排他制御機能を有する状態遷移機械に基づく音声対話コンテンツ制御手法

石川　博規† 　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: †hiroki@moss.elcom.nitech.ac.jp, ††{tsutsumi.shuhei,yamamoto.daisuke,naohisa}@nitech.ac.jp

あらまし　我々は音声インタラクションシステム構築ツールキット MMDAgent [1] を用いた研究を行ってきた．MMDAgent は FST(有限状態遷移機械) [12] に基づいた FST ファイルと呼ばれる対話スクリプトファイルを編集することで自由に音声対話内容を編集することができる [2]．従来の MMDAgent においては、特定の目的と状況に合わせてその都度 1 つの FST ファイルを作成することが一般的であった．そこで，本研究では，対話シナリオを独立した対話内容を持つ複数の FST ファイルを分割し，並行制御を行う手法を提案する．提案手法によって，FST ファイルの保守性や移植性が高まると考えている．

キーワード　音声対話システム, MMDAgent, FST, デジタルコンテンツ, コンテンツ管理, モジュール化, 排他制御

1. はじめに

近年，モジュール化されたコンテンツを配信する GooglePlay や AppStore 等のシステムやサービスが普及している．これらのシステムやサービスでは，独立したモジュール化されたコンテンツをユーザが任意に切り替えることで，自分好みの機能を容易に構築していくことが可能となっている．一方で我々は音声インタラクションシステム構築ツールキット MMDAgent [1] を用いた研究を行ってきた．MMDAgent は音声認識，音声合成，3D モデル描写を高度に統合したマルチモーダルな音声対話システムであり，FST ファイルと呼ばれるテキスト形式の対話スクリプトファイルを編集することで自由に音声対話内容を編集することができる [2]．MMDAgent の実行画面の例を図 1 に示す．音声対話を記述する言語としては XML ベースの VoiceXML [9] や XISL [10] 等がある．また，VoiceXML を用いたマルチモーダル音声対話システムを構築する試み [11] もあるが，MMDAgent においてはテキストベースの FST ファイルを用いて音声対話を記述する．FST ファイルは FST(有限状態遷移機械) [12] に基づいており，FST ファイルと FST は 1 対 1 に対応し，FST ファイルを読み込んだ数だけ FST が生成される．FST ファイルは図 2 のような形式で記述される．状態番号，遷移先状態番号，遷移条件 (入力イベント)，遷移時のコマンド (出力コマンド) の 4 つ組で表せられる．FST は独立した現在の状態番号を持ち，状態番号と遷移条件となっている入力イベントが共に合致した時に状態遷移を実行する．実行中の各 FST は MMDAgent の対話管理部によって制御されている．従来の MMDAgent においては，特定の目的と状況に合わせてその都度 FST ファイルを作成することが一般的であった．しかし，そのような利用法では状態数の増加によって，対話シナリオの編集が煩雑となる傾向があり，また，FST ファイルの保守性が低いといった問題があった．MMDAgent とは異なる音声対話システムではあるが，豊橋技術科学大学の小暮らのグループによる先行研究 [13] においても音声対話システムを容易にマルチタスクで使用できるようする必要性が述べられている．また，FST ファイルの保守性を向上させるため，FST ファイルを対話シナリオに応じて分割し，並列実行する方法も考えられていた．しかし，従来では後述する FST 競合といった問題から，FST ファイルの並列実行を効果的に実現することが出来なかった．そこで，本研究では，分割され独立した対話シナリオを持つ FST ファイルを，モジュール化した FST ファイルと定義し，複数のモジュール化した FST ファイルを並行制御する手法を実現することを目的とする．これにより，個々の FST ファイルに記述される状態数が少なくなり保守性が向上した．また，目的に合わせて動作させる FST ファイルを動的に変更することも可能になった例えば，図 3 のように，大学の南門と北門それぞれにおいて構内を案内する音声対話システムを考えると，従来では構内の案内を 1 つの FST ファイルで網羅した結果，数千行という膨大なサイズとなっていたものが，個々の建物の案内に分割することで数十行単位までサイズが小さくなり，保守性を高められると考えられる．また，1 号館や 2 1 号館の案内のように両方の門で共通している FST ファイルを，それぞれの音声対話システムに容易に追加することができると考えられる．

