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

2004－SLP－54（46）
2004／12／22

異なる端末環境から利用可能な MMIアプリケーション開発における記述負担の軽減

青木　一峰† 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: †{aoki,katurada,nitta}@vox.tutkie.tut.ac.jp, ‡yamada@mc.tutkie.tut.ac.jp

あらまし 我々はこれまでマルチモーダル対話（MMI）記述言語 XISL を検討してきた．従来の XISL は，特にスロットフィリング対話などで記述量が多いこと，また XML コンテンツの操作に CGI を用いる必要がある，などの理由からアプリケーション開発者の負担が大きかった．そこで，新たに提案する XISL2.0 では，モダリティに対する扱いの差異を反映した U-M-A (Uni-modal, Multi-modal, A-modal)三階層モデルを基に言語仕様を新たに作成し，入力記述をモダリティ非依存にすると共に，VoiceXML のタグセットを導入することによって記述量を大幅に削減した．後者は，対話シナリオを A-modal で記述できることで可能になった．さらに，XForms-like なデータモデルを導入し，XISL シナリオ上から XML コンテンツを容易に操作できるようにした．本報告では，これらの改善を加えた XISL2.0 と記述例について述べる．

キーワード マルチモーダル対話，MMI 記述言語，XISL

A solution for the issues in descripting MMI applications accessed from various types of terminals

Kazumine AOKI† 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: †{aoki,katurada,nitta}@vox.tutkie.tut.ac.jp, ‡yamada@mc.tutkie.tut.ac.jp

Abstract We have developed an MMI description language XISL1.1 (eXtensible Interaction Scenario Language) for providing seamless web services. Although XISL has some desirable features, it has some issues concerning descriptive abilities especially in slot-filling style dialogs, large amount of description and complicated design when handling XML contents. In this paper we provide a new version of XISL(XISL2.0) that is based on a U-M-A (Uni-modal, Multi-modal, A-modal) three layer model and resolves these issues by using some syntax in VoiceXML, employing modality-independent input description, and introducing XForms like data-model description which enables to handle XML contents from XISL.

Keyword Multimodal Interaction, 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 アプリケーションへのマルチモーダル対話（MMI）応用が活発に議論されている．Web 技術の標準団体である W3C では，MMI を記述する言語の要求仕様や，MMI システムのフレームワークについての検討を始めている[1]．その中で，SALT[2]や XHTML＋Voice[3]，VoiceXML のマルチモーダル化[4]など，既存のマークアップ言語のマルチモーダル対応が試みられている．一方，我々もこれまで MMI 記述言語 XISL1.1 （eXtensible Interaction Scenario Language）[5][6]および XISL1.5[7]を提案してきた．XISL は，入力モダリティ/出力メディア記述に関する拡張性が高いため，多様なモダリティを持つ端末で実行可能な MMI アプリケーションを開発することができる．また，インタラクションとコンテンツを分離す

ることで，XML コンテンツの再利用性を高めている．我々は，XISL の有用性を示すために，XISL の実行システムを構築すると共に，オンラインショッピングシステム（OLS）[8] などの MMI アプリケーションを開発してきた．その結果，XISL の優れたモダリティ拡張能力を示すことができたが，同時に XISL の記述量が膨大なため開発者の負担が大きいという問題が明らかになった．記述量が増える原因は以下の三つである．一つ目は，入力記述がモダリティ依存なため，モダリティごとの記述が必要なことである．二つ目は，個々の入力スロットに対する入力促しが複雑な記述になることである．三つ目は，対話中に XML コンテンツを操作する時，XISL の作成以外に CGI を作成しなければならないことである．

