〔公　開〕

ＴＲ－Ｃ－０１０１

ドメインモデルを用いた通信サービス仕様生成手法

中　村　光　宏
Mitsuhiro NAKAMURA

１９９４　８．２４

ＡＴＲ通信システム研究所

ＡＴＲテクニカルレポート表紙

〔公　開〕

ドメインモデルを用いた通信サービス仕様生成手法

中村　光宏
Mitsuhiro　Nakamura

１９９４．８．７

ＡＴＲ通信システム研究所

目　次

1. はじめに …… 1

2. ドメイン知識 …… 2

3. 要求の表現方法 …… 5

4. 辞書 …… 6

5. 到達ルート抽出メカニズム …… 7

6. 要求理解としての到達点と今後の課題 …… 10

付録：　適用例

1. はじめに

従来、仕様記述に関する多くの研究では、ユーザが要求を計算機処理可能な形式言語で明確に記述することを前提としている。その場合、形式言語で記述されたユーザ要求から対象システムのソフトウェアを設計する過程を計算機支援により自動化することが課題となる。支援手法として、記述内容に関し一定の論理基準に基づく検証を行ったり、記述した仕様の動作をわかりやすく表示するプロトタイピング手法が提案されている。これに対し、そもそもユーザから要求を抽出することが困難であるという認識に立ち、要求仕様を獲得することを支援する要求理解（Requirements engineering）の重要性が指摘されている。

一方、ソフトウェア工学の分野では、専門家の知識をドメインモデルとして構造化し、それを用いて要求から対象システムのソフトウェアを生成することを支援する試みが研究されている。そのためのドメインモデルとして必要な知識の構造化や記述方法に関するさまざまな試みが報告されている。しかし、一般にモデルができたからといって、即、動くシステムができたことにならないという問題点がある。

我々は非専門家を対象とした通信サービス仕様の作成支援をめざして研究を行っている。我々の手法の基本的なコンセプトは、既存のサービス仕様をドメインモデルに基づいて抽象化して事例ベースに蓄積し、「ある状態を起点として、指定した目的状態に遷移する仕様がほしい」といった部分的な情報で入力された要求に対し、要求された2状態の属性値を一致させる操作の組み合わせを抽出することによって、状態遷移形式の仕様を生成しようというものである。ドメインモデルは、①ドメイン固有の概念を表現し、結果的にモデル空間を規定する属性情報、②状態遷移の抽象表現である属性に対する操作方法、③状態および遷移に関する制約条件、④準正常動作生成のための必須条件の4種類の情報から構成される。非専門家が要求表現で使用する具体的な用語とモデル上の抽象表現との相互変換のために辞書を用意する。図1に我々が検討している仕様生成システムの基本構成を示す。

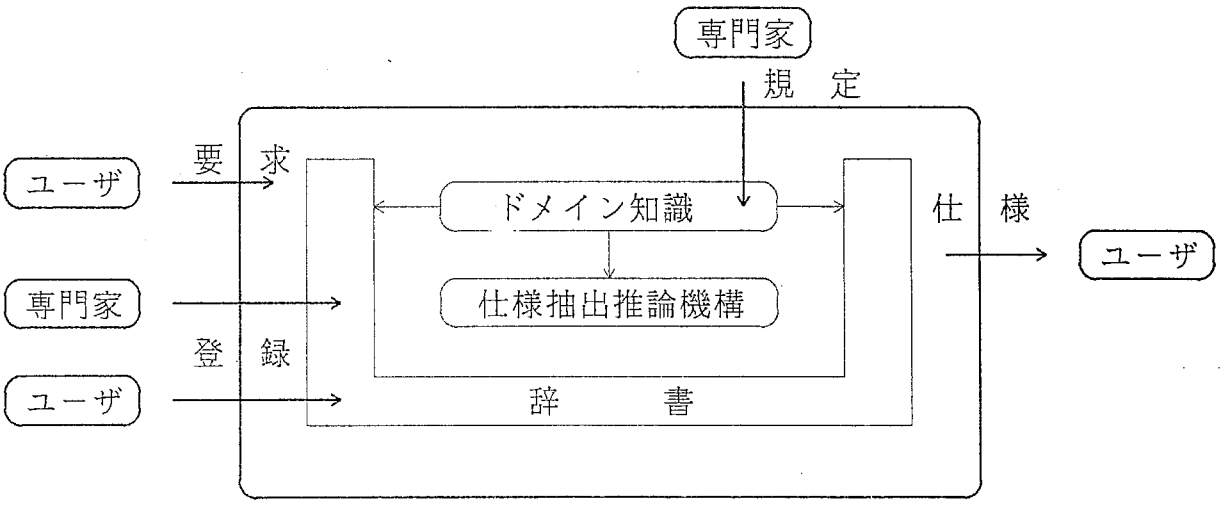


図1　仕様生成システムの構成

2. ドメイン知識

表1に、ドメインモデルにおける知識の種類を示す。

表1　ドメイン知識の種類

種別	内　　容
属性	対象ドメイン固有の属性（概念）、属性間の関係を記述。 何に着目してモデル化するかに依存する。
属性の操作方法	各属性に対する、値の設定／削除方法を記述。 他の属性値から自動的に値を算出するような、属性の関係に関する知識も含む。到達仕様抽出時の推論の単位となる。
制約条件	属性間の関係および属性と操作の組み合わせに関する制約条件を記述。
必須条件	特定の属性において、必ず付加すべき操作法を記述。

