社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－SLP－50 (12)
2004／2／7

MMIシステムにおける意味解釈と統合に関する記述方法の提案

中島 将宏† 桂田 浩一† 山田 博文‡ 新田 恒雄†

†豊橋技術科学大学 大学院工学研究科 知識情報工学専攻
‡豊橋技術科学大学 マルチメディアセンター

〒441-8580 愛知県豊橋市天伯町雲雀ヶ丘1－1
E-mail: †{nakajima,katurada,nitta}@vox.tutkie.tut.ac.jp, ‡yamada@vox.tutkie.tut.ac.jp

あらまし 本報告では，我々が提案してきたマルチモーダル対話（MMI）シナリオ記述言語XISLの抱える入力記述に関する問題点を述べ，それらを解決する記述方法を提案する．従来のXISLを用いた対話シナリオ作成では，多様なモダリティを複合的に用いたユーザ入力を詳細に記述しなければならなかった．新たに提案する改良版では，モダリティに関する記述を外部のファイルに分離し，シナリオにはモダリティに依存しないamodalな記述のみを行う方針を採用している．新たな方針の採用に伴い，amodalな記述とマルチモーダル入力を関連付けるクラス定義記述とルールベースの入力統合方式を導入し，システムアーキテクチャを変更した．また，記述の負担を軽減するため，様々なシナリオで共用できるクラス定義と統合ルールの標準セットを用意した．シナリオ記述の簡略化と標準セットの提供による開発コストの軽減を比較するため，我々の研究グループで先に作成したオンラインショッピングのアプリケーションを提案の記述方法で書き直した．この結果，入力操作に関する記述量が大幅に削減され，システム全体の記述コストを削減できることがわかった．
キーワード マルチモーダル対話，入力統合，MMI記述言語，XISL

A Proposal to Describe Semantic Interpretation and Multimodal Integration in an MMI System

Masahiro NAKAJIMA† Kouichi KATSURADA† Hirobumi YAMADA‡ and Tsuneo NITTA†

†Graduate School of Technology, Toyohashi University of Technology
‡Multimedia Center, Toyohashi University of Technology
1-1 Hibarigaoka, Tempaku-cho, Toyohashi 441-8580, JAPAN
E-mail: †{nakajima,katurada,nitta}@vox.tutkie.tut.ac.jp, ‡yamada@vox.tutkie.tut.ac.jp

Abstract In this paper, we clarify issues concerning of annMMI scenario description language XISL and propose a modified XISL to resolve them. The previous XISL requires detailed description of complex input pattern when multiple modalities are used in application. To resolve heavy description work, we propose the modified XISL in which dialog scenarios only include amodal description of inputs, and modality-dependent descriptions are separately descried in external files. In addition to this modification, we redesigned the architecture of a previous MMI system, and specified two types of external files: a class definition file which defines relations between amodal input representation and multimodal inputs, and a rule file which defines the procedure to integrate inputs and synchronize them. In addition, we provided a default set of semantic definitions and rules to ease developers' description work. The investigation concerning an online shopping application task shows that the modified XISL significantly reduces the amount of description.

Keyword Multimodal Interaction, Multimodal Integration, MMI Description Language, XISL

1　はじめに

近年，携帯電話やPDA・タブレットPC等のユーザ端末多様化に伴い，Webアクセスへのマルチモーダル対話（MMI）技術応用が活発に議論されている．またW3Cは，WebにおけるMMI標準化作業のため，ワーキンググループを結成し[1]，MMIに関する要求仕様，MMIのシナリオ記述言語，MMIのフレームワークを検討している．一方，我々もこれまでMMIシナリオ記述言語XISL[2]とMMIシステムアーキテクチャ[3]を開発すると共に，機能実証のためオンラインショッピングシステム[4]や航空チケット案内システムを構築してきた．

XISLで記述したMMIシナリオは，多様なモダリティを複合的に用いた入力とそれに対する処理や出力の組み合わせで構成されている．このため，MMIシナリオには扱うモダリティを複合的に組み合わせた入力パターンを全て列挙して記述しなければならず，開発者の負担が大きいという問題があった．