そこで，従来の XISL の記述負担を軽減する XISL2.0 を策定した．XISL2.0 は，U-M-A 三階層モデル[9]を基に言語設計を行ない，上に述べた三つの原因に対してそれぞれ改善を行なっている．まず一つ目は，対話記述層を A-modal，すなわちモダリティ非依存にすることで，モダリティごとの記述を不要にした．複数モダリティの統合は，Multi-modal 統合層で行なう．二つ目は，対話記述言語として W3C で策定された VoiceXML[10]を参考に言語設計を行なうことで対話記述性を改善した．これは，対話シナリオを A-modal で記述できることで可能になったものである．三つ目は，XForms[11]のデータモデルと XML 操作用の新たな要素の導入により，XISL 上から容易に XML 操作を可能にした．

本報告では，2 章で従来の XISL である XISL1.1 と XISL1.5 の言語仕様について説明を行ない，3 章で従来の XISL の問題点を述べる．続いて，4 章で問題点を改善する XISL2.0 を説明する．

2. 従来の XISL と記述例

我々は，マルチモーダル対話機能を持つ Web アプリケーションを開発する言語として，MMI 記述言語 XISL を提案し，実装してきた．XISL はコンテンツをインタラクションから分離して記述できるため，コンテンツの再利用性が高いという特徴を持つ．また入力モダリティ/出力メディアの拡張性が優れているため，多用な端末から利用するアプリケーションを実現できる．以下に，XISL のバージョン 1.1 とその改良版 1.5 の言語仕様を簡単に説明する．

2.1. XISL1.1

図 1 に OLS において，「商品」と「個数」の二つのスロットを埋めるタスクを XISL1.1 を用いて記述する例を示す．<xisl>要素は XISL 文章全体を表し，更新履歴等のメタ情報を記述する<head>と，対話シナリオを記述する<body>を持つ．

```
<xisl>
<head>・・・</head>
<body id="Shop">
  <dialog>・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ ①
    <exchange>・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ ②
     <!--「商品」と「個数」が同時入力される対話-->
      <prompt>
      <!--「商品と個数を入力してください。」-->
      </prompt>
      <operation comb="alt">・・・・・・・・・・・・・・・・・・・ ③
        <par_input>・・・・・・・・・・・・・・・・・・・・・・・・・入力 1
          <input type="touch" ...
                    match="商品" return="GOODS"/>・ ④
          <input type="speech" ...
               match="sp.grm#個数" return="NUM"/>・ ⑤
        </par_input>
        <par_input>・・・・・・・・・・・・・・・・・・・・・・・・・入力 2
          <input type="touch" match="個数" .../>
          <input type="speech" match="sp.grm#商品"..../>
        </par_input>
        <input type="speech"
                match="sp.grm#商品_個数".../>・・入力 3
      </operation>
      <action>・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ ⑥
        <output><!--「○を○個ですね」--></output>
        <submit next="set_goods.cgi"
                        namelist="GOODS"/>・・・・・・・ ⑦
        ・・・
      </action>
    </exchange>
    <exchange>・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ ⑧
        <!--「商品」だけが入力される対話 -->
      <operation comb="alt">・・・</operation>
      <action>
       <submit next="set_goods.cgi"
                        namelist="GOODS"/>
        <if cond="NUM==undefined">
          <then><!--「個数を入力して下さい」--></then>
          <else>・・・</else>
        </if>
      </action>
     </exchange>
     <exchange>・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ ⑨
        <!--「個数」だけが入力される対話 -->
        ・・・
     </exchange>
  </dialog>
```

図 1. XISL1.1 の記述例

一組の対話を表す<dialog>は，対話の最小単位である<exchange>の集合から成る．<exchange>は，<prompt>（ユーザへの入力促し），<operation>（待ち受け入力），<action>（アクション）で構成される．<prompt>には入力促しのための出力を，また<operation>には指定した待ち受け入力を記述する．<operation>には，待ち受ける全ての入力パターン（逐次，並列，択一，およびこれらの組合せ）を列挙することになる．パターンの記述には，<operation>の comb 属性，もしくは入力パターン記述用の要素（<par_input>等）を用いる．例えば，③の<operation>は，comb 属性に“alt（択一的）”が指定されているので，入力 1，入力 2，入力 3 のい

