

**Extensible Interaction Sheets Language  
XISL**

**version: 1.0  
January 30, 2002**

|                           |    |
|---------------------------|----|
| 概要                        | 5  |
| 1. はじめに                   | 5  |
| 2. XISL 実行システム            | 6  |
| 3. XISL の目標               | 7  |
| 4. XISL の範囲               | 7  |
| 5. 対話レベル                  | 8  |
| 5.1. エクスチェンジ              | 8  |
| 5.2. ダイアログ                | 9  |
| 5.3. ドキュメント               | 9  |
| 5.4. アプリケーション             | 9  |
| 5.5. セッション                | 9  |
| 6. XISL 構成要素              | 10 |
| 7. XISL ドキュメントの構造と実行      | 11 |
| 7.1. ドキュメントの構造            | 11 |
| 7.2. アプリケーションとその実行        | 14 |
| 7.3. ダイアログの遷移とアプリケーションの変化 | 16 |
| 8. 変数                     | 17 |
| 8.1. 変数宣言 - var           | 17 |
| 8.2. 変数のスコープ              | 18 |
| 8.3. 変数参照                 | 18 |
| 8.4. アプリケーション変数とドキュメント変数  | 19 |
| 8.5. ダイアログ変数              | 20 |
| 8.6. ローカル変数               | 20 |
| 8.7. セッション変数              | 21 |
| 9. 式                      | 21 |
| 10. 対話の記述                 | 22 |
| 10.1. 一組の対話 - dialog      | 24 |

|       |                           |                                              |    |
|-------|---------------------------|----------------------------------------------|----|
| 10.2. | 対話の最小単位                   | - begin , exchange , end                     | 25 |
| 10.3. | 複雑な対話の流れ                  | - seq_exchange , par_exchange , alt_exchange | 27 |
| 11.   | ユーザに入力を促すためのシステムの出力       | - prompt , reprompt                          | 29 |
| 12.   | ユーザの入力                    |                                              | 31 |
| 12.1. | operation                 |                                              | 31 |
| 12.2. | input                     |                                              | 32 |
| 12.3. | 複雑な入力の組み合わせ               |                                              | 34 |
| 13.   | システムの動作 , ユーザへの出力         | - action                                     | 34 |
| 14.   | 変数宣言                      | - var                                        | 35 |
| 15.   | 演算処理                      | - assign                                     | 35 |
| 16.   | 条件処理                      |                                              | 35 |
| 16.1. | if , elseif , then , else |                                              | 35 |
| 16.2. | switch                    |                                              | 36 |
| 17.   | 繰り返し処理                    | - while                                      | 37 |
| 18.   | 対話の遷移                     |                                              | 38 |
| 18.1. | call                      |                                              | 38 |
| 18.2. | return                    |                                              | 39 |
| 18.3. | goto                      |                                              | 39 |
| 19.   | 値の送受信                     | - submit                                     | 40 |
| 20.   | セッションの終了                  | - exit                                       | 41 |
| 21.   | プロンプトの再実行                 | - reprompt                                   | 41 |
| 22.   | 変数の参照                     | - value                                      | 41 |
| 23.   | ユーザ,システムへの出力              | - output                                     | 42 |
| 24.   | 値の読み取り                    |                                              | 45 |
| 24.1. | get_element               | - XML 要素の読み取り                                | 45 |
| 24.2. | get_value                 | - XML 要素の内容の読み取り                             | 47 |
| 24.3. | get_attribute             | - 属性値の読み取り                                   | 48 |

|       |                           |    |
|-------|---------------------------|----|
| 25.   | 値の書き出し                    | 49 |
| 25.1. | set_element - XML 要素の書き出し | 50 |
| 25.2. | set_value - XML 要素の値の書き出し | 51 |
| 25.3. | set_attribute - 属性値の書き出し  | 53 |
| 26.   | 要素の削除 - delete_element    | 54 |
| 付録 1. | XISL の DTD                | 56 |

## 概要

本文書では XISL( Extensible Interaction Sheet Language)を規定する．XISL はマルチモーダル入出力を用いたインタラク션을記述するための言語である．XISL の目標は，マルチモーダルを利用した，高度なインタラクシヨンの実現である．

### 1. はじめに

XISL は任意の XML ドキュメント内の要素（以後，XML 要素と呼ぶ）とユーザとのインタラクシヨンを記述するための言語である．ここでいうインタラクシヨとは，XML 要素（例えばブラウザにより表示されている）に対するユーザの操作（クリック，音声入力等），およびそれに対するシステムの動作（画面の更新，音声出力等）を指す．XISL ではマルチモーダルインタラクシヨン，すなわち複数の入出力手段を統合利用したインタラクシヨンを取り扱うことができる．これを実現するために，並行入出力，逐次入出力，択一入力の記述が可能になっている．また対話（インタラクシヨンの集合）の流れを記述するために，対話の遷移，割り込み等の処理をはじめとする，対話制御のための様々な命令が用意されている．

図 1に伝統的な“Hello World”を出力する対話例を示す．“hello.xml”はインタラクシヨンを付加する XML ドキュメントであり，“hello.xsl”(XSL スタイルシート)を用いて，<page>の内容がブラウザに表示されていると想定する．“hello.xisl”は XISL ドキュメントであり，<page>がクリックされたときにシステムが“Hello World”と発話する対話を記述している．

注意：XISL では<input>，および<output>の仕様に関して，属性の種類のみを規定している．つまり，本仕様では，属性値および内容は規定しない．これは<input>および<output>の記述に自由度を持たせることで，入出力モダリティの拡張や変更に備えるためである．<input>および<output>の属性値の記法や，要素の内容の仕様策定は，入出力端末（フロントエンド）の開発者に委ねる（フロントエンドのタイプごとに何らかの標準化が行なわれることが望ましい）．

#### hello.xml

```
<?xml version= " 1.0 " encoding= " Shift-JIS " ?>
<?xml-stylesheet type= " text/xsl " href= " hello.xsl " ?>
<!DOCTYPE hello SYSTEM " hello.dtd ">
<page>
  Touch this page to hear " Hello World! "
</page>
```

## hello.xisl

```
<?xml version=" 1.0 " encoding=" Shift-JIS " ?>
<!DOCTYPE xisl SYSTEM " xisl.dtd " >
<xisl version=" 1.0 " >
  <head> ..... </head>
  <body>
    <dialog id=" Hello World " >
      <exchange>
        <operation target=" hello.xml " >
          <input type=" touch " event=" click " match=" /page " />
        </operation>
        <action>
          <output type=" speech " event=" tts-speech " >
            <![CDATA[
              <param name=" speech_text " > Hello World! </param>
            ]]>
          </output>
        </action>
      </exchange>
    </dialog>
  </body>
</xisl>
```

図 1: XML ドキュメントと XISL ドキュメントの記述例

## 2. XISL 実行システム

XISL 実行システムは図 2 に示すように Front-end ( フロントエンド ) , Dialogue Manager ( 対話制御部 ) , Documents Server ( ドキュメントサーバ ) から構成される。ドキュメントサーバは、一般的な Web サーバである。XISL , XML , およびその他のドキュメント ( XSL で記述されたスタイルシートなど ) を保持し、これらのドキュメントを必要に応じて対話制御部に提供する。対話制御部では XISL をはじめとする各ドキュメントを解釈し、対話の進行状況を管理すると同時に、入出力の制御を行なう。フロントエンド部は端末部分であり、音声認識・合成エンジンやブラウザ等を用いて入出力処理を行なう。理想的には対話制御部は端末に非依存になるよう実装し、移植性を高くすることが望ましい。

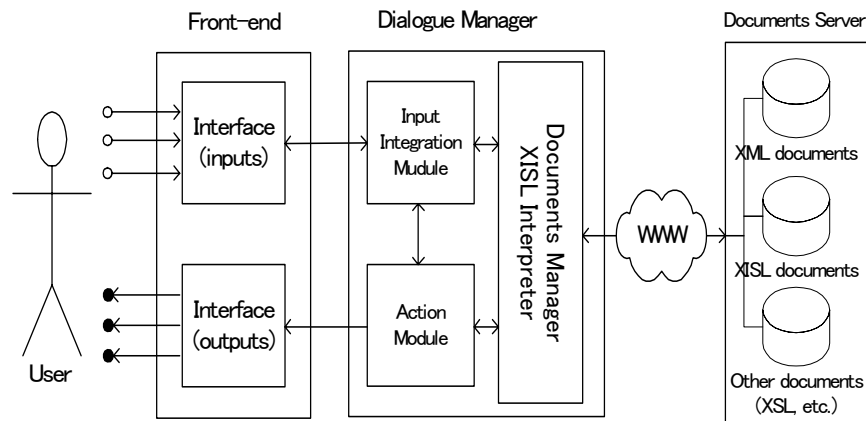


図 2: XISL 実行システムのアーキテクチャ

### 3. XISL の目標

XISL の主な目標は、

入出力モダリティの異なる様々なフロントエンド(携帯電話, PC, デジタル TV etc.)  
の利用を可能にし, シームレスな Web サービスを実現すること,  
複雑なマルチモーダル対話を容易に記述できるようにすること,  
コンテンツと対話の分離により双方の再利用性を高めること,  
ユーザ主導, システム主導, 混合主導の対話が記述できること,

である。を可能とするために, XISL は入出力モダリティを任意に拡張できる自由度を持つ。を実現するために, XISL には, 並行的, 逐次的, 択一的な対話の進行, 割込み対話, 条件処理, 繰り返し処理, 簡単な演算処理, XML ドキュメントへの読み書き等を実行するための命令が用意されている。を可能とするために, XISL は XML コンテンツから分離させ, 対話のみを記述する構造をとっている。を可能とするために, ユーザへのプロンプトを出力する対話と, 出力しない対話の双方が記述できるようになっている。

### 4. XISL の範囲

XISL は任意の XML 要素とユーザとのインタラクションを記述するための言語である。XML 要素は何らかの形でユーザに提示されており(画面上に表示されている, あるいは, ハードとして提供されている), それに対してユーザが何らかの操作を実行することによって, 対話が進行する。システムとしては, 一般的には, パーソナルコンピュータが想定できるが, XISL では特定のシステムを対象とせず, 利用モダリティも制限しない。(例えば様々な感知センサを入力モダリティとして持ち, 出力として 3 次元的な動作を行なうロボットを想定することも可能である。)

## 5. 対話レベル

XISL は、セッション、アプリケーション、ドキュメント、ダイアログ、エクスチェンジの 5 つの対話レベルを持つ。これらは上記の順に階層関係になっている。ユーザは常に一つのダイアログを実行していることになり、同時に、そのダイアログを含むドキュメント、アプリケーション、セッションを実行していることになる。また、ユーザは、ダイアログに含まれる一つのエクスチェンジの実行状態、もしくは一つ以上のエクスチェンジの待機状態にある。セッションおよびエクスチェンジを除く全てのレベルの対話に遷移でき、セッションを除く全てのレベルで変数を設定できる。対話レベルの関係を図 3 に示す。

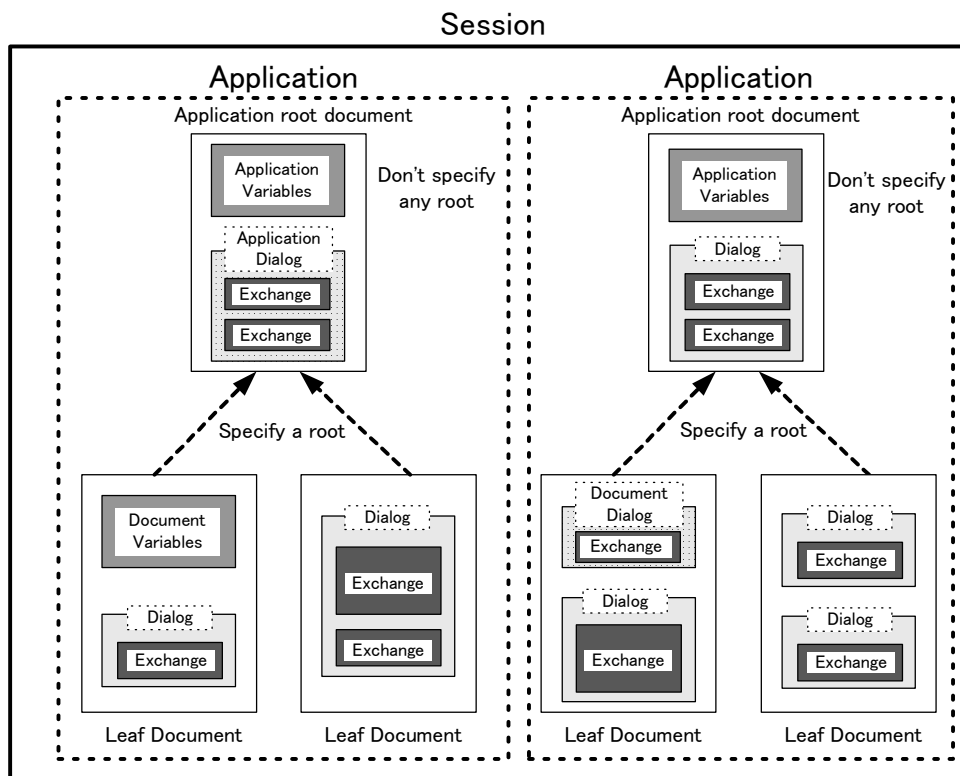


図 3: 対話レベルの関係

### 5.1. エクスチェンジ

**エクスチェンジ**には、ユーザとシステムとの対話の最小単位であるインタラクションが記述される。エクスチェンジには 2 種類の記述方法がある。一つは、XML 要素に対するユーザの操作を最初に記述し、それらの入力に対するシステムの動作を続いて記述する方法である。これはユーザ主導の対話を記述する際に用いる。もう一つは、ユーザに入力を促すためのシステムの出力をまず記述し、続いてユーザの操作を、最後にシステムの動作を記述する方法である。これはシステム主導の対話を記述する際に用いる。

## 5.2. ダイアログ

**ダイアログ**はエクスチェンジの集合であり、一組の対話が記述される。ユーザは常に1つのダイアログを実行していることになる。ダイアログ内に列挙されたエクスチェンジは、対話制御に関する記述がない限り、記述された順に実行される。ダイアログ内では、次に遷移するダイアログを指定できる。

