平川,竹中,「断片的要求からのソフトウエア自動作成手法」
信学会全国大会, B-412, 平成２年１０月
[7]原田,平川,竹中「通信サービス追加時のサービス絡み動作設計支援方法」
信学会全国大会, B-534, 平成３年３月
[8]河田,平川,竹中「通信サービス記述からプロセス動作仕様への変換の概要」
信学会全国大会, B-535, 平成３年３月
[9]柴田,平川,竹中「通信システムにおける断片的要求の充足性判定の解析」
信学会全国大会, B-536, 平成３年３月
[10]原田,平川,竹中「通信サービス動作競合検出手法」
信学会全国大会, B-339, 平成３年９月

[11]柴田,平川,竹中「通信サービスに対する断片的要求の理解」
信学会全国大会, B-336, 平成３年９月
[12] Hirakawa, Harada, Takenaka,
"A Description Method for Advanced Telecommunication Services"
IJECE Technical Report, SSE89-87, pp.37-42, October 1989
[13] 平川、原田、竹中、「高度通信サービスのためのサービス記述手法」
アドバンストネットワーク研究会、1989
[14] Shibata, Hirakawa, Takenaka,
"Reachability Analysis for a Behavior Description Independent of the Number of Processes", JWCC'90, pp. 85-93, July 1990
[15]原田、平川、竹中「通信サービス追加設計支援方法の一考察」
信学会交換研究会、SSE90-137, pp.13-18, March 1991
[16]原田,平川,竹中「通信サービス動作競合検出手法」
情報ネットワーク研究会、IN 91-86, pp. 43 - 48, Sept., 1991
[17]河田,平川,竹中「通信サービス仕様からプロセス動作仕様の生成」
第三回ネットワーキングアーキテクチャワークショップ, B-7, Oct., 1991
[18]高見,平川,河田,竹中「ビジュアルプロトタイピングによる通信サービス仕様生成法」信学会交換研究会、SSE91-103, pp. 53 - 58, Nov., 1991
[19]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.
[20]Hirakawa, Takenaka, "Telecommunication Service Description Using State Transitrion Rules", Int. Workshop on Software Specification and Design, pp. 140 - 147, Oct., 1991
[21]Harada, Hirakawa, Takenaka, "A Design Support Method for Service Interactions", Globecom'91, 46 - 3, Dec., 1991
[22]Hirakawa, Kawata, Takenaka, "Rule Descriptions for Telecommunication Serbvices and Its Transformation into Standardized Specification Descriptions", SETSS'92, March 1992

付録１：ＳＴＲのBNF記述

```
################################################

  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
```

STR-Manual(13)

```
        := "{" 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
```

付録２：通信サービスのＳＴＲ記述例

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)
#Ill-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.

付録３：Introduction to STR

Introduction to STR

Y. Hirakawa
ATR Communication Systems Research Laboratories

１　通信サービスの記述

本稿では、ＳＴＲ記述手法の意義と性質について紹介します。

ソフトウエア開発にあたって、まず、ソフトウエアで実現すべきことを仕様として記述することが行なわれます。これは、ソフトウエアで実現すべき機能を事前に記述することにより、早期段階で設計上の問題点を明らかにすることができ、ソフトウエア開発における重要な段階となっています。

ここではまず、電話交換システムのソフトウエアを例に、従来の仕様記述手法の問題点を明らかにします。

２　電話サービスの特徴

まず、具体的な電話サービスから、三者通話サービスと話中着信サービスを紹介します。

（１）三者通話

通話中の端末でフラッシュフックすることにより、今通話中であった相手を保留し、ダイヤルトーン受信状態となり、第三者が呼び出せる状態となります。第三者と通話状態となったとき、再びフラッシュフックすると、保留中の相手を加えた三者間で通話が可能となります。このとき、三者通話を要求し、成立させた端末は、三者通話のコントローラと呼ばれます。

（２）話中着信

通話中の端末に対してダイヤルしたとする。このとき、通話中の相手が話中着信サービスに加入している場合には、ダイヤルした端末は呼び返し音受信の状態となり、着側では、通話中に着信のあったことが音により通知される。この状態で、フラッシュフックにより、通話相手を切り替えることができます。

これらのサービスは同時提供され、またシステム内には不特定多数の端末が存在するため、互いに影響を及ぼしあうことがあります。

例えば、三者通話サービスでは、一旦三者通話のコントローラになった端末は、その三者通話サービスを終了までは、新たな三者通話を要求することはできません。しかし、三者通話に参加していてもコントローラでなければ、新たな三者通話サービスを要求することができます。これは、既存の三者通話サービスの状態が新たな三者通話サービスの動作に影響を与えている例と考えることができます。

