音声対話シナリオのための複合状態をもつ状態遷移記述形式の提案

若林敬太郎† 堤 修平† 山本 大介† 高橋 直久†

† 名古屋工業大学大学院工学研究科情報工学専攻 〒 466–8555 愛知県名古屋市昭和区御器所町
E-mail: †k.wakabayashi.698@nitech.jp, ††{tsutsumi.shuhei,yamamoto.daisuke,naohisa}@nitech.ac.jp

あらまし 本学では音声対話システムの研究・開発を促進するため MMDAgent と呼ばれる音声対話システム構築ツールキットを開発・公開している．MMDAgent には独自の書式で記述されたテキストファイルを読み込ませることで音声対話の流れをカスタマイズできる機能が存在する．その書式は FST 書式と呼ばれ，状態遷移として対話の流れを記述するようになっている．しかし，FST 書式ではプログラムを構造的に記述することができない．そのため，対話シナリオの規模が大きくなるほど処理の流れを把握しにくくなったり，似たような遷移パターンを持ったものでもひとつひとつ処理を記述しなければならないといった問題があった．そこで本研究では音声対話シナリオのための複合状態をもつ状態遷移記述手法を提案する．複合状態とは内部遷移をもつ状態であり，状態遷移の階層的な表現を可能にする．本研究の目的は，音声対話システムで用いる対話シナリオとなるプログラムを，どのような書式で表現すると開発者にとって書きやすく読みやすいものになるかを研究することである．

キーワード MMDAgent, 音声対話, 状態遷移, 複合状態, プログラミング, XML

1. はじめに

1.1 研究の背景

近年，スマートフォン，スマートウォッチとともに，Apple の Siri [1] や NTT ドコモのしゃべってコンシェル [2] など，音声対話システムが普及しつつある．そこで，本学では音声対話システムの研究・開発を促進するため MMDAgent [3] と呼ばれる音声対話システム構築ツールキットを開発・公開している．

MMDAgent は音声認識，音声合成，3D-CG 描画，対話管理を統合したシステムである．図 1 のようなエージェントと音声による対話が可能である．本学正門前に設置されている双方向音声案内デジタルサイネージ [4] などにも応用されている．

MMDAgent の特徴として，ユーザによって任意に対話シナリオの編集が行えるという点が挙げられる．対話シナリオとは「ユーザがこう言ったなら，キャラクタはこう言い返してこういう動きをする.」といった対話の流れやその台本のようなものを意味する．対話シナリオは FST ファイルと呼ばれる外部ファイルに記述する．MMDAgent の起動時に FST ファイルが読み込まれ，記述した対話を行うことができる．

図 1: MMDAgent の画面

1.2 研究の目的

本研究の目的は，音声対話システムで用いる対話シナリオとなるプログラムを，どのような書式で表現すると開発者にとって書きやすく読みやすいものになるかを研究することである．特に，MMDAgent で利用する対話シナリオであること想定する．

従来 MMDAgent で利用されてきた書式では，構造的にプログラムを書く方法がない．これにより，状態番号の意図しない重複や，変更をおこなうときに編集する必要のある部分を探し出すことに時間がかかるなど様々な問題が発生していた．そのため，そのような問題が解決できるような書式を考案する必要があった．

1.3 関連研究

1.3.1 音声対話記述言語

関連研究として VoiceXML [5] が挙げられる．VoiceXML は音声対話パターンの記述言語で，電話での自動応答システムや音声ブラウザで利用することを目的に考案された．モダリティの対象は音声と電話のプッシュ信号 (DTMF) と限定されており，マルチモーダルな対話には利用できない．VoiceXML はシステム主導型の対話が記述しやすいという特徴がある．

他の関連研究として MMI 記述言語 XISL [6] が挙げられる．XISL はマルチモーダル対話記述言語で，音声だけでなく，画像による出力や，ポインティングデバイスによる入力など，様々な入出力を扱うことができる．モダリティの追加に対応しており，マルチモーダルの取り扱いに柔軟性がある．マルチモーダルな対話を記述できる点が VoiceXML と大きく異なる．XISL はユーザ主導の対話を記述しやすくなっている．

XISL は VoiceXML をもとに提案された記述手法である．VoiceXML と XISL の比較は「音声対話記述言語 VoiceXML と MMI 記述言語 XISL の比較」[7] で述べられている．

