音声言語情報処理 19－19
(1997. 12. 12)

時間関係と対話性を考慮したマルチモーダル対話記述用スクリプト

秋葉友良, 神嶌敏弘, 伊藤克亘

電子技術総合研究所

茨城県つくば市梅園 1-1-4
0298-54-5925
t-akiba@etl.go.jp

あらまし

複数モードを利用した対話システムを構築するための対話記述言語 (対話スクリプト) と、それを解釈・実行する処理系について報告する。このスクリプトは、ユーザにとって自然で役に立つマルチモーダル対話を実現するために、対話中のイベントの時間関係と割り込みなどを始めとする対話性について注意を払って設計した。

キーワード　マルチモーダル対話, 対話システム, システム設計, 対話モデル

MILES: Multimodal Interaction LEading Script, which can express time relations between events and communicative elements in dialogues

AKIBA Tomoyosi, KAMISHIMA Toshihiro, ITOU Katunobu

Electrotechnical Laboratory

1-1-4, Umezono, Tsukuba, Ibaraki, 305 JAPAN
+81-298-54-5925
t-akiba@etl.go.jp

Abstract

In this paper, we will propose a programming language, with which a system designer can design dialogues between a human and an artifact through multi-modes. We also show the system, which interprets and acts on a script written in it. The language was carefully designed so that it can deal with both the precise time relation between events in dialogues and the communicative elements in dialogues, like interruptions, in order to make multi-modal interaction spontaneous and helpful for users.

key words　multimodal interaction, dialogue systems, system designing, dialogue model

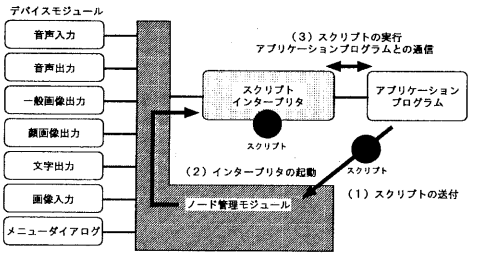

図 1: システムの構成と実行の手順

筆者らは、複数のモードを通した人と計算機との間の対話について研究を行っている。これまで、いくつかの領域 (道案内 [1]、電子秘書 [2]、論争支援 [3]) についての対話システムを構築してきた。これらのシステムの対話に関する処理を担当するモジュールは、領域 (アプリーケーション・プログラム) に大きく依存しおり、個別に設計・実装されている。

しかし、新しい領域の対話システムを構築する度に対話モジュールを一から設計し直すことは、システム開発のコストを考えた場合大きな負担となる。さらに、複数のモードを用いた複雑な対話処理が要求されるマルチモーダル対話システムにおいて、その対話処理のプログラミング自体が大きな負担となる。

そこで我々は、対話に関する処理だけを独立に記述する言語を設計し、それを解釈実行するモジュールをアプリケーション・プログラムとは切り放して実装することを試みた。この枠組みによって、アプリケーション・プログラムに大きく手を加えることなく対話に関する処理だけを独立に開発することができる上、対話に関する資源の共有や再利用が容易となり、システム開発のコストが軽減されるはずである。

本稿では、このようなシステム設計者が対話を記述するのに用いる言語 (対話スクリプト) と、それを解釈実行する処理系について報告する。

1 システムの構成

マルチモーダル対話システムは、顔画像合成、音声合成、顔画像認識、音声認識など、各モードを担当するデバイス・モジュール、複数のデバイス・モジュールを一元的に管理するノード管理モジュール、対話スクリプトを解釈する対話スクリプト・インタープリタ、から構成される (図 1)。

ユーザとの対話によって情報を獲得しようとするアプリケーション・プログラムは、ノード管理モジュールに対して、対話スクリプトとともにインタープリタ起動のメッセージを送付する。ノード管理モジュールによって起動される対話スクリプト・インタープリタは、対話スクリプトの記述に従ってデバイス・モジュールを使用しユーザとの対話を遂行する。アプリケーション・プログラムは、インタープリタと直接通信することができ、対話によって獲得した情報を受け取ったり、インタープリタの実行の制御を行うことができる。