1 .XISL1.1 の<submit>はサーバで実行した CGI の戻り値を return 属性に指定した変数へ代入する要素である．

ずれかの入力を待ち受ける．<par_input>で囲まれた入力 1 は④の<input>（「商品」のタッチ）と同時に⑤の<input>（「個数」の発話）の入力パターンを指定している．<action>には，<operation>で指定した入力を受けた際に実行する処理内容を記述する．処理内容は，単一の出力を表す<output>と CGI を実行する<submit>[1]に加え，条件式，演算処理等の要素を記述できる．

<input>と<output>の仕様は，モダリティ/メディアの拡張性を高めるために，XISL の仕様外となっている．

2.2. XISL1.5

XISL1.1 の問題は，<operation>に指定する入力パターンの記述量が多く，アプリケーション開発者の負担が大きいという点であった．例えば，図 1 の③の<operation>では，タッチと音声による入力パターンを三つ列挙する必要がある．この問題を改善するため，XISL1.5 では<operation>の仕様を変更した．具体的には，<operation>のモダリティや統合に関する記述を外部ファイルに分離し，XISL をスロットフィリング的な記述に変更することにより，<operation>をモダリティ非依存にした．また，外部ファイル化した統合記述に，独自のルール記述を用いることで，システム開発に必要な記述量を削減している．

図 1 ③の<operation>の記述は，XISL1.5 の仕様では図 2 のようになる．このように，新しい<operation>には<slot>が列挙される．(i)と(iii)の<slot>はそれぞれ「商品」と「個数」のスロットを表している．<slot>の return 属性は入力により埋められる値を格納する変数を指定する．<slot>内の<value>の type 属性には，<slot>と入力結果を結びつけるためのクラスを宣言する．クラスやルールに関する定義は，<operation>の grammar 属性に指定した外部ファイルに記述する．

図 2 に示した各<slot>に対応する，クラスとルール定義の例を図 3 に示した．[grammar_type] は，モダリティ毎の入力結果を<slot>へ埋めるため，<value>の type 属性で宣言したクラスと，モダリティを具体的に結びつける．図 3 の(ア)は，図 2 の(ii)の<value>の type 属性で定義した“goods”クラスに音声認識文法“＄商品”を結びつけている．この結びつけにより，“＄商品”で認識された入力結果は図 2 の(i)の<slot>に埋まる．また，[grammar] では，有効にしたい音声認識文法を指定する．ルール定義は，統合の際の同期を取る event セクション，<slot>を選択する model セクション，選択した<slot>に値を埋めるなどの処理をする act セクションから構成される．図 3 の［100］の“100”は，ルールの優先順位を示し，小さいほど優先順位が高い．図 3 のルールでは，タッチと音声による同時入力が 2 秒以内に起きた場合，それぞれのスロットに値を埋めることを示す．

```
<operation grammar="shop.txt">
  <slot return="GOODS">····· (i)
    <value type="goods"/>····· (ii)
  </slot>
  <slot return="NUM">····· (iii)
    <value type="num"/>
  </slot>
</operation>
```

図 2. XISL1.5 の例（<operation> 部のみ）

```
[grammar-type]{
  [goods]{SPEECH($商品);}···(ア)
  [num]{ SPEECH($個数);}
}
[grammar]{
$商品 = リンゴ|ミカン・・・;
$個数 = ひとつ{1}|ふたつ{2}・・・;
$商品_個数 =$商品 を $個数;
[100]{
  event{
    par(2){ /* 音声+タッチ */
      user_input(#i1, TOUCH, *);
      user_input(#i2, SPEECH, *);
    };
  }
  model{
    match_input(#m1, #i1, 1);
    match_input(#m2, #i2, 1);
  }
  act{
    fill(#i1, #m1, _all, 1);
    fill(#i2, #m2, _all, 1);
}}
```