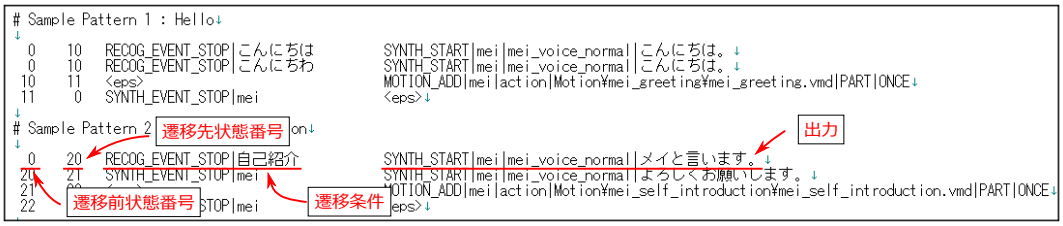

図 2: FST 書式の例

どちらの記述方式もフローチャートのように対話シナリオを記述するようになっている．本論文で提案する記述手法も状態遷移で対話シナリオを記述する点が上記の記述手法とは異なる．

1.3.2 MMDAgent 用対話シナリオ編集エディタ

さらに，関連研究として，音声対話エージェントのための Web ブラウザを用いたシナリオエディタ MMDAE [8] も挙げられる．このエディタでは，本学で開発した MMDAgent と呼ばれる音声対話システム構築ツールキットで利用する対話シナリオを編集することができる．

本研究で提案する記述書式は MMDAgent で利用することを想定している点で MMDAE と同じである．MMDAE では，MMDAgent の対話シナリオ記述書式である FST 書式のテキストを編集しやすいようなエディタを開発することで，FST 書式の書きづらさを軽減しようという解決策をとっている．しかし，本研究では FST 書式より書きやすい記述手法を提案している点で MMDAE の研究とは異なる．

2. 予 備 知 識

2.1 MMDAgent

2.1.1 MMDAgent の概要

MMDAgent は図 1 のように，画面上に表示される 3D キャラクターと音声を主としてマルチモーダルな対話をすることができるシステムである．音声認識，音声合成，3D-CG 描画，対話管理を統合することで実現している．

MMDAgent は音声認識，音声合成，対話管理，その他すべての追加機能をプラグインとして動作させる．プラグイン同士の連携はイベントを通しておこなわれる．MMDAgent では任意のプラグインから受信したイベントを全てのプラグインに送信する．他のプラグインはそのイベントを受け取り，そのイベントに見合った動作をおこなう（もしくはイベントを無視する）．

2.1.2 MMDAgent の対話管理

MMDAgent の対話管理の特徴として，ユーザによって任意に対話シナリオの編集が可能であるという点が挙げられる．対話シナリオとは「この状況でユーザがこう言ったなら，キャラクタはこう言い返してこういう動きをする．」といった対話の流れやその台本のようなものを意味する．

対話シナリオは FST ファイルと呼ばれる外部ファイルに記述する．FST ファイルは FST 書式で記述されたテキストファイルである．MMDAgent の起動時に FST ファイルが読み込まれ，記述した対話を行うことができる．

先に説明した通り，MMDAgent の対話管理も 1 つのプラグインとして実装されている．

2.2 FST 書式

2.2.1 FST 書式の概要

MMDAgent の対話管理は FST 書式で書かれたテキストファイルをもとにおこなわれる．FST 書式では対話シナリオを状態遷移で表現する．FST 書式で書かれたファイルの例を図 2 に示す．

FST 書式では 1 行が 1 つの遷移を表す．各行では，先頭から遷移前状態番号，遷移後状態番号，遷移条件，出力の 4 つをスペース区切りで記述する．

2.2.2 FST 書式の問題点

FST 書式では構造的なプログラムが書けない．コードに構造はあるのにも関わらず FST 書式自体に構造を表す方法が用意されていないため，コメントを使って無理やり構造を表す手法が取られていた．

このように FST 書式では構造的なプログラムが書けないということから次のような問題が発生する．

- 状態番号の意図しない重複に特に注意する必要がある
- プログラムを読むことが難しい
- 同様な処理をする部分の再利用が難しい

FST 書式では状態を番号で管理する．別の状態であるならば別の番号を割り当てなければならない．しかし，FST 書式で書かれたプログラムでは，他のプログラミング言語にはあるスコープという概念が存在しない．そのため，状態番号はプログラム全体で意図しない重複がおこらないように注意しなければならない．プログラムの規模が大きくなるにつれて状態数が多くなり，人手で状態番号の空きを管理するのは困難である．

一般的なプログラム言語では，サブルーチンを利用して階層的に処理を抽象化させてプログラムを作成していく．適切に階層的に処理を抽象化させて作られたプログラムは，サブルーチンの処理が想像できるようにそのサブルーチンに対して適切に命名がされているならば，プログラムの全てを読むことをしなくてもプログラムの概要を理解することが可能である．しかし，FST 書式では処理を抽象化する記述手段が用意されていない．したがって，FST 書式でかかれたプログラムを理解しようと思うと，プログラムの全ての部分を読まなければならない．そのためプログラムを読むことが難しくなる．

