言語理解 XML

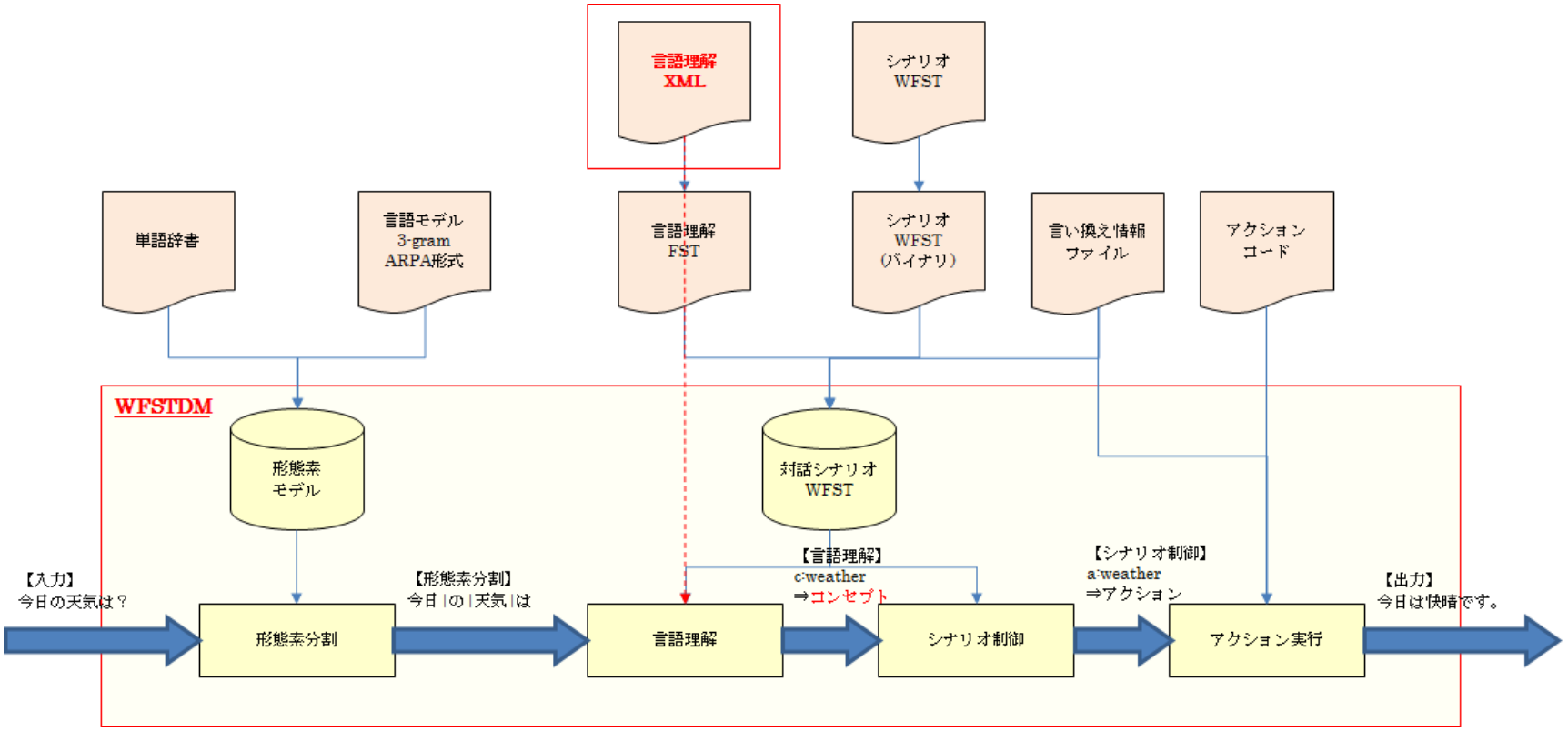
ファイル仕様書

1. はじめに

本書は言語理解 XML のファイル仕様について記載した文書である。

2. 言語理解 XML

言語理解 XML は、WFSTDM への入力を意味理解し、シナリオ WFST(シナリオ制御)の入力となるコンセプトを決定するためのファイルである。



言語理解 XML は 3 つのクラスタグ(word-class, slot, concept)で構成され、各クラスタグはラベル(半角文字列)で一意に識別される。いずれのクラスタグにおいても label 属性は必須であり、label 属性を省略すると FST の生成時にエラーが発生する。 クラスタグで定義されたラベルを他のクラスタグ内で使用する（値を参照する、あるいは値に展開する）ときは、ラベル名を半角のカッコで囲んだシンボル を用いる。例えば、GREETING というラベルの値を参照するときは、“(GREETING)”というシンボルを用いる。 クラスタグ内には複数の表現を記述することができ、それらは改行で区切られる。各行の要素を「表現要素」と呼ぶ。(例えば以下の例における「おはようご ざいます」など。)

言語理解 XML の例：

```
<word-class label="GREETING">
おはようございます
こんにちは こんば
んは
</word-class>

<slot label="COOKING_MENU">
エビとトマトのパスタ かぼちゃサ
ラダ いちじくのレアチーズタルト
鳥とキャベツのレモン煮
</slot>

<concept label="c:greeting">
(GREETING)
```

```
</concept>

<concept    label="c:request_cooking_menu">
(COOKING_MENU)
</concept>
```

2-1. word-class 表現要素を類似性、関連性でまとめることを目的としたクラスタグである。ラベルを他のクラスタグの表現要素として定義することで、表現要素を展開する　ことができる。