2.1　属性

属性を呼の属性と外部属性に2分する。

2.1.1　呼の属性

我々は、通信サービスを”呼び”の状態変化に着目してモデル化した。即ち、発呼により生成され、切断によって消滅する呼のライフサイクルモデルである。通信サービスをどのような回線をどの端末間に設定するかを決定していく過程として考え、”どのような”という部分を属性によって表現する。各属性は(1)に示す形式で規定する。値はその属性がとりうる値の範囲を、初期値は呼が生成された状態で、その属性が特定の値を持つ場合にその値を規定する。回線交換サービスの場合の呼の属性を構成する要素例を表2に示す。

表2　呼の属性規定の例

属性名称	取りうる値の範囲	初期値
接続状態	[通信前／通信中／通信後]	通信前
発アドレス	端末アドレス	―――
着アドレス	端末アドレス	―――
通信速度	[64kb/s／192kb/s／・・・・]	64kb/s
透過性	[透過／非透過]	非透過
遅延特性	ms	―――
セル廃棄率	%	―――
課金手段	[課金ﾒｰﾀ／ｸﾚｼﾞｯﾄｶｰﾄﾞ／ｷｬｯｼｭ]	課金メータ
課金アドレス	[発ｱﾄﾞﾚｽ／着ｱﾄﾞﾚｽ／第3者]	発アドレス

取りうる値の範囲を［・・・・］で規定した属性は、その中のどれかが選択される。

他の属性は、規定情報を単位とする値が設定される。

すべての呼状態は属性値の集合で規定される。逆に、属性値の集合による表現は現実の呼状態の抽象表現だと見なすこともできる。多くの属性は初期値をもたない。これらの属性は呼の発生後、通信中状態に至るまでにユーザ動作によって値が決定していく。属性種別は不変ではなく、ネットワークの進歩や要求の多様化に応じて追加、変更もありうる。表2の例では、アナログネットワークでは接続状態、発アドレス、着アドレスの3種類だった属性が、Ｎ－ＩＳＤＮ(Narrow-band ISDN) で通信速度や透過性が追加になり、さらにＢ－ＩＳＤＮ(Broad-band ISDN) では遅延特性やセル廃棄率が加わる。また、着側課金やクレジットカード課金のような要求の多様化に対応した属性規定もある。以下に記述法と記述例を示す。

記述法

〔%ATTRIBUTE (name), $REQUIRE (Data Type), $DEFAULT (Data)〕 ―――――― (1)

%ATTRIBUTE ：属性の名称を記述
$REQUIRE ：該当属性が取りうる値の範囲を指定
$DEFAULT ：属性の初期値を指定

記述例

%ATTRIBUTE (Path state)
$REQUIRE (Initial/Connected/Disconnected)
(パス状態属性の取りうる値は Initial,Connected または Disconnected)
$DEFAULT (Disconnected)
(パス状態属性の初期値は Disconnected)

2.1.2 外部属性

呼の属性がすべての状態に対応するのに対し、外部属性は選択的である。また、それらの属性は呼状態の変遷とは独立であり、ルール適用の条件として使用されるという特徴がある。外部属性のユーザによる状態変更ルールはユーザ要求とは別に考慮する。呼以外の属性には、さまざまな種類が存在するが、現在、以下の3種類を考慮している。

(1)日時
指定した日時（日、曜日、時刻）に限って要求を実現する。
（時刻なら何時から何時）

(2)グループ関係
指定したグループ関係が成立する場合に限って要求を実現する。

(3)サービス活性化
常時起動でなく、ユーザ動作による活性化(activate)時にのみ要求を実現する。

2.2　属性に対する操作

モデル上での状態遷移の種類を規定するものであり、すべてのイベントは、特定の属性に対する特定の操作にマッピング可能である。属性に対する操作は、属性個別の操作と属性間の関係に関する操作に分類される。

(1)属性個別の操作種別

①＄VALUE

属性値を直接設定。

②＄CLEAR

属性値を削除。対象の属性が初期値を持てばその値を設定し、持たない場合は未設定状態とする。

③＄IF-NEEDED (Data Type)