一般的なプログラム言語ではサブルーチンを利用して，同様な処理をおこなう部分を 1 カ所にまとめることができる．また，よく使う処理をライブラリとしてあらかじめ外部ファイル

で定義しておき，利用するときはそのファイルを読み込むなどして，再利用することも多い．しかし，FST 書式ではそのような仕組みが用意されておらず，プログラムの再利用ができない．同じ処理であっても繰り返し記述する必要がある．

また，道案内生成システムや Web FST 編集システムなど，FST 書式ファイルの自動生成をおこなうシステムが開発されている．機械的に FST 書式ファイルを出力する場合であっても，状態番号を重複しないようにするといった FST 書式の妥当性に考慮した多くの処理が多く必要になり，煩わしくなる．

このような問題があるため，構造的に音声対話シナリオを記述できる書式を考案する必要があると考えた．

3. 提 案 手 法

3.1 提案手法の概要

本論文では従来の FST 書式で発生していた問題を解決するために，音声対話シナリオのための複合状態をもつ状態遷移記述形式を提案する．

第 2. 章で述べたように， FST 書式では構造的なプログラムが書けない．構造的なプログラムが書けないことによりさまざまな問題が発生していた．

3.2 提案手法の目的

提案する音声対話記述書式で達成する目標は次の通り．

- 構造的に音声対話シナリオを記述できる
- プログラムの再利用がしやすい
- MMDAgent 自体には手を加えずに利用できる

構造的に音声対話シナリオを記述できるようにすることで，プログラムの正しさを考慮しなければならない範囲が局所化される．これは，状態番号の重複を注意しなければならない範囲が狭くなることで，状態番号が管理しやすくなる利点がある．さらに，プログラムを理解するために読む必要のある範囲が局所化される．これによって，全体の内容を一度に把握する必要がなく，部分ごとで順番に理解していくことができる．

プログラムの再利用性を高めることで，1 度プログラムを書いてしまえば，次から同じ処理はそのプログラムを再利用すればよいので，プログラムを簡単に書けるようになる．

提案書式を利用するために MMDAgent に手を加える必要があるということは，提案書式を利用できるシステムが少なくなってしまう．また，システムに変更を加えると正しく動作しなくなるというリスクがある．従来の FST 書式の対話シナリオを利用しているシステムであれば，手を加えることなく提案手法を用いて作成された対話シナリオを利用できることが望ましい．

3.3 提案手法の特徴

書式の特徴は次の通り．

- 複合状態を導入
- 複合状態テンプレート
- 状態を数値ではなく文字列で識別する
- FST 書式に変換可能

複合状態とは，状態遷移機械において内部状態をもつ状態のことである．状態遷移機械は，状態と遷移でその動作を定義する．状態の中にさらに状態遷移があるものを複合状態と定義される．複合状態を導入することにより，状態遷移を階層的に表現することができる．その結果，構造を持った状態遷移を記述することができる．

複合状態テンプレートとは，複合状態にパラメータを定義できるようにしたものである．複合状態テンプレートにパラメータとして具体的な値を代入し，複合状態を生成する．遷移パターンは同じで，具体的な値だけが違う処理を複合状態テンプレートとして作成することで同じパターンを再利用し，繰り返し記述する必要がなくなる．

FST 書式では状態の識別のために，状態を状態番号で管理していた．しかし，状態が番号で表されていた場合，その番号の状態にいるということがどういう状況であるのかを推測することが難しい．状態の識別子を文字列にすることで，その状態が何を表すのか推測しやすいプログラムを書くことができる．また，自動生成プログラムによって対話シナリオを作成する場合でも，適切な命名規則を定めることで，状態名の重複を防ぐことが可能となる．

3.3.1 複 合 状 態

複合状態とは状態遷移機械において内部遷移をもつ状態のことである．状態遷移機械は，状態と遷移でその動作を定義する．Harel の提案したステートチャートや UML のステートマシン図でも用いられている．複合状態もほかの状態と同じように遷移をつなぐことができる．各記述形式によって複合状態の条件は異なることがある．

複合状態を導入することによって構造をもった状態遷移を記述することができる．

本論文での複合状態の定義は次の 4 つ組で表せる．

$$(Q, T, q_0, E)$$

Q は状態の集合である．複合状態もそうでない状態も含む．T は遷移の集合である．q_0 は初期状態である．$q_0 \in Q$ である．E は exit point の集合を表す．$E \subset 2^Q$ である．1 つの exit point に複数の状態を対応づけられる．