また、いま、三者通話を行っている全ての加入者が話中着信サービスに加入している場合を考えてみましょう。ある端末からダイヤルしたとき、その相手が通話中でかつ、話中

着信サービスに加入していた場合でも、三者通話のコントローラである場合には、話中着信できません。一方、相手が三者通話を行っている場合でも、コントローラでなければ話中着信が受け付けられます。これは、既存の三者通話サービスの状態が、別の話中着信サービスの動作に影響を与えている例です。

このように、電話サービスでは、既存のサービスの状態が他のサービス動作に影響を与えることがあります。これは、一般にはサービスインタラクションと呼ばれるものです。これは、多くのサービスを提供しているシステムに、新たなサービスを導入しようとした場合、既存のサービス全てとの間でサービスインタラクションを考慮する必要があり、ソフトウエアを設計する上で極めて深刻な問題となっています。

次節では、この複数サービスが絡んだ場合の動作の規定方法という観点から、従来の仕様記述手法を考えてみます。

3　従来の仕様記述手法

現在、仕様記述言語として用いられているもののほとんどは、手順的な記述を用いています。手順的とは、時系列的な処理の流れを記述することをいいます。通常は、まず、対象とするシステムの構造（どのようなプロセスがあり、どのプロセスとどのプロセスが通信チャネルを持つかなど）を定義し、１つのシステム中の各々のプロセスが周辺プロセスとどのようなやり取りを行い、また各プロセスでどのような処理を行うかを記述します。

この手法には次の２つの難点があります。

（１）複数サービスが絡んだ動作の簡潔な記述が困難

従来仕様記述では、プロセス間の通信信号の授受など、システムの外側からは認識できない動作も厳密に規定します。そのため、通信信号のデータ構造など、インプリメント手法に影響を受けた記述となります。

実際現状では、設計の初期工程では、単一サービスに対して典型的な動作のみを記述し、複数サービスが絡んだ場合の動作は自然言語で規定される程度です。

設計の後工程になって初めて複数サービスが絡んだ場合の動作規定ができあがるわけですが、それは、１つの端末に対して全てのサービスの動作を含み、極めて複雑でプログラム動作に近い詳細なアルゴリズム記述となっています。

（２）構造が異なる場合には、同じサービス動作でも記述が異なる

交換システムの場合には、システムによりさまざまな構造を持つものが存在します。しかし、利用者側から見た場合、システムの構造とは無関係に、その上で実現されるサービス動作は同じ場合がほとんどです。現在の仕様記述手法では、同じサービス動作を実現する場合でも、目的システムの構造が異なる場合には、新たな仕様記述を作成する必要があります。

これらの問題点を解決するための技術を次節以降で議論します。

4　サービス記述手法

本節では、前節の問題点を解決するための研究の一環として開発したサービス記述手法であるＳＴＲ（State Transition Rule）手法を紹介します。ＳＴＲ記述では、これまでのような手順的な記述を採用せず、宣言的な記述のアプローチを採用しています。また、目的

システムの内部構成には無関係な記述となるよう、システムの内部構造はブラックボックスと捉え、利用者からの信号入力（イベント）と利用者から認識できる端末の状態を基本としてサービス動作の規定を行います。

（1）状態の記述

ＳＴＲ手法では、状態をモデル化して表現します。以下に例を示します。

状態の記述例：

アイドル状態の端末Aの記述　　{idle(A)}

ダイヤルトーン受信中の端末Aの記述　{dial-tone(A)}

B端末と通話中のA端末の状態記述　　{path(A,B)}

B端末を保留し、C端末と通話中の端末Aの状態記述

{path-passive(A,B),path(A,C)}

状態記述で用いられている「idle(A)」などは状態記述要素と呼ばれます。状態記述要素は、端末が状態で用いる装置をモデル化したものです。「path(A,B)」などは、端末Aが端末Ｂとの間で通話路を持っていることを表現しています。

この状態記述を応用し、複数の端末で構成されるシステム内の部分的な状況は、例えば以下のように表現することができます。

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

上記の例は、４つの端末で構成される状況を記述しています。この状況の中では、端末Aと端末Ｂは互いに通話しており、また端末Ｃは端末Ｄを呼び出しています。

（2）動作規則の記述

サービス動作は、ＳＴＲ記述による状態遷移規則の集合で規定されます。各規則は以下の３つの要素で構成されます。

「現状態記述」　「イベント記述」　「次状態記述」

現状態記述及び次状態記述では、先に述べた部分的な状況（一般に複数の端末の状態記述で構成）の記述が用いられます。規則の記述例を以下に示します。（図１に同じものを掲示）

規則１）idle(A)　offhook(A):　dial-tone(A).

規則２）dial-tone(A),idle(B)　dial(A,B):　ringback(A,B),ringing(B,A).

規則３）dial-tone(A),not[idle(B)]　dial(A,B):　busy(A).

規則４）ringback(A,B),ringing(B,A)

offhook(B):　path(A,B),path(B,A).