デバイス・モジュール、デバイス管理モジュール、スクリプト・インタープリタ間は、一定のプロトコルに従って通信を行う。ノード管理モジュールによってデバイスを一元的に管理することで、新しいデバイス・モジュールの追加は容易である。現在、上記のデバイスモジュールに加え、テキスト表示、静止画像表示、GUI によるメニュー・ダイアログ、が実装されている。

2 マルチモーダル対話記述言語 MILES

システム設計者は、マルチモーダル対話スクリプトによって、多種のモード・デバイスを媒介としたシステムとユーザとの対話を記述する。我々は、スクリプトの設計にあたり、とくに以下の 2 点について注意を払った。

時間関係 マルチモーダル対話では、複数モード間の時間関係を扱う必要がある。特に、入力モードの同時性、出力モードの時系列上の順序。

対話性 自由な対話の実現 [4]。システム発話への割り込みを可能にする。

マルチモーダル対話記述言語 MILES(Multimodal Interaction LEading Script) は、「状況」と呼ばれる内部表現の操作によって、ユーザとの対話を記述する。「状況」は、システムの内部状態、システム外部の状態、対話の文脈など、対話に関係する資源を一つの表現にまとめたものである。システムからの外部への出力、外部からの入力等は、この「状況」を介して実行される (図 2)。

「状況」は、「状況要素」の集合である。「状況要素」は Prolog の term で表す。「状況要素」は、その functor によって同一性が保証される。すなわち、一つの「状況」内で、同じ functor をもつ 2 つの「状況要素」が現れることはない。

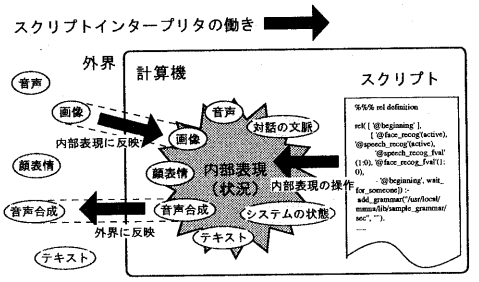

図 2: インタープリタ

MILES では、このように表現された「状況」に対して、ある「状況」においてどのように「状況」を変化させるかを記述する (別の言葉で言えば、「状況」間の関係を記述する) ことによって、対話を記述していく。MILES は、次の要素より構成される。

- rel 定義部
- sim 定義部
- Prologプログラム部

rel 定義部と sim 定義部では「状況」の間の関係を記述するが、その際に指定する「状況」は、その「状況」に含まれるべき部分的な「状況要素」の集合を指し示した以下のような表現<状況>を使用する。

```
<状況> ::= [ <TERM>, <TERM>, ... ]

<TERM> ::= <状況要素> | - <状況要素>
```

以下がすべて成り立つとき、<状況>は現在の状況に適合するという。

1. <状況>に含まれる<状況要素>が、現在の状況に含まれる (Prolog で unification 可能)。
2. <状況>に含まれる- <状況要素>が、現在の状況に含まれない (Prolog で unification 可能なものが存在しない)。

また、<状況>は、現在の状況から次の状況への差分の表現としても使用される。<状況>によって、次の状況は以下のように決まる。

1. <状況>に含まれる<状況要素>は、次の状況に必ず含まれる。現在の状況に、同じ functor を持つ状況要素が存在する場合、<状況要素> に置き換わる。
2. <状況>に含まれる- <状況要素>が、現在の状況に含まれる (Prolog で unification 可能なものが存在する) 場合、次の状況には含まれない。

2.1 rel 定義部

rel 定義によって、状況間の関係を記述する。システムは、複数の rel 定義から、現在の状況と比べて適切な rel 定義を一つ選び、その差分の記述に従って状況を遷移させる。

rel 定義の書式は以下の通り。

```
rel( <(現在の) 状況>, <状況 (差分)> ) :-
        <Prologプログラム>.
```

(Prolog のホーン節と同様、:-以下は省略可能)

手続き的に解釈すると、rel 定義の第一項の<状況>が現在の状況に適合する場合、第二項の<状況>(差分表現) に従って状況を変更する、ことを意味する。

<Prologプログラム>では、この述語を適用する場合に同時に実行する任意の Prologプログラムが記述できる。<Prologプログラム>での、(最初に現れる) カット ('!') の位置は重要である。カット以前に記述するプログラムは、状況との適合性判定の条件となる。カット以降に記述するプログラムは、適合性判定の後、状況の遷移と同時に実行される。<Prologプログラム>中にカットが現れない場合、プログラムの最後にカットが存在するものとみなされる。