遷移の定義は次の 4 つ組で表せる．

$$(q_{src}, q_{dst}, \sigma, \gamma)$$

q_{src} は遷移前状態を表している．q_{src} は，状態もしくは複合状態の exit point が指定できる．つまり，$q_{src} \in Q \cup \{x|$ 全ての $s \in Q$ について $s = (Q', T', q'_0, E')$ ならば $x \in E'$ となる $x\}$ q_{dst} は遷移先状態を表していて，$q_{dst} \in Q$．σ は入力, γ は出力を表している．

そして，複合状態を含む状態遷移機械の定義は複合状態の定義と同様である．つまり，状態遷移機械自体がひとつの複合状態であると見なすことができる．

複合状態の内部状態として複合状態がありそのまた内部に複合状態があるというように，階層的に状態遷移を表現することができる．複合状態を表した図を図 3 に示す．

複合状態をもつ状態遷移機械の現在の状態は $(q_{(0)}, q_{(0,0)}, q_{(0,0,0)}, \cdots)$ と表せる．これは最上位の複合状態

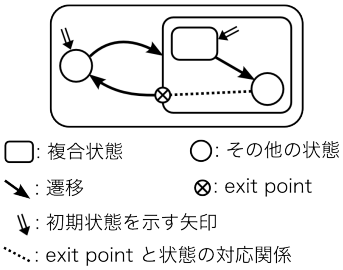

図 3: 複合状態を図示したもの

が $q_{(0)}$ で，$q_{(0)}$ の現在の状態が複合状態 $q_{(0,0)}$ であり，さらに $q_{(0,0)}$ の現在の状態は複合状態 $q_{(0,0,0)}$ であるということを示している．

この状態であるときに入力 i があったとき，まず $q_{(0)}$ を参照し $q_{(0,0)}$ から入力 i で遷移するものがあるか調べる．ある場合，遷移先の状態を $q_{(0,1)}$ とすると，現在の状態を $(q_{(0)}, q_{(0,1)}, \cdots)$ に変化させる．入力 i で $q_{(0,0)}$ から遷移するものがないときは，$q_{(0,0)}$ を参照し $q_{(0,0,0)}$ から入力 i で遷移するものがあるか調べる．このようにして遷移するものがあるまで下の階層を参照し，遷移するものがある場合はそこで遷移を起こして次の入力があるまで待機．最下層まで参照しても遷移を起こすものがなければ今回の入力は無視して次の入力があるまで待機する．

3.3.2 複合状態テンプレート

複合状態テンプレートとは複合状態にパラメータを埋め込んだものである．複合状態で，パラメータが違うだけで遷移パターンは同じものをひとつのテンプレートにまとめることができる．複合状態テンプレートによって複合状態を抽象化して定義することができる．

例えば,「キャラクターがこんにちはと話しながらお辞儀をする」状態遷移機械と,「キャラクターがさようならと話してから手を振る」状態遷移機械を比較すると，台詞や動作は違うが遷移パターンは同じである．そこで，あらかじめ「キャラクターが＜台詞＞を話してから＜動作＞をする」という複合状態テンプレートを定義する．実際に利用するときは台詞と動作のパラメータを与えて複合状態を生成することで重複する記述を減らすことができる．

これはクラスベースのオブジェクト指向言語におけるクラスの概念と共通する部分がある．クラスという設計図から，コンストラクタを通して内部パラメータの違ういくつものインスタンスを生成するように，複合状態テンプレートという設計図からパラメータの違ういくつもの複合状態を生成する．

3.3.3 文字列による状態識別

FST 書式では状態の識別のために，状態を状態番号で管理していた．しかし，状態が番号で表されていた場合，その番号の状態にいるということがどういう状況であるのかを推測することが難しい．状態の識別子を文字列にすることで，その状態が何を表すのか推測しやすいプログラムを書くことができる．また，自動生成プログラムによって対話シナリオを作成する場合でも，適切な命名規則を定めることで，状態名の重複を防ぐことが可能となる．

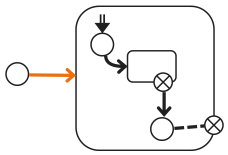

図 4: 入遷移の展開前

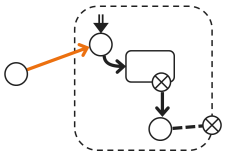

図 5: 入遷移の展開後

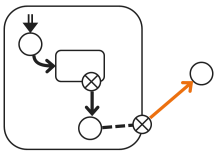

図 6: exit point から出る遷移の展開前

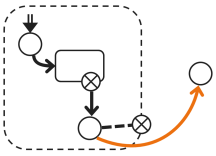

図 7: exit point から出る遷移の展開後

3.3.4 FST 書式への変換

FST 書式に変換が可能であることにより， MMDAgent に手を加えなくても提案書式で記述したプログラムを実行可能という効果がある．

FST 書式へ変換を可能にするために提案書式が満たすべき条件は，提案書式で表現される状態遷移機械が決定性をもつことである．状態遷移機械が決定性をもつこととは，現在の状態と入力によって遷移先状態と出力が一意に決まることである．