そこで，本研究ではユーザ入力の記述をモダリティに依存しない表現（amodalな表現）にすることで，この負担を軽減する方法を提案する．この改良に伴い，amodalな表現とマルチモーダル入力の関係，および入力の同期制御を別途記述する必要が生じる．本研究では前者をクラス定義ファイル，後者をルールベースの入力統合方式の導入によって解決する．

以下，2章では従来のXISLとその実行システム，およびそれらの問題について述べ，3章では提案する記述とシステムアーキテクチャを説明する．続いて4章で二つの記述方法を比較し，5章でまとめる．

2　従来のMMIシステムとXISL

2.1　MMIシステム

MMIシステムは図1に示すようにフロントエンドと対話制御部から構成される．フロントエンドはPCやPDA・電話のようにユーザとインタラクションするための端末で，対話制御部はXISLで記述されたMMIシナリオに従い対話の進行を制御するモジュールである．対話制御部やフロントエンドはWebサーバ上のXISLドキュメントやその他のコンテンツを利用する事を前提としている．

入力統合は図1に示すようにフロントエンドから送られる入力情報を元に，対話制御部内の入力統合部で行われる．入力統合部は，XISLの入力記述から構文解析表を生成し，複数モダリティからの入力情報を構文解析処理で統合する[5]．

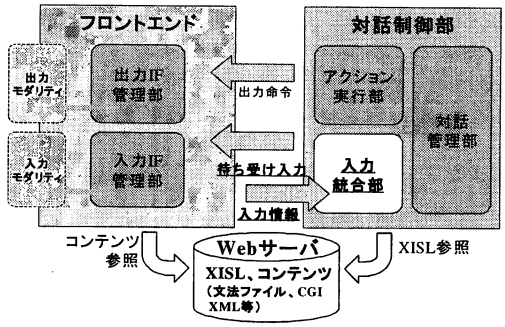

図1．MMIシステム

```
<operation comb="seq"> /* 入力1 */
  <input type="touch" event="click" match="..."/>
  <input type="touch" event="click" match="..."/>
</operation>
<operation comb="alt"> /*入力2と3は択一制御*/
  <par_input>   /* 入力2 */
    <input type="touch" event="click" match="..."/>
    <input type="speech" event="recognize"
           match="grammar.txt#ここから"/>
  </par_input>
  <par_input>   /* 入力3 */
    <input type="touch" event="click" match="..."/>
    <input type="speech" event="recognize"
           match="grammar.txt#ここまで"/>
  </par_input>
</operation>
<operation>          /* 入力4 */
  <input type="speech" event="recognize"
         match="grammar.txt#from_to"/>
</operation>
```

図2．出発空港と到着空港の入力（従来記述）

2.2　XISL

XISLはMMIのシナリオを記述するための言語である．前述したように，XISLを用いたシナリオには，複数モダリティを用いたユーザの入力・システムの応答とその同期制御および比較演算等が記述される．本節では，本研究で検討の対象とする入力の記述について述べる．

図2にXISLの入力記述例を示す．XISLはXMLに準拠した言語で，ユーザ入力を記述するために<operation>や<input>等のタグセットを規定している．<operation>には複数モダリティによる入力とその同期制御が記述される．<input>は単一モダリティの入力を表し，モダリティを指定するtype属性，入力イベントを指定するevent属性，および入力統合用の追加パラメータを指定するmatch属性を持つ．<input>を<par_input>等の要素で括るか，あるいは<operation>のcomb属性を指定することによって，入力の逐次/同時/

択一的な制御を記述できる．

XISL では，<input>の記述に関して，要素と属性のみを規定しており，具体的な内容や属性値を規定していない．これにより，フロントエンド開発者は入力記述を自由に定めることが可能になるため，多様な端末を容易に導入することが可能となる．

図2では航空チケット案内のアプリケーションにおいて，音声とタッチの利用を想定した出発空港と到着空港の入力方法を四通り記述している．図の入力1はタッチの連続入力，入力2と入力3はタッチと音声の同時入力，入力4は音声のみの入力を表している．