これらの規則はそれぞれ以下のような動作を規定しています。

規則１）電話機を使用していない状態（idle状態と呼ばれる）で、受話器を上げると、発信音（dial-tone）受信の状態となります。

規則２）端末Aが発信音受信の状態で、端末Ｂが空状態の時、A端末からＢ端末にダイヤルすると、A端末は呼び返し（ringback）音受信の状態に、Ｂ端末は呼び出し（ringing）音の鳴っている状態になります。

規則３）端末Aが発信音受信の状態で、端末Ｂが空でない状態の時、A端末からＢ端末にダイヤルすると、A端末はビジー音受信の状態になり、Ｂ端末の状態変化はありません。

規則４）端末Aが呼び返し音受信の状態で、端末Ｂが呼び出し音の鳴っている状態のと

き、端末Ｂが応答する（受話器を上げる）と、両者は通話中の状態になります。

比較のために、上記４つの規則と対応する従来仕様記述（ＳＤＬ）を図２に示します。図１の記述と図２の記述は等価なものです。図１の規則２と規則３に対応する動作は、次ような手順として、図２に記述されています。

端末Aが発信音受信状態の時、端末ｘを指定するダイヤルを受信したとき、端末ｘに対して、信号１（着信要求）を送出します。端末ｘが信号１を空状態で受信した場合には、信号２（着信応答）が返送され、端末ｘは呼び出し音受信状態へ、また、端末Aは呼び返し音受信状態へ遷移します。一方、端末ｘが空以外の状態（not[idle(B)]に対応する発信音受信状態など）で信号１を受信した場合には、信号３（棄却）が返送され、端末Aはビジー音受信状態へ遷移し、端末ｘでは状態遷移は起こりません。

従来の設計手法では、設計者は、図１に記載したような、各状況でどの様な動作をすべきかということを理解し、それを実現する図２の形式の制御手順を作成していました。図１の各状況に矛盾せず、しかも漏れのない手順を作成することは容易なことではありません。これに対し、ＳＴＲ手法では、各状況を個別に記述するため、人間の理解と親和性のよい記述となっています。

5　設計の進展に応じた記述

設計を進めていく場合、例えば、次のような過程を経験することがあります。

ステップ１）基本通話サービスでは、ダイヤルした相手が通話中のときにはビジー音受信となる。これを実現するための動作を設計します。

ステップ２）設計が進み、話中着信サービスに対する動作を組み込むことを考えます。話中着信サービスでは、ダイヤルした相手が通話中でも、相手が話中着信サービスに加入（m-cw(B)）している場合には、着信します。このため、先に設計した基本通話サービスの動作を修正する必要があります。

このように一般には、設計が進展していく過程において設計手戻りが発生します。

この各ステップでの動作規定をＳＴＲで記述した場合について議論を進めますが、その説明のために、ＳＴＲで導入している規則適用上の約束を説明しまする。以下の２つの規則を考えましょう。

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

規則２）dial-tone(A),idle(B),m-cfv(B,C),idle(C)

　　dial(A,B):　ringback(A,C),ringing(C,A),pingring(B).

規則１は基本通話サービスにおける呼び出し動作の規則です。また、規則２は、着信転送サービスの動作規則です。着信転送サービスでは、端末Ｂが着信転送サービスを起動し、その転送先として端末Ｃを指定している（m-cfv(B,C)）とします。端末Aから端末Ｂにダイヤルされた場合、転送先の端末Ｃが空状態であれば、その呼びは転送され、端末Ｂには転送のあったことを知らせる特殊な呼び出し音（pingring）がごく短時間だけ鳴らされます。

ここで次のような状況を考えましょう。

端末A：dial-tone(A)

端末B：idle(B)、m-cfv(B、C)

端末C：idle(C)

この状況で、端末Aが端末Bにダイヤルした場合には、上記規則の両者が共に適用可能となります。このような場合、以下の約束に従うものとします。

規則適用上の約束：２つの規則（規則１と規則２）が共に適用可能な状況において、規則１の適用条件が規則２の適用条件に完全に包含されている場合には、規則２を適用する。

準備ができたので、先に述べた２つのステップをＳＴＲ記述したものを以下に示します。

ステップ１）dial-tone(A)、path(B、C)　dial(A、B)：　busy(A)、path(B、C).

ステップ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).