2.2 sim 定義部

rel 定義と同様に状況間の関係を記述するが、定常的に状況を変化させたい場合に使用する。システムは、(指定された順序で)sim 定義それぞれについて、現在の状況と適合する場合、その差分の記述に従って状況を遷移させる。

ユーザとしては、認識結果の解釈 (2.6節参照) のために sim 定義を用いるのが好ましい (認識結果が得られた場合、常に解釈を実行するため)。システム内部では、外界の認識結果を状況に反映させたり、逆に状況の記述をを外界に反映させたりするために、あらかじめ用意された sim 定義を用いている。

sim 定義の書式は以下の通り。

```
sim( <全順序>, <(現在の) 状況>,
        <状況 (差分)> ) :- <Prologプログラム>.
```

<全順序>では、sim 定義を解釈する順序を整数により指定する。小さいものから順番に解釈される。第２引数以降とボディ部の記法は、rel 定義と同じである。

2.3 Prologプログラム部

ユーザは任意のPrologプログラムを記述し、rel定義やsim定義から呼び出す事が出来る。このようなPrologプログラムには、システム内に用意された解釈プログラム(2.6節参照)等が参照するデータベースが含まれる。

2.4 状況要素

'@'で始まるfunctorを持つ状況要素はシステムで予約されている。これらは、システム外部の状態やシステムの設定を表現し、それを操作することでシステム外部を変化させたり(システムからの出力)システムの設定を変更したりすることができる。このような状況要素の一部を以下に示す。

- `'@speech_recog_result'(C)`
  音声認識結果
- `'@speech_recog_interp'(C)`
  音声解釈結果
- `'@act'(Seq)`
  動作系列の指定(2.5節参照)。
- `'@message_from_aa'(Msg)`
  アプリケーション・プログラムからのメッセージ
- `'@message_to_aa'(Msg)`
  アプリケーション・プログラムへのメッセージ。
- `'@speech_recog'(X)`
  音声認識デバイスのON/OFF。
- `'@beginning'`
  インタープリタ実行開始。初期設定に使用。
- `'@finish'`
  インタープリタ終了。
- `'@timeval'(Val)`
  rel定義の第一項で指定する状況が生起する時間間隔を表す。

それ以外の任意のfunctorを持ち任意の引数を持つ状況要素はユーザが自由に定義し使用することができる。これらは主に、対話の文脈を表すために使用される。3節でみるように、状態遷移的に対話の流れを表現することもできる。(Prologプログラム部に)planningモジュールを用意すれば、plan認識・生成による対話のモデルを持ち込むことも可能であろう。

2.5 動作系列

システムの外界への働きかけ(動作)は、動作系列として管理される。動作系列は、基本動作のシークエンスで表される。システムは、このシークエンスから、順番に基本動作を取り出し、外界への働きかけを行う。ユーザは、状況要素の操作によって、このシークエンスを自由に制御することができる。

動作系列の操作は、状況要素`'@act'(Seq)`の操作によって行う事が出来る。この引数に与えられる<Seq>は、次のような表現で表される。

```
<Seq> ::= [ <動作単位>, ... ]
<動作単位> ::= <基本動作> | [ <基本動作>, ... ]
```

すなわち、<Seq>は動作単位のリストであり、動作単位は基本動作あるいは基本動作のリストである。一つの動作単位中の基本動作は、そのリストの順序に従い、間隔を置かずに動作が実行される。動作単位の実行中には動作が割り込まれることはない。一方、動作単位の実行の間では、状況の操作が行われる事で、動作の割り込みを行うこともできる(3.2節)。

基本動作の一部を以下に示す。

- `set_agent(Agent)`
  表示エージェントの切り替え
- `speak(Str, Args)`, `speak(Str)`
  発声と口の動き
- `face(Cmd)`
  顔表情の変化
- `+`<状況の差分>
  状況の操作
- `reset`
  未処理の動作系列を捨てる
- `insert(Seq)`
  現在の動作系列の先頭に、新たな動作系列Seqを挿入する