提案手法を実現する上で，従来の MMDAgent において次の

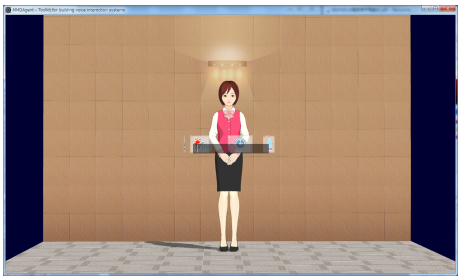

図 1　MMDAgent の実行画面

ような問題点が挙げられる．

問題点 1(FST 競合)

全ての FST が独立かつ並列に動作するため，複数の FST を実行した場合に FST ファイルの作成者が予期しない動作をする可能性がある．

問題点 2(FST 協調の欠如)

1 つの入力イベントに対して，複数の FST が協調的に動作することができない．

FST 競合の例を図 4 に示す．同じ認識キーワードで状態遷移する複数の FST が実行中であるとき，従来の MMDAgent では遷移条件に合致した全ての FST で同時に状態遷移が起きてしまい，結果として音声合成やモーションなど同時に 2 つ以上が共存できないコマンドがほぼ同時に実行されてしまう問題があった．ほぼ同時に音声合成やモーションのコマンドが実行されると後から実行されたコマンドで上書きされてしまうため，音声再生の途中で次の音声再生が始まってしまう，モーションの途中で次のモーションが始まってしまうといった現象が発生する．これらはどれもシステムのユーザにとって不自然な動作に見えてしまうと考えられる．

次に FST 協調の欠如の例を図 5 に示す．従来の MMDAgent では，1 回の音声入力に対して複数の FST を協調的に動作させたい時，例えば「今日の天気と占いと教えて」という音声入力に対して天気予報をする FST と占いをする FST を順番に実行するといったことができない問題があった．より具体的には，従来の MMDAgent において 1 つの FST ファイルに天気予報と占いの両方の音声対話を記述した場合，「今日の天気と占いを教えて」という音声入力に対して天気予報と占いのそのどちらかしか実行されない．また，2 つの FST ファイルにそれぞれ天気予報と占いの音声対話を記述した場合，前述の FST 競合が発生し，不自然な動作となってしまう．