複合状態が含まれていても，決定性をもっていれば状態遷移機械が取りうる現状態は 1 つである．そのため，複合状態を展開して複合状態のない状態遷移に置き換えれば FST 書式に変換できる．

複合状態を展開する方法は次の通り．

a） 複合状態へ入る遷移の場合

複合状態の初期状態に遷移をつなぎかえる

b） 複合状態の外へ強制的に出る遷移の場合

複合状態の全ての内部状態から遷移先の状態へ遷移をつなぎかえる

c） 複合状態の終了状態から外へ出る遷移の場合

終了状態から遷移先の状態へ遷移をつなぎかえる

複合状態に入遷移がつながっている場合の展開例を図 4，5 に示す．図 4 のように，ある状態から複合状態に入る遷移がある場合，図 5 のように，複合状態の初期状態につなぎかえる．

出遷移が複合状態の exit point につながっている場合の展開例を図 6，7 に示す．図 6 のように複合状態の exit point から，ある状態に出る遷移がある場合，図 7 のように，その exit poin につながっている全ての状態から，遷移先の状態に遷移をつなぐようにする．

出遷移が複合状態に直接つながっている場合の展開例を図 8，9 に示す．図 8 のように，複合状態からある状態に出ていく遷移がある場合，図 9 のように，その複合状態の内部状態の全てから外の状態に出て行くように遷移をつなぎかえる．

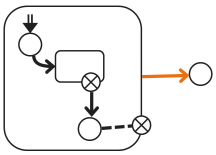

図 8: 複合状態から出る遷移の展開前

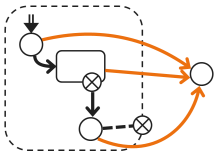

図 9: 複合状態から出る遷移の展開後

4. プロトタイプ

4.1 プロトタイプ: XFST 書式

プロトタイプとして XFST 書式を考案した．XFST 書式は XML 形式で独自のタグを定義したものである．

XFST 書式の例をソースコード 1 に示す．

ソースコード 1: XFST 書式の例

```
1  <xfst>
2    <template>
3      <var_list>
4        <var name="agent">mei</var>
5      </var_list>
6      <template_list>
7        <template name="Initialize"
8          import="./initialize.xfst" />
9        <template name="Contents">
10         <var_list>
11           <var name="agent"><val name="$1" /></var>
12           <var name="speech"><val name="$2"></var>
13         </var_list>
14         ...（中略）...
15       </template>
16     </template_list>
17     <inner_state_list>
18       <composite name="initialize"
19         base="Initialize">
20       </composite>
21       <composite name="contents"
22         base="Contents">
23         <arg_list>
24           <arg><val name="agent"></arg>
25           <arg>おはようございます．</arg>
26         </arg_list>
27       </composite>
28     </inner_state_list>
29     <initial_state name="initialise"/>
30     <exit_point_list />
31     <transition_list>
32       <transition>
33         <prev>initialize.exit</prev>
34         <next>contents</next>
35         <event>eps</event>
36         <command>eps</command>
37       </transition>
38     </transition_list>
39   </template>
40 </xfst>
```

2 行目から 39 行目や 7, 8 行目，9 行目から 15 行目で複合状態テンプレートが定義されている．7, 8 行目では `import` 属性に `./initialize.xfst` が指定されている．このように，外部ファイルで定義されている複合状態テンプレートを読み込むことができる．

17 行目から 28 行目では複合状態テンプレートに引数を与えて複合状態を生成している．ここでは 18 行目からの部分に注目して説明する．

21, 22 行目で複合状態の属性 `base` に `Contents` が指定されている．これは 9 行目で定義した `Contents` テンプレートをもとに複合状態を生成することを意味している．そして 23 行目以降で `Contents` テンプレートに渡す引数を指定している．これらの値は `Contents` テンプレートが定義されている 10 行目からの部分で変数に代入されている．引数は順番に，`$1`, `$2`, ... という変数名で参照することができる．

4.1.1 外部ファイルのインポート機能

XFST 書式では外部ファイルで定義された複合状態テンプレートをインポートする機能を有する．この機能により，よく利用する複合状態テンプレートをライブラリとして作成し，適宜インポートするようにして再利用しやすくなっている．