2.6 システム予約述語

対話スクリプトの枠組に加え、対話システム開発を補助するためのProlog述語をいくつか用意した。それらの多くは、複雑な音声と顔画像認識の結果を解釈するための述語である。システム設計者は、rel定義部およびsim定義部のPrologプログラム呼び出

しによって、これらの述語を自由に利用することができる。

3 スクリプト記述例

3.1 複数のモードの同時性

マルチモーダル対話では、個別のモードからの入力を処理することに加え、複数のモードからの入力が同時に生じたことを処理する必要がある。MILESでは、「状況」の指定において、モードに対応する「状況要素」を列挙することでモードの同時性を判断する。また、「同時」と判断する時間間隔を状況要素@timevalで指定する。

次のrel定義では、「1秒の間に画像Someと音声helloが同時に現れる状況」を表す。

```
rel( [ '@timeval'(1:0),
       '@face_recog_interp'(Some),
       '@speech_recog_interp'(hello) ],
     [ '@act'( [ speak("こんにちわ") ] ) ]
   ) :- someone(Some).
```

3.2 システムの発話への割り込み

状況要素@actに記述する動作系列(2.5節)は、記載順に漸次的に実行される。その過程(どこまで実行したかという情報)も「状況」として管理されるため、「状況」の操作によって実行の中止や割り込みを行うことができる。

次に示すスクリプトは、質問に答えて道順を説明するための定義である。最初のrel定義によって、Prolog述語route_to_etlに与えられた説明文は漸次的に発話される。説明文の発話中、ユーザから発話があると、第2のrel定義によって、動作系列が捨てられ発話がキャンセルされる。説明のための動作系列の最後には状況の操作(+[-explaining,explained])が記述されており、説明が完了した時点で実行され、文脈explainedが状況に加えられる。説明が完了しない限り、状況にexplainedが与えられることはない。このような方法で、発話の完了を文脈として管理することができる。

```
rel( [ '@timeval'(1:0),
       'speech_recog_interp'(how_to_go) ],
     [ explaining,
       '@act'([ speak(Way),
                +[ -explaining,explained ]
              ] ) ]
   ) :- !, route_to_etl(Way).

rel( [ explaining,
       '@timeval'(1:0),
       '@speech_recog_interp'(stop) ],
     [ - explaining,
       '@act'( [ reset ] ) ] ).

route_to_etl("並木大橋バス停から、電総研、本館正面入り口までの、生き方を説明します。東京駅から、乗ってきた、高速バスを、最初のバス停である...").
```

3.3 ユーザ主導対話

図4は、電子秘書システム[2]の一部を再現したスクリプトである。ユーザがシステムの前に現れると、顔画像を表示しあいさつをする。そしてユーザからの質問に答える。ユーザが立ち去ると、顔画像を消して、次のユーザが現れるのを待つ(図3)。

このスクリプトでは、主に外界からの刺激(ユーザの出現、退去、ユーザからの質問)を手がかりに対話を行う。システムが利用する文脈は、システムの前にいるユーザの情報(誰かいるか?誰がいるか?)で、「状況要素」"wait_for_someone"と"face_to(Some)"によって表現している。

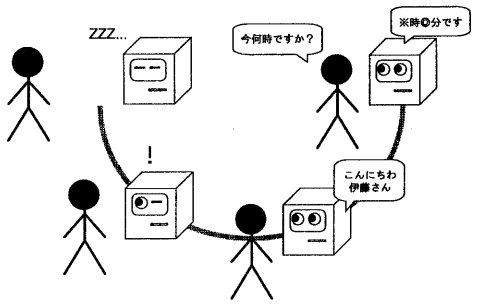

図3: スクリプト例1(ユーザ主導対話)

3.4 システム主導対話

図6は、アプリケーションプログラムが人探しをするために、ネットワーク上に点在する複数の計算機に送り付けることを想定したスクリプトである。スクリプトが起動すると、人がいない場合は人物が現れるまで待つ。探す対象となる人物が現れると、見つけたことをアプリケーションプログラムに伝える。

```
%%% rel 定義部

rel( [ '@beginning' ],
     [ '@face_recog'(active), '@speech_recog'(active), '@speech_recog_fval'(1:0),
       '@face_recog_fval'(1:0), -'@beginning', wait_for_someone]) :-
 add_grammar("/usr/local/mmna/lib/sample_grammar/sec", "").

rel( [ wait_for_someone, '@timeval'(1:0), '@face_recog_interp'(Some) ],
     [ - wait_for_someone, face_to(Some),
       '@act'( [ [ reset, set_agent(jony), face(be(cool)), face(come_here), face(be(happy)) ],
                 speak("こんにちわ、 sさん。", [Name]),
                 face(be(cool)) ]                          )]) :-
 name_of(Some, Name), !.

rel( [ face_to(Some), '@timeval'(1:0), '@face_recog_interp'(nobody) ],
     [ - face_to(Some), wait_for_someone, last(Some),
       '@act'( [ reset, stay(3), set_agent(jony), face(go_away), face(be(cool)) ]  )] ).

rel( [ face_to(Some), '@timeval'(1:0), '@speech_recog_interp'(what_date)  ],
     [ '@act'( [ reset, set_agent(jony), face(be(angry)),
                 speak("d月、 d日、です。", [M, D])  ] )  ]) :-
 datime(datime(_, M, D, _, _, _)).

rel( [ face_to(Some), '@timeval'(1:0), '@speech_recog_interp'(what_time) ],
     [ '@act'( [ [ reset, set_agent(jony), face(be(angry)) ],
                 speak("d時、 d分、です。", [H, M])] )        ] ) :-
 datime(datime(_, _, _, H, M, _)).

%%% sim 定義部

sim(100, ['@speech_recog_result'(SR)],
         ['@speech_recog_interp'(SI) ]) :-
 top_n_result(SR, 3, SR1), interp_with_regexp(interp_x, SR1, SI).

sim(110, ['@face_recog_result'(FR)],
         ['@face_recog_interp'(FI) ]) :-
 interp_a_series(interp_s, FR, FI).

%%% Prologプログラム部

interp_with_regexp_rule(interp_x, [何時], what_time).
interp_with_regexp_rule(interp_x, [何日], what_date).
interp_with_regexp_rule(interp_x, [はい], yes).
interp_with_regexp_rule(interp_x, [いいえ], no).
interp_with_regexp_rule(interp_x, [fp(_),'$'], fp).

interp_a_series_rule(interp_s, ['^',N, N, N, N], N).
interp_a_series_rule(interp_s, _, giveup).

name_of('t-akiba', "秋葉").
name_of('kamisima', "神嶌").
name_of('kito', "伊藤").
```

文脈 wait_for_someoneにおいてこの状況が１秒の間隔内で生じる

新しい文脈 face_to(Some)へ遷移

これまでの表出列を捨て(reset:割り込み)表出列に挨拶の手順(顔を表示し、微笑み、「こんにちわ」と発声、普通の表情に戻る)を加える

音声認識結果(解釈済み)がwhat_time(何時ですか)である

音声認識結果を取り出す

解釈結果を「状況」に加える

(インタープリタ内に用意された)正規表現による解釈プログラムを呼び出す

解釈規則の指定

音声認識解釈規則

画像認識解釈規則(４回連続で人物を認識した場合のみその人物であると判断)

図 4: スクリプト例 1

別の人物がいる場合は、居場所を質問して、計算機の前まで来てもらうようにお願いする (図 5)。

このスクリプトでは、システム側が積極的に働くことで情報の収集を試みる。システムは、状態遷移的に管理された対話の流れを利用する。これは「状況要素」"state(X)" によって表される。

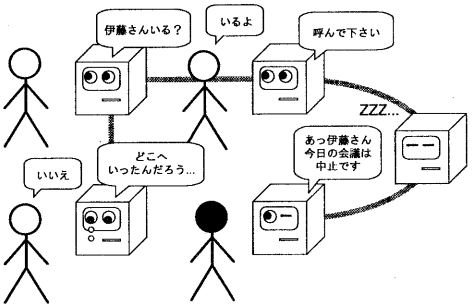

図 5: スクリプト例 2(システム主導対話)

4 関連研究と今後の課題

対話モデルの研究では、プラン生成とプラン認識によるモデル [5] が有名である。しかしこれらは、発話を行為として捉えた抽象度の高いレベルでのモデルである。行為と各モードとの関係には別のモデルが必要であるし、また高度な言語処理が必要となる。また、これらの対話モデルで、対話のダイナミズム (割り込み、漸時的な発話) をどのように扱うか (そもそも扱えるのか)、ということも十分に明らかになっているとはいえない。