属性に＄REQUIRE で指定した範囲外のデータが書き込まれると範囲内のデータに変換して設定する。

例：着アドレス属性に短縮アドレスを書き込もうとして＄VALUE (Short address)を実行すると、＄IF-NEEDED (Short address) が動作し、Short address を端末アドレスに変換して着アドレス属性に書き込むという推論が生成される。

(2)属性間の関係に関する操作種別

④＄IF-ADDED (属性名)

パラメータで指定した属性に値が書き込まれると起動する。

例：発アドレスが決まれば、自動的に着アドレスが決まる。（ホットライン）
着アドレス属性に「＄IF-ADDED (Originating address)＋＄VALUE (Data)」を記述する。

⑤＄IF-REMOVED (属性名)

パラメータで指定した属性の値が削除されると起動する。

属性間の関係に関する操作は、属性個別の操作と組み合わせて規定する。

2.3　制約条件

回線属性間には、属性値相互の排他性や属性値と操作種別の間の排他性がある。専門家の視点からみるとあってはならないという意味の知識である。本節では、こうした制約の規定方法について述べる。制約は2タイプに分類できる。

(1)属性値相互の排他性の記述

同時に取りえない属性値の組合せを記述する。

#<Si(Ai), Sj(Aj), ・・・>

例：#＜Connection State(Connection), Transparency (non Transparent),
Capacity(not 64Kb/s)＞
意味： 非透過で通信中の時、通信速度が64Kb/s以外であってはならない。

(2)属性値と操作種別の間の排他性の記述
ある属性が特定の属性値を持つ場合に、禁止すべき操作種別を記述する。

#<Si(Ai), Sj(Aj), ・・・:Sk(操作種別)>

例：# ＜Connection State(Connection):Capacity($VALUE)＞
意味：通信中に通信速度を変更してはならない。

2.4 必須条件

必須条件とは、ユーザの要求動作に付随して、ユーザの意図とかかわりなく必要になる準正常動作のことをいう。制約記述があってはならないという意味の知識の表現なのに対し、必須条件は無くてはならないという意味の知識表現である。必須条件の記述は、付随動作を付加すべき条件と、条件が充足した場合に生成する動作とからなり、以下の形式で記述する。

＆＜条件：属性名，属性値，動作：属性名，操作種別＞

例：＆＜条件：Ａｉ，not(DEFAULT(Ai))，動作：Ａｉ， $CLEAR(Ai) ＞
（Ａｉは発アドレスと着アドレス属性）
意味：任意の状態でユーザの動作に起因する切断動作が必要。

3. 要求の表現方法

２章で述べたように呼びの状態変化に着目してモデル化した場合、要求も状態で表現すると扱いやすい。状態遷移モデルにおけるサービス仕様は(2)式によって表される。開始状態Ｓｓから、いくつかの遷移を経て目的状態Ｓｅに到達する、その過程のユーザ動作や途中状態を含めて規定されるわけである。我々は、要求をサービス仕様の部分情報として表現することにした。部分情報としては図２に示すようなさまざまなパターンが考えられる。現実に、なるだけ多様な要求表現方法が有効であると考えられる。また、外部属性はメニューから選択後、詳細情報を入力させる。

{Ss, B1, S1, B2, S2, ・・・・, Bn, Se} ―――――― (2)
(Biはユーザ動作、Siはサービス状態、特にSsは開始状態、Seは目的状態)

i　開始状態と目的状態のみ規定

ii　開始状態と目的状態に加え、最初のユーザ動作を規定

iii　開始状態と目的状態に加え、最後のユーザ動作を規定

iv　開始状態と目的状態に加え、途中の１状態を規定

図２　要求の表現例

4. 辞書

現実の状態やイベントとモデル上の状態やイベントの相互変換の働きを持つ。また、状態やイベントを具体的な名称でなく抽象化した表現で扱うことで、ユーザとシステムで言葉の意味を共有することが可能になる。状態について、現実の状態は状態プリミティブの集合で、モデルでは属性値の集合で表現する。

例：dial-tone(A)　⇦⇨　〔非接続中，A，＊〕
　　path(A,B)　　⇦⇨　〔接続中，A，B〕

イベントについて、モデルでは属性種別＋操作で表現する。

例：dial(A,B) ⇦⇨着アドレス属性に対する$VALUE(B)
　　offhook(A)⇦⇨発アドレス属性に対する$VALUE(A)

ユーザが、新規の状態で要求を入力した場合、対応する用語が辞書に無いという問題がある。この場合、システムはユーザに対し全ての属性値を問い合わせて対応するモデル上の状態を決定し、自動的に辞書に登録する。また、逆に、モデル上の状態やイベントに対応する現実の状態やイベントが１種類も登録されていない場合のユーザへの問い合わせ方法は今後の課題である。

5. 到達ルート抽出メカニズム

3章で述べた形式で表現された要求は、4章で述べた辞書によって、ドメインモデルにマッピングされる。本章では、図2のパターンiの形式の要求で入力された2状態間の到達ルートを抽出する推論メカニズムについて述べる。これによって、前状態＋イベント⇨次状態の状態遷移形式の仕様を抽出する。全体の処理フローを図3に示す。

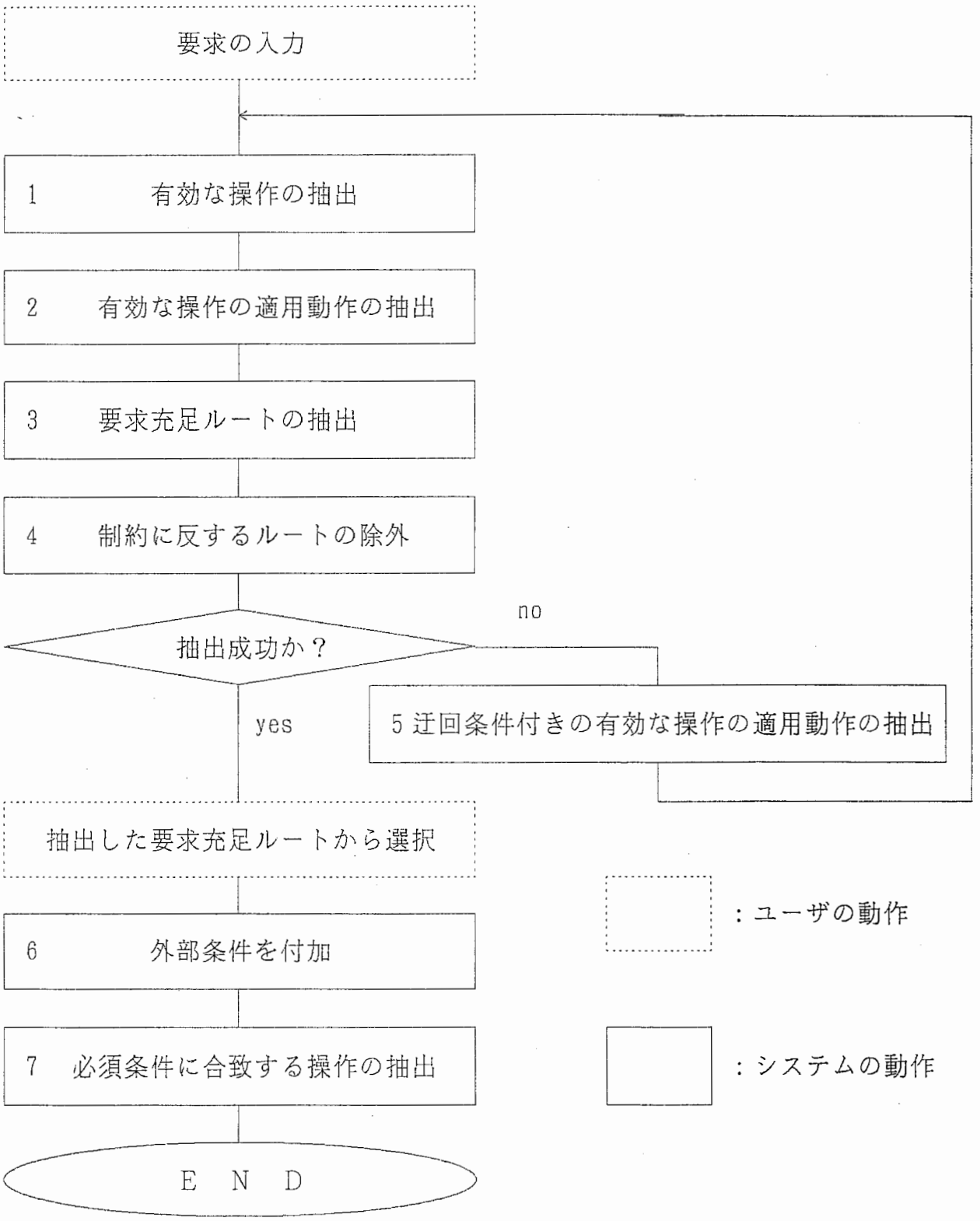


図3 仕様抽出の全体フロー

5.1　有効な操作の抽出

Ｓｓ：〔s1, s2, ･･･, sn〕とＳｔ：〔t1, t2, ･･･, tn〕の各属性値を比較。異なる値を持つ属性の個数を「状態間の距離」と定義する。１回の操作で状態間の距離が属性個別の操作を用いる場合に１、属性の関係に関する操作を用いる場合に２、それぞれ縮まる操作を選択する。状態間の距離＝３で値の異なる属性をAi, Aj, Ak、それらの値をsi, sj, sk、ti, tj, tkとする。この時、①の操作はＳｓ、Ｓｔ間の距離を１、②の操作は距離を２縮める。

$VALUE(Ai)

①　　ｓｉ ――――→ ｔｉ

（ｓｉを持つ属性Ａｉに対しｔｉを書き込む）

$VALUE(Ai)　$IF-ADDED (Aj, Ai) $VALUE(Aj)

②　ｓｉｓｊ ――――――→ ｔｉｔｊ

（ｓｉを持つ属性Ａｉに対しｔｉを書き込み、それが契機となって属性Ａｊに対しｔｊを書き込む）

5.2　有効な操作の適用動作の抽出

5.1で抽出した操作が実際に適用される状態と適用後の状態を求める。

ｉ）開始状態Ｓｓ：[s1, s2, ･･･sn] において適用可能な操作を5.1で抽出した操作から選択する。

ii）適用可能な操作があれば、前後の状態とともに操作適用動作テーブルに登録する。

iii）操作適用後の状態が新規状態であれば状態テーブルに登録する。

iv）状態テーブルに新規状態が登録されなくなるまでｉ）～iii）を繰り返す。

5.3　要求充足ルートの抽出

5.2で抽出した操作の適用動作を用いて、ＳｓからＳｔに至るルートを求める。

ｉ）開始状態Ｓｓを初期状態として探索を行う。

ii）自己の状態を前状態とする操作適用状態から１個選択し、次状態に操作適用状態idとともに探索を依頼する。全遷移を探索終了すれば、依頼元の状態に戻す。

iii）終状態Ｓｔに到達すれば、遷移系列を要求充足ルートテーブルに登録し、前状態に戻る。

iv）開始状態Ｓｓにおいて、探索が終了するまでii）とiii）を繰り返す。

5.4　制約に反するルートの除外

属性値相互の排他性は状態の制約、属性値と操作の間の排他性はイベントの制約に当たる。5.3で抽出した要求到達ルートのうち、制約に反する有効な操作の適用動作を持つルートを除外する。

5.5　迂回ルートの抽出

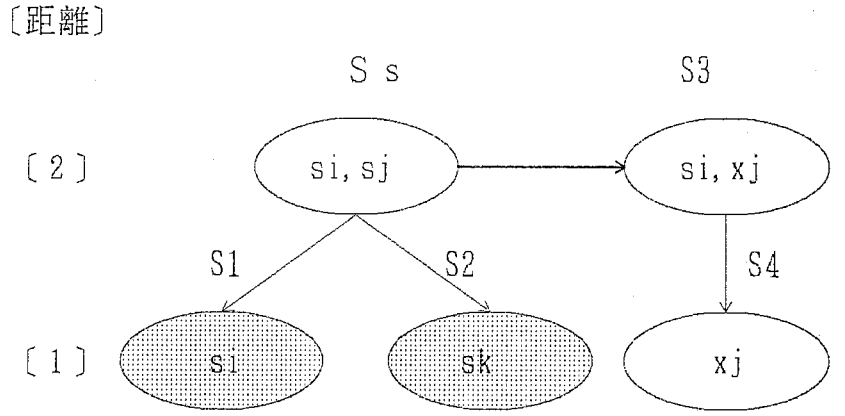
5.4の制約に反するルートの除外によって、1ルートも抽出できない場合に、迂回ルートを抽出する。遷移先が存在しない場合、①②の順に迂回ルートを抽出する。

（いずれも失敗した場合は、到達ルート無しと判断する。）

図4、図5にそれぞれの例を示す。

①横滑り（目的状態と差異のある値を持つ属性が制約の原因になっている場合で、かつ、その属性値を変更することで、制約を回避できる場合に適用）

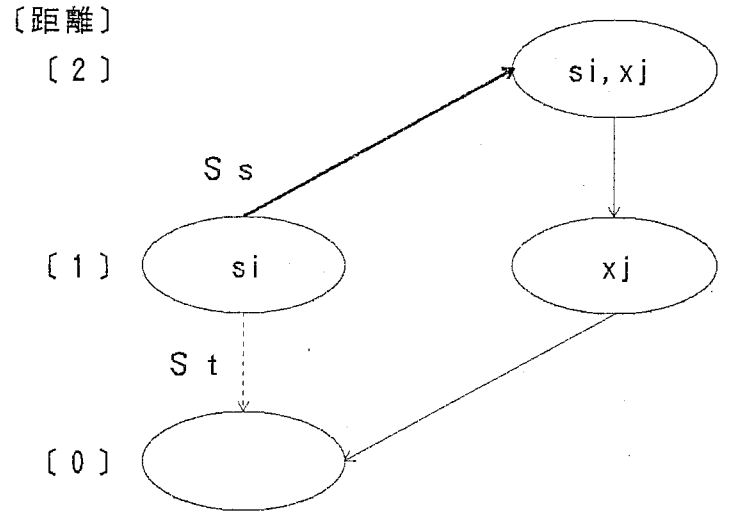
②遠回り（目的状態と差異のない値を持つ属性が制約の原因になっている場合で、かつ、その属性値を変更することで、制約を回避できる場合に適用）



（楕円内は目的状態と異なる値を持つ属性の属性値）

Ｓ１とＳ２が制約違反状態のケースで属性ｊを書き換えて迂回している。

図4　横滑り迂回の例



（楕円内は目的状態と異なる値を持つ属性の属性値）

Ｓｓ⇨Ｓｔが制約違反遷移のケースで属性ｊを書き換えて迂回している。

図5　遠回り迂回の例

5.6　外部条件を付加

要求入力時にユーザが選択、指定した外部条件について、これまでに生成した全ルールの前状態に付加する。

5.7　必須条件に合致する操作の抽出

ユーザが選択した要求充足ルートの各状態と、必須条件の条件部を比較し、条件が合致する場合に、動作部を適用した、有効な操作の適用動作を作成する。

6.　要求理解としての到達点と今後の課題

本稿では、既存サービス仕様を抽象化したドメインモデルを用いて、新たな要求に対応する仕様を自動的に生成する手法について述べている。この手法で生成できない要求を原因で分類すると以下の3種類になる。

①属性が不足
②属性操作が不足
③属性として表現できない要求

上記のうち、①②は既存の仕様を抽象化してモデルを構築する限り、既存の仕様とかけ離れた要求への対応が困難となるのはやむをえない。

属性の不足は、要求のモデルへのマッピングができないという現象として表れ、その時点で属性の追加規定が必要である。要求からモデルへのマッピングを辞書を利用して行う場合、属性追加が辞書の全面的な修正につながる恐れが大きく、この点がネックとなる。その他の知識では、制約条件と必須条件の追加が伴うが、これは比較的、容易に実現できると考えられる。

一方、属性操作の不足は、候補として得られる仕様の中に満足する仕様が無いという現象となる。これに対しては、属性操作は5種類と限定されているため、満足する仕様が得られなかった時点で、全属性に5種類の属性操作を付加してルール抽出を行うことで、可能な全ての仕様を得ることができる。ただし、この方法では、開始状態と目的状態の距離が大きい要求に対して、候補となる仕様の数が大きくなる問題がある。

属性として表現できない要求の例としては、暗証番号の認証動作等がある。接続条件の決定過程として構築したモデルの属性として認証を扱うのはかなり無理があり、このような要求に対応する仕様の生成は現状では不可能である。現実の通信サービス仕様は、セキュリティ、輻輳時の規制、再開時の呼救済のような多様な視点からの独立した要求を統合して作成されている。こうした仕様の生成のためには、それぞれの立場で独立に構築した複数のモデルを自動的に統合、合成する技術が必要である。

現状の技術の評価として、非専門家をユーザとする場合、前記の3つのケース以外の場合、即ち、既存のモデルで対応可能な場合に限定すれば効果はあると考えられる。適用範囲の拡大のために、本稿の5章で述べた到達ルート抽出方法を拡張し、複数呼間の制約の記述や到達ルート抽出を実現することが課題となる。

一方、ドメインモデルの応用面では、要求の合成、即ちフィーチャーインタラクショ

ンの問題への適用が考えられる。この分野では、望ましくない現象を定義し、その状態を検出する手法が一般的であるが、ドメインモデルから生成した知識を用いることにより、望ましくない現象の発生する原因にさかのぼった解析が可能であり、より効率的な検出、解消が行える可能性がある。

現実の仕様を抽象化して構築するモデルは、抽象度が高くなればなるほど、適用範囲は大きくなるが、適用時に得られる情報が減少してしまう。これは、抽象化に伴って、モデルから個々の具体的な情報が消えてしまうことによる。この際、消えた情報の存在とその内容の提示をどのように行うかは、モデル推論の一般的かつ重要な問題であり、今後の課題である。

付録 ： 適用例

本稿で述べた手法でサービス仕様を生成する事例を3種類紹介する。
（必須条件知識による、要求付随動作の生成は省略）

以下のドメイン知識および辞書を前提とする。

属性
① A1:path-state =[initial, connected, disconnected]
② A2:originating-address = terminal address(T)
③ A3:terminating-address = terminal address(T)

属性操作
① A1:path-state = $VALUE
② A2:originating-address = $VALUE, $CLEAR
③ A3:terminating-address = $VALUE, $CLEAR, $IF-NEEDED,
$IF-ADDED(originating-address)
$IF-ADDED(terminating-address)

制約
① #<path-state(connected), $VALUE(originating-address)>
② #<path-state(connected), $VALUE(terminating-address)>
③ #<path-state(connected), originating-address(*))>
④ #<path-state(connected), terminating-address(*))>

辞書

①	path(A, B)	A1:connected, A2:A, A3:B
②	hold(A, B)	A1:disconnected, A2:A, A3:B
③	Calling(A, B)	A1:initial, A2:A, A3:B
④	dial-tone(A)	A1:initial, A2:A, A3:*
⑤	idle(*)	A1:initial, A2:*, A3:*
⑥	busy-tone(A)	A1:disconnected, A2:A, A3:*
⑦	dial(A, B)	P1:originating-address, P2:terminating-address, $VALUE(terminating-address), B
⑧	offhook(A)	P1:originating-address, $VALUE(originating-address), A
⑨	offhook(B)	P1:terminating-address, $VALUE(path-state), connected
⑩	flash(A)	P1:originating-address, $VALUE(path-state), disconnected
⑪	onhook(A)	P1:originating-address, $CLEAR(originating-address)
⑫	onhook(A)	P1:terminating-address, $CLEAR(terminating-address)
⑬	dial(A, S)	P1:originating-address, P2:short-address, $VALUE(terminating-address)

事例〔1〕

要求：dial-tone(A)⇨ path(A, B)　　辞書①④によってモデルに変換

モデル：[initial, A, *] ⇨ [connected, A, B]

前後の状態の属性値を比較すると、パス状態属性と着アドレス属性の値が異なる。
パス状態属性の操作には$VALUE しかないので、(1)の操作が抽出される。

$$\text{initial} \xrightarrow{\$\text{VALUE}} \text{connected} \quad \text{——————} \ (1)$$

着アドレス属性の操作は③によって、以下の4種類。
$VALUE, $CLEAR, $IF-NEEDED, $IF-ADDED(originating-address)
次状態での着アドレス属性の値は、初期値でないため、$CLEAR は対象外。
また、$IF-ADDEDの対象属性(originating-address) に差異が無いため、これも対象外。
よって、$VALUE と $IF-NEEDED が残り、(2X3)の操作が抽出される。

$$B \xrightarrow{\$\text{VALUE}} C \quad \text{——————} \ (2)$$

$$B \xrightarrow{\$\text{IF-NEEDED}} C \quad \text{——————} \ (3)$$

操作の適用場面として、(4)〜(9)が抽出される。

$$[\text{initial}, A, *] \xrightarrow{\$\text{VALUE}(A1)} [\text{connected}, A, *] \quad \text{——————} \ (4) \quad \times$$

$$[\text{initial}, A, *] \xrightarrow{\$\text{VALUE}(A3)} [\text{initial}, A, B] \quad \text{——————} \ (5)$$

$$[\text{initial}, A, *] \xrightarrow{\$\text{IF-NEEDED}(A3)} [\text{initial}, A, B] \quad \text{——————} \ (6)$$

$$[\text{initial}, A, B] \xrightarrow{\$\text{VALUE}(A1)} [\text{connected}, A, B] \quad \text{——————} \ (7)$$

$$[\text{connected}, A, *] \xrightarrow{\$\text{VALUE}(A3)} [\text{connected}, A, B] \quad \text{——————} \ (8) \quad \times$$

$$[\text{connected}, A, *] \xrightarrow{\$\text{IF-NEEDED}(A3)} [\text{connected}, A, B] \quad \text{——————} \ (9) \quad \times$$

これらのうち、(4X8X9)は制約④に違反しているため没。
到達ルートとして、(10X11)が抽出される。

$$[\text{initial}, A, *] \xrightarrow{\$\text{VALUE}(A3)} [\text{initial}, A, B] \xrightarrow{\$\text{VALUE}(A1)} [\text{connected}, A, B] \quad \text{————} \ (10)$$

$IF-NEEDED(A1)　　$VALUE(A1)
[initial, A, *] → [initial, A, B] → [connected, A, B] ———— (11)

辞書によって(10)から(12)を、(11)から(13)を得る。

dial-tone(A)　　dial(A, B):　　Calling(A, B).
Calling(A, B)　　offhook(B):　　path(A, B).　　——— (12)

dial-tone(A)　　dial(A, S):　　Calling(A, B).
Calling(A, B)　　offhook(B):　　path(A, B).　　——— (13)

事例〔2〕

要求：dial-tone(A)⇨ path(A, C)　　最初のイベントが　dial(A, B)

辞書①④⑦によってモデルに変換

モデル：[initial, A, *] ⇨ [connected, A, C]
イベントは　$VALUE(terminating-address), B

$VALUE(A3), B
開始状態に最初のイベントを適用すると、[initial, A, *] → [initial, A, B]