クラス定義

文法定義

ルール定義

図 3. クラスとルール定義

3. 従来の XISL の問題点

我々はこれまで，XISL1.1 や XISL1.5 を用いて OLS や航空チケット予約サービスなどのアプリケーションを開発したが，その結果，XISL の記述量に関する三つの問題が明らかになった．以下に，これらを説明する．

3.1. 入力記述量の問題

XISL1.1 から 1.5 への改良により，<operation>の記述量が減少して入力記述の可読性は増したが，一方で外部ファイルに定義するクラスとルールの記述は新たな負担となった．

まずクラス定義を考察する．XISL1.5 では，<slot>と入力結果を結びつけるために，クラス定義が必要であった．クラス定義が必要な理由は，モダリティごとに入力結果の表現形式が統一されていないことにある．例えば，音声認識による入力結果は「認識文法名」と「値」で表現される．また，HTML 要素へタッチしたときの入力結果は「要素 id」と「値」で表現される．そのため，クラス定義はモダリティごとに定義する必要があり記述量が多くなる．

次にルール定義は，基本的にデフォルトのものが用

意されるため，単純な入力に対してはルールを定義する必要がない．しかし，独自のルールを定義する場合には，モダリティ依存な記述が必要になる．

3.2. 複雑なプロンプトの問題

従来の XISL は，未入力のスロットに対する入力促しを行なう際に多くの記述を必要とした．図 1 の「商品」と「個数」のスロットを埋める例では，「商品」だけが埋められたとき，「個数」に対する入力促しを行ないたい．このような対話を実現するためには，「商品」と「個数」が同時に埋められる②の<exchange>のほかに，「商品」と「個数」それぞれが単独で埋められた時の処理に対応する⑧と⑨の<exchange>を用意しなければならない．それぞれの<exchange>には，<action>内で条件式を用意し，片方のスロットに値が埋まっていないなら入力促しを行なうといった処理を記述している．このように，適切な入力促しを行なうには，<exchange>を複数用意しなければならず，記述量が増えてしまう．記述方法は他にもいくつか考えられるが，どの方法も条件式などを多用する必要があり記述量が多くなる．従来の XISL では，スロットフィリング的なタスクを効率的に記述することが難しかった．

3.3. XML コンテンツ操作の問題

XISL は，シナリオとコンテンツの再利用性を高めるために，コンテンツ部分を XML コンテンツとして分離記述していた．XML コンテンツは，出力データ等に使用されるが，データはアプリケーション実行中に変更されることがある．例えば，OLS で用いるショッピングカートにユーザが商品を入れると，商品データが追加される．

従来の XISL は，このような XML 操作に CGI を用いている．例えば，図 1 の⑥の<action>では，⑦の<submit>を用いて CGI を実行し，ドキュメントサーバ側にある XML へ直接データを追加した後，クライアント側に再度 XML を読込んでいる．この例から分かるように，アプリケーション開発者は，シナリオと別に CGI を作成する必要があり，開発の大きな負担となるだけでなく，アプリケーション実行中サーバ/クライアント間の通信が頻繁に起こりパフォーマンスの劣化の原因ともなる．

4. XISL2.0 の提案

以下では，XISL2.0 の設計思想を述べたのち，3 章で述べた問題の改善策についてそれぞれ詳しく説明する。

4.1. 設計思想

XISL2.0 では，図 4 に示す U-M-A 三階層モデルを基に言語設計を行なった．また，各層のインターフェースは後述の EMMA 形式を採用した．これにより，対話シナリオはモダリティ非依存となり，記述が簡潔になると共に，マルチモーダル記述がなくなった結果，VoiceXML (Uni-modal 記述言語)のタグセットや FIA（Form Interpretation Algorithm）を導入し易くなる．また，新仕様では XForms-like なデータモデルを導入して，データ管理を行なえるようにした（データは音声認識文法中の意味付与，表示・音声合成等に利用される）．