むしろ、我々が目指すのはより抽象度の低いレベルのモデルである。各モードの間の関係を、直接、かつ、できるだけ容易に、記述することが目的である。このようなスクリプトとして、状態遷移図 (有限状態オートマトン) によるモデル化の手法が提案されている ([6] など)。このモデルでは、システムの行う対話を全体的にはっきりと定義できる (定義しなければならない) という利点を持つ。しかし、対話の流れを網羅的に記述する必要があり、割り込みなどの対話のダイナミズムを記述するのは容易でないように思われる。

我々のスクリプトでの記述の単位は、より局所的である。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事が出来る。また、3節でみたように、ユーザ定義の状況要素を利用する事で状態遷移的な対話の記述も可能である。しかし逆に、スクリプトの記述からそこに表現された対話の流れを把握することは、適切なコメントが与えられていない限り、容易ではないかもしれない。言語として、直接的な対話の流れの表現方法が用意されている方が好ましいとも考えられる。

我々のスクリプトにおけるもう一つの課題は、記述のモジュール性の確立である。部分的な対話を記述した２つのスクリプトを組み合わせて１つの対話スクリプトを記述する場合、単順にアペンドするだけではだめで、２つのスクリプトの整合性を保つために多少の手直しをする必要がある。大きな対話スクリプトを記述するためには、小さな対話を記述したスクリプトを複数組み合わせても対話として破綻しないような記述のモジュール性が必要となるであろう。

参考文献

[1] Itou, K., Akiba, T., Hasegawa, O., Hayamizu, S. and Tanaka, K. Collecting and analyzing non-verbal elements for maintenance of dialog. In ICSLP, pp.907-910, 1994.

[2] Hayamizu, S. et al, Multimodal interaction system at the Electrotechnical Laboratory, Proceedings of Real World Computiong Symposium, pp.16-22, 1997.

[3] 新田克己, 他. 論争支援マルチモーダル実験システム MrBengo. 電子情報通信学会論文誌 D-II, Vol.80-D-II No.8, 1997.

[4] 伊藤克亘, 秋葉友良, 上條俊一, 田中和世. 休止を区切りとした対話処理. 音声言語情報処理, 95-SLP-7-22, 1995.

[5] Allen, J., Analyzing Intention in Utterances, Artificial Intelligence, Vol.15, pp.143-178, 1980.

[6] 松浦 博, 神尾 広幸, 内山 ありさ, 郡田 美香, 田村 正文, 新田 恒雄, マルチモーダル対話システムのための UI 設計支援ツール, 情処学研報, 94-SLP-3-7, 1994.