各ダイアログは、ダイアログレベル、ドキュメントレベル、アプリケーションレベルのいずれかのレベルに属する。ダイアログレベルのダイアログは、明示的にそのダイアログが遷移先として指定されたときにのみ実行できる。つまり、遷移先として指定されない限り実行されることはない。

## 5.3. ドキュメント

**ドキュメント**はダイアログの集合であり、一つの XISL ファイルに相当する。各 XISL ドキュメントではドキュメントレベルの変数、およびダイアログを宣言できる。ドキュメントレベルの変数はそのドキュメントの実行中、アクセス可能であり、他のドキュメントに移った時にアクセス不可能になる。ドキュメントレベルのダイアログはそのドキュメントの実行中、常時実行可能であり、任意の時点でそのダイアログに遷移できる。他のドキュメントに移った時、元のドキュメントレベルのダイアログは実行不可能になる。

## 5.4. アプリケーション

**アプリケーション**は一つのタスクを遂行するためのドキュメントの集合である。一つのアプリケーションには、一つのアプリケーションルートドキュメントと、0 個以上のリーフドキュメントが含まれる。アプリケーションルートドキュメントではアプリケーションレベルの変数、およびダイアログを宣言できる。

一つのアプリケーションの実行中、必ず一つのドキュメントが実行される。リーフドキュメントが実行される場合には、同時にアプリケーションルートドキュメントが読み込まれ、アプリケーションレベルの変数やダイアログが利用可能になる。

アプリケーションレベルの変数はそのアプリケーションの実行中、アクセス可能であり、アプリケーション外のドキュメントに移った時にアクセス不可能になる。アプリケーションレベルのダイアログはそのアプリケーションの実行中、常時実行可能であり、任意の時点でそのダイアログに遷移できる。アプリケーション外のドキュメントに移った時、元いたアプリケーションのアプリケーションレベルのダイアログは実行不可能になる。

## 5.5. セッション

**セッション**は、ユーザが XISL 実行システムに接続した時に始まる。XISL ドキュメントを読み込み、対話を進行、遷移させながら継続し、ユーザ、XISL 実行システムのいずれか

の要求によって終了する。

## 6. XISL 構成要素

| 要素               | 目的                                                                                              | ページ |
|------------------|-------------------------------------------------------------------------------------------------|-----|
| <action>         | システムの動作，ユーザへの出力を行なう。                                                                            | 35  |
| <alt_exchange>   | 複数の<exchange>を囲み，択一的な対話の流れを作る。                                                                  | 29  |
| <alt_input>      | 複数の<input>を囲み，択一的な入力のリ組み合わせを記述する。                                                               | 34  |
| <assign>         | 値を変数に割り当てる。                                                                                     | 35  |
| <author>         | 著者名を記述する。                                                                                       | 13  |
| <begin>          | <dialog>の導入処理を記述する。                                                                             | 27  |
| <body>           | 対話内容を記述する。                                                                                      | 12  |
| <call>           | 他のダイアログへ遷移する。                                                                                   | 39  |
| <case>           | 属性 value の値と，<switch>の属性 expr の値とが一致した場合に行なう処理を記述する。                                            | 37  |
| <date>           | 更新日を記述する。                                                                                       | 13  |
| <details>        | 詳細情報を記述する。                                                                                      | 13  |
| <delete_element> | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素に含まれる任意の XML 要素を削除する。                                      | 55  |
| <dialog>         | 一組の対話を記述する。                                                                                     | 25  |
| <dialog_var>     | ダイアログ変数を宣言する。                                                                                   | 20  |
| <document_var>   | ドキュメント変数を宣言する。                                                                                  | 19  |
| <else>           | 親要素である<if>，または<elseif>の属性 cond で指定された論理式が，偽の場合実行される。                                            | 36  |
| <elseif>         | 条件処理を行なう。                                                                                       | 36  |
| <end>            | <dialog>の終了処理を記述する。                                                                             | 27  |
| <exchange>       | ユーザとの対話の最小単位を記述する。                                                                              | 27  |
| <exit>           | 実行中のダイアログ，ドキュメント，アプリケーションを終了する。                                                                 | 41  |
| <get_attribute>  | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素の属性値を取得する。                                     | 49  |
| <get_element>    | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素を取得する。                                         | 47  |
| <get_value>      | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素の内容を取得する。ただし，XML 要素の直接の子供となる内容のテキストデータのみを取得する。 | 48  |
| <goto>           | 他のダイアログへ遷移する。ただし元の対話へ復帰できない。                                                                    | 40  |
| <head>           | XISL ドキュメントに関するメタ情報を記述する。                                                                       | 12  |
| <history>        | 更新履歴を記述する。内部に<item>を含む。                                                                         | 13  |
| <if>             | 条件処理を行なう。                                                                                       | 36  |
| <input>          | ユーザの 1 入力を記述する。                                                                                 | 33  |

|                 |                                                                |    |
|-----------------|----------------------------------------------------------------|----|
| <item>          | 一つの更新履歴を記述する．内部に<date> , <details>を含む．                         | 13 |
| <name>          | 対話の名前を記述する．                                                    | 13 |
| <operation>     | ユーザから受け付ける入力を記述する．                                             | 32 |
| <other>         | 全ての<case>が実行されなかった場合の処理を記述する．                                  | 37 |
| <output>        | ユーザへの 1 出力を記述する．                                               | 43 |
| <par_exchange>  | 複数の<exchange>を囲み，並行的な対話の流れを作る．                                 | 29 |
| <par_input>     | 複数の<input>を囲み，並行入力の組み合わせを記述する．                                 | 34 |
| <par_output>    | 複数の<output>を囲み，同時出力を行なう．                                       | 45 |
| <prompt>        | ユーザに入力を促すための出力を記述する．                                           | 31 |
| <reprompt>      | <exchange>内の<prompt>を再実行する．                                    | 31 |
| <return>        | 呼び出し元のダイアログへ戻る．                                                | 39 |
| <seq_exchange>  | 複数の<exchange>を囲み，逐次的な対話の流れを作る．                                 | 29 |
| <seq_input>     | 複数の<input>を囲み，逐次的な入力の組み合わせを記述する．                               | 34 |
| <seq_output>    | 複数の<output>を囲み，逐次的な出力を行なう．                                     | 45 |
| <set_attribute> | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素に含まれる任意の XML 要素の属性値を更新する． | 54 |
| <set_element>   | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素に含まれる任意の XML 要素を更新する．     | 51 |
| <set_value>     | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素に含まれる任意の XML 要素の内容を更新する．  | 53 |
| <submit>        | 変数の値を HTTP GET または POST リクエスト経由でドキュメントサーバに送信し，戻り値を受信する．        | 40 |
| <switch>        | 多分岐処理を行なう．                                                     | 37 |
| <then>          | 親要素である<if> , または<elseif>の属性 cond で指定された論理式が，真の場合実行される．         | 36 |
| <var>           | 変数を宣言する．                                                       | 17 |
| <value>         | <output>内に変数の値を埋め込む．                                           | 42 |
| <while>         | 属性 cond に記述される論理式の値が真である間 , <while>内の処理を行なう．                   | 37 |
| <xisl>          | XISL ドキュメントのルート要素                                              | 12 |

## 7. XISL ドキュメントの構造と実行

### 7.1. ドキュメントの構造

XISL は XML 文書である．XISL の利用者は XML1.0 で定義されている概念と用語に馴染んでいることを期待される．XISL の文法は付録 1.の DTD によって定義される．XISL の基本構造を以下に示す．

```

<?xml version= " 1.0 " encoding= " Shift-JIS " ?>
<!DOCTYPE xisl SYSTEM " ./xisl.dtd " >
<xisl version= " 1.0 " >
  <head>
    --title, author, history, etc. --
  </head>
  <body>
    --contents--
    .
    .
  </body>
</xisl>

```

XISL のルート要素は<xisl>であり，内部には子要素である<head>と<body>を，この順に記述する．属性 version は必須であり，XISL のバージョンを指定する．本仕様の XISL のバージョンは“ 1.0 ”である．属性 application にはアプリケーションルートドキュメント（7.2節参照）の URI を指定する．指定したドキュメントが存在しなければ実行時エラーになる．

<head>には，製作者，更新履歴などの，対話とは直接関係のないドキュメントのメタ情報を記述し，<body>には，対話の内容を記述する．

|    |                   |                                     |
|----|-------------------|-------------------------------------|
| 要素 | <xisl>            |                                     |
| 目的 | XISL ドキュメントのルート要素 |                                     |
| 属性 | version           | XISL のバージョンを指定する．本バージョンは“ 1.0 ”である． |
|    | application       | アプリケーションルートドキュメントの URI を指定する．       |

|    |                           |  |
|----|---------------------------|--|
| 要素 | <head>                    |  |
| 目的 | XISL ドキュメントに関するメタ情報を記述する． |  |
| 属性 | なし．                       |  |

|    |            |  |
|----|------------|--|
| 要素 | <body>     |  |
| 目的 | 対話内容を記述する． |  |
| 属性 | なし．        |  |

<head>の記述例を以下に示す．

```

<head>
  <name> OnLine Shopping System </name>
  <author> Takanori Kobayashi </author>
  <date> Dec.26 2001 </date>
  <details> Version 1.0 </details>
  <history>
    <item>
      <date> Jun.18 2001 </date>
      <details> Version 0.0 </details>
    </item>
    .
    .
  </history>
</head>

```

<head>内の各記述は以下の目的を持つ．

|    |           |
|----|-----------|
| 要素 | <author>  |
| 目的 | 著者名を記述する． |
| 属性 | なし．       |

|    |           |
|----|-----------|
| 要素 | <date>    |
| 目的 | 更新日を記述する． |
| 属性 | なし．       |

|    |            |
|----|------------|
| 要素 | <details>  |
| 目的 | 詳細情報を記述する． |
| 属性 | なし．        |

|    |                         |
|----|-------------------------|
| 要素 | <history>               |
| 目的 | 更新履歴を記述する．内部に<item>を含む． |
| 属性 | なし．                     |

|    |                                      |
|----|--------------------------------------|
| 要素 | <item>                               |
| 目的 | 一つの更新履歴を記述する．内部に<date>，<details>を含む． |
| 属性 | なし．                                  |

|    |             |
|----|-------------|
| 要素 | <name>      |
| 目的 | 対話の名前を記述する． |
| 属性 | なし．         |

<body>にはユーザ - システム間の対話とその制御，その他の処理を記述する．<body>は<document\_var>（8.4節参照）と<dialog>を子要素として持つ．ドキュメントは，最初の

<dialog>から実行される．遷移先が指定されずに<dialog>の実行が終了したとき，ダイアログ，ドキュメント，アプリケーションの実行が終了する．

```
<body>
  <document_var>
    --Declaration of document variables--
  </document_var>
  <dialog> --A unit of dialogue-- </dialog>
    .
    .
</body>
```

## 7.2. アプリケーションとその実行

アプリケーションルートドキュメントとリーフドキュメントの区別は，<xisl>の属性 application に URI が記述されているかどうかで行なう．属性 application にアプリケーションルートドキュメントの URI が指定されているとき，そのドキュメント自身はリーフドキュメントであり，URI で指し示されるドキュメントがアプリケーションルートドキュメントである．属性 application に URI が指定されていないとき，そのドキュメント自身がアプリケーションルートドキュメントである．1つのアプリケーションを複数のドキュメントによって構成する場合には，まず 1つのドキュメントをアプリケーションルートドキュメントとして選択し，その URI を他のドキュメントの属性 application に記述すればよい．

XISL インタプリタは，アプリケーション内のドキュメントを読み込む度に，アプリケーションルートドキュメントが既に読み込まれているかどうかチェックする．アプリケーションルートドキュメントが読み込まれていないなら，それを読み込む．アプリケーションルートドキュメントは，アプリケーションが変化しない限り読み込まれたままになっている．したがって下記の 2つの条件のうち 1つは，処理の間，常に成立する．

1. アプリケーションルートドキュメントが読み込まれ，それを実行している．
2. アプリケーションルートドキュメントとリーフドキュメントが共に読み込まれ，リーフドキュメントを実行している．

複数ドキュメントからなるアプリケーションには 2つの利点がある．1つは，アプリケーションレベルの変数をアプリケーション内の他のドキュメントで利用できることである．これにより同一アプリケーション内での情報の共有と保持が可能になる．もう 1つは，ユーザがリーフドキュメントにいるときでも，アプリケーションレベルのダイアログが常時実行可能になることである．これにより同一アプリケーション内での共通の割り込み処理が記述できるようになる．

以下に 2つのドキュメントからなるアプリケーションの例を示す．

### app-root.xisl (root document)

```
<?xml version= " 1.0 " encoding= " Shift-JIS " ?>
<!DOCTYPE xisl SYSTEM " xisl.dtd " >
<xisl version= " 1.0 " >
  <head> ..... </head>
  <body>
    <dialog id= " Help " scope= " document " >
      <exchange>
        <operation target= " help.grxml " >
          <input type= " speech " event= " recognize "
            match= " /grammar/rule[@id=help] " />
        </operation>
        <action>
          <output type= " speech " event= " tts-speech " >
            <![CDATA[
              <param name= " speech_text " > Click the Page! </param>
            ]]>
          </output>
        </action>
      </exchange>
    </dialog>
  </body>
</xisl>
```

### hello.xisl (leaf document)

```
<?xml version= " 1.0 " encoding= " Shift-JIS " ?>
<!DOCTYPE xisl SYSTEM " xisl.dtd " >
<xisl version= " 1.0 " application= " app-root.xisl " >
  <head> ..... </head>
  <body>
    <dialog id= " Hello World " >
      <exchange>
        <operation target= " hello.xml " >
          <input type= " touch " event= " click " match= " /page " />
        </operation>
        <action>
          <output type= " speech " event= " tts-speech " >
            <![CDATA[
              <param name= " speech_text " > Hello World! </param>
            ]]>
          </output>
        </action>
      </exchange>
    </dialog>
  </body>
</xisl>
```