なお，今回は出力メディア（画面表示・音声プロンプトなど）の記述を対話シナリオに直接書いており，マルチメディア分化生成は今後の課題になっている．

4.2. 入力記述のモダリティ非依存化

XISL1.5 は，改良により対話シナリオについては，A-modal な記述となっていた．しかし，外部ファイルのクラスとルール定義がモダリティに依存しているため，開発者の記述負担となっていた．

そこで，クラス定義を削除するために，クラス定義が必要な理由であった，モダリティごとに異なる入力結果の表現を統一化する方法を考えた．その表現規格としては現在 W3C で検討中の EMMA[12]を採用することにした．EMMA は，入力結果を「値」とその「意味」の構造に XML を用いて表現する．図 5 に EMMA の例を示す．図 5 の(1)では，“リンゴ” が「値」，<GOODS> がその「意味」を表している．また，「値」を XISL2.0 のスロットを表す<field>と結びつけるために，name 属性に「意味」を指定する．例えば<field>の name 属性が「GOODS」と指定されている場合，入力結果が図 5 の通りであった場合，そのスロットは“リンゴ”が埋まる．

また，ルールの記述をモダリティ非依存化するために，入力統合は，図 6 の例のように「意味」の構造を用

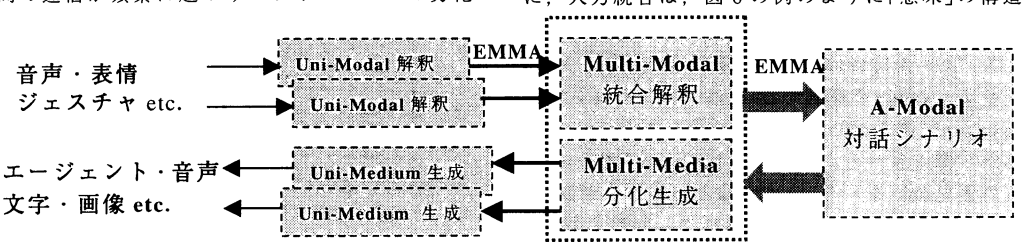

図 4 U-M-A 三階層モデル

いる．この例では，まず，(あ)(い)の EMMA はユーザが「ここから」と発話しながら「奈良駅」をタッチしたときに生成される．(あ)の EMMA はユーザの「ここから」という発話から生成され，(い)の EMMA は「奈良駅」をタッチした入力から生成される．この二つの EMMA を統合するとき，「unknown」には同じ「意味」を持つ値が上書きされる，というルールを用いる．(あ)の EMMA の値「unknown」は意味<駅名>を持ち，(い)の EMMA の値「奈良駅」が意味<駅名>を持つので，「unknown」は「奈良駅」に上書きされる．その結果が(ウ)の EMMA になる．このように，意味構造を用い入力統合することで，ルールをモダリティ非依存化できる．

XISL2.0 は，入力結果の表現とルールをモダリティ非依存化することで，これまでのようなモダリティごとの記述を不要にしている．また，対話シナリオだけでなく外部ファイルのルールもモダリティ非依存にしたことで，開発者は，出力記述を除き A-modal な記述を行なうだけでよくなる．

4.3. VoiceXML を参考にした設計

3.1 節で述べた通り，従来の XISL では，複雑な記述なくして個々のスロットに対する適切な入力促しを行なうことができなかった．このような問題を解決している言語として，対話記述言語としてよく知られている VoiceXML がある．VoiceXML は，個々のスロットに入力促しを指定することができ，独自のスロットを埋めるアルゴリズムである FIA により，自動的に埋まっていないスロットのプロンプトが実行される．

XISL2.0 では，対話シナリオを A-modal にすることで，VoiceXML のタグセットを導入した言語の再設計が可能となった．しかし，VoiceXML は入力モダリティとして音声と DTMF，出力メディアとして合成音声とオーディオのみの利用を想定しており，入出力の拡張性を持っていない．そこで XISL2.0 では，XISL1.1 同様に，モダリティ/メディアの記述仕様を XISL2.0 の仕様外とし，システム開発者が自由に入力モダリティや出力メディアの仕様を規定できるようにした．