そこで本研究ではこれらの問題点を解決するため，次のアプローチをとる．

アプローチ 1

FST に優先度を付与し，それと同時に MMDAgent の対話管理部へ排他制御を導入し FST の状態遷移を制御する．

アプローチ 2

アプローチ 1 に合わせて，各 FST への入力イベントを一時的に保存するキューを用いて，順序良く FST を状態遷移できるようにする．

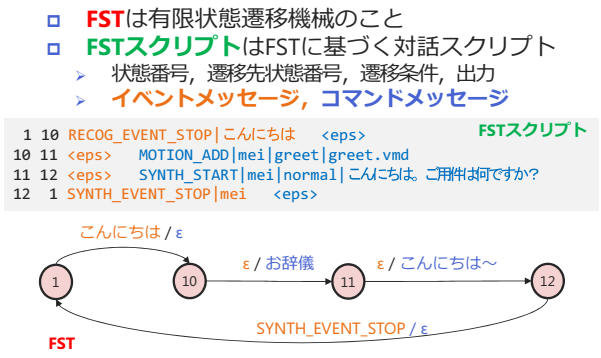

図 2　FST ファイルの記述形式

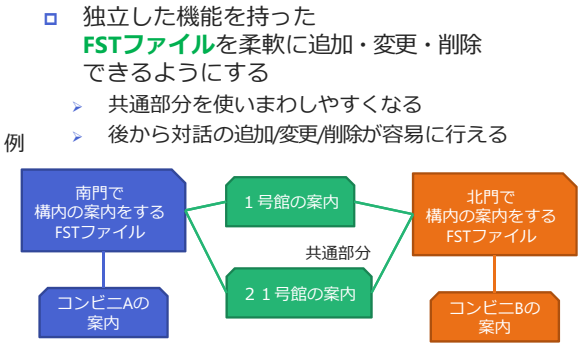

図 3　モジュール化された FST ファイルの活用例

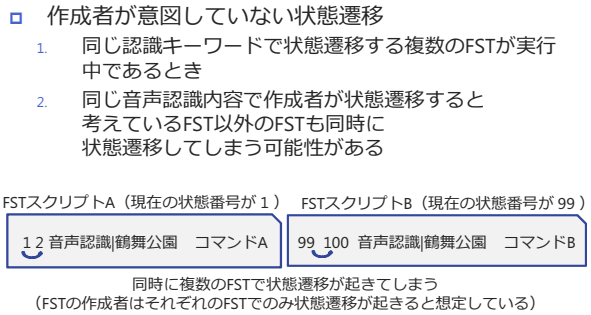

図 4　FST 競合の例

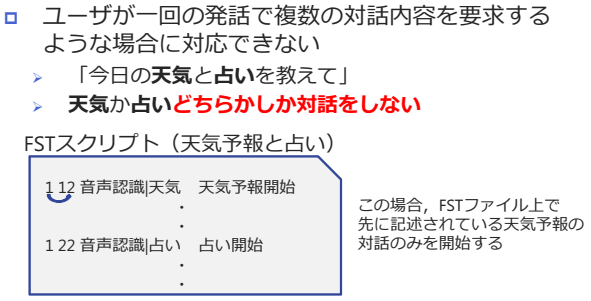

図 5　FST 協調の欠如の例

2.　実現上の課題

1. のアプローチに基づき MMDAgent の対話管理部を改良する上で次のような実現上の課題が挙げられる．

実現上の課題 1

従来の MMDAgent の対話管理部では，入力イベントを図 6 のように全ての FST に同一処理フレームで流し込むようになっている．また，実行中の FST 群の処理順が固定されている．さらに，各 FST は実行中の他の FST の状態を考慮していない．そのため，状態遷移条件に合致した全ての FST で状態遷移が並列に実行されることになり，FST ファイルの作成者が意図していない不自然な動作をする可能性がある．

実現上の課題 2

従来の MMDAgent の対話管理部では，前述のように入力イベントを全ての FST に同一フレームで流し込むようになっており，また，入力イベントは処理フレーム終了後に破棄されている．そのため，入力イベントに合致する FST の状態遷移をタイミングをずらして実行するようなことが出来ない．

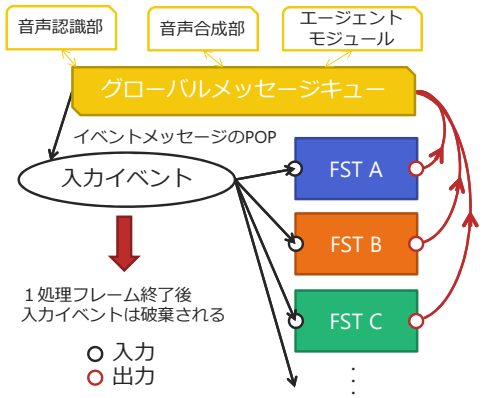

図 6　従来の MMDAgent の対話管理部

3. 実 現 手 法

3.1 FST の優先度付けと排他制御

MMDAgent において，FST はクラスを用いて実装されており，実行時には FST の数だけクラスからオブジェクトがインスタンス化される．MMDAgent の対話管理部はそれらオブジェクトをリスト構造で保持し制御している．従来の MMDAgent の対話管理部での FST の制御方式では，リストの先頭から順番に全ての FST に入力イベントを与えて状態遷移を行っていた．そこで，先頭から順番に処理をする部分は変えずに，FST のリストを動的に入れ替えられるようにすることに加え，同一処理フレームでは最初に遷移条件を満たした FST のみを状態遷移させ，以降の FST の処理をスキップする制御方式に変更した．これにより，同一処理フレーム内では状態番号と入力イベントから遷移条件を満たしている FST を最大 1 つだけ状態遷移させることになる．また，リストの最初から順番に処理を行うためリストの順番を入れ替えることで FST の優先度を変更することができる．