この例では、hello.xisl がリーフドキュメントとして実行されることを想定している。hello.xisl の<xisl>の属性 application には app-root.xisl がアプリケーションルートドキュメントとして指定されている。app-root.xisl では、<dialog>の属性 scope が document になっている。これはダイアログ「Help」がアプリケーションレベルで宣言されていることを意味する。したがってユーザの「Help」に関する発話はそのアプリケーション全体で常に受け付けられることになる。<dialog>タグの属性 scope とダイアログのレベルの詳細な関係については10.1節を参照されたい。

### 7.3. ダイアログの遷移とアプリケーションの変化

<goto>もしくは<call>によってダイアログが遷移する場合、アプリケーションが変化するときと、変化しないときがある。また遷移によってドキュメントが変化する場合がある。以下にダイアログの遷移とアプリケーション、ドキュメントの変化の関係を示す。また、図 4にダイアログの遷移とアプリケーションの変化の関係を示す。

- ドキュメントが変化しない遷移  
実行中のダイアログを含むドキュメントと、遷移先のダイアログを含むドキュメントが同じであるとき、ドキュメントは変化しない。
- ドキュメントが変化する遷移  
実行中のダイアログを含むドキュメントと、遷移先のダイアログを含むドキュメントが異なるとき、ドキュメントは変化する。
- アプリケーションが変化しない遷移  
アプリケーションが変化しない遷移は、以下いずれかに当てはまるダイアログの遷移である。
  - 実行中のダイアログのアプリケーションルートドキュメントと、遷移先のダイアログのアプリケーションルートドキュメントが同じである。
  - 実行中のダイアログがドキュメント D に含まれ、遷移先のダイアログのアプリケーションルートドキュメントが D である。
  - 実行中のダイアログのアプリケーションルートドキュメントがドキュメント D であり、遷移先のダイアログが D に含まれる。
  - 実行中のダイアログがアプリケーションルートドキュメント D に含まれ、遷移先のダイアログも D に含まれる。
- アプリケーションが変化する遷移  
上記のアプリケーションが変化しない遷移に当てはまらないダイアログの遷移は、アプリケーションを変化させる。

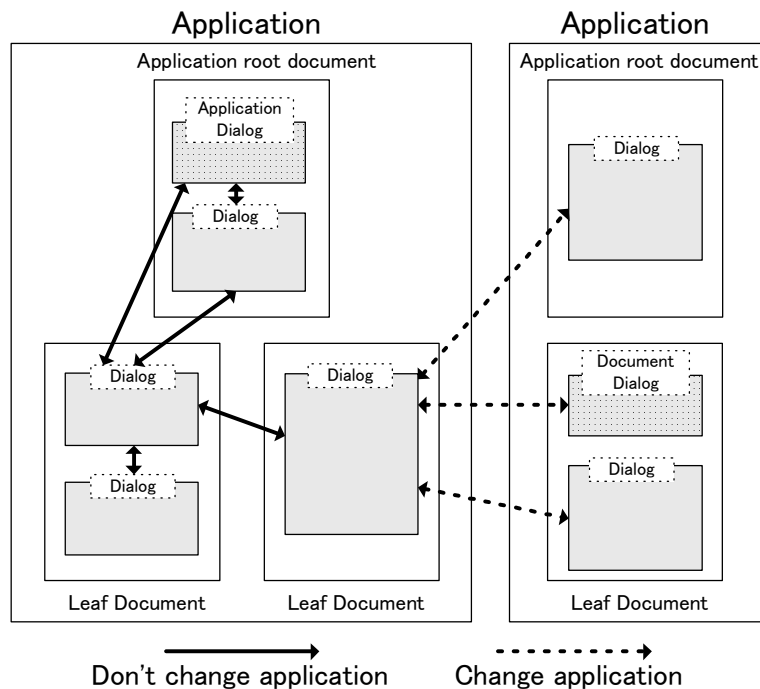


図 4: ダイアログの遷移とアプリケーションの変化の関係

## 8. 変数

### 8.1. 変数宣言 - var

変数は、`<var>`要素により宣言される。属性 `name` は必須属性であり、宣言する変数の名前を記述する。属性 `expr` には初期値を指定する。変数名の命名規則は、ECMAScript の変数の命名規則に準拠する。変数名の指定は必須であるが、初期値の指定は任意とする。初期値を指定しない場合のデフォルト値は、ECMAScript の `undefined` 値とする。スコープが `session`、`local`(8.2節参照)の変数を除き、使用前に必ず宣言しなければならない。

```
<var name= " home_phone " />
<var name= " one " expr= " 1 " />
<var name= " person " expr= " 'Yusaku Nakamura ' " />
```

|    |                          |              |
|----|--------------------------|--------------|
| 要素 | <code>&lt;var&gt;</code> |              |
| 目的 | 変数を宣言する。                 |              |
| 属性 | name                     | 変数の名前を記述する。  |
|    | expr                     | 変数の初期値を指定する。 |

## 8.2. 変数のスコープ

変数は以下のスコープに対して宣言できる．

| スコープ        | 概要                                                                                                                                                                                                     |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| session     | セッション全体で有効な変数であり，セッション内の任意のダイアログからアクセス可能である．対話制御部で予め宣言されており，新しいセッション変数を XISL ドキュメントで宣言することはできない．                                                                                                       |
| application | アプリケーション全体で有効な変数であり，アプリケーション内の任意のダイアログからアクセス可能である．アプリケーションルートドキュメント中の<document_var>内の<var>によって宣言される．アプリケーション変数は，アプリケーションルートドキュメントが読み込まれたときに初期化され，アプリケーションルートドキュメントが読み込まれている間アクセスできる．                   |
| document    | ドキュメント全体で有効な変数であり，ドキュメント内の任意のダイアログからアクセス可能である．XISL ドキュメント中の<document_var>内の<var>によって宣言される．ドキュメント変数はドキュメントが読み込まれたときに初期化され，当該ドキュメントが読み込まれている間アクセスできる．                                                    |
| dialog      | ダイアログ全体で有効な変数であり，そのダイアログ内の各要素からアクセス可能である．<dialog_var>内の<var>により宣言される．また，<dialog>の属性 arg で指定された引数は，ダイアログ変数として扱われる．ダイアログ変数はダイアログの実行が始まったときに初期化され，当該ダイアログが実行されている間アクセスできる．                                |
| local       | <begin>，<end>，<exchange>内で閉じたスコープをもつ変数であり，それぞれの要素内からのみアクセス可能である．変数宣言は特に必要としないが，<var>で行なう事も可能である．ローカル変数は<begin>，<end>，<exchange>内で宣言もしくは新規に検出されたときに初期化され，当該<begin>，<action>，<exchange>が実行されている間アクセスできる． |

## 8.3. 変数参照

属性 cond，expr，namelist，および match では変数を参照できる．属性 cond，expr，namelist 内では，変数名によって参照し，属性 match 内では，“\$変数名”により変数を参照する．未定義の変数を参照した場合には，実行時エラーとなる．

属性 cond，expr，namelist での変数参照の例

```
<if cond= " person == ' Yusaku Nakamura ' " >  
  <assign name= " character " expr= " ' kind ' " />  
</if>
```

```
<assign name= " new " expr= " x+y " />
```

```
<return namelist= " person, character " />
```

#### 属性 match での変数参照の例

```
<input match= " /personal/item[@name= "$person" ] " type= " touch "
      event = " click " />
```

XISL では、スコープが異なる場合に限って、同じ名前の変数を宣言できる。ただし、セッション変数と同じ名前の変数は宣言できない。スコープが同じであり、かつ同じ名前の変数が宣言された場合には、実行時エラーとなる。またセッション変数と同じ名前の変数宣言を行なった場合も、実行時エラーとなる。

同じ変数名が存在したときは、スコープの小さい方の変数（よりローカル側の変数）を優先して参照する。スコープの大きい方の変数を参照したい場合には、下に示すように、“スコープ.変数名”の形式を用いる。（変数のスコープを明確化するために“スコープ.変数名”を用いてもよい。）

|               |            |
|---------------|------------|
| session.X     | セッション変数    |
| application.X | アプリケーション変数 |
| document.X    | ドキュメント変数   |
| dialog.X      | ダイアログ変数    |
| local.X       | ローカル変数     |

#### 8.4. アプリケーション変数とドキュメント変数

ドキュメント変数は XISL ドキュメント中の<document\_var>内の<var>で宣言する。アプリケーションルートドキュメントで宣言したドキュメント変数は、アプリケーション全体で有効なアプリケーション変数となる。

```
<body>
  <document_var>
    <var name= " user_ID " />
    <var name= " max_items " expr= " 15 " />
    .
    .
  </document_var>
  .
  .
</body>
```

|     |                |
|-----|----------------|
| 要素名 | <document_var> |
| 目的  | ドキュメント変数を宣言する。 |
| 属性  | なし。            |

## 8.5. ダイアログ変数

ダイアログ変数は<dialog>の最初の子要素である<dialog\_var>内の<var>で宣言する。また<dialog>の属性 arg の属性値である引数はダイアログ変数として宣言されたものとして扱われる。

```
<dialog id= " sample " arg= " person,character, . . . " >
  <dialog_var>
    <var name= " num_of_items "   expr= " 24 " />
    <var name= " greeting_word "  expr= " ' HELLO ' " />
    .
    .
  </dialog_var>
  .
</dialog>
```

|     |               |
|-----|---------------|
| 要素名 | <dialog_var>  |
| 目的  | ダイアログ変数を宣言する。 |
| 属性  | なし。           |

## 8.6. ローカル変数

ローカル変数は、<begin>、<exchange>、<end>内で有効な変数である。ローカル変数の宣言は特に必要としないが、<begin>、<exchange>、<end>内の子要素である<var>で行なう事も可能である。<begin>の子要素として宣言した変数は、<begin>内で有効な変数であり、他の<exchange>、<end>で宣言した場合も同様に、各要素内で有効な変数となる。

```
<exchange>
  <operation>
    <input type= " speech " event= " recognize " target= " number.grxml "
      match= " /grammar/rule[@id=number] " return= " num " />
  </operation>
  <action>
    <var name= " price " expr= " 150 " />
    <assign name= " sum " expr= " num * price " />
    .
    .
  </action>
</exchange>
```

アプリケーション変数、ドキュメント変数、ダイアログ変数は宣言が必須であるが、ローカル変数は宣言なしでも利用できる。例えば、ある変数 x とある変数 y の加算結果を変数宣言していない変数 new に格納する場合、以下のように記述すればよい。<assign>は、属性 expr の計算結果を 属性 var で指定した変数に代入するという働きを持つ要素である。

詳細については，15節を参照されたい．

```
<assign name= " new " expr= " x+y " />
```

この例の場合，変数 new は新しいローカル変数として自動的に登録され，値がセットされる．

8.7. セッション変数

XISL には，以下のセッション変数が予め準備されている．値を明示的に更新するまでは，対話制御部が設定したデフォルト値が格納される．

|         |                                  |
|---------|----------------------------------|
| TIMEOUT | 入力のタイムアウト時間を設定する変数．単位は[msec]．    |
| ERROR   | 正常状態では 0，実行時エラーが起きるとエラー番号が格納される． |

セッション変数 ERROR のエラー番号とその意味は付録 2.に示す．

```
<exchange>
  <prompt>
    .
    .
  </prompt>
  <operation>
    <input type= " speech " event= " recognize " target= " number.grxml "
      match= " /grammar/rule[@id=number] " return= " num " />
  </operation>
  <action>
    <if cond= " ERROR != 0 " />
      <then>
        <reprompt />
      </then>
      .
      .
    </action>
</exchange>
```

9. 式

XISL では，四則演算(整数のみ)，文字や数値の大小比較，AND，OR，NOT の論理演算を，属性 cond，expr 内で記述できる．式は以下の文法に従う．

|                                                                                                                                                                                                                                        |                                                         |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| 計算式 ::= 項 { { '+'   '-' } 項 }*<br>項 ::= 因子 { { '*'   '/'   '%' } 因子 }*<br>因子 ::= 数値   変数   '(' 計算式 ')'   '-' 因子   '+' 因子                                                                                                               |                                                         |
| 条件式 ::= and 式 ( '  ' and 式 )*<br>and 式 ::= 論理式 ( '&&' 論理式 )*<br>論理式 ::= '(' 条件式 ')'   '!' 論理式   比較式   '1'   'true'   '0'   'false'                                                                                                     |                                                         |
| 比較式 ::= 計算式 { '='   '!='   '<'   '>'   '<='   '>=' } 計算式  <br>文字列 { '='   '!='   '<'   '>'   '<='   '>=' } 文字列<br>数値 ::= ( 0   1   2   3   4   5   6   7   8   9 ) +<br>文字列 ::= ' シングルクォートを含まない全ての文字* '<br>変数 ::= ECMAScript で宣言可能な変数名 |                                                         |
| <b>算術演算子</b>                                                                                                                                                                                                                           |                                                         |
| 算術演算子                                                                                                                                                                                                                                  | 意味                                                      |
| +<br>-<br>*<br>/<br>%                                                                                                                                                                                                                  | 加算<br>減算<br>乗算<br>整数除算<br>剰余                            |
| <b>論理演算子</b>                                                                                                                                                                                                                           |                                                         |
| 論理演算子                                                                                                                                                                                                                                  | 意味                                                      |
| &&<br>  <br>!                                                                                                                                                                                                                          | 論理積<br>論理和<br>否                                         |
| <b>関係演算子</b>                                                                                                                                                                                                                           |                                                         |
| 演算子                                                                                                                                                                                                                                    | 意味                                                      |
| ==<br>!=<br><<br><=<br>><br>>=                                                                                                                                                                                                         | 等しい<br>等しくない<br>より小さい<br>より小さいか等しい<br>より大きい<br>より大きい等しい |

## 10. 対話の記述

<dialog>は<body>の子要素であり、一組の対話を記述する。<dialog>は<begin>（最初に実行される処理ユニット）、複数の<exchange>（対話の最小単位）、<end>（最後に実行される処理ユニット）の順に記述する。