図 7 に XISL2.0 の記述例を示す．この例は図 1 の①の<dialog>と同じ対話を記述したものである．最初に先頭の(a)の<form>を訪れると，VoiceXML の FIA とほぼ同様のアルゴリズムが実行される．まず，そのアルゴリズムにより，フォームレベルにある(b)の<fe>が実行される．<fe>は，XISL2.0 独自の要素で，モダリティ/メディアの動作を定義する．この例では詳細を省略しているが，html の表示と音声認識文法を有効にしている．次に，(c)の<initial>が選択され(d)の<prompt>によって「商品」と「個数」に対する入力促しが行なわれ，ユーザ入力を待つ．この例では，(e)の<field>と(h)の<field>がそれぞれ「商品」と「個数」のスロットを表す．

```
<emma:emma version="1.0"
 xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:interpretation emma:id="Shop">
    < GOODS >リンゴ</ GOODS >·····(1)
  </emma:interpretation>
</emma:emma>
```

図 5. EMMA

```
<emma:emma version="1.0"
xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:interpretation id="Select_Section">
      <出発><駅名>unknown</駅名></出発>          (あ)
  </emma:interpretation>
</emma:emma>
<emma:emma version="1.0"
xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:interpretation id="Select_Section">
      <駅名>奈良駅</駅名>                        (い)
  </emma:interpretation>
</emma:emma>
<emma:emma version="1.0"
xmlns:emma="http://www.w3.org/2003/04/emma">
  <emma:interpretation id="Select_Section">
      <出発><駅名>奈良駅</駅名></出発>           (う)
  </emma:interpretation>
</emma:emma>
```

図 6. EMMA の統合

```
<form id="Shop">·····(a)
  <fe><!--html 文章--></fe>
  <fe><!--音声認識文法--></fe>  }  ·····(b)
  <initial>·····(c)
    <prompt>·····(d)
      <fe><!--「商品と個数を入力して下さい」--></fe>
    </prompt>
  </initial>
  <field name="GOODS">·····(e)
    <prompt>
      <fe><!--「商品を入力してください」--></fe>
    </prompt>
    <filled>·····(f)
      <backend action="set_goods.cgi"
                    namelist="GOODS"/>·····(g)
    </filled>
  </field>
  <field name="NUM">·····(h)
    <prompt>·····(i)
      <fe><!--「個数を入力してください」--></fe>
    </prompt>
    <filled>
      <backend action="set_num.cgi" namelist="NUM"/>
    </filled>
  </field>
  <filled namelist="GOODS NUM">
    <goto next="next_dialog"/>
  </filled>
</form>
```

図 7. XISL2.0

<field>はスロットを表す点で XISL1.5 の<slot>と類似しているが，個々にプロンプトとアクションを持つ点で異なる．ここで，ユーザ入力によって(e)の<field>が埋められた場合，アクションである(f)の<filled>が実行される．(f)の<filled>内の(g)の<backend>は，XISL2.0 独自の要素で，XISL1.1 の<submit>と同様の動作を行なう[2]．続いて，まだ埋まっていない(h)の<field>が選

```
<xisl version="2.0">
  <head>
    <model>
      <instance>
        <sc:Shop>
          <sc:Tax>1.05</sc:Tax>
          <sc:ShoppingCart>
            < sc:Goods/>
          </sc:ShoppingCart>
        </sc:Shop>
      </instance>
      <bind nodeset="/sc:Shop/sc:Tax" readonly="true">
    </model>
  </head>
  ...
```

図 8. データモデル

```
<form id="Shop">
  <field name="Goods">
    <filled>
      <set_value expr="'リンゴ'"
          select="/sc:Shop/sc:ShoppingCart/sc:Goods"/>
    </filled>
  </ field>
  . . .
```

図 9. <set_value>

択され(i)の<prompt>によって「個数」に対する入力促しが行なわれる.