排他制御を行う手法として，FST を Active と Inactive の 2 つの状態に分けるようにした．各状態は FST の待機状態番号と現在の状態番号から判定する．待機状態番号とは MMDAgent がユーザの入力を待っておりかつ対話の途中でない状態番号を指す．主に対話開始のトリガーとなっている音声入力待ちの状態番号が待機状態番号に当たる．Active/Inactive の 2 状態は次のように定義される．

Active　FST の現在の状態番号が待機状態番号以外

Inactive　FST の現在の状態番号が待機状態番号

Active な状態は対話を実行中である状態を指し，また，Active な FST は同時に 1 つまでしか存在しないように制御する．そして，Active な FST を優先して状態遷移させる．Active/Inactive を用いた排他制御のアルゴリズムを次に示す．

- 全ての FST に対して入力イベントと現在の状態番号から遷移条件をみたしているか判定する
- Active な FST がないときは，優先度順に各 FST に対して次のことを実行する
  - 遷移条件を満たしているとき
    - 状態遷移させる
    - 遷移後の状態番号を見て，待機状態番号以外であったら Active な状態に切り替える
  - 1 回でも状態遷移が実行されていたら，そのフレームの状態遷移処理を終了する
- Active な FST があるときは，Active な FST に対してのみ次のことを実行する
  - 遷移条件を満たしているとき
    - 状態遷移させる
    - 遷移後の状態番号を見て，待機状態番号であったら Inactive な状態に切り替える

3.2 イベントキューを用いた状態遷移の遅延実行

従来の MMDAgent の対話管理部では入力イベントを同一処理フレームで全ての FST に流し込んだ後，破棄するという制御を行っていた．本研究における提案手法では 3.1 で述べた FST の優先度付けと排他制御に加え，1 処理フレームで入力イベントを破棄するのではなく，適切なタイミングで FST に入力イベントを流し込み状態遷移を制御するという状態遷移の遅延実行を行う．実際には何度も遷移条件に合致しているのか判定することを防ぐため，状態遷移情報を保持し，適切なタイミングでその情報を用いて FST を状態遷移させるようにしている．状態遷移情報には遷移先状態番号と出力コマンドが含まれている．このような情報を保持するキュー構造をイベントキューと定義する．また，イベントキューは FST のクラスに含まれている．イベントキューを用いた状態遷移の遅延実行のアルゴリズムを次に示す．

- 全ての FST に対して入力イベントと現在の状態番号から遷移条件をみたしているか判定する
- Active な FST がないとき
  - 優先度順に各 FST に対して先にイベントキューを用いた状態遷移を行う
    - イベントキューから状態遷移情報を pop 可能であれば，状態遷移情報を pop する
      - pop された状態遷移情報があった場合
        - 状態遷移させる
        - 遷移後の状態番号を見て，待機状態番号以外であったら Active な状態に切り替える
        - イベントキューをクリアする
    - 1 回でも状態遷移が実行されていたらそのフレームの状態遷移処理を終了する
  - 優先度順に各 FST に対して次のことを実行する
    - 遷移条件を満たしているとき
      - 状態遷移させる
      - 遷移後の状態番号を見て，待機状態番号以外であったら

Active な状態に切り替える
　＊　1 回でも状態遷移が実行されていたら，そのフレームの状態遷移処理を終了する
- Active な FST があるとき
– 全ての Inactive な FST において，

遷移条件を満たしているとき，状態遷移情報を push する
– Active な FST が遷移条件を満たしているとき
　＊　状態遷移させる
　＊　遷移後の状態番号を見て，待機状態番号であったら Inactive な状態に切り替える