<exchange>は、XISL で記述できる対話の最小単位であり、ユーザに入力を促すためのシステムの出力を記述する<prompt>、ユーザからの受け付ける入力を記述する<operation>、受け付けた入力に対するシステムの処理・出力を記述する<action>の順に記述する。まず<dialog>の記述例を示す。

```

<body>
  <dialog id= " order " comb= " seq " repeat= " 1 " scope= " dialog " arg= " x " >
    <begin>
      <output type= " mmi-browser " event= " open " >
        <![CDATA[
          <param name= " window-name " > order_page </param>
          <param name= " uri " > burger.xml</param>
        ]]>
      </output>
      <output type= " speech " event= " tts-speech " >
        <![CDATA[
          <param name= " speech_text " > May I help you? </param>
        ]]>
      </output>
    </begin>
    <exchange>
      <operation target= " burger.xml " >
        <input type= " touch " event= " click " match= " /hamburger " />
        <input type= " speech " event= " recognize " target= " burger.grxml "
          match= " /grammar/rule[@id=order] " />
      </operation>
      <action>
        <output type= " speech " event= " tts-speech " >
          <![CDATA[
            <param name= " speech_text " >
              O.K. You ordered a hamburger.
            </param>
          ]]>
        </output>
      </action>
    </exchange>
  </dialog>
</body>

```

|                                   |                                           |
|-----------------------------------|-------------------------------------------|
| S : [Display hamburger.xml]       | (modality: display)                       |
| May I help you?                   | (modality: speech)                        |
| U : This burger please.           | (modality: speech, grammar: burger.grxml) |
| [Push the button " hambueger " ]  | (modality: touch_panel)                   |
| S : O.K. You ordered a hamburger. | (modality: speech)                        |

<dialog> (一組の対話) は、基本的に以下の流れで進行する。

対話の初期処理 (<begin>の実行)

<exchange>を属性 comb に従って実行していく。

属性 repeat で繰り返しが指定されていれば の処理に戻る。

## 対話の終了処理（<end>内の実行）

<end>の処理が終了するまでに別<dialog>への遷移命令（<call>や<goto>）がなければ，<end>の処理が終了した時点でダイアログ，ドキュメント，アプリケーションの実行が終了する．また<begin>内や<exchange>の<action>内で，別<dialog>への遷移命令があれば，<end>の処理は実行せず，指定された<dialog>に遷移する．

### 10.1. 一組の対話 - dialog

<dialog>は，属性 id，comb，repeat，scope，arg の各属性を持つ．属性 id は，必須属性であり，対話名を記述する．対話名はドキュメント内でユニークなものとする．ドキュメント内で重複した対話名が存在した場合，実行時エラーとなる．

属性 comb では対話の流れを指定する．流れの種類は図 5に示すように，“seq”，“par”，“alt” の 3 種類がある．明示的に記述しない場合は，“seq” が指定されたとみなす．

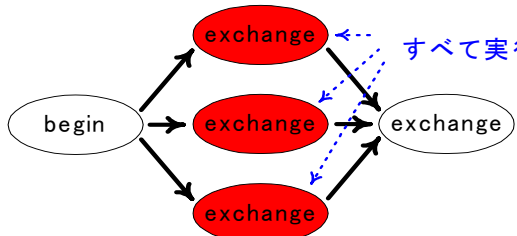
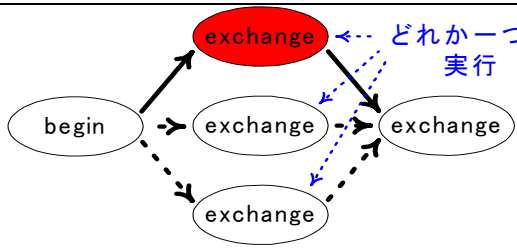
| 属性値 | 対話の流れ                                                                                                                                                                                                                                                   |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| seq |  <p style="text-align: center;"><b>seq</b><br/><b>(逐次)</b></p> <p>まず&lt;begin&gt;を実行し，続いて&lt;exchange&gt;を記述された順に実行．全ての&lt;exchange&gt;の処理が終われば，&lt;end&gt;に移る．</p>  |
| par |  <p style="text-align: center;"><b>par</b><br/><b>(並行)</b></p> <p>まず&lt;begin&gt;を実行し，続いて&lt;exchange&gt;を任意の順に実行．全ての&lt;exchange&gt;の処理が終われば，&lt;end&gt;に移る．</p>   |
| alt |  <p style="text-align: center;"><b>alt</b><br/><b>(択一)</b></p> <p>まず&lt;begin&gt;を実行し，続いてどれか 1 つの&lt;exchange&gt;を実行．その&lt;exchange&gt;の処理が終われば，&lt;end&gt;に移る．</p> |

図 5: <dialog>の属性 comb の記述

属性 repeat では対話を繰り返す回数を指定する．明示的に記述しない場合は，1 が指定されたものとみなす．また無限に繰り返す場合は“\*”を指定する．なお繰り返しの対象は<exchange>のみであり，<begin>，<end>は繰り返しの対象ではない．

属性 scope では対話レベルを指定する．“dialog”，“document”のいずれかを指定できる．“dialog”が指定された<dialog>は，ダイアログレベルの対話であり，後述する<call>や<goto>（18節参照）等で明示的にその対話が遷移先として指定されたときにのみ実行される．つまり，遷移先として指定されない限り，実行されることはない．一方，“document”が指定された<dialog>は，ドキュメントレベルの対話であり，同じドキュメント内の別の<dialog>が実行中であっても，常時実行可能である．この様な<dialog>を割り込み対話といい，割り込み対話を実行するきっかけになる入力を割り込み入力という．割り込み入力は，割り込み対話の<begin>に続いて実行可能な<exchange>内の<operation>である．ある<dialog>を実行中に割り込み入力があった場合，実行中の<dialog>を一時，停止状態にする．次に入力を受け付けた割り込み対話を実行状態にし，割り込み対話の処理を進める．この時，入力を受け付けた<exchange>から実行し，その割り込み対話内の<begin>は実行されない．その後の処理は，10節の ～ に示した<dialog>の基本的な流れと同じである．ドキュメントレベルの対話のうち，特に，アプリケーションルートドキュメント内の対話は，アプリケーションレベルの対話であり，そのアプリケーション実行中に常時実行可能である．属性 scope を明示的に記述しない場合は“dialog”が指定されたものとみなす．

属性 arg では，<call>や<goto>によりこの<dialog>に遷移した時に，<dialog>が受け取る変数を指定する．変数は“，”で区切って列挙する．属性 arg に列挙された変数の個数と，<call>，および<goto>の属性 namelist に列挙された変数の個数が一致しない場合，実行時エラーになる．

|    |             |                                                                            |
|----|-------------|----------------------------------------------------------------------------|
| 要素 | <dialog>    |                                                                            |
| 目的 | 一組の対話を記述する． |                                                                            |
| 属性 | id          | 対話名を記述する．                                                                  |
|    | comb        | <dialog>内の<exchange>の流れを記述する．“seq”，“par”，“alt”のいずれかの値をとる．デフォルト設定は“seq”である． |
|    | repeat      | 対話の繰り返し回数を記述する．1以上の整数もしくは“*”（無限大を表す）のいずれかの値をとる．デフォルト設定は“1”である．             |
|    | scope       | 対話レベルを指定する．“dialog”，“document”のいずれかの値をとる．デフォルト設定は“dialog”である．              |
|    | arg         | 対話に引き渡される引数を列挙する．複数の引数は“，”で区切る．                                            |

## 10.2. 対話の最小単位 - begin，exchange，end

<dialog>は対話の最小単位を表す<begin>，<exchange>，<end>から構成される．<begin>には対話の導入処理を，<exchange>にはユーザとの対話の最小単位を，<end>には対話の

終了処理を記述する。

<begin> , <end>には後述する<action>とほぼ同じ内容が記述できる (<reprompt>のみが記述できない) . <exchange>は表 1に示すように , 2 種類の方法で記述できる .

表 1: <exchange>の記述方法

|                       |                                                                   |
|-----------------------|-------------------------------------------------------------------|
| ユーザ主導型<exchange>の記述   | 1 XML 要素に対するユーザの操作を記述<br>2 システムの動作を記述                             |
| システム主導型の<exchange>の記述 | 1 ユーザに入力を促すためのシステムの出力を記述<br>2 XML 要素に対するユーザの操作を記述<br>3 システムの動作を記述 |

ユーザ主導型の<exchange>は , XML 要素に対するユーザの操作と , それに対するシステムの動作に関する記述からなる . 一方 , システム主導型の<exchange>は , ユーザに入力を促すためのシステムの出力 , それを受けたユーザの操作 , それに対するシステムの動作に関する記述からなる . ユーザ主導型の<exchange>の記述例を図 6に , システム主導型の<exchange>の記述例を図 7にそれぞれ示す .

```
<exchange>
  <operation target= " burger.xml " >
    <input type= " touch " event= " click " match= " /hamburger " />
    <input type= " speech " event= " recognize " target= " burger.grxml "
      match= " /grammar/rule[@id=order] " />
  </operation>
  <action>
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
        <param name= " speech_text " > O.K. You ordered a hamburger. </param>
      ]]>
    </output>
  </action>
</exchange>
```

図 6: ユーザ主導型の<exchange>の記述例

U : This burger please. (modality: speech, grammar: burger.grxml)  
[Push the button " hambueger " ] (modality: touch\_panel)  
S : O.K. You ordered a hamburger. (modality: speech)

```
<exchange>
  <prompt>
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
```

```

    <param name= " speech_text " > Would you like a hamburger? </param>
  ]]>
</output>
</prompt>
<operation target= " burger.xml " >
  <input type= " speech " event= " recognize " target= " yes.grxml "
    match= " /grammar/rule[@id=yes] " />
</operation>
<action>
  <output type= " speech " event= " tts-speech " >
    <![CDATA[
      <param name= " speech_text " > O.K. You ordered a hamburger. </param>
    ]]>
  </output>
</action>
</exchange>

```

図 7: システム主導型の<exchange>の記述例

S : Would you like a hamburger? (modality: speech)  
 U : Yes. (modality: speech, grammar: yes.grxml)  
 S : O.K. You ordered a hamburger. (modality: speech)

|     |                      |
|-----|----------------------|
| 要素名 | <begin>              |
| 目的  | <dialog>の導入処理を記述する . |
| 属性  | なし .                 |

|     |                     |
|-----|---------------------|
| 要素名 | <exchange>          |
| 目的  | ユーザとの対話の最小単位を記述する . |
| 属性  | なし .                |

|     |                      |
|-----|----------------------|
| 要素名 | <end>                |
| 目的  | <dialog>の終了処理を記述する . |
| 属性  | なし .                 |

### 10.3. 複雑な対話の流れ - seq\_exchange , par\_exchange , alt\_exchange

<dialog>の属性 comb では , 図 5に示すような<exchange>の流れを指定できる . これに加えて , XISL では , 局所的に逐次 , 並行 , あるいは択一を指定するための要素も用意している .

これらの要素の使用例を以下に示す

```

<dialog comb= " alt " >
  <seq_exchange>

```

```

<exchange>
  <operation target= " burger.xml " >
    <input type= " touch " event= " click " match= " /hamburger " />
  </operation>
  <action>
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
        <param name= " speech_text " >
          How many?
        </param>
      ]]>
    </output>
  </action>
</exchange>
<exchange>
  <operation>
    <input type= " speech " event= " recognize " target= " burger.grxml "
      match= " /grammar/rule[@id=number] " />
  </operation>
  <action>
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
        <param name= " speech_text " >
          O.K.
        </param>
      ]]>
    </output>
  </action>
</exchange>
</seq_exchange>
<exchange>
  <operation>
    <input type= " speech " event= " recognize " target= " burger.grxml "
      match= " /grammar/rule[@id=order] " />
  </operation>
  <action>
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
        <param name= " speech_text " >
          O.K.
        </param>
      ]]>
    </output>
  </action>
</exchange>
:/dialog>

```

U : [Push the button “ hambueger ”] (modality: touch\_panel)  
 S : How many? (modality: speech)  
 U : Two. (modality: speech)  
 S : O.K. (modality: speech)

or

U : Two hamburgers, please. (modality: speech)  
 S : O.K. (modality: speech)

|    |                                |
|----|--------------------------------|
| 要素 | <seq_exchange>                 |
| 目的 | 複数の<exchange>を囲み，逐次的な対話の流れを作る． |
| 属性 | なし．                            |

|    |                                |
|----|--------------------------------|
| 要素 | <par_exchange>                 |
| 目的 | 複数の<exchange>を囲み，並行的な対話の流れを作る． |
| 属性 | なし．                            |

|    |                                |
|----|--------------------------------|
| 要素 | <alt_exchange>                 |
| 目的 | 複数の<exchange>を囲み，択一的な対話の流れを作る． |
| 属性 | なし．                            |

## 11. ユーザに入力を促すためのシステムの出力 - prompt, reprompt

<prompt>は，システム主導型の<exchange>において，ユーザに対して入力を促すために用いられる．内部に複数の<output>を含む．また，<reprompt>は<exchange>内の<prompt>を再実行するために使用する．<exchange>内に<prompt>を記述せずに<reprompt>を記述した場合には，実行時エラーとなる．

一つの<exchange>には属性 count の値が異なる<prompt>を列挙できる．このとき最初に実行される<prompt>は count の値が 1 のもの(あるいは count が指定されていないもの)である．その後，<exchange>内の<reprompt>が実行されたとき，<prompt>が再び実行される．<prompt>の再実行が繰り返され，別の<prompt>の属性 count で指定された回数繰り返された(繰り返しの回数と count の値が一致した)とき，それ以降は count の回数一致した<prompt>が実行される．

属性 timeout には<prompt>を自動的に繰り返す際の間隔をミリ秒で記述する．属性 timeout に何らかの値が指定されている場合にユーザの入力がなかった場合，属性 timeout に指定された時間ごとに<prompt>が繰り返される．属性 bargein では，<prompt>の実行中にユーザの入力があった場合に，プロンプトを中断すべきかどうか指定する．中断してもよい場合には“ true ”を，中断してはならない場合には“ false ”を記述する．