2.3 XISL の入力記述に関する問題点

先に述べたように，入力統合に用いる構文解析表はXISL の入力記述に基づいて生成される．このため，開発者には統合すべきユーザ入力を網羅して記述することが求められる．このような記述方法は，入力を詳細に記述・制御できるメリットを持つ反面，モダリティの数が増えると記述の負担が大きくなる．

3 提案システム

新しい XISL ではモダリティに依存しない amodal な入力記述を採用した．<operation>内にはマルチモーダル入力の統合を格納する要素と統合結果の種類および関係（この二つを“クラス”と呼ぶ）を記述する．さらに，クラスとマルチモーダル入力を関連付ける「クラス定義」，入力統合に用いる「ルール」を外部ファイルとして導入した．

以下，上で述べた新しい XISL と外部ファイルの導入について説明した後，システムアーキテクチャの変更点を示す．

3.1 XISL の変更

XISL に対し，次に示す四項目の変更を行った．

(a) <input>を廃止

amodal な記述を行うには，モダリティや同期制御に関わる記述を除く必要がある．

(b) <operation>に grammar 属性を導入

この属性は 3．2 節で述べるクラス定義と文法，および 3．3 節で述べる統合ルールを記述したファイルを参照するために用いる．

(c) <slot>を導入

マルチモーダル入力の統合結果を格納するために用いる．この要素は，識別子を記述する name 属性と XISL の変数を記述する return 属性を持つ．

(d) <group>，<value>を導入

この二つの要素は<slot>に対してクラスを指定するために用いる．<group>は複数の<slot>を括り，その関係を指定する．<value>は<slot>の子要素となり，当該

```
<operation grammar=" air.txt" >
  <group type=" range" >
   <slot name=" Departure" return=" dep" >
    <value type=" AIRPORT" ></value></slot>
   <slot name=" Arrival" return=" arr" >
    <value type=" AIRPORT" ></value></slot>
 </group>
</operation>
```

図3．出発空港と到着空港の入力（amodal 表現）

```
[grammar-type]{ /* <value>の type 属性値を定義 */
  [AIRPORT]{
    SPEECH($airport);SPEECH(…);}}
[group-type]{   /* <group>の type 属性値を定義 */
  type:range;
  size:2;
  label[1]:[始点][開始]…;
  label[2]:[終点][終了]…;}
[grammar]{ /* 音声認識文法を定義 */
  $airport = なごや|ふくおか;
  $FromTo=$airport<始点/>から$airport<終点/>まで;
  $ToFrom=$airport<終点/>まで$airport<始点/>から;}
```

図4．クラス定義と文法の記述

<slot>に格納すべき統合結果の種類を指定する．クラスの指定には<group>，<value>の type 属性を用いる．クラス定義に関する詳細な説明は次節で述べることとし，ここでは図3のクラス指定による，情報の種類および関係を大まかに説明する．図 3 では<value>の type 属性値“AIRPORT”（AIRPORT クラス）によって，各<slot>が格納する統合結果の種類が空港名となることが規定されている．また，<group>の type 属性値“range”（range クラス）によって，<group>の子要素である二つの<slot>が範囲（始点と終点）を表し，出発空港と到着空港の関係となることが規定される．

3.2 クラス定義と文法の記述

amodal な記述とマルチモーダルな記述は，クラス定義で関連付けられる．クラス定義はシナリオ外のファイルに記述され，シナリオにはファイル名のみ記述する．これにより典型的な入力（数字や日付の入力，範囲の入力など）の再利用が可能になり，アプリケーション開発コストを低減できる．

ここで，図 3 の XISL に記述されているクラス（“AIRPORT”，“range”）のクラス定義を図4に示す．まず，“AIRPORT”のクラス定義は[grammar-type]に記述されている．ここでは AIRPORT クラスに音声認識文法$airport を対応させている．一方，“range”のクラ