4. プロトタイプシステム

4.1 プロトタイプシステム概要

既存の MMDAgent のプログラムの一部を改造する形でプロとタイムシステムの実装を行った．MMDAgent のプログラムは基本的に C++で記述されており，本研究で実装した内容も全て C++で記述した．MMDAgent は音声認識，音声合成，3D 描写，対話管理などの各機能がモジュール化されており，それぞれプラグインとして実装されている．そこで本研究においては対話管理を行うプラグインのみを改造し，それ以外の部分は変更していない．これにより対話管理部以外の機能に変更があっても容易にマージすることができる．また，プロトタイプシステムでは対話管理部において，3 つの排他制御レベルを選択可能にするように考えている．そのようにする理由としは，排他制御を実装したことで従来の MMDAgent で可能であった対話処理ができなくなる場合があるからである．そこで，FST ファイルの用途と内容によって FST ファイルの作成者が排他制御のモードを選択可能にしたほうが良いと考えた．3 つの排他制御モードは，NoSyncMode，QueueSyncMode，SyncMode である．各排他制御モードは FST ファイル単位で設定することができるようにする．FST ファイルにソースコード 4.1 のように記述することで，排他制御のモードを設定する．@マークが先頭につけられた行が FST の排他制御モードの設定するコマンドとなっている．また，3.1 で述べた待機状態番号もソースコード 4.1 のように FST ファイル中に記述することで，待機状態番号の範囲を FST ファイルごとに設定することができる．

ソースコード 4.1 待機状態番号と排他制御モードの設定

```
# 待機状態番号の設定
@ WAITSTATE 0 1
# 排他制御モードの設定
@ TRANSITIONMODE QueueSync

0    31    RECOG_EVENT_STOP|こんにちは     <eps>
...
```

4.2 各排他制御モードの実装

次に，各排他制御モードのプロトタイプシステムにおける実装を述べる．

NoSyncMode

NoSyncMode では排他制御を行わない従来の MMDAgent と同じ制御方式である．Active/Inactive 関係なく全ての FST を独立かつ並列に実行する．入力イベントが全ての FST に同一処理フレームで流し込まれる制御を行う．このモードを使用する場面としては，音声認識内容をカウントするだけの FST ファイルを実行する場面などが考えられる．例えば，「こんにちは」と音声認識した回数をカウントするだけで，音声合成やモーションの再生等の 2 つ以上が同時に共存不可能なコマンドを使用しないのであれば，排他制御を行わなくても 1. で述べたような FST 競合は発生しないと考えられる．また，このような内容の FST ファイルを排他制御した場合，正しいカウントが取れないなど意図した通りの結果が得られないことが考えられる．

QueueSyncMode

3.1 で述べたアルゴリズムを用いて FST の排他制御を行う．また，それに加えて 3.2 で述べたアルゴリズムを用いて FST の状態遷移の遅延実行を行う．このモードでは排他制御により，同時に 2 つ以上の FST を並列実行しない．Active な FST を優先して状態遷移させ，また，Active な FST は 1 つまでしか存在しないように制御する．しかし，Inactive な FST が遷移条件を満たす入力イベントがあったとき，イベントキューに遷移情報を push し，Active な FST が存在しなくなったときに順次 pop するため，入力イベントの取りこぼしが起きない．また，1 つの入力イベントによって複数の FST が優先度順に順次状態遷移するような制御が可能である．このモードを使用する場面としては，ユーザが 1 回の発話で複数の対話内容を要求した時が考えられる．例えば，「1 号館と 2 1 号館がどこにあるか教えて.」というユーザの発話に対して音声認識部は 1 号館、2 1 号館といったキーワードを認識する．従来の MMDAgent の対話管理部では，1 つの FST ファイルに両方の記述があった場合は入力イベントが 1 処理フレームで破棄されてしまう問題からどちらか一方の案内しかせず，また，それぞれ別の FST ファイルに記述した場合は入力イベントを常に全ての FST に流し込んでしまう制御の問題から FST 競合が発生していた．提案システムの QueueSyncMode では FST の排他制御とイベントキューによる状態遷移の遅延実行により，各案内を別の FST ファイルに記述すると，それぞれの案内を優先度順に順番に実行することができ，より自然な音声対話を可能にする．