以下に<prompt>の記述例を示す．

```

<exchange>
  <prompt>
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
        <param name= " speech-text " >
          Please input the title of the book you want to buy.
        </param>
      ]]>
    </output>
    <output type= " timer " event= " create " >
      <![CDATA[
        <param name= " timer-id " > noinput </param>
        <param name= " interval " > 60000 </param>
      ]]>
    </output>
  </prompt>
  <prompt count= " 3 " timeout= " 60000 " >
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
        <param name= " speech-text " >
          Title please.
        </param>
      ]]>
    </output>
  </prompt>
  <operation comb= " alt " >
    <!-- Waiting for the input of user ' s speech.-->
    <input type= " speech " event= " recognize " target= " book.grxml "
      match= " /grammar/rule[@id=title] " return= " result " />
    <!-- Waiting for the input of " noinput " . -->
    <input type= " timer " event= " timeout " target= " book.xml "
      match= " /page " return= " result " />
  </operation>
  <action>
    <!-- " noinput " ? -->
    <if cond= " result == ' noinput ' " >
      <!-- If the value of cond is true, the prompt is executed again. -->
      <then>
        <output type= " timer " event= " destroy " >
          <![CDATA[
            <param name= " timer-id " > noinput </param>
          ]]>
        </output>
        <reprompt/>
      </then>
    <else>

```

```

    <output type= " speech " event= " agent-speech " >
      <![CDATA[
        <param name= " speech-text " > 0.K. You ordered a book. </param>
      ]]>
    </output>
  </else>
</action>
</exchange>

```

S : Please input the title of the book you want to buy. (modality: speech)  
 U : [noinput 60s] (modality: timer)  
 S : Please input the title of the book you want to buy. (modality: speech)  
 U : [noinput 60s] (modality: timer)  
 S : Title please. (modality: speech)  
 :

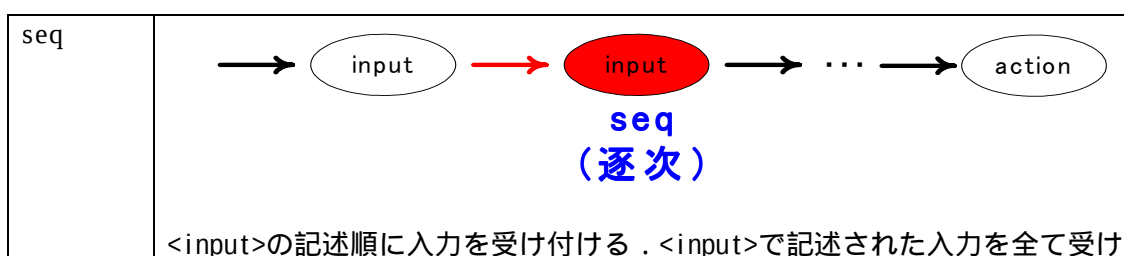
|    |                      |                                             |
|----|----------------------|---------------------------------------------|
| 要素 | <prompt>             |                                             |
| 目的 | ユーザに入力を促すための出力を記述する． |                                             |
| 属性 | count                | 当該<prompt>の実行のタイミングを整数値で記述する．デフォルト設定は“ 1 ”． |
|    | timeout              | 自動的に<reprompt>を実行する間隔（ミリ秒）を記述する．            |
|    | bargain              | ユーザがプロンプトを中断できるかどうかを指定する．デフォルト設定は“ true ”．  |

|    |                             |  |
|----|-----------------------------|--|
| 要素 | <reprompt>                  |  |
| 目的 | <exchange>内の<prompt>を再実行する． |  |
| 属性 | なし．                         |  |

## 12. ユーザの入力

### 12.1. operation

<operation>は，<input>（12.2節参照）の集合から構成される．属性 target は，入力の対象となる XML 要素が含まれるドキュメントの場所を URI で指定する．属性 comb は入力の組み合わせを指定する．属性 comb は図 8に示す値を取ることができる．



|     |                                                                                      |
|-----|--------------------------------------------------------------------------------------|
|     | 付ければ，<action>内の実行に移る．                                                                |
| par | <p>&lt;input&gt;で記述された入力を同時もしくは順不同に受け付ける．記述された入力を全て受け付ければ，&lt;action&gt;内の実行に移る．</p> |
| alt | <p>&lt;input&gt;で記述された入力のうち，どれか一つの入力を受け付けば，&lt;action&gt;内の実行に移る．</p>                |

図 8: <operation>の属性 comb の記述

以下に<operation>の記述例を示す．

```
<operation target= " burger.xml " comb= " par " >
  <input type= " touch " event= " click " match= " /hamburger " />
  <input type= " speech " event= " recognize " target= " burger.grxml "
    match= " /grammar/rule[@id=order] " />
</operation>
```

| 要素 | <operation>        |                                                                             |
|----|--------------------|-----------------------------------------------------------------------------|
| 目的 | ユーザから受け付ける入力を記述する． |                                                                             |
| 属性 | target             | 入力の対象となる XML 要素を含むドキュメントの場所を示す URI を記述する．                                   |
|    | comb               | 要素内の<input>の入力の組み合わせを記述する．“ par ”，“ alt ”，“ seq ” のいずれか値をとる．デフォルトは “ par ”． |

## 12.2. input

<input>は，ユーザからの 1 入力を記述するために用いる．属性 type，属性 event，および属性 match は必須属性である．

属性 type は入力モダリティを指定する．例えば，入力モダリティとして音声を使用するなら音声( “ speech ” )，マウスを使用するならタッチ( “ touch ” )等を指定する．属性 type の取り得る値はフロントエンドの開発者が決定するものとし，XISL ではその詳細を規定しない．

属性 event は入力イベントを指定する．例えば，音声認識ならば “ recognize ” といった値を，マウス操作ならばクリック( “ click ” )やダブルクリック( “ d\_click ” )等といった値を記述する．属性 type の取り得る値はフロントエンドの開発者が決定するものとし，XISL ではその詳細を規定しない．

属性 target は入力対象となる XML 要素が含まれるドキュメントの場所を URI で指定する．<operation>と<input>の双方において属性 target が指定されていた場合，<input>での指定を優先する．

属性 return には変数を “ , ” で区切って列挙する．ユーザから入力された値，文字列，認識語彙など，フロントエンドから送信される入力データが格納される．属性 return に記述された変数は，その場でローカル変数として宣言されたことになる．属性 return に記述する変数の数，およびそれぞれに代入される値はフロントエンドの開発者が決定するものとし，XISL ではその詳細を規定しない．

属性 match では，入力対象とする XML 要素名を指定する．指定方法は原則として XPath の書式に従うが，利便性のために，入力対象となった XML 要素の属性値を変数に代入できるようにしている．以下に要素の属性値を変数に格納する方法を示す．

```
<input match= “ XML 要素名[変数名 := @属性名] ” />
```

記述例を以下に示す．

```
<input match= “ /goodslist/goods[goods_id:=@id] ” type= “ touch ” event= “ click ” />
```

これにより，入力対象となった要素 goods の属性 id の値を変数 goods\_id に格納することができる．属性 match で指定する XML 要素はフロントエンドの開発者が決定するものとし，XISL ではその詳細を規定しない．

|    |                 |                                           |
|----|-----------------|-------------------------------------------|
| 要素 | <input>         |                                           |
| 目的 | ユーザの 1 入力を記述する． |                                           |
| 属性 | type            | 入力モダリティを指定する．                             |
|    | event           | 入力イベントを指定する．                              |
|    | target          | 入力の対象となる XML 要素を含むドキュメントの場所を示す URI を記述する． |
|    | match           | 入力対象である，XML コンテンツ内の要素名を指定する．              |

|  |        |                                         |
|--|--------|-----------------------------------------|
|  | return | ユーザから入力された値を，格納する変数を列挙する．複数の変数は“，”で区切る． |
|--|--------|-----------------------------------------|

### 12.3. 複雑な入力の組み合わせ

<operation>の属性 comb では，図 8に示すような<input>の入力の組み合わせを記述できる．これに加えて，XISL では<operation>内における局所的な，逐次入力，並行入力，択一入力を記述するための要素も用意している．

記述例を以下に示す．この例は，オンラインショッピングシステムにおいて，ある商品のボタンを押しながら音声で購入個数を入力するか，音声のみで商品名と購入個数を入力するかの，2通りの入力の受け付けが可能であることを示している．

```
<operation target=“ ./ols.xml ” comb=“ alt ”>

  <par_input>
    <input type=“ touch ” event=“ click ”
           match=“ /goods_list/goods[goods:=@id]/button[1] ” />
    <input type=“ speech ” event=“ recognize ” target=“ ols.grxml ”
           match=“ /grammar/rule[@id=number] ” />
  </par_input>

  <input type=“ speech ” event=“ recognize ” target=“ ols.grxml ”
         match=“ /grammar/rule[@id=goods_number] ” />
</operation>
```

|    |                                  |
|----|----------------------------------|
| 要素 | <seq_input>                      |
| 目的 | 複数の<input>を囲み，逐次的な入力の組み合わせを記述する． |
| 属性 | なし．                              |

|    |                                |
|----|--------------------------------|
| 要素 | <par_input>                    |
| 目的 | 複数の<input>を囲み，並行入力の組み合わせを記述する． |
| 属性 | なし．                            |

|    |                                  |
|----|----------------------------------|
| 要素 | <alt_input>                      |
| 目的 | 複数の<input>を囲み，択一的な入力の組み合わせを記述する． |
| 属性 | なし．                              |

## 13. システムの動作，ユーザへの出力 - action

<action>内には，ユーザの動作およびユーザへの出力を記述する．<action>では，変数宣言（var），演算処理（assign），条件処理（if，switch），繰り返し処理（while），対話の遷移（call，return，goto），値の送受信（submit），セッションの終了（exit），プロンプ

トの再実行 ( reprompt ), ユーザ・システムへの出力 ( output , seq\_output , par\_output ), 値の読み取り ( get\_element , get\_value , get\_attribute ), 値の書き出し ( set\_element , set\_value , set\_attribute ), 要素の削除 ( delete\_element ) を行なうことができる .

次章以降では<action>内部の記述について , 詳細を示す .

|    |                         |
|----|-------------------------|
| 要素 | <action>                |
| 目的 | システムの動作 , ユーザへの出力を行なう . |
| 属性 | なし .                    |

#### 14. 変数宣言 - var

<var>の詳細は8.1節を参照されたい .

#### 15. 演算処理 - assign

<assign>は , 値を変数に割り当てるために使用する . 属性 name には新しい値を格納する変数名を記述し , 属性 expr には代入する新しい値を記述する . 属性 name , 属性 expr は共に必須属性である . 属性 name で指定された変数名が , 事前に宣言されていない場合 , <assign>により宣言されたとみなす . その場合の変数のスコープはローカルである .

```
<assign name= " price " expr= " price + 1000 " />
```

```
<assign name= " capital " expr= " ' Tokyo ' " />
```

|    |              |                     |
|----|--------------|---------------------|
| 要素 | <assign>     |                     |
| 目的 | 値を変数に割り当てる . |                     |
| 属性 | name         | 新しい値を格納する変数名を記述する . |
|    | expr         | 変数の新しい値を記述する .      |

#### 16. 条件処理

##### 16.1. if , elseif , then , else

<if>は , 条件処理を行なうために用いる . <if>は子要素として<then>を持たなければならない . <if>の必須属性 cond で示す論理式が真の場合 , <then>内の処理が行なわれる . また , 論理式が偽の場合は , <else>内の処理が行なわれる . <then> , <else>で指定できる要素は<action>内で記述できる要素と同等である . また , <if>はオプションとして子要素に<elseif>を持つことができる . なお論理式については9節を参照されたい .

```

<if cond= " product == ' hamburger ' " >
  <then>
    <assign name= " price " expr= " 180 " />
  </then>
  <elseif cond= " product == ' cheeseburger ' " >
    <then>
      <assign name= " price " expr= " 220 " />
    </then>
    <elseif cond= " product == ' orange_juce ' " />
      <then>
        <assign name= " price " expr= " 120 " />
      </then>
    <else>
      <assign name= " price " expr= " 100 " />
    </else>
  </elseif>
</if>

```

|    |               |           |
|----|---------------|-----------|
| 要素 | <if>,<elseif> |           |
| 目的 | 条件処理を行なう．     |           |
| 属性 | cond          | 論理式を記述する． |

|    |                                                      |  |
|----|------------------------------------------------------|--|
| 要素 | <then>                                               |  |
| 目的 | 親要素である<if>，または<elseif>の属性 cond で指定された論理式が，真の場合実行される． |  |
| 属性 | なし．                                                  |  |

|    |                                                      |  |
|----|------------------------------------------------------|--|
| 要素 | <else>                                               |  |
| 目的 | 親要素である<if>，または<elseif>の属性 cond で指定された論理式が，偽の場合実行される． |  |
| 属性 | なし．                                                  |  |

## 16.2. switch

<switch>は，多分岐判断処理を行なうために用いる．<switch>は内容として，複数の<case>タグと，一つの<other>タグを持つことができる．また，必須属性 expr に分岐する条件となる計算式を指定する．属性 expr に記述した計算式の値が，<case>の必須属性 value の変数の値と等しい場合，<case>内の処理を実行する．いずれの<case>の属性 value の値とも一致しない場合は，<other>内の処理を実行する．<case>，<other>内で記述できる要素は，<action>内で記述できる要素と同等である．

```

<switch expr= " product " >
  <case value= " ' hamburger ' " >

```

```

    <assign name= " price " expr= " 180 " />
  </case>
  <case value= " ' orange_juce ' " >
    <assign name= " price " expr= " 120 " />
  </case>
    .
    .

  <other>
    <assign name= " price " expr= " 100 " />
  </other>
</switch>

```

|    |            |                   |
|----|------------|-------------------|
| 要素 | <switch>   |                   |
| 目的 | 多分岐処理を行なう． |                   |
| 属性 | expr       | 分岐の条件となる計算式を記述する． |

|    |                                                      |                                |
|----|------------------------------------------------------|--------------------------------|
| 要素 | <case>                                               |                                |
| 目的 | 属性 value の値と，<switch>の属性 expr の値とが一致した場合に行なう処理を記述する． |                                |
| 属性 | value                                                | <switch>の属性 expr と一致させる値を記述する． |

|    |                               |  |
|----|-------------------------------|--|
| 要素 | <other >                      |  |
| 目的 | 全ての<case>が実行されなかった場合の処理を記述する． |  |
| 属性 | なし．                           |  |

## 17. 繰り返し処理 - while

<while>は，繰り返し処理を行なうために用いる．必須属性 cond に論理式を記述し，論理式が真である間，<while>内に記述した処理を繰り返す．<while>内で記述できる要素は，<action>内で記述できる要素と同等である．次の例は 1 から 100 までの和算を行なうものである．

```

<var name= " number " expr = " 100 " />
<var name= " sum " expr = " 0 " />
<while cond= " number != 0 " >
  <assign name= " sum " expr= " sum + number " />
  <assign name= " number " expr= " number - 1 " />
</while>