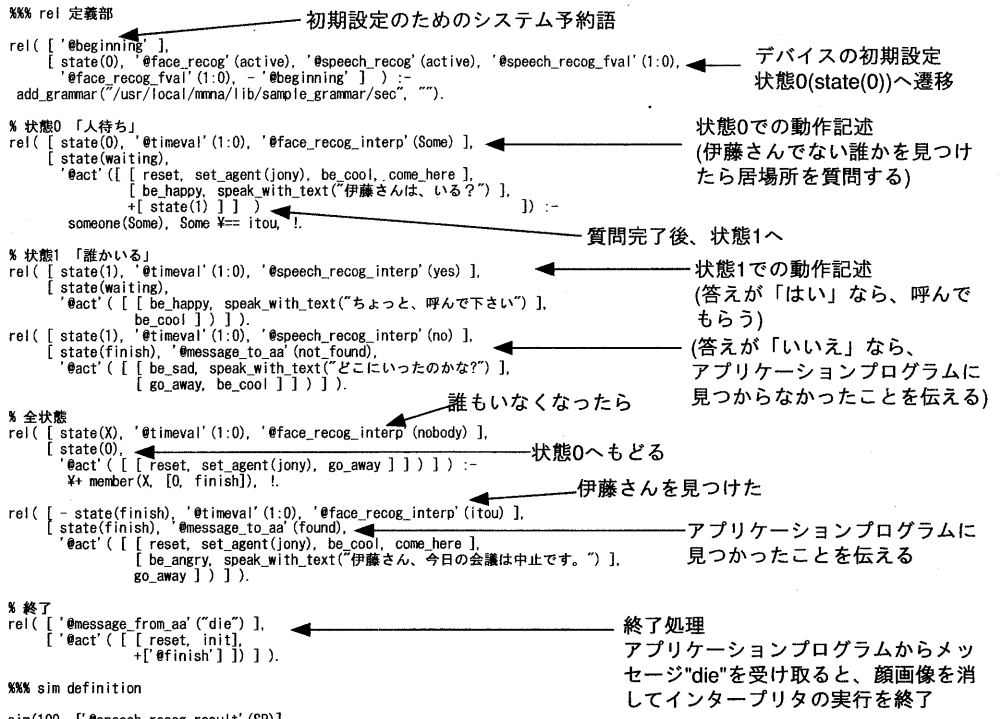
%%% rel 定義部
初期設定のためのシステム予約語
rel(['@beginning'],
[state(0), '@face_recog'(active), '@speech_recog'(active), '@speech_recog_fval'(1:0),
'@face_recog_fval'(1:0), - '@beginning']) :-
add_grammar("/usr/local/mmna/lib/sample_grammar/sec", "").
デバイスの初期設定
状態0(state(0))へ遷移
% 状態0 「人待ち」
rel([state(0), '@timeval'(1:0), '@face_recog_interp'(Some)],
[state(waiting),
'@act'([[reset, set_agent(jony), be_cool, come_here],
[be_happy, speak_with_text("伊藤さんは、いる?")],
+[state(1)]])]) :-
someone(Some), Some ¥== itou, !.
状態0での動作記述
(伊藤さんでない誰かを見つけたら居場所を質問する)
質問完了後、状態1へ
% 状態1 「誰かいる」
rel([state(1), '@timeval'(1:0), '@speech_recog_interp'(yes)],
[state(waiting),
'@act'([[be_happy, speak_with_text("ちょっと、呼んで下さい")],
be_cool])]).
rel([state(1), '@timeval'(1:0), '@speech_recog_interp'(no)],
[state(finish), '@message_to_aa'(not_found),
'@act'([[be_sad, speak_with_text("どこにいったのかな?")],
[go_away, be_cool]])]).
状態1での動作記述
(答えが「はい」なら、呼んでもらう)
(答えが「いいえ」なら、アプリケーションプログラムに見つからなかったことを伝える)
% 全状態
誰もいなくなったら
rel([state(X), '@timeval'(1:0), '@face_recog_interp'(nobody)],
[state(0),
'@act'([[reset, set_agent(jony), go_away]])]) :-
¥+ member(X, [0, finish]), !.
状態0へもどる
伊藤さんを見つけた
rel([- state(finish), '@timeval'(1:0), '@face_recog_interp'(itou)],
[state(finish), '@message_to_aa'(found),
'@act'([[reset, set_agent(jony), be_cool, come_here],
[be_angry, speak_with_text("伊藤さん、今日の会議は中止です。")],
go_away])]).
アプリケーションプログラムに見つかったことを伝える
% 終了
rel(['@message_from_aa'("die")],
['@act'([[reset, init],
+['@finish']])]).
終了処理
アプリケーションプログラムからメッセージ"die"を受け取ると、顔画像を消してインタープリタの実行を終了
%%% sim definition
sim(100, ['@speech_recog_result'(SR)],
['@speech_recog_interp'(SI)]) :-
top_n_result(SR, 3, SR1), interp_with_regexp(interp_x, SR1, SI).
sim(110, ['@face_recog_result'(FR)],
['@face_recog_interp'(FI)]) :-
interp_a_series(interp_s, FR, FI).
%%% Prolog プログラム部
interp_with_regexp_rule(interp_x, [はい], yes).
interp_with_regexp_rule(interp_x, [いいえ], no).
interp_with_regexp_rule(interp_x, [fp(_), '$'], fp).
interp_a_series_rule(interp_s, ['^',N, N, N, N], N).
interp_a_series_rule(interp_s, _, giveup).
someone(Some) :-
name_of(Some, _).
name_of('t-akiba', "秋葉").
name_of('kamisima', "神嶋").
name_of('itou', "伊藤").

図 6: スクリプト例 2