ス定義は[group-type]に記述されている．[group-type]の type，size，label[]にはそれぞれ<group>の type 属性値，子要素となる<slot>の数，<slot>に与えるラベルを記述する．ラベルは次に述べる発話の認識結果がもつあいまいさを解決するために導入した．例えば図 4 の[grammar]で$airport は，$FromTo と$ToFrom 内で二回参照されている．このような場合，$FromTo や$ToFrom で認識された二つの$airport が出発空港なのか到着空港なのかを付加情報無しに判断することはできない．そこでラベルを認識文法に付与して，$airport の認識結果と<slot>を対応付ける．図 4 では，<始点/>が与えられた$airport の認識結果は一つ目の<slot>に，<終点/>が与えられた$airport の認識結果は二つ目の<slot>に対応付けられる．

3.3 ルールを用いた入力統合

3.3.1 ルールの導入

3.1 節で述べたように，新しい XISL では amodal な入力記述を採用した．これに伴い，入力統合に関する記述をシナリオ外で行う必要が生じる．本システムでは，統合ルールの導入によりこれを実現した．また，典型的な統合方法をデフォルトのルールとして提供することにより，ルール記述の負担軽減を試みた．

3.3.2 ルールの記述

統合ルールは，event セクション，model セクション，act セクションから構成される．event セクションには入力のモダリティや逐次/同時/択一的等の同期制御を記述する．model セクションには統合対象となる<slot>の構造を記述する．また，act セクションには統合の振る舞いを記述する．それぞれのセクションには複数の条件を記述できる．図 5 に「“ここまで” という発話とタッチが同時に発生した時に，タッチ入力に対応する<slot>を二つ検索し，その二つ目を埋める」というルールを示す．なお，このルールは図 3 で示した<operation>においてクリックした対象を到着空港として受け付けるためのルールである．

```
[40]{
  event{
   par(2){ /* ここまで+タッチ */
    user_input(#i41, TOUCH, *);
    user_input(#i42, SPEECH, $kokomade);
   };
  }
  model{  /* タッチ入力に対応する<slot>が二つ*/
   match_input(#m41, #i41, 2);
  }
  act{    /* 二つ目の<slot>を埋める*/
   fill(#i41, #m41, _all, 2);
  }
}
```

図 5．ルールの例

表 1．統合ルールに記述できる条件

	条件	説明
event	user_input	モダリティや時間制御を用いた入力イベントの指定
	seq,par,alt	
	no_input	
model	match_input	入力が対応する<slot>の検索
	match_grammar	
	find_type	特定の<slot>を検索
	pickup	
	is_filled_by_name	<slot>が埋まっているかを判定
	is_filled_by_type	
act	fill_matched	要素を埋める
	fill	
	hold	処理を中断
	submit	統合途中の結果を対話制御部へ強制送信

```
[1000]{
  event{
    par(2){ /* 音声+タッチ */
      user_input(#i1, TOUCH, *);
      user_input(#i2, SPEECH, *);
    };
  }
  model{ /* <operation>内の<slot>が一つだけ*/
    pickup(#m1, *, 1);
  }
  act{   /* 音声を無視してタッチで<slot>を埋める */
    fill(#i1, #m1, _all, 1);
  }
}
```

図 6．デフォルトルールの一つ

図中[40]はルールの識別子で，一組の event/model/act をまとめるために用いる．event セクション内の user_input はユーザの一入力に対する条件で，#i41・#i42 はその識別子を表す．例では二つの user_input を par(2)で括ることにより，「“ここまで” という発話とタッチ入力が 2 秒以内に発生」という条件を記述している．model セクション内では match_input を用いることにより，一つの<operation>内から二つのタッチ入力（#i41）に対応する<slot>（Departure と Arrival）を検索している．検索された<slot>は，match_input の識別子である#m41 を用いて参照される．act セクションで

は event と model を満たした時の振る舞いが記述される．図5では識別子（#i41, #m41）とパラメータ（_all, 2）を用い，model で検索した全<slot>（#m41, _all）の内，二つ目（2）をタッチ入力（#i41）の結果で埋めることを記述している．