```
<word-class label="GREETING">
おはようございます
こんにちは　こんば
んは
</word-class>

<concept label="c:greeting">
<!-- 「(ラベル)」の書式で、他のクラスタグの表現要素として定義することができる -->
(GREETING)
</concept>

↓以下の定義に同じ
<concept label="c:greeting">
```

おはようございます
こんにちは こんば
んは
</concept>

ラベルの定義はネスト（入れ子）にすることができる。すなわち、word-class タグで定義されたラベルは、他の word-class, slot, concept タグの中で使用することができる。許容されるネストの深さは、最大で 10000 である。

```
<word-class label="NEST1">
ネスト 1
</word-class>

<word-class label="NEST2">
(NEST1)
ネスト 2
</word-class>

<word-class label="NEST3">
<!-- ラベル NEST2 は NEST1 を含む -->
(NEST2)
ネスト 3
</word-class>

<concept label="c:nest">
<!-- ラベル NEST3 は NEST1, NEST2 を含む -->
(NEST3)
</concept>
↓以下の定義に同じ
```

```
<concept label="c:nest">
ネスト1
ネスト2
ネスト3
</concept>
```

2-2. slot

word-class 同様、表現要素を類似性、関連性でまとめることを目的としたクラスタグであり、word-class との差分は下記の通り。

(1) アクションコード部で入力データを参照することができる

入力データのうち、スロットに対応する部分は、KEYWORD という変数に値が格納されており、アクションコード部においてその値を参照することができる。値の参照方法については後述する。