SyncMode

3.1 で述べたアルゴリズムを用いた FST の排他制御を行う．ただし，こちらのモードではイベントキューによる状態遷移の遅延実行は行わないため，Active な FST が存在するときは Inactive な FST では遷移条件に合致した入力イベントがあっても状態遷移を行わない．これにより入力イベントの取りこぼしが発生する可能性があるが，3 つの排他制御モードの中で最も排他制御のレベルが高い制御方式であると考えられる．

5. 評 価 実 験

5.1 目的と方法

評価実験の目的は，作成したプロトタイプシステムを用いて実験を行い，複数のモジュール化された FST ファイルを並行実

行した場合に MMDAgent の対話管理部が改良されているかどうか評価することである．また，上手く動作しなかったパターンが存在した場合は，可能ならばその原因と対処法について考察する．

実験で用いた FST ファイルは次の 6 個である．

2goukan.fst

2 号館を案内する FST ファイル．案内パネルは用いない．

51goukan.fst

51 号館を案内する FST ファイル．案内パネルを使用する．この FST ファイルでは，案内が開始されると標準で表示されているメニューパネルが削除され，代わりに 51 号館の位置を示すパネルが追加される．そして，案内が終了すると 51 号館のパネルが削除され，再び標準のメニューパネルが追加されるという流れで動作する．

seiza.fst

黄道 12 星座の説明をする FST ファイル．簡単化のため，今回の実験では双子座のみの説明を行う．

uranai.fst

黄道 12 星座に基づいた星座占いを行う FST ファイル．簡単化のため，今回の実験では占うのはふたご座のみで，占いの結果は完全ランダムである．

weatherforecast.fst

明日の天気予報を行う FST ファイル．簡単化のため，今回の実験では天気予報の結果は完全ランダムである．

countup.fst

音声認識の回数をカウントする FST ファイル．カウントを確認できるように MMDAgent のプログラムを若干変更している．SYNTH_EVENT_START（何かしら音声認識を開始すると発行されるイベント）が発行された回数をカウントして画面上に表示する．

6 個のモジュール化された FST ファイルを用意し，それら 6 個の中から 3 個を選択し並行実行する．実行中の各 FST ファイルに対して，正常に対話が可能かどうか検証する．正確に測定するため対話に用いる入力としてキーボードによる入力を用い，擬似的に音声入力を再現した．FST ファイルの組み合わせは全部で ${}_6C_3 = 20$ 通りとなる．さらに，各組み合わせに対して次のようなパターンで実験を行う．

パターン A　全て従来手法で実行した場合（3 個全て NoSyncMode で排他制御）

パターン B　全てイベントキューなしの排他制御手法で実行した場合

（3 個全て SyncMode で排他制御）

パターン C　それぞれの FST ファイルの排他制御モードを表 1 のようにした場合

よって，全体の実験総数は $20 \times 3 = 60$ 通りとなる．

パターン C をこのように設定した理由を述べる．2goukan.fst，51goukan.fst，weatherforecast.fst を QueueSyncMode にした理由としては，これらの FST ファイルの間では認識キーワードなどの対話が開始する遷移条件となっている入力イベントが重複していないことが挙げられる．基本的には入力イベントの取りこぼしが発生しない QueueSyncMode を用いた方がより自然な対話を実行可能であると考えている．例えば，パネルを出して説明する 51goukan.fst は Active 状態で説明を続ける時間が比較的長く，対話の途中で 2 号館の案内や天気など，別の FST についてユーザから入力がある場合が考えらるが，そのような場合であっても，QueueSyncMode であれば，現在 Active 状態になっている FST ファイルの対話が終了してから，途中で入力のあった FST ファイルの方へと切り替えていくことが可能である．また,「2 号館と 5 1 号館を案内して」といった入力があった場合，2goukan.fst と 51goukan.fst の両方で遷移条件が満たされることになるが，そのような場合であっても，2 号館と 5 1 号館の案内を両方とも順番待ちをして実行していくことが可能である．