このように, XISL2.0 では, スロットを埋める対話を, 従来の XISL よりも簡単に記述することが可能である.

4.4. データモデルの導入

XISL のシナリオ上から XML コンテンツの操作を行なうためには, XISL と XML を明確に関連付ける必要がある. そこで, XForms を参考にしたデータモデルを導入した. XForms は HTML フォームの拡張規格で, コンテンツとプレゼンテーションを分離することでそれぞれの再利用性を高めている. データモデルは, XForms のコンテンツを定義する部分で, 任意の XML とデータの制約やデータ同士の依存関係を定義することができる.

XISL2.0 では, データモデルを, XISL のヘッダ部分に定義することで, シナリオ中からの XML コンテンツの操作を可能にする. この結果, XISL 上からデータ管理のための XML 操作が可能となり, CGI 作成が減少した. また, XML の操作はクライアント側で行なわれるためサーバ/クライアント間の通信によるパフォーマンスの劣化も防止できる. 以下に, データモデルと XML コンテンツ操作のための要素について簡単に説明する.

4.4.1. <model>

図 8 に XISL2.0 のデータモデルの例を示す. 上記でも述べた通り, データモデルは, XISL の<head>の<model>に定義する. XML コンテンツは<instance>の中に定義される. また, データの制約やデータ同士の依存関係は<bind>で定義する. 図 8 の<bind>は, コンテンツ中の<sc:Tax>内の "1.05" というデータは読取専用であるという制約を定義している.

4.4.2. XML コンテンツの操作

XML コンテンツを容易に操作できるように, データの取得や設定, またノードの作成や削除などを行なうために独自の要素を導入した. 図 9 の<set_value>では, 図 8 の XML コンテンツ中の<sc:Goods>に「リンゴ」を設定する操作を行なっている. また, 更に複雑な XML 操作を可能にするために, <script>中で DOM を使用することができる.

5. まとめ

XISL2.0 では, U-M-A 三階層モデルを基に, 入力記述をモダリティ非依存化にすることで, 出力記述を除き A-modal な対話シナリオを記述するだけでよくなった. また, VoiceXML を用いた言語設計によって, スロットフィリング的なタスクを効率的に記述できるようになった. XML コンテンツは, XForms を参考にしたデータモデルの導入により, 対話シナリオから容易に操作できるようになった. 以上の結果, アプリケーション開発において開発者の記述負担を軽減できた.

今後, 出力に関しても A-modal な記述を目指すと共に, XISL2.0 の有用性を示すため, 実行システムを開発する予定である.

2 .VoiceXML の<submit>との混同を避けるため, タグ名を変更した.

文　献

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

[2] 植田喜代志 他, "VoiceXML のマルチモーダル化の検討", 情報処理学会研究報告, 2001-SLP-38, pp.43-48 , Oct.2001.

[3] http://www.saltforum.org/

[4] http://www.voicexml.org/specs/multimodal/x+v/12/spec.html

[5] 桂田浩一 他, "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.

[6] http://www.vox.tutkie.tut.ac.jp/XISL/XISL.html

[7] 中島将宏 他, "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", 情報処理学会研究報告, 2004-SLP-50, pp.75-80, Feb.2004.

[8] 小林剛典 他, "MMI 記述言語 XISL によるオンラインショッピングシステムの開発", 情報処理学会研究報告, 2002-SLP-42, pp.11-16, Jul.2002.

[9] 新田恒雄,"マルチモーダル対話の深化と記述言語の今後" , 情報処理学会研究報告, 2004-SLP-50, pp.15-22, Feb.2004.

[10] http://www.w3.org/TR/voicexml20/

[11] http://www.w3.org/TR/xforms/

[12] http://www.w3.org/TR/emma/