```

|    |                                            |           |
|----|--------------------------------------------|-----------|
| 要素 | <while >                                   |           |
| 目的 | 属性 cond に記述される論理式の値が真である間，<while>内の処理を行なう． |           |
| 属性 | cond                                       | 論理式を記述する． |

## 18. 対話の遷移

対話を遷移させるためには、<call>または<goto>を用いる。<call>、<goto>は、以下のよう  
に用いられる。

1. 同ドキュメント内の任意のダイアログへの遷移
2. 他のドキュメント内の任意のダイアログへの遷移

同ドキュメント内のダイアログに遷移する際には、必須属性 next に URI または URI  
フラグメントを指定する。他のドキュメントへ遷移する際には、属性 next に URI を指定す  
る。この場合、フラグメントにダイアログ名を指定した URI を指定する。もしフラグメン  
トが指定されなければ、そのドキュメントにおける最初の対話が遷移先になる。

<call>と<goto>の違いは、<call>が遷移した先のダイアログから戻ってくることを想定し  
ているのに対し、<goto>が想定していない点である、そのため、<call>は現在の対話のデー  
タを退避させるが、<goto>は現在の対話で利用中の一部のデータを破棄する。

以下に<call>、<return>と<goto>の詳細を示す。

### 18.1. call

<call>は指定したダイアログへ遷移するために使用する。属性 namelist には遷移先に渡  
す引数を列挙し、属性 return には遷移先から受け取る戻り値を列挙する。遷移先のダイア  
ログで<return>を発見した場合、元の対話へ戻る。<return>を発見することなくドキュメ  
ントの実行が終了した場合、実行時エラーとなる。また、<call>の属性 namelist に記述さ  
れた変数の個数と、遷移先の<dialog>の属性 namelist に記述された変数の個数が異なる場  
合は、実行時エラーとなる。さらに属性 return に記述された変数の個数と、<return>の属  
性 namelist に記述された変数の個数が一致しない場合にも実行時エラーとなる。

以下に示す記述は、オンラインショッピングシステムにおいて、商品購入ボタンが押さ  
れた場合に、<call>により購入金額の合計と税金の額を計算するダイアログに遷移し、合計  
金額と税金の額を受け取る例を示したものである。

```
<?xml version=" 1.0 " encoding=" Shift_JIS " standalone=" no " ?>
<!DOCTYPE xisl SYSTEM " ./xisl01.dtd ">
<xisl ver=" 1.0 " application=" ./root.xisl ">
  <head>
    ...
  </head>
  <body>
    <dialog id=" Order_Form " comb=" alt " repeat=" * " namelist=" goods,flag ">
      ...
      <exchange>
        <operation target=" ./order.xml " comb=" alt ">
          <input match=" /order_form/button[1] " type=" touch " event=" click " />
```

```

        <input type= " speech " match= " " event= " ./grammar.txt#order " />
    </operation>
    <action>
        <call    next = " #Calc_Sum "
                namelist = " number, tax_rate " return = " total, tax " />
    </action>
</exchange>
...
</dialog>
<dialog id= " Calc_Sum " repeat= " 1 " namelist= " number, tax_rate " >
    <begin>
        <assign var= " sub_total " expr= " goods_price * number " />
        <assign var= " tax " expr= " sub_total * tax_rate " />
        <assign var= " total " expr= " subtotal + tax " />
        <return namelist= " total , tax " />
    </begin>
</dialog>
</body>
</xisl>

```

|    |               |                                |
|----|---------------|--------------------------------|
| 要素 | <call>        |                                |
| 目的 | 他のダイアログへ遷移する． |                                |
| 属性 | next          | 遷移先を示す URI を指定する．              |
|    | namelist      | 引数を列挙する．複数の引数は“，”で区切る．         |
|    | return        | 戻り値を格納する変数を列挙する．複数の変数は“，”で区切る． |

### 18.2. return

<return>は，<call>で呼び出された対話から元の対話へ戻るために使用する．戻る場所は元の対話の<call>が記述されている場所である．属性 namelist には，呼び出し側の<call>の属性 return に戻す値を記述する．<return>の属性 namelist に記述された変数の個数と，<call>の属性 return に記述された変数の個数が一致しない場合は実行時エラーとなる．

```
<return  namelist = " total, tax " />
```

|    |                 |                                                        |
|----|-----------------|--------------------------------------------------------|
| 要素 | <return>        |                                                        |
| 目的 | 呼び出し元のダイアログへ戻る． |                                                        |
| 属性 | namelist        | 呼び出し元の<call>の属性 return に戻す値を格納した変数を列挙する．複数の変数は“，”で区切る． |

### 18.3. goto

<goto>は指定したダイアログへ遷移するために使用する．ただし<goto>で遷移する場合，元の対話に復帰することはない．属性 namelist には遷移先に渡す引数を列挙する．以下に遷移先とデータ破棄の関係を示す．

### 1. 同一ドキュメント内のダイアログへの遷移

たとえ同一ダイアログへの遷移であっても、現在のダイアログ内のデータ（ダイアログ変数、ローカル変数）は破棄される。セッション変数、ドキュメント変数、アプリケーション変数は保持される。

### 2. 同一アプリケーション内における他のドキュメントへの遷移。

現在のドキュメント内のデータ（ドキュメント変数、各ダイアログのダイアログ変数、各 exchange のローカル変数）は破棄される。セッション変数、アプリケーション変数は保持される。

### 3. 別アプリケーションのドキュメントへの遷移

アプリケーション内のデータ（アプリケーション変数、アプリケーション内の各ドキュメント変数、各ドキュメント内のダイアログ変数、各 exchange のローカル変数）を全て破棄する。セッション変数のみ保持される。

<call>で呼び出された対話内で<goto>を発見した場合においても、同様に上記に示した動作をとる。この場合、<call>で呼ばれたという情報も破棄され、呼び出し側の対話へ戻る事は無くなる。もし、<goto>で遷移した先のダイアログで<return>を発見した場合、<call>で呼ばれたという情報が破棄されているため、実行時エラーとなる。なお、<goto>の属性 namelist に記述された変数の個数と、遷移先の<dialog>の属性 namelist に記述された変数の個数が異なる場合も<call>と同様に、実行時エラーとなる。

```
<goto next = " Help.xisl " namelist = " keyword " />
```

|    |                              |                        |
|----|------------------------------|------------------------|
| 要素 | <goto>                       |                        |
| 目的 | 他のダイアログへ遷移する。ただし元の対話へ復帰できない。 |                        |
| 属性 | next                         | 遷移先を示す URI を指定する。      |
|    | namelist                     | 引数を列挙する。複数の引数は“,”で区切る。 |

## 19. 値の送受信 - submit

<submit>は、変数の値を HTTP GET または POST リクエスト経由でドキュメントサーバに送信し、戻り値を受信する。属性 next は必須属性であり、値の送信先となる URI を記述する。属性 method にはリクエストのメソッド“get”もしくは“post”を指定する。属性 namelist には、サーバに引き渡す変数を列挙し、サーバからの戻り値を格納する変数を属性 return で指定する。

```
<submit next= " /cgi-bin/weather.cgi " method= " post " namelist= " city "
return= " result " />
```

|    |          |
|----|----------|
| 要素 | <submit> |
|----|----------|

|    |                                                         |                             |
|----|---------------------------------------------------------|-----------------------------|
| 目的 | 変数の値を HTTP GET または POST リクエスト経由でドキュメントサーバに送信し，戻り値を受信する． |                             |
| 属性 | next                                                    | 値の送信先を示す URI を記述する．         |
|    | method                                                  | リクエストのメソッドを記述：get または post． |
|    | namelist                                                | 送信する変数を列挙する．                |
|    | return                                                  | サーバからの戻り値を格納する変数を記述する．      |

## 20. セッションの終了 - exit

<exit>は，実行中のダイアログ，ドキュメント，アプリケーションを終了するために使用する．

```
<exit/>
```

|    |                                 |
|----|---------------------------------|
| 要素 | <exit/>                         |
| 目的 | 実行中のダイアログ，ドキュメント，アプリケーションを終了する． |
| 属性 | なし．                             |

## 21. プロンプトの再実行 - reprompt

<reprompt>の詳細は11節を参照されたい．

## 22. 変数の参照 - value

<value>は，<output>内に変数の値を埋め込むために使用する．参照する変数は，必須属性 expr に記述する．属性 expr に記述した変数に格納されている値は，<value>と置き換えられてフロントエンドに送信される．

下記の例で，変数 title に XML and Java が格納されている場合，フロントエンドには “ Please speak the author of XML and Java ” という内容が送信され，それに応じた出力がフロントエンドで実行される．

```
<exchange>
  <prompt>
    <output type= " speech " event= " tts-speech " >
      <![CDATA[
        <param name= " speech-text " >
          Please speak the author of <value expr= " title " /> .
        </param>
      ]]>
    </output>
```

```

</prompt>
<operation>
...
</operation>
<action>
...
</action>
</exchange>

```

|    |                      |              |
|----|----------------------|--------------|
| 要素 | <value>              |              |
| 目的 | <output>内に変数の値を埋め込む． |              |
| 属性 | expr                 | 参照する変数を記述する． |

### 23. ユーザ,システムへの出力 - output

<output>は，ユーザへの1出力を記述するために使用する．属性 type に出力モダリティを指定し，属性 event に出力イベントを指定する．属性 type，属性 event で指定する内容は，フロントエンドに依存した部分であり，XISL の仕様では規定しない．

<output>に記述する内容は CDATA セクションとする．出力する内容を全て CDATA セクションとすることで，システムに依存した独自の要素を記述することができる．なお，<output>内の CDATA セクションには，<value>を記述することができる．

以下に，ブラウザにコンテンツを表示し，擬人化エージェントに音声出力をさせる場合を想定して，記述例を示す．

```

<exchange>
  <operation>
    ...
  </operation>
  <action>
    <output type= " mmi-browser " event= " open " >
      <![CDATA[
        <param name= " window-name " > Top_Page </param>
        <param name= " uri " > ./top.xml </param>
        <param name= " position " > center </param>
      ]]>
    </output>
    <output type= " speech " event= " agent-speech " >
      <![CDATA[
        <param name= " agent-name " > Wizard </param>
        <param name= " speech-text " > May I help you? </param>
      ]]>
    </output>
  </action>

```

</exchange>

|    |                |               |
|----|----------------|---------------|
| 要素 | <output>       |               |
| 目的 | ユーザへの1出力を記述する． |               |
| 属性 | type           | 出力モダリティを記述する． |
|    | event          | 出力イベントを記述する．  |

<output>は、原則として記述した順に実行される。出力を制御する場合は、<par\_output>、<seq\_output>を使用する。図 9に示すように、<output>を<par\_output>で囲むと同時出力となる。また<par\_output>内で、局所的に逐次出力を行ないたい場合は、<output>を<seq\_output>で囲む。

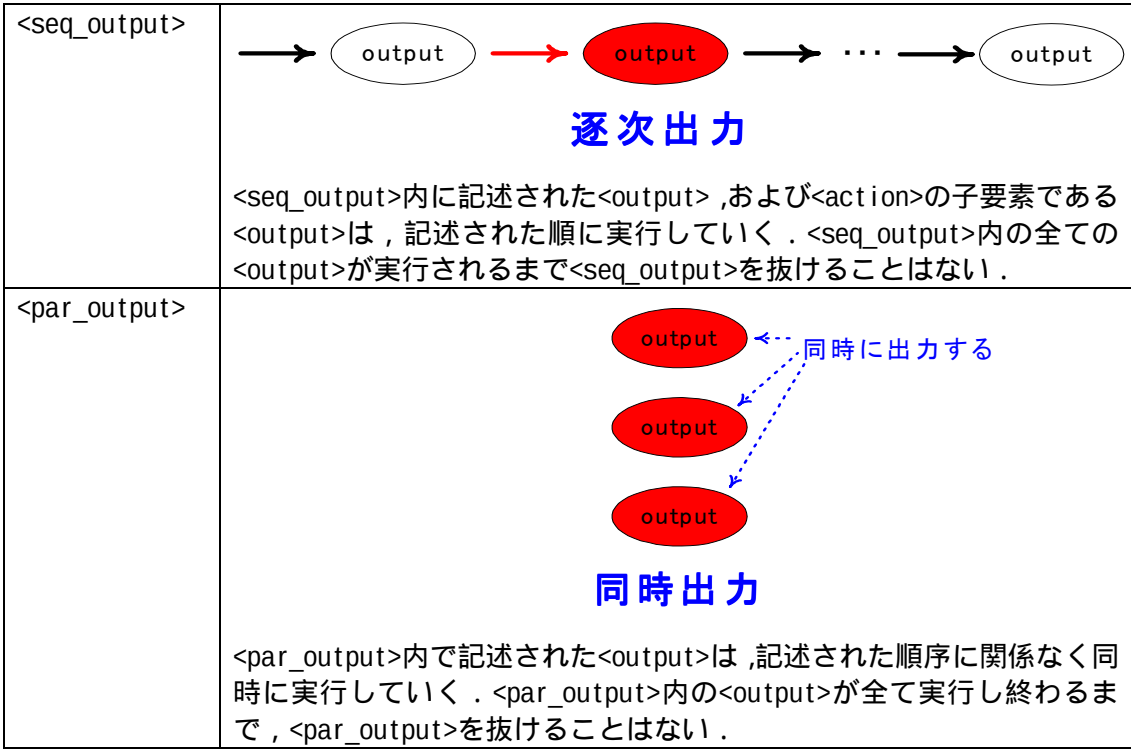


図 9: 逐次出力, 同時出力の記述

以下に、<para\_output>、<seq\_output>を<action>内に記述した例を示す。また図 10に実行の概念図を示す。以下の例では、映像と音楽と擬人化エージェント出力を同時に行なう。ただし、擬人化エージェントの動作は逐次的に実行される。

```
<exchange>
  <operation>
    ...
  </operation>
</action>
```

```

<par_output>
  <output type= " sound " event= " play " >
    <![CDATA[
      <param name= " file_name " > BGMusic.mp3 </param>
    ]]>
  </output>
  <output type= " movie " event= " play " >
    <![CDATA[
      <param name= " file_name " > BGMovie.mpg </param>
    ]]>
  </output>

  <seq_output>
    <!-- Create Agent -->
    <output type= " agent " event= " create " >
      <![CDATA[
        <param name= " agent-name " > Wizard </param>
        <param name= " agent-file " > Merlin.acs </param>
      ]]>
    </output>
    <output type= " speech " event= " agent-speech " >
      <![CDATA[
        <param name= " agent-name " > Wizard </param>
        <param name= " speech-text " > Please wait a moment. </param>
      ]]>
    </output>
  </seq_output>
</par_output>
</action>
</exchange>