4.2 変換システム

4.2.1 変換システムの概要

XFST 書式で書かれたテキストファイルを FST 書式に変換するプログラムを Java で実装した．

このプロトタイプシステムを使用して MMDAgent に対話シナリオを読み込ませるまでの流れは以下のとおりである．

（1） XFST 書式で対話シナリオを作成

（2） プロトタイプシステムを用いて FST 書式に変換

（3） 生成された FST 書式のファイルを MMDAgent に読み込ませる

4.2.2 システム構成

システム構成図を 10 に示す．

図 10: 変換システムの構成図

まず XFST 書式ファイルを XFST パーサに読み込ませ，デシアライズし，XFST オブジェクトとして出力する．XFST オブジェクトは XFST 書式で記述されたファイルの記述内容をそのまま表現するデータである．

次に XFST オブジェクトをテンプレート展開器に通して，複合状態遷移オブジェクトを出力する．テンプレート展開器は複合状態テンプレートのパラメータ部分を値に置き換える．

さらに 複合状態遷移オブジェクトを複合状態展開器に通して，遷移リストを出力する．複合状態遷移オブジェクトに存在する複合状態を展開し複合状態をなくす．出力される遷移リストの状態の識別子はこの段階ではまだ文字列である．

その後，遷移リストを状態番号割り当て器を通して，状態の識別子が数値である遷移リストを出力する．

最後に FST 書式出力器を通して FST 書式ファイルを出力する．

5. 評価実験

5.1 実験の目的

実験の目的は，提案システムが FST の問題点を解決し音声対話の編集が容易になったかを検証することである．今回の実験によって評価する項目は，読みやすさ，書きやすさ，再利用のしやすさ，バグの出にくさである．

5.2 実験方法

5.2.1 実験概要

指定した内容の対話を再現する対話シナリオを XFST 書式を用いた場合と FST 書式を用いた場合の両方を被験者に作成してもらった．作成にかかった時間と実験後のアンケートから各書式の評価を行った．

被験者として本研究室の学生 6 人に参加してもらった．被験者は，研究等で FST 書式の利用経験があるが，XFST 書式については，ゼミの発表を聞いて概要は知っているものの詳しい仕様については知らない者を選んだ．

作成した書式の順番に影響されないように，被験者をグループ A，グループ B の半々にわけ，グループ A には先に FST 書式で対話シナリオを作成してもらい，その後 XFST 書式で対話シナリオを作成してもらった．反対に，グループ B には先に XFST 書式で対話シナリオを作成してもらい，その後 FST 書式で対話シナリオを作成してもらった．

被験者は FST 書式については研究で利用しているため，FST 書式の仕様については説明をおこなわなかった．反対に，XFST 書式については約 20 分の説明をおこなった．

5.2.2 作成してもらった対話内容

ここからは，被験者に作成してもらった対話内容について説明する．作成してもらった対話内容は,「ユーザが「クイズ」と言ってきたら，画面上の 3D キャラクターのメイちゃんがクイズを 8 問出してくれるコンテンツ」とした．

クイズは 3 択問題と自由解答問題を交互に出す形式にした．ユーザが答えたらメイちゃんが正誤を伝え次の問題に進むという流れとした．

5.2.3 ライブラリの用意

再利用性について調査するため，今回の実験では，XFST 書式，FST 書式それぞれのライブラリと利用サンプルをあらかじめこちらで作成し，被験者に利用してもらった．あらかじめ用意したファイルは FST も XFST も問題 2 まで実装してあり，2 問目が終わるとそのまま終了してユーザの「クイズ」の発生を待つようになっている．

5.2.4 プログラムの作成について

対話プログラムの作成は，あらかじめ用意したサンプルを追加編集をしてもらうというようにした．被験者には，作成してもらったシナリオを MMDAgent に読み込ませ動作確認をして，被験者自身がバグがないと確信した段階で完成とした．

編集時間として計測するのは，シナリオを編集している時間と，シナリオを読んでいる時間を編集時間とした．MMDAgent を動かして動作の確認をしている時間は編集時間に含めなかった．

5.2.5 アンケートについて

実験の後，被験者にはアンケートに回答してもらった．アンケートは 7 段階評価によって回答してもらった．評価基準は「1: 大変そう思う，2: そう思う，3: どちらかといえばそう思う，4: どちらともいえない，5: どちらかといえばそう思わない，6: そう思わない，7: 全くそう思わない」として回答してもらった．

次にアンケート項目を表 1 に示す．アンケートの評価値が高いほど「良い評価」と言えるようになっている．

表 1: アンケート項目

項目番号	質問内容
1	プログラムの挙動を想像するのが難しかった
2	どこを編集するべきか分かりにくかった
3	単純な作業に時間がかかって面倒だった
4	編集時に気を配るところが多くて大変だった
5	なかなか思った通りにプログラムが動かなかった
6	ライブラリを利用してもらくにならなかった編集時に気を配るところが多くて大変だった
7	バグが発生しやすいと感じた
8	バグがあったときに原因箇所の特定が難しかった

5.3 実験結果と考察

5.3.1 編集時間について

各書式での編集時間の平均を表 2 に示す．

表 2: 編集時間の平均

	FST（秒）	XFST（秒）
グループ A	1451	450
グループ B	1342	1010
両グループ	1397	730

表 2 より，グループ A, B どちらとも FST 書式の編集にかかった時間と比べ XFST 書式の編集にかかった時間の方が短いことが分かる．両グループ合わせた平均時間でみると XFST 書式の編集にかかった時間は FST 書式の編集にかかった時間の半分になった．

被験者は全員研究で FST 書式を利用している人であり，反対に XFST 書式はほとんど利用したことがないのにも関わらず，被験者 6 人のうち 5 人は XFST 書式の編集時間の方が短かった．これより，おなじ内容の対話シナリオでは XFST 書式の方が速く作成できるのではないかと考えられる．

5.3.2 アンケート結果について

アンケートの回答結果を平均して棒グラフにしたものを図 11 に示す．評価基準はアンケートは全ての項目で，評価値が高い方が良い評価と言えるようになっている．

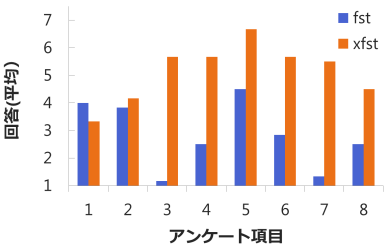

図 11: アンケート結果

項目 1 の評価値の平均は，FST 書式で 4.00，XFST 書式で 3.33 となっている．また，項目 2 の評価値の平均は，FST 書

式で 3.83，XFST 書式で 4.17 となっている．このように項目 1, 2 では XFST 書式と FST 書式の間で評価の差が小さかった．これより，FST 書式に慣れた人にとって XFST 書式の編集は，FST 書式の編集と変わらない感覚でおこなえると考えられる．項目 1 について XFST 書式の評価値がやや下回っているが，これは被験者が XFST 書式の編集がほぼ初めてだったためと考えられる．

次に，項目 3 の評価値の平均は FST 書式で 1.16，XFST 書式で 5.67 となっている．また，項目 4 の評価値の平均は FST 書式で 2.50，XFST 書式で 5.67 となっている．このように項目 3, 4 の結果から，今まで FST 編集時に面倒・大変であったりした部分が，XFST 書式では改善されて，プログラムが書きやすくなったことが分かる．

また，項目 5 の評価値の平均は FST 書式で 4.50，XFST 書式で 6.83 と良くなっている．この結果より，FST 書式と比較して XFST 書式の方が思った通りに動くプログラムを書きやすいことが分かった．

さらに，項目 6 の評価値の平均は，FST 書式で 2.83，XFST 書式で 6.67 と良くなっている．この結果から，複合状態テンプレートを利用した場合，編集が楽になることが確認できた．FST 書式では，あらかじめよく使うパターンのサンプルが用意してあっても変更しなければ多くて楽にはない．しかし，XFST 書式では複合状態テンプレートの機能を用いることで実装が楽なったと考えられる．

そして，項目 7 の評価値の平均は，FST 書式で 1.33，XFST 書式で 5.5 と良くなっている．この結果から，普段利用している人でも FST 書式ではバグが発生しやすいと大変感じていたが，XFST 書式では改善されているということが分かる．

最後に，項目 8 の評価値の平均は，FST 書式で 2.5 XFST 書式で 4.5 と良くなっている．この結果から，FST 書式ではバグの原因箇所を特定するのがやや難しかったが，XFST 書式では改善されてどちらともいえないくらいにはなったということが分かる．

このように，ほぼ全ての項目で FST 書式より XFST 書式の方が良い評価を得ており，FST 書式で大変だったり面倒だったりした部分が XFST 書式では改善されていることが分かった．

6. おわりに

6.1 まとめ

音声対話システム MMDAgent における従来の対話シナリオ記述書式である FST 書式の問題を解決するため，本論文では音声対話シナリオのための複合状態をもつ状態遷移記述手法を提案した．

FST 書式では構造的なプログラムを書くことは難しく，そのため以下のような問題が発生していた．

- 状態番号の意図しない重複に特に注意する必要がある
- プログラムを読むのが困難で，バグの原因箇所の特定も難しい
- 同様な処理をする部分の再利用が難しい

この問題を解決するため，本論文では音声シナリオのための複合状態をもつ状態遷移記述手法を提案した．提案した状態遷移記述手法には以下のような特徴がある．

- 複合状態を導入
- 複合状態テンプレート
- 状態を数値ではなく文字列で識別する
- FST 書式に変換可能

複合状態を導入することで階層的な状態遷移を記述となり階層的な構造をもつプログラムを書けるようになった．また，複合状態テンプレート機能によって，パラメータが違うだけで遷移パターンは同じである複合状態を少ない記述量で表現できるようになった．これはプログラミング言語でいわれる抽象化に値する．そして，FST 書式に変換可能であるという特徴から MMDAgent に手を加えることなく，提案手法を用いた対話シナリオを利用することができる．

また，プロトタイプシステムを実装し，実験を実施した．

実験では同じ内容のプログラムを XFST 書式と FST 書式で作成してもらい，編集時間の計測とアンケートをおこない XFST 書式と FST 書式を比較した．

XFST 書式での編集時間は平均で 730 秒で，FST 書式での編集時間は平均で 1397 秒となり，XFST 書式での編集時間が約半分にまで減少した．この結果より，XFST 書式の方が早く実装できるようになったことが分かった．

アンケートより，今まで FST 編集時に面倒であったり大変であったりした部分が，XFST 書式では改善されて，プログラムが書きやすくなったことが分かる．

さらに，複合状態テンプレートを利用した場合，編集が楽になることが確認できた．FST 書式では，あらかじめよく使うパターンのサンプルが用意してあっても変更しなければ多くて楽にはない．XFST 書式では複合状態テンプレートの機能を用いることで実装が楽なったと考えられる．

ほかのアンケート項目でもほぼ全ての項目で FST 書式より XFST 書式の方が良い評価を得ており，FST 書式で大変だったり面倒だったりした部分が XFST 書式では改善されていることが分かった．

6.2 今後の課題

重み付き状態遷移機械においてさまざまな演算について研究されている [9]．これらの研究から OpenFST [10] というプロジェクトで C++ で演算プログラムが実装されている．複合状態をもつ状態遷移機械においてもこのような演算のいくつかが可能であると考えられる．演算についても検証し，提案書式に取り入れることが今後の課題としてあげられる．

また，今回考案した XFST 書式は，XML でタグ付けしていく形式であるため記述するのは煩雑であると感じている．XML でのタグ付けではなく，よりプログラミング言語らしい記述方法を考えることも課題として挙げられる．

また FST 書式に変換するのではなく，直接 XFST 書式を実行する実行環境を作成することで，変換の手間を減らせるだけでなく，対話シナリオの新たな記述手法を用いることができる可能性がある．XFST 書式をを直接実行する実行環境の実装し新たな記述手法について検証することも課題として挙げら

れる．

文　　献

[1] Siri. http://www.apple.com/ios/siri/. Accessed: 2016-01-11.

[2] しゃべってコンシェル. https://www.nttdocomo.co.jp/service/information/shabette_concier/index.html. Accessed: 2016-01-11.

[3] MMDAgent. http://www.mmdagent.jp. Accessed: 2016-01-11.

[4] 大浦圭一郎, 山本大介, 内匠逸, 李晃伸, 徳田恵一. キャンパスの公共空間におけるユーザ参加型双方向音声案内デジタルサイネージシステム (¡特集¿音声対話システムの実用化に向けて). 人工知能学会誌, Vol. 28, No. 1, pp. 60–67, jan 2013.

[5] Voice Extensible Markup Language (VoiceXML) 2.1. http://www.w3.org/TR/voicexml21/. Accessed: 2016-01-11.

[6] 桂田浩一, 中村有作, 山田真, 山田博文, 小林聡, 新田恒雄. MMI 記述言語 XISL の提案. 情報処理学会論文誌, Vol. 44, No. 11, pp. 2681–2689, nov 2003.

[7] 桂田浩一, 中村有作, 山田真, 小林聡, 山田博文, 新田恒雄. 音声対話記述言語 VoiceXML と MMI 記述言語 XISL の比較. 情報処理学会研究報告音声言語情報処理（SLP）, Vol. 2001, No. 100, pp. 49–54, oct 2001.

[8] 西村良太, 山本大介, 打矢隆弘, 内匠逸. 音声対話エージェントのための web ブラウザを用いたシナリオエディタの開発. マルチメディア、分散協調とモバイルシンポジウム 2013 論文集, Vol. 2013, pp. 1796–1799, jul 2013.

[9] Mehryar Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pp. 213–254. Springer, 2009.

[10] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In Jan Holub and Jan rek, editors, *Implementation and Application of Automata*, Vol. 4783 of *Lecture Notes in Computer Science*, pp. 11–23. Springer Berlin Heidelberg, 2007.