表 1　パターン C の排他制御モード

FST ファイル	排他制御モード
2goukan.fst	QueueSyncMode
51goukan.fst	QueueSyncMode
seiza.fst	SyncMode
uranai.fst	SyncMode
weatherforecast.fst	QueueSyncMode
countup.fst	NoSyncMode

seiza.fst と uranai.fst を SyncMode にした理由としては，これらの FST ファイル間で対話開始の遷移条件となっている入力イベントが重複していることが挙げられる．seiza.fst の方は,「ふたご座」と入力すると MMDAgent がふたご座自体の説明をする．一方 uranai.fst の方は,「ふたご座」と入力すると生まれ月がふたご座に対応している人の運勢について MMDAgent が話す．これらの FST ファイルを QueueSyncMode で実行すると,「ふたご座」という入力に対して，ふたご座自体の説明とふたご座の人の運勢の両方を MMDAgent が行うことになる．このような動作の方が自然に感じられる可能性はあるが，今回の実験では事前に動作させてみた結果から，優先度の高いどちらか一方のみを話す方がより自然な対話であると仮定した．これらの FST ファイルを SyncMode で実行することで，優先度の高いどちらか一方のみを開始する対話の処理が可能である．

countup.fst を NoSyncMode とした理由としては，この FST ファイルが音声認識イベントやキー入力イベントの回数をカウントするだけであり，FST 競合を起こす可能性がある音声合成やモーション再生などのコマンドメッセージを発行しないものだからである．また，正確に入力イベントをカウントするためには，入力イベントが発行されたその時に状態遷移する方が良く，FST の排他制御が邪魔となる．そこで，排他制御を行わない NoSyncMode でこの FST ファイルを実行する．

5.2　結果と考察

FST ファイルの切り替えと排他制御モードに基づく実験の結果と，その結果からの考察を述べる．評価の基準を表 2 に，実験結果を表 3 に示す．

表 3 より，パターン C が最も結果を得られたことが分かる．

従来の MMDAgent の対話管理部の FST 制御手法に近いパ

表 2　実験の評価基準

OK	実行時の MMDAgent の動作に対して不自然な点がない.
OK_i^*	FST 競合や FST 協調の不足により特定の条件下では異常な動作をする.
NG_i	FST 競合や FST 協調の不足により致命的に異常な動作をする.

表 3　各排他制御パターンにおける評価結果

排他制御パターン	OK の割合	OK^* の割合	NG の割合
パターン A	0%	60%	40%
パターン B	20%	30%	50%
パターン C	80%	20%	0%

ターン A では，多くの場合で FST 競合に起因する問題が発生することが分かる．これは遷移条件となっている入力イベントの重複により，FST ファイルの作成者が意図していない状態遷移が発生し，音声が途切れたり 3D モデルが正しく追加されたり削除されたりしないすることが主な原因である．

また，提案手法である FST の排他制御のみを用いたパターン B においては，FST 競合の問題は解決できるが，その反面，入力イベントを複数の FST で共有することが出来ないため FST 協調が不足する問題が発生してしまう．例えば，遷移条件となっている入力イベントの重複があっても，より優先度が高い FST のみが状態遷移するため，音声が途中で途切れたりといった FST 競合の問題は発生しなくなったと考えられが，一方で，複数の FST で入力イベントを共有することができないため，「2 号館と 5 1 号館を案内して」という入力に対して，2 号館を案内する FST と 5 1 号館を案内する FST の内，より優先度の高いどちらかの FST しか状態遷移が発生しないといった問題が発生しうる．このような問題は FST 協調の不足に起因し，自然な対話処理ではないと考えられる．また，ある特定の入力イベントをカウントするような，排他制御を必要とせず，排他制御が逆に邪魔になってしまうような FST が正しく動作しない問題も発生する．よって，パターン B のように全ての FST を排他処理 (SyncMode) のみで実行した場合，FST 競合の問題は解決できるが，FST 協調の不足は解決できていないと考えれらる．