このようなルールを複数記述することで，<slot>を埋める処理を実現した．表1に現在使用できる条件の一覧を示した．

3.3.3 デフォルトのルール

デフォルトの統合ルールでは，アプリケーションに非依存な統合を行う．例えば一つの<slot>に対する競合入力の中から特定モダリティの入力を選択する場合や，ある入力値を格納するための<slot>を複数の中から一つ選択する場合などに用いる．

図6にタッチと音声の競合を解決するルールの記述を示した．event セクションでは「音声とタッチが2秒以内に同時発生」，model セクションでは「<slot>が一つのみ」という条件を記述している．これらを満たすと，タッチ入力の結果が act セクション内の fill によって<slot>に埋められる．

3.3.4 ユーザ定義ルールの追加

3.3.1節で述べたように，開発者は必要に応じてアプリケーション依存の統合ルールを記述・追加できる．

開発者が作成したルールを統合で用いるには，ルールを記述したドキュメントの URL を<operation>の grammar 属性で指定すればよい．これにより，当該<operation>に対し，開発者が記述したルールを適用できる．また，デフォルトのルールを利用する／しないの指定も可能なため，開発者は自分が望むように<operation>ごとの統合処理を規定できる．

3.4 システムアーキテクチャの変更

前節までに述べた XISL の記述方法の変更，クラス定義の記述及び入力統合処理の導入に伴い，システムのアーキテクチャを図7のように変更した．

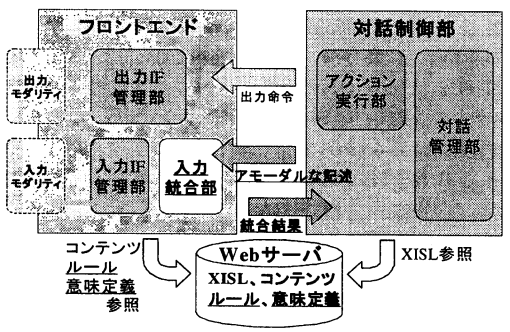

図7. 提案システムのアーキテクチャ

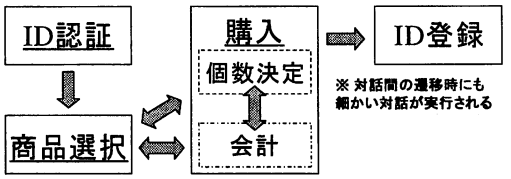

図8. OLS の対話の流れ

```
<operation comb="alt"> /* 商品のみを選択する */
  <input match="goods_list..."
         type="touch" event = "click"/> ･･･ 入力①
  <input match="grammar.txt#goods" return="goods"
       type="speech" event="recognize"/> ･･･ 入力②
</operation>
<operation comb="alt"> /* 商品と個数を選択する */
 <par_input> ･･･ 入力③
  <input match="goods_list..."
         type="touch" event="click"/>
  <input match="grammar.txt#num" return="num"
         type="speech" event="recognize"/>
 </par_input>
 <input match="grammar.txt#goods_num_buy_01"
        type="speech" event="recognize"
        return="goods,number"/> ･･･ 入力④
</operation>
```

図9. Top_Page の入力記述（従来記述）

入力統合部で取り扱うルールにはフロントエンド内で扱う情報（モダリティなど）が記述されているため，入力統合部をフロントエンド側に移した．また，ルールやクラス定義のファイルは Web サーバ上にあるものを取得・参照することにした．

4 評価

本節では，これまで我々の研究グループで作成してきたオンラインショッピング（OLS）のアプリケーションを提案記述で書き直した上で，<operation>の数および記述量を従来記述と比較する．記述量の比較には XISL ドキュメント，クラス定義，ルールの記述に含まれる文字数を用いた．ただし，XISL ドキュメント内のコメントとデフォルトのクラス定義，ルールの記述は比較の対象外とした．