```

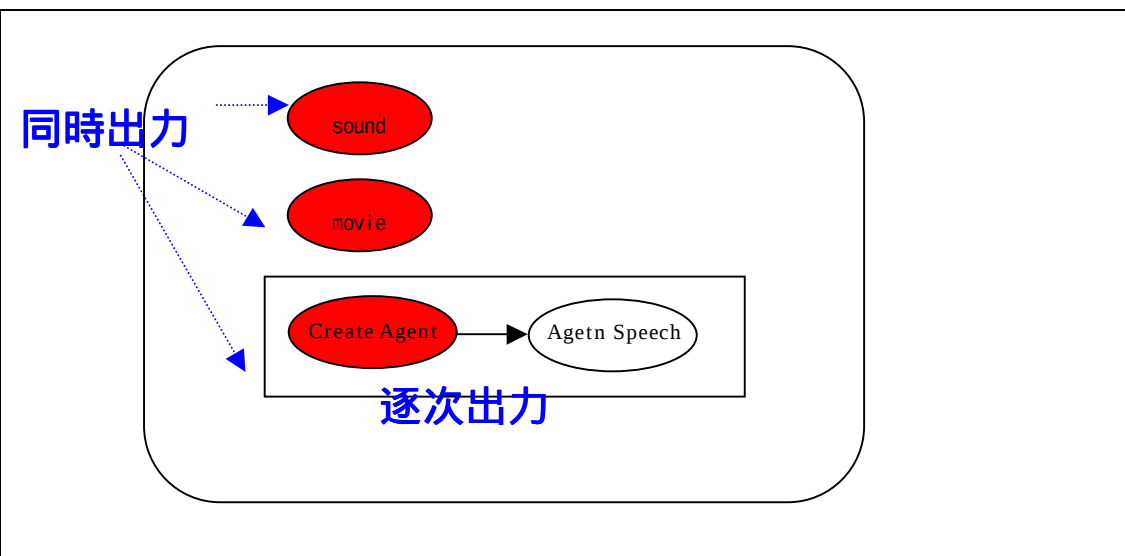


図 10: 出力実行の概念図

|    |                            |
|----|----------------------------|
| 要素 | <seq_output>               |
| 目的 | 複数の<output>を囲み，逐次的な出力を行なう． |
| 属性 | なし．                        |

|    |                          |
|----|--------------------------|
| 要素 | <par_output>             |
| 目的 | 複数の<output>を囲み，同時出力を行なう． |
| 属性 | なし．                      |

#### 24. 値の読み取り

<get\_element>，<get\_value>，<get\_attribute>は，指定した XML ドキュメントまたは変数から XML 要素，XML 要素の内容，属性値を取得するために用いられる．<get\_element>，<get\_value>，<get\_attribute>で指定する属性は target，var，match，place であり，このうち必須属性は target，var，match である．

属性 target には，対象となる XML ドキュメントを示す URI または，変数名を指定し，属性 var には，取得した値を格納する変数名を指定する．

属性 match には，取得したい XML 要素，XML 要素の内容，属性値を XPath の書式に従って指定する．なお，指定したドキュメントおよび変数，指定した XML 要素，XML 要素の内容，属性名が存在しない場合は，実行時エラーとなる．

属性 place では，読み取りの対象となる XML ドキュメントの場所を指定する．指定できる場所は，クライアント（temp）DB 内のコンテンツ，配信サーバ（master）内のコンテンツの何れかである．属性 place を明示的に指定しない場合は，temp とみなす．

図 11および図 12に，<get\_element>，<get\_value>，<get\_attribute>の説明の際に用いる，読み取りの対象となる XML ドキュメント，および XML 要素が格納されている変数を示す．

```
<?xml version= " 1.0 " encoding= " Shift_JIS " ?>
<goods_list>
  <category type= " food " >
    <goods id = " a001 " >
      <name> hamburger </name>
      <price value= " 180 " >¥180</price>
      <stock>200</stock>
    </goods>
    <goods id = " a002 " >
      <name> cheeseburger </name>
      <price value= " 220 " >¥220</price>
```

```

        <stock>150</stock>
    </goods>
    ...
</category>
<category type= " drink " >
    <goods id = " b001 " >
        <name>orange_juce</name>
        <price value= " 120 " >¥120</price>
        <stock>200</stock>
    </goods>
    ...
</category>
...
</goods_list>

```

図 11: XML ドキュメント—goods\_list.xml

```

<category type= " drink " >
    <goods id = " b001 " >
        <name>orange_juce</name>
        <price value= " 120 " >¥120</price>
        <stock>200</stock>
    </goods>
    ...
</category>

```

図 12: 変数—drink

#### 24.1. get\_element - XML 要素の読み取り

<get\_element>は、指定した XML ドキュメント、もしくは変数に格納されている XML 要素から、任意の XML 要素を取得するために使用する。

図 11に示した XML ドキュメント ( goods\_list.xml ) から XML 要素/goods\_list/category [@type='food']を取得するための記述例を以下に示す。

```

<get_element target = " goods_list.xml " var = " category "
             match = " /goods_list/category[@type= ' food ' ] " place = " master " >

```

上記の<get\_element>により、変数 category には以下の XML 要素が格納される。

```

<category type= " food " >
    <goods id = " a001 " >
        <name> hamburger </name>
        <price value= " 180 " >¥180</price>

```

```

    <stock>200</stock>
  </goods>
  <goods id = " a002 " >
    <name> cheeseburger </name>
    <price value= " 220 " >¥220</price>
    <stock>150</stock>
  </goods>
  ...
</category>

```

また、図 12に示した変数 drink から、XML 要素/cagtegrory/goods[@id='b001']を取得するための記述例を以下に示す。

```

<get_element target = " drink " var = " goods "
             match = " /category/goods[@id= ' b001 ' ] " place = " temp " >

```

上記の<get\_element>により、変数 goods には以下の要素が格納される。

```

<goods id = " b001 " >
  <name>orange_juce</name>
  <price value= " 120 " >¥120</price>
  <stock>200</stock>
</goods>

```

|    |                                                         |                                        |
|----|---------------------------------------------------------|----------------------------------------|
| 要素 | <get_element>                                           |                                        |
| 目的 | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素を取得する． |                                        |
| 属性 | target                                                  | XML ドキュメント名，または XML 要素を格納している変数名を指定する． |
|    | var                                                     | 取得した内容を格納する変数名を記述する．                   |
|    | match                                                   | 取得したい XML 要素名を指定する（Xpath で指定）．         |
|    | place                                                   | temp または master を指定する（デフォルトは temp ）．   |

#### 24.2. get\_value - XML 要素の内容の読み取り

<get\_value>は、指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素の内容を取得するために使用する．ただし、XML 要素の直接の子供となる内容のテキストデータのみを取得する．

図 11に示した XML ドキュメント ( goods\_list.xml ) から XML 要素/goods\_list/category [@id='food']/goods[@id='a001']/name の内容を取得するための記述例を以下に示す．

```

<get_value target = " goods_list.xml " var = " name "
           match= " /goods_list/category[@type= ' food ' ]/goods[@id= ' a001 ' ]/name "

```

```
place = " master " >
```

上記の<get\_value>により，変数 name には以下の内容が格納される．

```
hamburger
```

また，図 12に示した変数 drink から，XML 要素/category/goods[@id='b001']/stock の内容  
を取得するための記述例を以下に示す．

```
<get_value target = " drink " var = " stock "  
match = " /category/goods[@id= ' b001 ' ]/stock " place = " temp " >
```

上記の<get\_value>により，変数 stock には以下の内容が格納される．

```
200
```

| 要素 | <get_value>                                                                                     |                                               |
|----|-------------------------------------------------------------------------------------------------|-----------------------------------------------|
| 目的 | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素の内容を取得する．ただし，XML 要素の直接の子供となる内容のテキストデータのみを取得する． |                                               |
| 属性 | target                                                                                          | XML ドキュメント名，または XML 要素を格納している変数名を指定する．        |
|    | var                                                                                             | 取得した内容を格納する変数名を記述する．                          |
|    | match                                                                                           | 取得したいテキストデータを子供として持つ XML 要素名を指定する（Xpath で指定）． |
|    | place                                                                                           | temp または master を指定する（デフォルトは temp）．           |

#### 24.3. get\_attribute - 属性値の読み取り

<get\_attribute>は，指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素の属性値を取得するために使用する．

図 11 に示した XML ドキュメント(goods\_list.xml)から属性/goods\_list/category  
[@id='food']/goods[@id= ' a001 ' ]/price/@value を取得するための記述例を以下に示す．

```
<get_value target = " goods_list.xml " var = " price "  
match=" /goods_list/category[@type= ' food ' ]/goods[@id= ' a001 ' ]/price/@value "  
place = " master " >
```

上記の<get\_attribute >により，変数 price には以下の属性値が格納される．

```
180
```

また、図 12に示した変数 drink から、属性 /category/@type を取得するための記述例を以下に示す。

```
<get_element target = " drink " var = " type "
              match = " /category/@type " place = " temp " >
```

上記の<get\_element>により、変数 type には以下の要素が格納される。

|      |
|------|
| food |
|------|

|    |                                                             |                                        |
|----|-------------------------------------------------------------|----------------------------------------|
| 要素 | <get_attribute>                                             |                                        |
| 目的 | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素から任意の XML 要素の属性値を取得する． |                                        |
| 属性 | target                                                      | XML ドキュメント名，または XML 要素を格納している変数名を指定する． |
|    | var                                                         | 取得した属性値を格納する変数名を記述する．                  |
|    | match                                                       | 取得したい属性値を持つ属性名を指定する（Xpath で指定）．        |
|    | place                                                       | temp または master を指定する（デフォルトは temp）．    |

25. 値の書き出し

<set\_element>，<set\_value>，<set\_attribute>は，指定した XML ドキュメント，もしくは変数に格納されている XML 要素，XML 要素の内容，属性値を更新するために用いられる．<set\_element>，<set\_value>，<set\_attribute>で指定する共通の属性は target ,var ,match ,place であり，<set\_element>のみ属性 type を指定する．必須属性は target ,var ,match ，および<set\_element>の属性 type である．

属性 target には，対象となる XML ドキュメント名または，変数名を指定する．

属性 var には，更新する値または変数を記述する．

属性 match には，更新の対象となる XML 要素，XML 要素の内容，属性値のパスを XPath の書式に従って指定する．なお，指定したドキュメントおよび変数，指定した XML 要素，XML 要素の内容，属性名が存在しない場合は，実行時エラーとなる．

属性 place では，更新される対象を有している XML ドキュメントの場所を指定する．指定できる場所は，クライアント（temp）DB 内のコンテンツ，配信サーバ（master）内のコンテンツの何れかである．属性 place を明示的に指定しない場合は，temp とみなす．

なお，指定したドキュメントおよび変数，指定した XML 要素，XML 要素の内容，属性名が存在しない場合は，実行時エラーとなる．

### 25.1. set\_element - XML 要素の書き出し

<set\_element>は、指定した XML ドキュメント、もしくは変数に格納されている XML 要素に含まれる任意の XML 要素を更新するために使用する。

図 11に示した XML ドキュメント ( goods\_list.xml ) の XML 要素/goods\_list/category [@type='food']/goods[@id='a001']を更新するための記述例を以下に示す。

```
<set_element target = " goods_list.xml " var = " goods "
            match = " /goods_list/category[@type= ' food ' ]/goods[@id= ' a001 ' ] "
            type= " oversight " place = " master " >
```

上書きする値が格納されている変数 goods は図 13に示す通りである。

```
<goods id = " a001 " >
  <name>hamburger</name>
  <price value= " 200 " >¥200</price>
  <stock>150</stock>
</goods>
```

図 13: 変数—goods

上記の<set\_element>により、図 11に示した goods\_list.xml は以下のように書き換えられる。

```
<?xml version= " 1.0 " encoding= " Shift_JIS " ?>
<goods_list>
  <category type= " food " >
    <goods id = " a001 " >
      <name> hamburger </name>
      <price value= " 200 " >¥200</price>
      <stock>150</stock>
    </goods>
    <goods id = " a002 " >
      <name> cheeseburger </name>
      <price value= " 220 " >¥220</price>
      <stock>150</stock>
    </goods>
    ...
  </category>
  ...
</goods_list>
```

また、図 12に示した変数 drink の XML 要素/cagteory に子要素を挿入するための記述例を以下に示す。

```
<get_element target = "drink" var = "cola" match = "/category"
              type= "insert" place = "temp">
```

挿入する値が格納されている変数 cola は図 14に示す通りである .

```
<goods id = " b008 " >
  <name>cola</name>
  <price value= " 130 " >¥130</price>
  <stock>250</stock>
</goods>
```

図 14: 変数—cola

上記の<set\_element>により ,図 12に示した変数 drink は以下のように書き換えられる .

```
<category type= " drink " >
  <goods id = " b001 " >
    <name>orange_juce</name>
    <price value= " 120 " >¥120</price>
    <stock>200</stock>
  </goods>
  ...
  <goods id = " b008 " >
    <name>cola</name>
    <price value= " 130 " >¥130</price>
    <stock>250</stock>
  </goods>
</category>
```

|    |                                                               |                                           |
|----|---------------------------------------------------------------|-------------------------------------------|
| 要素 | <set_element>                                                 |                                           |
| 目的 | 指定した XML ドキュメント , もしくは変数に格納されている XML 要素に含まれる任意の XML 要素を更新する . |                                           |
| 属性 | target                                                        | XML ドキュメント名 , または XML 要素を格納している変数名を指定する . |
|    | var                                                           | 更新する要素を持つ変数名を記述する .                       |
|    | type                                                          | 上書き (overset) または挿入 (insert) を指定する .      |
|    | match                                                         | 更新の対象となる要素名を指定する (Xpath で指定) .            |
|    | place                                                         | temp または master を指定する (デフォルトは temp) .     |

## 25.2. set\_value - XML 要素の値の書き出し

<set\_value>は , 指定した XML ドキュメント , もしくは変数に格納されている XML 要素に含まれる任意の XML 要素の内容を更新するために使用する .