一方で，FST ごとに排他制御モードを調整し設定したパターン C の結果を見ると，多くの場合において，FST 競合がなく，かつ FST が協調して動作していることが分かる．実験結果から，FST ファイルの内容や用途に合わせて FST の排他制御モードを設定できるようにすることの効果が分かった．しかし，パターン C においても FST 競合と FST 協調の不足以外の点で，まだ問題が残っていると考えられる．今回のパターン C の実験において OK^* と判定されたものは 20% ある．それらにおける問題とは提案システムで改良された対話管理部を用いても，対話のコンテキストを正しくシステムが認識することができないことに起因していると考えられる．例えば，今回の実験においては，seiza.fst が「ふたご座」という入力で状態遷移し，uranai.fst が「占い」と「ふたご座」のどちらの入力でも状態遷移する．そこで，uranai.fst は「占い」と入力すると,「それでは占います．あなたの星座を教えてください.」と MMDAgent から応答があるが，その後に,「ふたご座」と入力しても seiza.fst の方が優先度が高い場合，先に占いでなく星座の説明を始めてしまう．今回の実験で発生した前述の問題を解決するためには，直前に実行された FST ファイルの優先度をシステムで自動的に上げるような処理が必要であると考えている．

文　献

[1] Akinobu Lee, Keiichiro Oura, Keiichi Tokuda, MMDAgent - A fully open-source toolkit for voice interaction systems, Proceedings of the ICASSP 2013, pp. 8382-8385, 2013.5

[2] 徳田 恵一, コンテンツ生成の循環系を軸とした音声技術基盤の構築を目指して, 電子情報通信学会技術研究報告. SP, 音声 111(365)，pp. 153-157，2011

[3] 石川 博規, 山本 大介, 高橋 直久, 音声対話コンテンツのパッケージ化に関する研究, 電子情報通信学会総合大会講演論文集 2014 年情報・システム (2), pp. 200, 2014.3

[4] 石川 博規, 山本 大介, 高橋 直久, 音声対話コンテンツのパッケージ化とその配信システム, マルチメディア、分散協調とモバイルシンポジウム 2014 論文集 2014, pp. 789-795, 2014.7

[5] 若林　敬太郎, 山本 大介, 高橋 直久, タブレット端末のための複合状態を用いた音声対話コンテンツ編集手法, マルチメディア、分散協調とモバイルシンポジウム 2014 論文集 2014, pp. 781-788, 2014.7

[6] Wakabayashi Keitaro, Daisuke Yamamoto, Naohisa Takahashi, A Voice Dialog Editor Based on Finite State Transducer Using Composite State for Tablet Devices, Computer and Information Science 2015, Springer International Publishing, pp. 125-139, 2016

[7] 柳倫浩, 山本 大介, 高橋 直久, ユーザによる案内情報の付与とそれに基づくモバイル音声道案内システム, マルチメディア、分散協調とモバイルシンポジウム 2014 論文集 2014, pp. 294-300, 2014.7

[8] Yanagi Tomohiro, Daisuke Yamamoto, Naohisa Takahashi, Development of mobile voice navigation system using user-based mobile maps annotations, Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on. IEEE, pp. 373-378, 2015.

[9] VoiceXMl Forum technical working group, Voice Extensible Markup Language (VoiceXML) version2.0, http://www.w3.org/TR/voicexml20/

[10] 小林 聡, 中村 有作, 植田 浩一, 山田 博文, 新田 恒雄, マルチモーダル対話記述言語 XISL の提案, 音声情報処理, pp. 43-48, 2001.7

[11] 植田 喜代志, 秋田 祥史, 荒木 雅弘, 西本 卓也, 新美 康永, VoiceXML のマルチモーダル化の検討, 音声言語情報処理, pp. 43-48, 2001.10

[12] Allauzen C., Riley M., Schalkwyk J., Skut, W., Mohri M., OpenFst: A general and efficient weighted finite-state transducer library, Implementation and Application of Automata Springer Berlin Heidelberg, pp. 11-23, 2007

[13] 小暮 悟, 伊藤 敏彦, 中川 聖一, 移植性の高い音声対話システムにおける対話スクリプト構築ツールの試作, 音声言語情報処理, pp. 139-144, 2002.2

[14] 奥 智岐, 西本 卓也. 荒木 雅弘. 新美 康永, タスクに依存しない音声対話の制御方式, 電子情報通信学会論文誌, D-Ⅱ, No.5, pp. 608-615, 2003.5