4.1 オンラインショッピング（OLS）

図8に OLS の対話の流れを示した．

OLS は ID 認証対話（Certify_id）から始まり，商品選択対話（Top_page），購入対話（Buy_Goods，

Order_Form)，ID 登録対話（Regist_id）を経て終了する．さらに，各対話間の遷移の際には Yes/No を問う対話が実行される．

4.2 従来記述と提案記述の比較

表 2 に各対話における<operation>の数を示す．OLS 全体では従来記述が 14，提案記述がその半数の 7 となり，<operation>の数を削減できることが確認できた．例えば Top_Page の対話では，図 9 に示した従来記述の商品名と個数に関する同時/択一的な入力記述が，図 10 に示すように一つの<operation>の記述に置き換えられた．同様に，Certify_Id および Buy_Goods の対話においても，従来記述で必要であったマルチモーダル入力の詳細な記述が amodal な表現で置き換えられたことにより，大幅に<operation>数を削減できた．一方，Regist_Id，Order_Form の各対話では元々<operation>の数が少なかったため，書き換えの効果が見られなかった．

次に，記述量の比較を行う．表 3 に従来記述に対する，提案記述の相対記述量を示した．表の左側は入力記述に限定した相対記述量であり，右側は出力も含めた全記述量の相対記述量である．まず，OLS 全体として，入力記述量は 45%，全記述量は 83%となり，特に入力記述量を大幅に削減できた．個別の対話について検証すると，Regist_Id を除く各対話では，入力記述量を大幅に削減できた．Regist_Id の対話において入力記述量が削減されていない理由は，表 2 の説明で述べたように元々の<operation>の記述が少なく，書き換えによる効果がほとんどなかったからである．

全記述量は，いずれの対話も従来記述に比べ減少し

表 2．従来記述と提案記述の<operation>数

対話名	従来記述の <operation>数	提案記述の <operation>数
Certify_id	3	1
Regist_Id	1	1
Top_Page	6	3
Buy_Goods	3	1
Order_Form	1	1
全体	14	7

表 3．従来記述に対する提案記述の相対記述量

対話名	入力記述量（%）	全記述量（%）
Certify_id	39	75
Regist_Id	102	89
Top_Page	49	81
Buy_Goods	50	83
Order_Form	61	90
全体	45	83

```
<operation grammar="select_goods.txt">
  <slot name="GoodsID" return="goods">
    <value type="html-async">GOODS</value>
    <value type="OLS_GOODS_ID"></value>
  </slot>
  <slot name="GoodsNum" return="num">
    <value type="TYPE_NUMERIC"></value>
  </slot>
</operation>
```

図 10．Top_Page の入力記述（提案記述）

た．入力記述量と全記述量の割合が比例関係にならないのは，<operation>数の変化や実行システムの仕様変更に伴って，一部の対話について入力記述外でのシナリオ修正が必要になったためである．

以上の比較により，提案記述は従来記述に比べ少ない記述量で MMI アプリケーションを記述できることが分かった．また，特に入力記述において大きな効果が得られる事を確認した．

5 まとめ

提案記述を用いることで，<operation>の数を大幅に削減してシナリオを記述できることが確認できた．また，入力記述を中心に，シナリオ全体の記述量も減らす事ができた．これにより，記述コストを低減したアプリケーション開発が可能となった．

今後は，アプリケーションで扱うデータの管理を XISL から行う方法を検討し，CGI 等の外部プログラムや比較演算・条件分岐に関わる記述の削減を試みる．また，本提案のクラス定義とルールの記述を XML ベースの言語として拡張・統合し，汎用性・応用性の高いマルチモーダル統合文法を検討する．

参考文献

[1] http://www.w3.org/2002/mmi/

[2] 桂田 他:"MMI 記述言語 XISL の提案,", 情報処理学会論文誌 第 44 巻 第 11 号, pp.2681-2689(2003).

[3] 中島 他:"シームレスな Web サービスを実現する MMI システムの構築," 情報処理学会 インタラクション２００３論文集, pp.59-60(2003-2).

[4] 桂田 他:"多様な端末からのアクセスが可能な OLS システムの実装," 情報処理学会研究報告, 2003-SLP-45, pp.41-46(2003-2).

[5] 大谷 他:"移植性に優れた MMI システムアーキテクチャの検討", 情報処理学会第 63 回全国大会論文集分冊 2, pp.177-178 (2001).