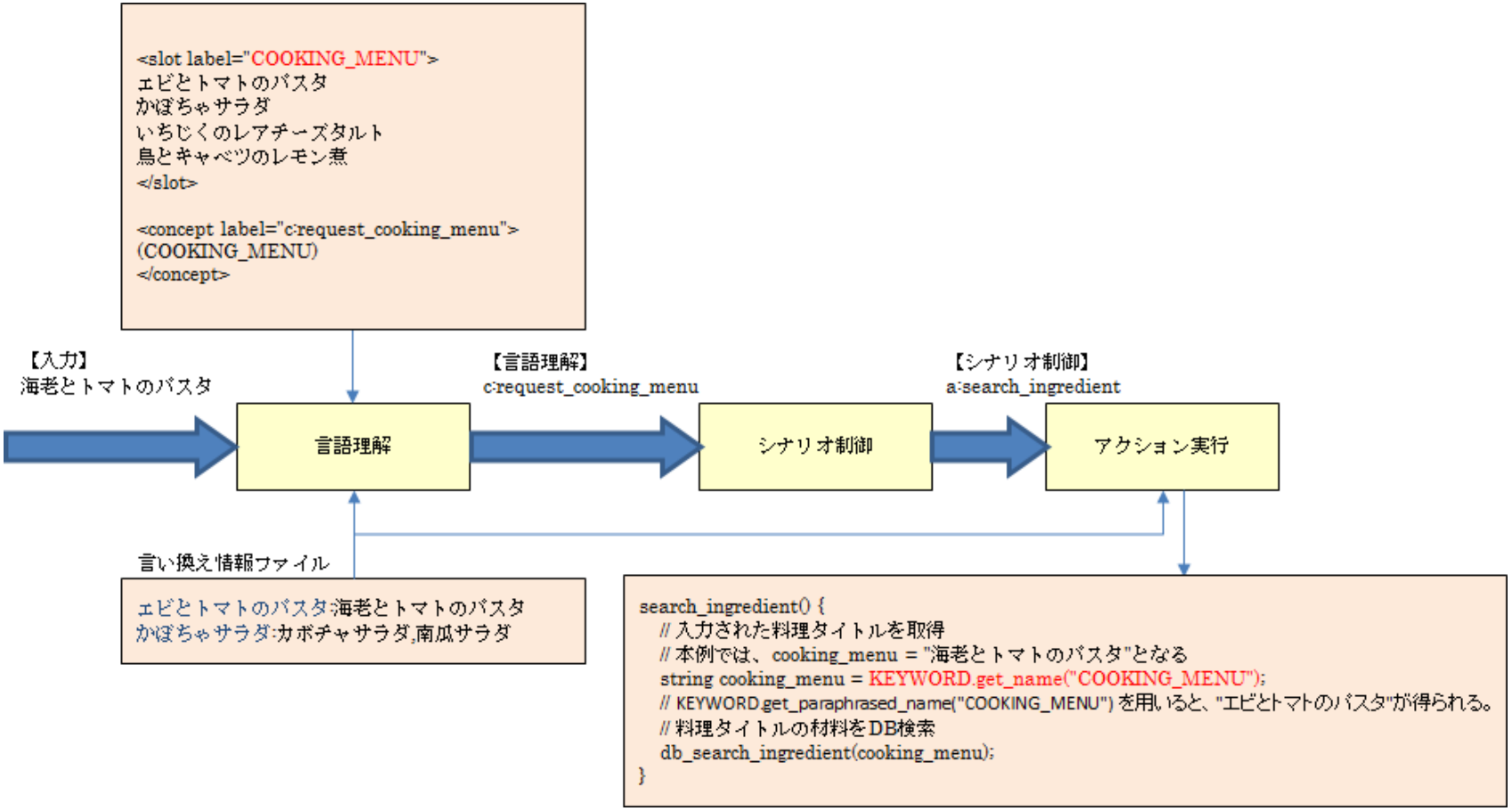
(2) concept でのみラベルを展開することができる

word-class は他の word-class、slot でラベルを展開することができるが、slot はこれらのクラスタグで展開することができない。slot タグで定義されたラベルを concept 以外のタグで使用すると、FST 生成時にエラーが発生する。

```
<slot label="SLOT">
表現要素
</slot>

<word-class label="WORD-CLASS">
(SLOT)　　<!-- NG: word-class では slot のラベルを展開することはできない -->
</word-class>

<concept label="c:concept">
(SLOT)　　<!-- OK -->
</concept>
```


<slot label="COOKING_MENU">
エビとトマトのパスタ
かