第4章　音声対話インタフェース汎用プラットフォーム

ここでは、音声対話インタフェース汎用プラットフォームの実装について述べる。まず、音声対話インタフェースの汎用化の際の問題点について述べた上で、音声対話インタフェース汎用プラットフォームの構成について説明する。

4.1　音声対話I/F汎用化の問題点

一般的に音声対話インタフェースを構築する際、汎用性を意識して音声認識や理解のための辞書や文法を外在化することは多いが、問題解決規則や対話制御規則については、特定のタスクやアプリケーションごとにそれぞれ個別に設計され、知識のモジュール化についてあまり注意が払われていない。これは、問題解決や対話制御の規則を抽象化することは実際には難しいこと、またそれらの一般的な表現形式として、フレームやプロダクションルールが存在するものの、記述が容易ではないことによると考えられる。そのため、他のタスクでの対話制御に関する知識の再利用が難しく、新しいシステムを構築する際には対話モジュールを最初から設計し直さねばならず、システム開発において大きな負担のかかる要因となっている。

これまでにもこうした問題意識から、音声対話システムの汎用化についていくつかの機関で研究されている。秋葉ら [12] は,対話に関する処理を独立に記述するマルチモーダル対話言語を設計した。この記述言語は、より抽象度の低いレベルでの対話のモデル化を目的とし、割り込みなどを含む対話中の任意の状態に対して、局所的なシステムの挙動を細かく指定することができるという特徴をもつ。田中ら [13] はタスクを情報検索に限定し、データベースからの情報をもとに文法と語彙の設定を半自動的に行う、汎用的な音声対話プラットフォームを構築している。このプラットフォームでは、GUI を採用することで

ユーザ発話を誘導し、限定された発話・対話パターンで音声対話を実現することで、より容易に対話を記述できるという特徴をもつ。また、荒木ら [14] は音声対話システムの対話管理部を作成する方法として、一般的なタスクドメインを情報の流れる方向に着目して分類し、適用できる対話ライブラリを構築し、汎用的な対話ライブラリを組み合わせるといったアプローチを提案している。

問題解決規則や対話制御規則などの知識を外在化して実装しようとすると、知識の記述の容易性と多様性のトレードオフが問題となる。つまり、多様な対話制御をシステムの外部で記述しようとすると、扱うパラメータが増えアルゴリズムが複雑になってしまう。また、外在化した知識の記述を容易にするために扱うパラメータを減らすと、多様な記述が不可能となってしまう。

本研究で提案する音声対話インタフェース汎用プラットフォームでは、システム行動の多様性のための対話記述の自由度と記述容易性のトレードオフを考慮して、システムの行動レベルの記述の上に、問題解決レベルとして対話を一段抽象化することによって対話の多様性と記述の容易性を実現する。具体的には、上位レベルとなるシステムの問題解決戦略を記述するプランニングルールをプロダクションルールで表現し、後ろ向き推論によって抽象的な行動プランを決定する。また、行動レベルの記述、すなわち実際に行う発話や、アプリケーションコマンドを決定する行動決定ルールは、行動プランごとに用意される。行動決定ルールには複数の心的状態を独立に扱う心的状態モデルを導入し、各心的状態と対話の進行に関しての対話の基本状態をそれぞれ独立に記述する。対話の基本状態は状態遷移図によって表現し、各心的状態に関してはそれぞれ状態の更新のタイミングやトリガとなるイベントを記述する。その際に、各記述の内容に特化した最小限のライブラリを用意するだけでシステムが動作できるように、最大限に状態を抽象化する。これにより、問題解決及びその手段としての対話の多様性と対話記述の容易性を実現する。さらに、それぞれの心的状態を独立に記述することにより、各心的状態モデルの再利用が可能となる。

また、システムを構成するモジュールを並列に動作させて、多様な対話制御方法に対応し、ユーザとシステムの自由なコミュニケーション形態を実現する。

4.2 プラットフォームの構成

本プラットフォームの構成を図 4.1 に示す。この図に示すように、音声認識部、システム制御部、行動管理部、応答生成部とタスクに応じたデータベースなどの具体的なアプリケーション部から構成される。

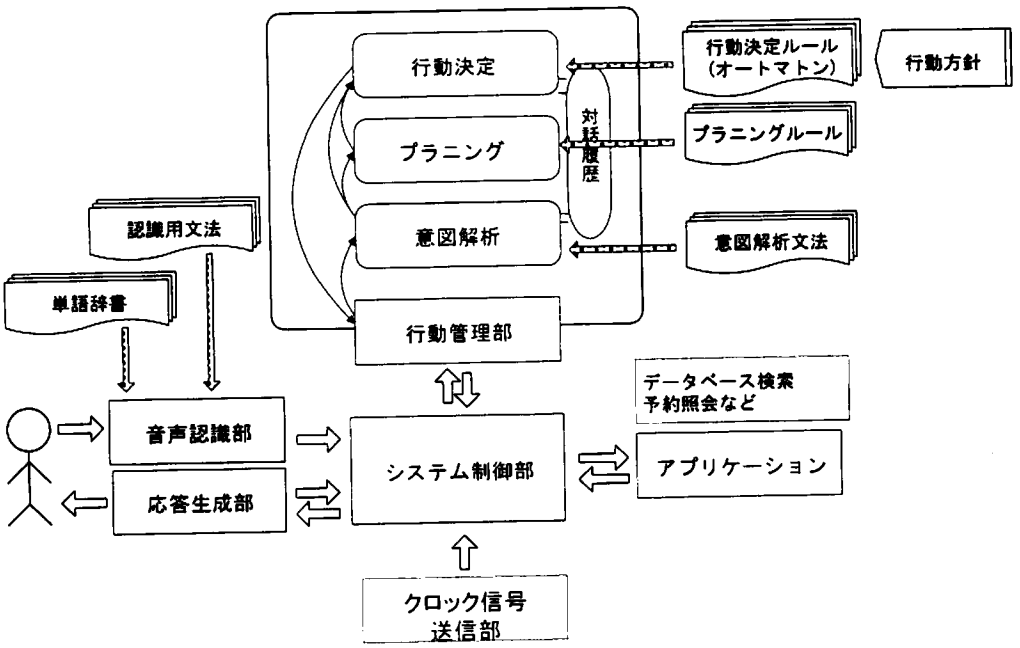


図 4.1: プラットフォームの構成

システムは、各部が並列に動作し、その間でメッセージ通信を行なって状況に応じた適切な処理を行なうマルチプロセス形式で構築されるが、実質的にはメッセージの全てがシステム制御部を介してやりとりされるために、実質的にはシステム制御部を中心とした集中制御となる。

典型的な処理の流れを図 4.2 を用いて説明する。

音声分析部 (Speech Analyzer) によってユーザ音声 (User Speech) の音声区間が検出されると、その旨がシステム制御部 (System Controller) に伝えられ、システム制御部は音

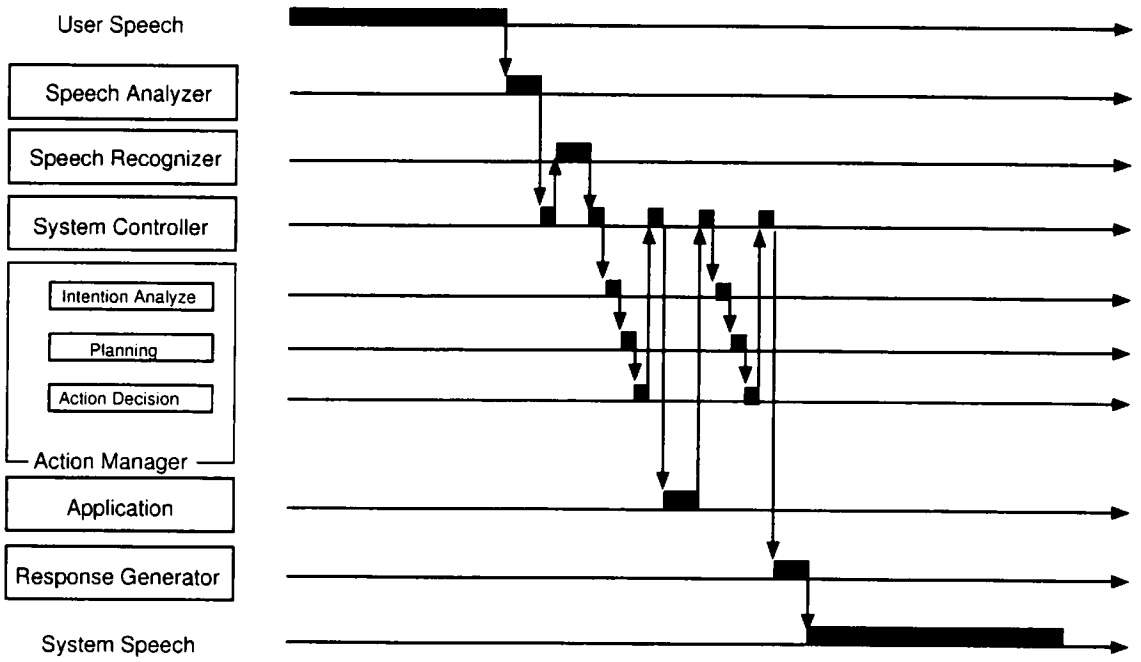


図 4.2: システムの典型的な処理の流れ

声認識部 (Speech Recognizer) に音声認識処理を行なう様、要求する。音声認識処理の結果は、同様にシステム制御部を通じて行動管理部 (Action Manager) に伝えられ、意図解析 (Intention Analyze)、プラニング (Planning)、行動決定 (Action Decision) の順に処理が行なわれる。その結果、システムがとるべき行動が決定され、システム制御部に伝えられる。図 4.2 においては、アプリケーション (Application) の結果を得た上で、応答生成部 (Response Generator) においてシステム発話 (System Speech) を生成、出力している。

以下に各部について説明する。

音声認識部

入力されたユーザ発話の区間検出を行い、検出されたユーザ音声区間の音声について、単語辞書、文法を用いて認識結果を単語列で出力する。音声認識エンジンは 3 章での比較実験で用いた Julian[38] を使用した。Julian に与えた有限状態文法も比較実験同様、対話の書き起こしデータが文法によって認識されるよう作成したものを用いた。

システム制御部

各構成部からメッセージを受け取り、メッセージタイプに応じて送信席を決定し、メッ

セージの送信を行う。メッセージ処理規則の汎用化は今回は行っていない。

行動管理部

システム制御部から受け取った認識結果や、アプリケーションから受け取った結果をもとにシステム設計者の記述した外部知識から、システムの動作を決定する。詳しくは次節で述べる。

音声合成部

行動管理部で生成されたシステム発話を合成音声により出力する。合成音声は2章での実験で使用したIBM Protalker97[39]を使用した。

アプリケーション

システム制御部から受け取ったメッセージとパラメータをもとに、アプリケーションを実行する。具体的にはタスクにより、データベース検索や、予約照会などを行う。

本研究で設計、構築する音声対話インタフェース汎用プラットフォームは、スロットフィリング型タスクを基盤とした、チケット予約や、データベース検索、会議室の予約など様々なタスクに対応できるものである。人生相談や、旅行相談など、ユーザによってスロットの構造が変化し多様化するタスクについては取り扱わない。システム開発者はプランニングルール、行動決定ルールを記述することにより、対話制御方法を決定するのみで対話制御方法を多様にかつ簡単に変化させることのできるプラットフォームを目指している。

4.3 各部の処理の詳細

4.3.1 システム制御部

システム制御部の処理内容は図4.3に示す様に、メッセージ処理ルール(Message Handling Rule)に従って、入力されたクロック信号*か、他部からのメッセージのタイプに応じて、出力するメッセージとその出力先を決定する。

*UNIX版の場合はクロッククライアントが送信し、Windows版の場合にはシステム制御部内でタイマーイベントが出力する。

メッセージ処理ルールは、現在のところ、表 4.1 に示すような単純な入力メッセージー行動（またはメッセージ）の対応で表現したものを扱い、一般化は行なわない。このメッセージ処理ルールに従うメッセージの流れを図 4.3 に示す。

表 4.1: 他部からのメッセージと制御の対応

メッセージ名	送信元	送信先	行動または（出力メッセージ）
SttofUtt	音声分析部	送信せず	U_UTT_FLAG=ON
EndofUtt	音声分析部	音声認識部	U_UTT_FLAG=OFF、ポーズ長計測開始
RecogRes	音声認識部	行動管理部	
StartApp	行動管理部	アプリケーション	
StartRes	行動管理部	応答生成部	
AppRes	アプリケーション	行動管理部	
ClockSig	信号送信部	送信せず	ポーズ長判定
		行動管理部	しきい値を超えたら PauseOver
Event	行動管理部	行動管理部	
Library	行動管理部	行動管理部	
ActRes	行動管理部	行動管理部	

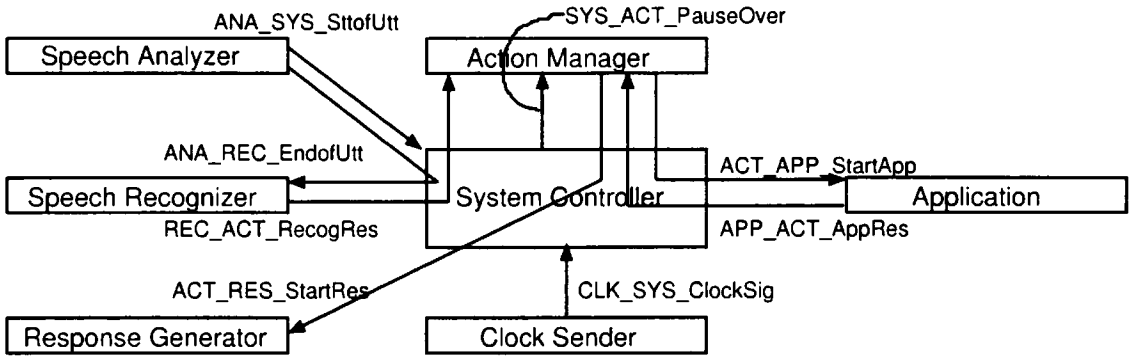


図 4.3: メッセージフロー

表 4.1 および図 4.3 に示したメッセージの書式を以下に示す。

ANA_SYS_SttofUtt

意　味 音声区間始端の検出

書　式 **ANA_SYS_SttofUtt** *utterance-ID*

引　数 int *utterance-ID* 発話ＩＤ。起動時に初期化し、発話毎にインクリメントされる。整数型。

ANA_REC_EndofUtt

意　味 音声区間終端の検出および音声認識処理開始

書　式 **ANA_REC_EndofUtt** *utterance-ID recognition_results*

引　数 int *utterance-ID* 発話ＩＤ。整数型。

char **speech file name* 音声データを格納してあるファイルのパス。文字列型。

REC_ACT_RecogRes

意　味 音声認識処理結果の取得

書　式 **REC_ACT_RecogRes** *utterance-ID recognition_results*

引　数 int *utterance-ID* 発話ＩＤ。整数型。

char ***recognition_results* 音声認識結果としての単語列。[単語 1, 単語 2, 単語 3,...] のように記述する。

ACT_APP_StartApp

意　味 アプリケーションコマンド実行

書　式 **ACT_APP_StartApp** *utterance-ID command_type command_parameter*

引　数 int *utterance-ID* 発話ＩＤ。整数型。

int *command_type* アプリケーションコマンドのタイプ。整数型。

char ***command_parameter* アプリケーションコマンドのパラメータ。アプリケーションコマンドのタイプに依存する。[引数 1, 引数 2, 引数 3,...] のように記述する。文字列の配列。

ACT_RES_StartRes

意　味 応答生成処理開始

書　式 **ACT_RES_StartRes** *utterance-ID response_type response_parameter*

引　数 int *utterance-ID* 発話ＩＤ。整数型。

int *response_type* システムの発話タイプ。整数型。

char ***response_parameter* システム応答のパラメータ。システムの発話タイプに依存する。[引数 1, 引数 2, 引数 3,...] のように記述する。文字列の配列。

APP_ACT_AppRes

意　味 アプリケーションコマンド実行結果の取得

書　式 **APP_ACT_AppRes** *utterance-ID command_type applicatoin_result_parameter*

引　数 int *utterance-ID* 発話ＩＤ。整数型。

int *command_type* アプリケーションコマンドのタイプ。整数型。

char ***application_result_parameter* アプリケーションコマンド実行結果のパラメータ。アプリケーションコマンドのタイプに依存する。[引数 1, 引数 2, 引数 3,...] のように記述する。文字列の配列。

CLK_SYS_ClockSig

意　味 クロック信号

書　式 **CLK_SYS_ClockSig**

引　数

SYS_ACT_PauseOver

意　味 ポーズ長超過

書　式 SYS_ACT_PauseOver *utterance_id*

引　数 int *utterance-ID* 発話ＩＤ。ポーズ長を計測し始めた時の発話ＩＤ。整数型。

4.3.2 行動管理部

ここでは行動管理部の処理フロー、各種ルール、サブルーチンの仕様などについて詳細に示す。

(1) 処理フロー

行動管理の処理フローを図 4.4 に示す。

この図に示す様に、行動管理部はまず起動時に初期化としてプラニングルール（後述）と行動決定ルール（後述）とを読み込む。通常時は、システム制御部からメッセージを受けとる毎に、意図解析以降の処理が開始される。

まず、メッセージが音声認識処理結果の取得を意味する REC_ACT_RecogRes の場合、意図解析文法（後述）を用いて、REC_ACT_RecogRes のパラメータである音声認識結果から意図解析を行なう。さらに、その結果に従ってプラニングの必要性を判定する。基本的には、あいづちを除く全てのユーザ発話意図、アプリケーションコマンド実行結果 (APP_ACT_AppRes) に対してプラニングを行なう。なお、入力メッセージが APP_ACT_AppRes の場合には意図解析を経ずにプラニングの必要性を判定し、ユーザ発話後のポーズ長超過を意味する SYS_ACT_PauseOver の場合には直接行動決定に進む。

プラニングを行なう場合は、初期化の際に読み込んだプラニングルールを用いて後向き推論を行ない、状況に応じて問題解決を進める。後向き推論のアルゴリズムとしては、まず始めに意図解析の結果から目標を設定し†、その目標を達成するための条件を探すために、設定された目標を結論部に持つルールを検索する。適合するルールを検索した後そのルールの前提部の条件を評価する。その条件が満たされればその目標は達成されたこと

†目標はメニュー選択により設定できるものとする。

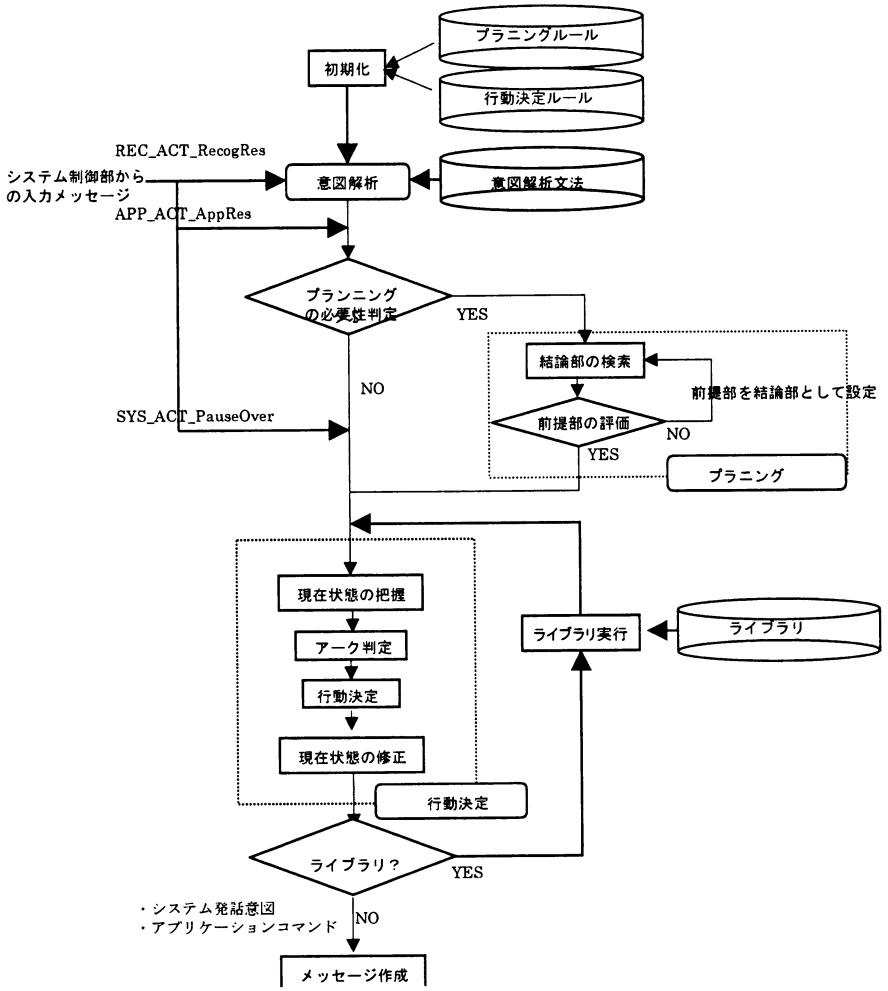


図 4.4: 行動管理の処理フロー

になる。その条件が満たされていなければ、その条件を満たすべく、条件を結論部に持つルールを検索した後そのルールの前提部の条件を評価する。これを繰り返して、最終的に目標が達成されるまで後向きにルールを検索していく。

その過程で、問題解決の目標を達成するために行動プランの実行が必要な場合には、その行動プランを選択して行動決定処理に進む。なお、上述したプラニング必要性の判定で、プラニングの必要性がないと判定された場合には、直接、行動決定処理に進む。

行動決定処理では、初期化の際に読み込んだ行動決定ルールに基づいて、基本的にシステムの行動を状態、ユーザの行動をアークとした状態遷移制御を行なう。ただしアークに

はシステム内部のアプリケーションコマンドの実行結果などのイベントも含む。状態遷移制御のアルゴリズムとしては、図 4.4 に示した様に、現在の状態を把握した上で、システム制御部から、あるいは意図解析処理から得られたシステムあるいはユーザの行動が、現在の状態から継るアークに示されているか否かを調べる。該当するアークがあれば、現在の状態をその遷移先に修正し、そこに記述されている行動を次にシステムがとるべき行動として決定する。システムがとるべき行動としては、システムからの応答生成か、アプリケーションコマンドの実行か、ライブラリ（後述）の実行がある。始めの２つに関しては、決定された行動としてメッセージを作成してシステム制御部に送るのみであるが、ライブラリの実行に関しては、ライブラリを実行した上で、その行動を現在の状態としライブラリの実行結果をアークとして行動決定処理に再度進む必要がある。その際、ライブラリの種類によってはシステム設計者によるスロット記述（後述）を参照して処理を行なう。

行動決定処理において最終的に行動プランが終了した場合、次のプラニング処理で、問題解決の目標を達成するための次の行動プランを決定する。

この様にして、行動管理部では現在の状態や入力メッセージなどに応じて、プラニングルールや行動決定ルールなどのルール群を用いて、システムが次にとるべき行動を決定し、最後に出力するメッセージを作成する。以下には行動管理部で用いるルール群とサブルーチンについて、詳細に説明する。

(2) ルールなどの仕様

(a) プラニングルール

プラニングルールは、目標に応じた方法で問題解決を行なうことを可能にするために、システム設計者によって記述される。

以下にはプラニングルールの記述言語の仕様を説明する。一般的に、後向き推論ではルールは以下のように記述される [40]。

(ルール名
　＜前提部＞
　⟹

＜結論部＞)

これを以下の記述形式で表現することにする。

プラニングルール記述形式

```
<rule name=''ルール名''
<cond ''前提部''>
<conc ''結論部''>
</rule>
```

ただし、前提部と結論部は以下のように表現される。

手続き名 第一引数 [第二引数 第三引数, ...]

手続きとしては、現在のところ表 4.2 に示したものが利用可能である。

表 4.2: プラニングルールにおける手続き

	手続き名	意味	第一引数	第二引数
値の大小関係	MoreThan	$>$	数値	数値
	LessThan	$<$	数値	数値
	MoreOrEqual	$\geq$	数値	数値
	LessOrEqual	$\leq$	数値	数値
	Equal	$=$	数値	数値
	NearlyEqual	$\approx$	数値	数値
値の操作	Increment	$+=$	数値	数値
	Decrement	$-=$	数値	数値
論理関係	Include	$\supset$	数値	文字
行為の実行	Act	行動決定処理実行	文字	
その他	Goal	ユーザの最終目標	文字	
	SubGoal	最終目標への仮目標	文字	

以下に、論文検索タスクを例にしたプラニングルールを示す。

プラニングルール（論文検索タスクの場合の一例）

```
<rule id=1>
<cond ‘‘SubGoal 検索条件取得’’
and ‘‘Act 論文表示実行’’>
<conc ‘‘Goal 特定論文表示’’>
</rule>

<rule id=2>
<cond ‘‘Act 検索条件取得_AND’’
and ‘‘Act 論文検索実行’’
and ‘‘Equal 検索結果件数 1’’>
<conc ‘‘SubGoal 検索条件取得’’>
</rule>

<rule id=3>
<cond ‘‘Act 検索条件絞り込み’’>
<conc ‘‘Equal 検索結果件数 1’’>
</rule>

<rule id=4>
<cond ‘‘Act 検索条件取得_OR’’>
and ‘‘Act 論文検索実行(検索条件)’’
and ‘‘AlmostEqual 検索結果件数 10’’
and ‘‘Act 論文リスト表示実行’’
<conc ‘‘Goal 関連論文リスト表示’’>
</rule>

<rule id=5>
<cond ‘‘Act 検索条件追加\_OR’’>
<conc ‘‘AlmostEqual 検索結果件数 10’’>
</rule>
```

上記のプラニングルール群のうち最初の３つは、「ユーザが属性値を知っている特定の（１件の）論文を表示する」という目標に関するルールであり、後の２つは「ユーザが指定する属性値に関係する（10件の）論文リストを表示する」という目標に関するルールである。目標をどのタイミングでどのように設定するかは、システム開発者に任されているが、現段階ではメニュー選択により設定されることにする。なお、ルール中の (*) で示

した節は、条件部にあるが評価の必要がない旨を示している。

(b) 行動決定ルール

行動決定ルールは、基本的にシステムの行動を状態、ユーザの行動をアークとした状態遷移の規則を示したものである。前述したプラニングルールの Act 手続きで指定される行動プラン毎に、サブオートマトンとしてシステム設計者が半自動的に記述する。先に示したプラニングルールの例と同じ論文検索タスクの場合を想定して具体化したサブオートマトンどうしの関係を図 4.5 に示す。

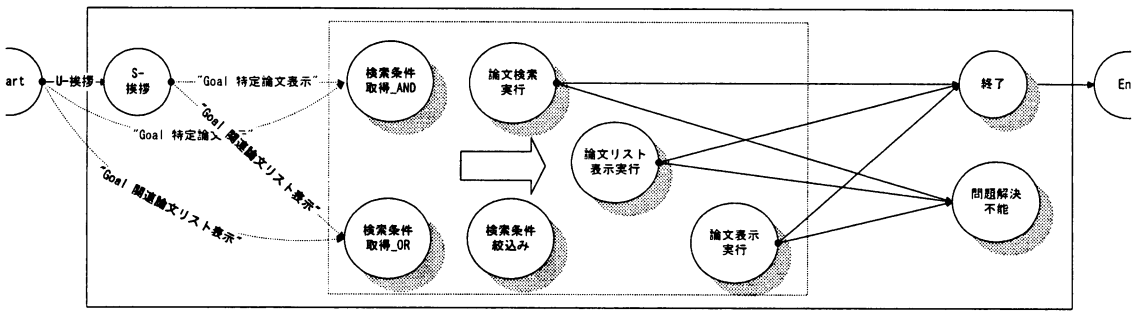
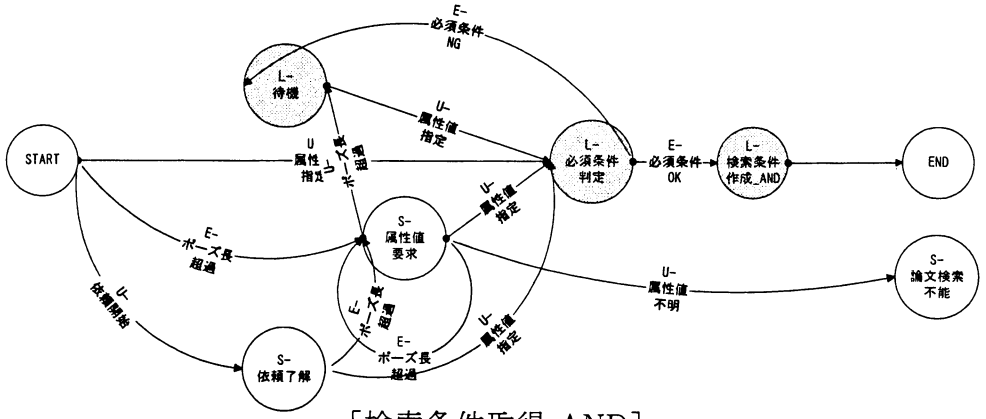


図 4.5: サブオートマトンの関係（論文検索タスクの場合の一例）

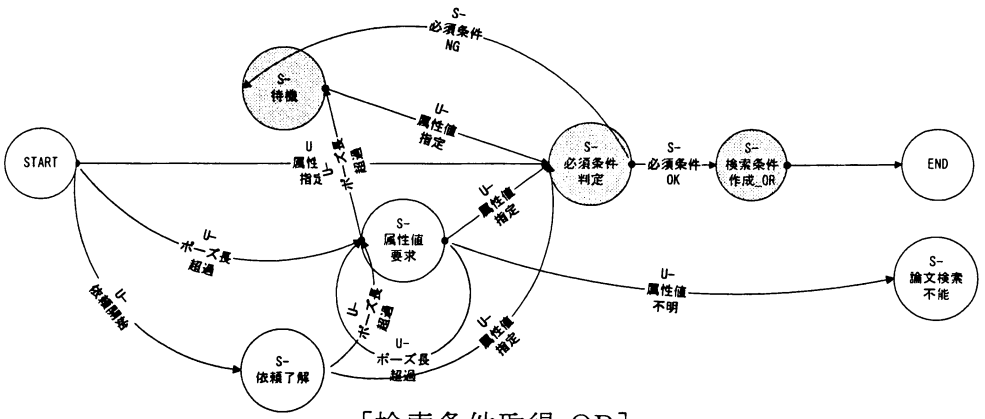
図 4.5 において、最初のユーザ発話の意図解析結果で「挨拶」の発話タイプが得られた場合のみ、挨拶サブオートマトンに遷移する。挨拶サブオートマトンへの遷移に関わらず、問題解決の目標として「特定論文表示」や「関連論文リスト表示」が得られた場合には、プラニングによって最終的にそれぞれ検索条件取得_AND あるいは検索条件取得_OR の行動プランが選択され、その結果それぞれの行動プランに対応したサブオートマトンの

状態「START」に遷移する。サブオートマトンの状態「END」に達した時、サブオートマトンに対応する行動プランが達成されたことになる。サブオートマトンの状態「END」に達していないのに、問題解決の目標が変わりプラニングで異なるサブオートマトンが選択された場合には、本来ならサブオートマトン間の遷移が必要であるが、今回は単にサブオートマトンを切替えて状態「START」から開始することにする。

図 4.5 中のサブオートマトンを詳細にしたものを図 4.6、図 4.7、図 4.8 に示す。

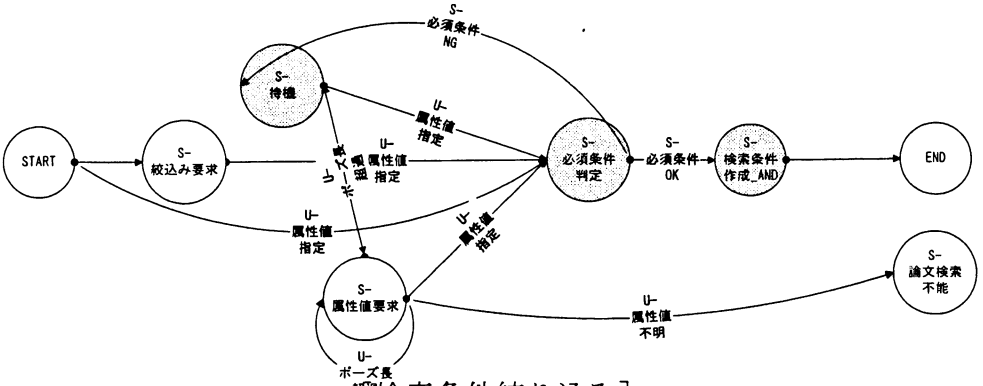


［検索条件取得_AND］

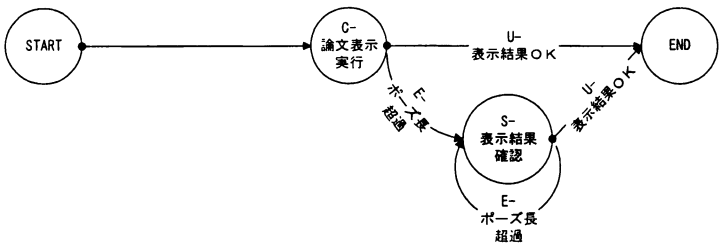


［検索条件取得_OR］

図 4.6: サブオートマトン (1)

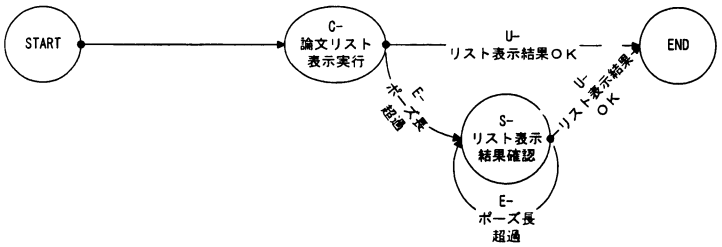


[検索条件絞り込み]

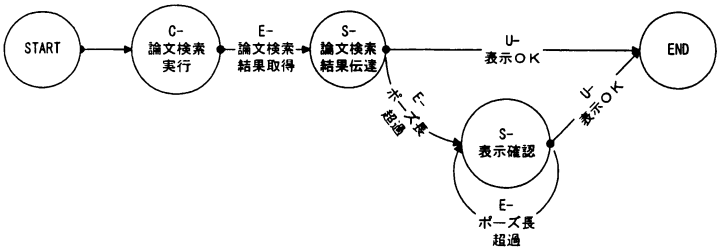


[論文検索実行]

図 4.7: サブオートマトン (2)



［論文リスト表示実行］



［論文表示実行］

図 4.8: サブオートマトン (3)

サブオートマトンは、行動決定ルール記述言語により記述される。例えば、図 4.6 図 4.7 図 4.8 に示したサブオートマトンは以下のように記述される。

なお、図中の “E_” はライブラリ実行結果などのイベント（後述）、”U_” はユーザ発話意図における発話タイプ、“S_” はシステム発話意図における発話タイプ、“L_” はライブラリ、“C_” はアプリケーションコマンドをそれぞれ意味する。

検索条件取得_AND

```
<rule id=0>
<state id=0 act=''Start''>
<arc act=''U_依頼開始'' next_state_id=1 next_state_act=''S_依頼了解''>
<arc act=''E_ポーズ長超過'' next_state_id=2 next_state_act=''S_属性値要求''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act=''L_必須条件判定''>
</rule>

<rule id=1>
<state id=1 act=''S_依頼了解''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act=''L_必須条件判定''>
<arc act=''E_ポーズ長超過'' next_state_id=2 next_state_act=''S_属性値要求''>
</rule>

<rule id=2>
<state id=2 act=''S_属性値要求''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act=''L_必須条件判定''>
<arc act=''E_ポーズ長超過'' next_state_id=2 next_state_act=''S_属性値要求''>
<arc act=''U_属性値不明'' next_state_id=6 next_state_act=''S_問題解決不能''>
</rule>

<rule id=3>
<state id=3 act=''L_必須条件判定''>
<arc act=''E_必須条件 NG'' next_state_id=4 next_state_act=''L_待機''>
<arc act=''E_必須条件 OK'' next_state_id=5
next_state_act=''L_検索条件作成_AND'' end=1>
</rule>

<rule id=4>
<state id=4 act=''L-待機''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act=''L_必須条件判定''>
</rule>

<rule id=5>
<state id=5 act=''L_検索条件作成_AND''>
<arc act='''' next_state_id=6 next_state_act=''End''>
</rule>
```

検索条件取得_OR

```
<rule id=0>
<state id=0 act=''Start''>
<arc act=''U_依頼開始'' next_state_id=1 next_state_act=''S_依頼了解''>
<arc act=''E_ポーズ長超過'' next_state_id=2 next_state_act= ''S_属性値要求''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act ''L_必須条件判定''>
</rule>

<rule id=1>
<state id=1 act=''S_依頼了解''>
<arc act=''E_ポーズ長超過'' next_state_id=2 next_state_act=''S_属性値要求''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act=''L_必須条件判定''>
</rule>

<rule id=2>
<state id=2 act=''S_属性値要求''>
<arc ''E_ポーズ長超過'' next_state_id=2 next_state ''S_属性値要求''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act=''L_必須条件判定''>
<arc act=''U_属性値不明'' next_state_id=6 next=state_act=''S_問題解決不能''>
</rule>

<rule id=3>
<state id=3 act=''L_必須条件判定''>
<arc act=''L_必須条件判定 OK'' next_state_id=5 next_state_act=''L_検索条件作成
_OR''>
<arc act=''L_必須条件判定 NG'' next_state_id=4 next=state_act=''L_待機''>
</rule>

<rule id=4>
<state id=4 act=''L_待機''>
<arc ''E_ポーズ長超過'' next_state_id=2 next_state ''S_属性値要求''>
<arc act=''U_属性値指定'' next_state_id=3 next_state_act=''L_必須条件判定''>
</rule>

<rule id=5>
<state id=5 act=''L_検索条件作成_OR''>
<arc ''' next_state_id=7 next_state ''End''>
</rule>
```