図 11に示した XML ドキュメント ( goods\_list.xml ) の XML 要素/goods\_list/category

[@type='food']/goods[@id='a001']/stock の内容を更新するための記述例を以下に示す .

```
<set_value target = " goods_list.xml " var = " 198 "
           match= " /goods_list/category[@type= ' food ' ]/goods[@id= ' a001 ' ]/stock "
           place = " master " >
```

上記の<set\_value>により , 図 11に示した goods\_list.xml は以下のように書き換えられる .

```
<?xml version= " 1.0 " encoding= " Shift_JIS " ?>
<goods_list>
  <category type= " food " >
    <goods id = " a001 " >
      <name> hamburger </name>
      <price value= " 180 " >¥180</price>
      <stock>198</stock>
    </goods>
    <goods id = " a002 " >
      <name> cheeseburger </name>
      <price value= " 220 " >¥220</price>
      <stock>150</stock>
    </goods>
    ...
  </category>
  ...
</goods_list>
```

また , 図 12に示した変数 drink の XML 要素/cagtegrory/goods[@id='b001']/price の内容を更新するための記述例を以下に示す .

```
<set_value target = " drink " var = " ¥130 "
           match = " /cagtegrory/goods[@id= ' b001 ' ]/price "
           place = " temp " >
```

上記の<set\_value>により , 図 12に示した変数 drink は以下のように書き換えられる .

```
<category type= " drink " >
  <goods id = " b001 " >
    <name>orange_juce</name>
    <price value= " 120 " >¥130</price>
    <stock>200</stock>
  </goods>
  ...
```

```
</category>
```

|    |                                                               |                                        |
|----|---------------------------------------------------------------|----------------------------------------|
| 要素 | <set_value>                                                   |                                        |
| 目的 | 指定した XML ドキュメント，もしくは変数に格納されている XML 要素に含まれる任意の XML 要素の内容を更新する． |                                        |
| 属性 | target                                                        | XML ドキュメント名，または XML 要素を格納している変数名を指定する． |
|    | var                                                           | 上書きする値または変数名を記述する．                     |
|    | match                                                         | 更新の対象となる XML 要素名を指定する（Xpath で指定）．      |
|    | place                                                         | temp または master を指定する（デフォルトは temp）．    |

### 25.3. set\_attribute - 属性値の書き出し

<set\_attribute>は，指定した XML ドキュメント，もしくは変数に格納されている XML 要素に含まれる任意の XML 要素の属性値を更新するために使用する．

図 11に示した XML ドキュメント（goods\_list.xml）の属性値/goods\_list/category[@type='food']/goods[@id='a001']/@id を更新するための記述例を以下に示す．

```
<set_attribute target = " goods_list.xml " var = " a0001 "
               match = " /goods_list/category[@type= ' food ' ]/goods[@id= ' a001 ' ]/@id "
               place = " master ">
```

上記の<set\_attribute>により，図 11に示した goods\_list.xml は以下のように書き換えられる．

```
<?xml version= " 1.0 " encoding= " Shift_JIS " ?>
<goods_list>
  <category type= " food ">
    <goods id = " a0001 ">
      <name> hamburger </name>
      <price value= " 180 ">¥180</price>
      <stock>200</stock>
    </goods>
    <goods id = " a002 ">
      <name> cheeseburger </name>
      <price value= " 220 ">¥220</price>
      <stock>150</stock>
    </goods>
    ...
  </category>
  ...
</goods_list>
```

また，図 12に示した変数 drink の，属性値/cagtegrory/goods[@id='b001']/price/@value

を更新するための記述例を以下に示す．

```
<set_attribute target = " drink " var = " 130 "
               match = " /cagtegrory/goods[@id= ' b001 ' ]/price/@value "
               place = " temp " >
```

上記の<set\_attribute>により ,図 12に示した変数 drink は以下のように書き換えられる．

```
<category type= " drink " >
  <goods id = " b001 " >
    <name>orange_juce</name>
    <price value= " 130 " >¥120</price>
    <stock>200</stock>
  </goods>
  ...
</category>
```

|    |                                                                   |                                           |
|----|-------------------------------------------------------------------|-------------------------------------------|
| 要素 | <set_attribute>                                                   |                                           |
| 目的 | 指定した XML ドキュメント , もしくは変数に格納されている XML 要素に含まれる任意の XML 要素の属性値を更新する . |                                           |
| 属性 | target                                                            | XML ドキュメント名 , または XML 要素を格納している変数名を指定する . |
|    | var                                                               | 上書きする値または変数名を記述する .                       |
|    | match                                                             | 更新の対象となる XML 要素の属性を指定する ( Xpath で指定 ) .   |
|    | place                                                             | temp または master を指定する ( デフォルトは temp ) .   |

## 26. 要素の削除 - delete\_element

<delete\_element>は , 指定した XML ドキュメント , もしくは変数に格納されている XML 要素に含まれる任意の XML 要素を削除するために使用する .

図 11に示した XML ドキュメント ( goods\_list.xml ) の XML 要素/goods\_list/category [@type='food']/goods[@id='a001']/を削除するための記述例を以下に示す .

```
<delete_element target = " goods_list.xml "
                 match= " /goods_list/category[@type= ' food ' ]/goods[@id= ' a001 ' ] "
                 place = " master " >
```

上記の<delete\_element>により ,図 11に示した goods\_list.xml は以下のように書き換えられる .

```
<?xml version= " 1.0 " encoding= " Shift_JIS " ?>
```

```

<goods_list>
  <category type= " food " >
    <goods id = " a002 " >
      <name> cheeseburger </name>
      <price value= " 220 " >¥220</price>
      <stock>150</stock>
    </goods>
    ...
  </category>
  ...
</goods_list>

```

また、図 12に示した変数 drink の XML 要素/cagtegrory/goods[@id='b001']/stock を削除するための記述例を以下に示す。

```

<delete_element  target  = " drink "
                  match   = " /cagtegrory/goods[@id= ' b001 ' ]/stock "
                  place   = " temp " >

```

上記の<delete\_element>により、図 12に示した変数 drink は以下のように書き換えられる。

```

<category type= " drink " >
  <goods id = " b001 " >
    <name>orange_juce</name>
    <price value= " 120 " >¥120</price>
  </goods>
  ...
</category>

```

|    |                                                             |                                         |
|----|-------------------------------------------------------------|-----------------------------------------|
| 要素 | <delete_element>                                            |                                         |
| 目的 | 指定した XML ドキュメント ,もしくは変数に格納されている XML 要素に含まれる任意の XML 要素を削除する。 |                                         |
| 属性 | target                                                      | XML ドキュメント名 ,または XML 要素を格納した変数名を指定する。   |
|    | match                                                       | 削除したい XML 要素名を指定する ( Xpath で指定 ) .      |
|    | place                                                       | temp または master を指定する ( デフォルトは temp ) . |

## 付録 1 . XISL の DTD

```
<!--
  XISL 1.0 DTD
  version 2002-01-21
-->

<!ENTITY % exchange.type " par_exchange | seq_exchange | alt_exchange |
exchange ">

<!ENTITY % input.type " input | par_input | seq_input | alt_input ">

<!ENTITY % output.type " output | seq_output | par_output ">

<!ENTITY % begin.end.content " var | assign | if | switch | while |
                                call | return | goto | submit |
                                exit |
                                seq_output | par_output | output |
                                get_value | get_element | get_attribute |
                                set_value | set_element | set_attribute |
                                delete_element ">

<!ENTITY % action.content " %begin.end.content; | reprompt ">

<!ENTITY % boolean "( true | false )" >

<!ENTITY % duration "CDATA" >

<!ENTITY % conditional.logic "CDATA">

<!ENTITY % expression "CDATA" >

<!ENTITY % variable.name "CDATA" >

<!ENTITY % variable.names "CDATA" >

<!ENTITY % integer "CDATA" >

<!ENTITY % target.name "CDATA" >

<!ENTITY % uri "CDATA" >

<!ENTITY % path "CDATA" >

<!ENTITY % get.set.attrs " target %target.name; #REQUIRED
```

```

                                var      %variable.name;   #REQUIRED
                                match     %path;             #REQUIRED
                                place     (temp | master)    'temp' ">

<!--===== Root =====>

<!ELEMENT xisl (head?,body) >
<!ATTLIST xisl  version          CDATA  #REQUIRED
               application       %uri;  #IMPLIED>

<!ELEMENT head    (name?,author?,date?,details?,history?)>

<!ELEMENT name    (#PCDATA)>

<!ELEMENT author  (#PCDATA)>

<!ELEMENT date    (#PCDATA)>

<!ELEMENT history (item+)>

<!ELEMENT item    (date,details)>

<!ELEMENT details (#PCDATA)>

<!ELEMENT body    (document_var?,dialog*)>

<!ELEMENT document_var (var+)>

<!--===== Dialog =====>

<!ELEMENT dialog  ( dialog_var?, begin?, (%exchange.type;)* , end?) >

<!ELEMENT dialog_var (var+) >
<!ATTLIST dialog  id          ID      #REQUIRED
               comb          (par | seq | alt)    "seq"
               repeat        CDATA          "1"
               scope         (dialog | document) "dialog"
               arg           %variable.names;    #IMPLIED>

<!--===== Begin =====>

<!ELEMENT begin   (%begin.end.content;)*>

<!--===== End =====>

```

```

<!ELEMENT end      (%begin.end.content;)*>

<!--===== Exchanges =====>

<!ELEMENT par_exchange  (%exchange.type;)+>

<!ELEMENT seq_exchange  (%exchange.type;)+>

<!ELEMENT alt_exchange  (%exchange.type;)+>

<!ELEMENT exchange      (prompt?,operation,action)>

<!--===== Prompt =====>

<!ELEMENT prompt  (%output.type;)+ >
<!--ATTLIST prompt  count      %integer;    "1"
                  timeout    %duration;    #IMPLIED
                  bargein    %boolean;    "true" >

<!--===== Operation =====>

<!ELEMENT operation  (%input.type;)+ >
<!--ATTLIST operation  target %target.name;    #IMPLIED
                  comb ( par | seq | alt )  "par">

<!ELEMENT par_input (%input.type;)+>

<!ELEMENT seq_input (%input.type;)+>

<!ELEMENT alt_input (%input.type;)+>

<!ELEMENT input EMPTY >
<!--ATTLIST input  match      %path;    #REQUIRED
                  type      CDATA    #REQUIRED
                  event      CDATA    #REQUIRED
                  target     %target.name;    #IMPLIED
                  return     %variable.names; #IMPLIED >

<!--===== Action =====>

<!ELEMENT action  (%action.content;)*>

<!ELEMENT var EMPTY>
<!--ATTLIST var  name %variable.name;    #REQUIRED
                  expr %expression;      #IMPLIED>

```

```

<!ELEMENT assign EMPTY>
<!ATTLIST assign  name  %variable.name;  #REQUIRED
                  expr  %expression;      #REQUIRED>

<!ELEMENT if (then, (else? | elseif*))>
<!ATTLIST if cond  %conditional.logic; #REQUIRED>

<!ELEMENT then  (%action.content;)*>

<!ELEMENT else  (%action.content;)*>

<!ELEMENT elseif  ( then, (else? | elseif*) )>
<!ATTLIST elseif  cond %conditional.logic;  #REQUIRED>

<!ELEMENT switch  ( case+, other? )>
<!ATTLIST switch  expr  %expression;  #REQUIRED>

<!ELEMENT case  (%action.content;)*>
<!ATTLIST case  value CDATA #REQUIRED>

<!ELEMENT other (%action.content;)*>

<!ELEMENT while (%action.content;)*>
<!ATTLIST while cond %conditional.logic; #REQUIRED>

<!ELEMENT call EMPTY>
<!ATTLIST call  next      %uri;  #REQUIRED
                  namelist  %variable.names;  #IMPLIED
                  return    %variable.names;  #IMPLIED>

<!ELEMENT return  EMPTY>
<!ATTLIST return  namelist  CDATA  #IMPLIED>

<!ELEMENT goto EMPTY>
<!ATTLIST goto  next      %uri;          #REQUIRED
                  namelist  %variable.names;  #IMPLIED>

<!ELEMENT submit EMPTY>
<!ATTLIST submit next      %uri;  #REQUIRED
                  method    (get | post) "get"
                  namelist  %variable.names;  #IMPLIED
                  return    %variable.names;  #IMPLIED >

<!ELEMENT exit EMPTY>

```

```

<!ELEMENT reprompt EMPTY>

<!ELEMENT seq_output (%output.type;)*>

<!ELEMENT par_output (%output.type;)*>

<!-- The <value> elements appear in the CDATA section of <output> elements.
      <!ELEMENT value  EMPTY>
      <!ATTLIST value  expr  %expression;  #REQUIRED>
-->

<!ELEMENT output  (#PCDATA) >
<!ATTLIST output  type      CDATA  #REQUIRED
              event      CDATA  #REQUIRED>

<!ELEMENT get_value EMPTY>
<!ATTLIST get_value  %get.set.attrs;>

<!ELEMENT get_element EMPTY>
<!ATTLIST get_element %get.set.attrs;>

<!ELEMENT get_attribute  EMPTY>
<!ATTLIST get_attribute  %get.set.attrs;>

<!ELEMENT set_value  EMPTY>
<!ATTLIST set_value  %get.set.attrs;>

<!ELEMENT set_element  EMPTY>
<!ATTLIST set_element  %get.set.attrs;
                    type (insert | overset) #REQUIRED>

<!ELEMENT set_attribute  EMPTY>
<!ATTLIST set_attribute  %get.set.attrs;>

<!ELEMENT delete_element EMPTY>
<!ATTLIST delete_element target  %target.name;  #REQUIRED
                        match    %path;          #REQUIRED
                        place    (temp | master) "temp">

```