[initial, A, B] ⇨ [connected, A, C] のルート抽出問題になる。

また、$IF-ADDED(terminating-address) を持つ属性を検索するとterminating-address が該当する。

$VALUE(A3), B 適用後の状態　[initial, A, B]に対し、A3に対する操作で、距離を縮められるものを抽出して下記を得る。

$VALUE(A3), B: $IF-ADDED(A3, A2): $VALUE(A3), C
[initial, A, *] → [initial, A, C]

その後は、　[initial, A, C]⇨[connected, A, C] のルート抽出問題になる。

事例〔3〕

要求：path(A, B) ⇨ path(A, C)　　辞書①によってモデルに変換

モデル：[connected, A, B] ⇨ [connected, A, C]

前後の状態の属性値を比較すると、着アドレス属性の値のみが異なる。（B→C）
着アドレス属性の操作は③によって、以下の4種類。
$VALUE, $CLEAR, $IF-NEEDED, $IF-ADDED(originating-address)
次状態での着アドレス属性の値は、初期値でないため、$CLEAR は対象外。
また、$IF-ADDEDの対象属性(originating-address) に差異が無いため、これも対象外。
よって、$VALUE と $IF-NEEDED が残り、(1X2)の操作が抽出される。

$VALUE
B ——————→ C　　—————— (1)

$IF-NEEDED
B ——————→ C　　—————— (2)

(1X2)の操作の適用場面として(3X4)が抽出される。

$VALUE(terminating-address), C
[connected, A, B] ⇨ [connected, A, C]　　—————— (3)

$IF-NEEDED(terminating-address), C
[connected, A, B] ⇨ [connected, A, C]　　—————— (4)

(3X4)はそのまま、要求充足仕様となっている。
ところが、(3X4)は制約②に違反するため没。$IF-NEEDED による書き込みも$VALUE と同じとみなす。
そこで、迂回アルゴリズムを実行する。
制約条件をながめると、path-stateの値(connected) がネックになっているのがわかる。
path-stateに、他の値(disconnected, initial)を書き込むことを考える。

[connected, A, B] ⇨ [disconnected, A, B]　　—————— (5)
[connected, A, B] ⇨ [initial, A, B]　　—————— (6)

辞書でイベントを検索した結果、⑩から

A
[connected, A, B] ⇨ [disconnected, A, B]　　—————— (7) が可能とわかる。

(path-stateに対する$CLEAR は属性操作に無いため(6)の可能性は除去される)

[disconnected, A, B] ⇨ [connected, A, C] への到達ルートを抽出する。

前後の状態の属性値を比較すると、パス状態と着アドレス属性の値が異なる。
(8X9X10)の操作が抽出される。

$VALUE(path-state)
disconnected ——————→ connected　　—————— (8)

$VALUE(terminating-address)
B ——————→ C　　—————— (9)

$IF-NEEDED(terminating-address)

B ⟶ C ——— (10)

これらの適用場面として、以下が抽出される。

$VALUE(path-state)

[disconnected, A, B] ⟶ [connected, A, B] ——— (11)

$VALUE(terminating-address)

[disconnected, A, B] ⟶ [disconnected, A, C] ——— (12)

$IF-NEEDED(terminating-address)

[disconnected, A, B] ⟶ [disconnected, A, C] ——— (13)

$VALUE(terminating-address)

[connected, A, B] ⟶ [connected, A, C] ——— (14) ×

$IF-NEEDED(terminating-address)

[connected, A, B] ⟶ [connected, A, C] ——— (15) ×

$VALUE(path-state)

[disconnected, A, C] ⟶ [connected, A, C] ——— (16)

(14)(15)は制約条件違反で没。
要求充足ルートとして、(17)(18)が抽出される。

$VALUE(path-state), disconnected　　$VALUE(terminating-address), C

[connected, A, B] ⇨ [disconnected, A, B] ⇨ [disconnected, A, C]

$VALUE(path-state), connected

⇨ [connected, A, C] ——— (17)

$VALUE(path-state), disconnected　　$IF-NEEDED(terminating-address), C

[connected, A, B] ⇨ [disconnected, A, B] ⇨ [disconnected, A, C]

$VALUE(path-state), connected

⇨ [connected, A, C] ——— (18)

辞書によって(17)から(19)を、(18)から(20)を得る。

```
path(A, B)        flash(A):     hold(A, B).
hold(A, B)        dial(A, C):   Calling(A, C).   ——— (19)
Calling(A, B)     offhook(B):   path(A, B).

path(A, B)        flash(A):     hold(A, B).
hold(A, B)        dial(A, S):   Calling(A, C).   ——— (20)
Calling(A, B)     offhook(B):   path(A, B).
```