このように、ステップ１の動作は１つの規則により規定され、ステップ２での動作は、２つの規則により規定することができます。ここで注意すべきは、ステップ１からステップ２への設計の進展の際に、ＳＴＲ記述は単に新たな規則の追加のみで対処できている点です。

設計を進める際、見落としがあった場合、手順的な記述の際には設計手戻りが不可避ですが、ＳＴＲ記述では、多くの場合、「規則適用上の約束」により、設計の手戻りが必要ありません。これは、設計を進める際に極めて重要な性質です。

6　複数サービスが絡んだ動作の記述例

ここでは、この章の最初に紹介したサービスインタラクションの２つの例に関して、具体的に記述を行います。

例１：三者通話間の相互影響の記述

三者通話を要求するときの動作記述を以下に示します。

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

例えば、三者通話のコントローラの状態を「m-3wc(A)、3wc１(A、B)、3wc2(A、C)、path(A、B)、path(A、C)」と設計し、他の参加者の状態を「path(B、C)」、「path(C、A)」とします。上記の規則はこの全ての三者通話参加者に関して適用可能となっています。そこで、コントローラのみ適用不可能とすることは、次の規則の導入により規定されます。

規則２）path(A、B)、m-3wc(A)　flash(A)：　path(A、B)、m-3wc(A).

例２：三者通話中の端末への話中着信

話中着信を実現する規則を次に示します。

規則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).

例１で述べた三者通話の各端末状態を仮定すると、各端末が話中着信に加入している場合には、上記規則により、全ての端末で話中着信が可能となります。ここで、三者通話の

コントローラにのみ話中着信を禁止することは、次の規則の導入により規定されます。

規則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).

このように、さまざまな形のサービスインタラクションはＳＴＲ記述手法を用いることにより厳密に規定することができます。

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

図１　ＳＴＲ記述例

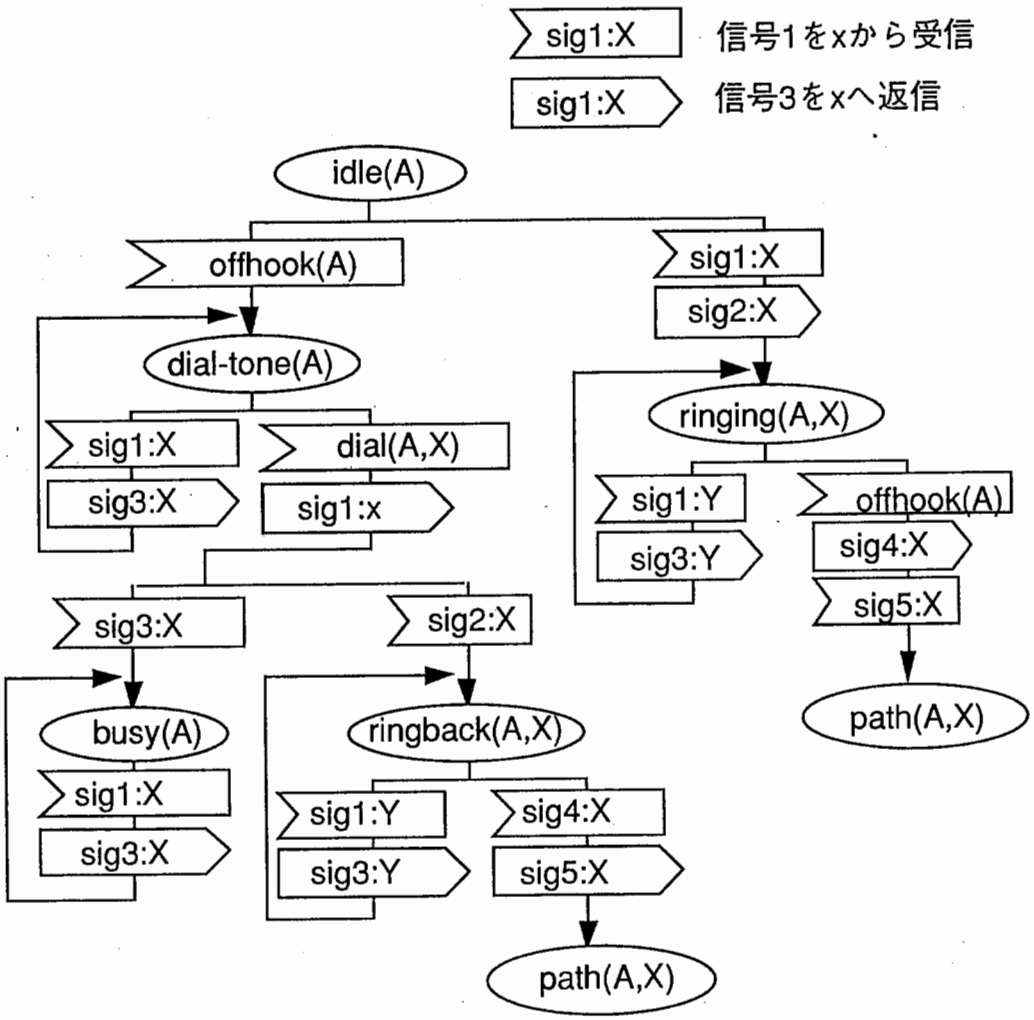


図２　従来仕様記述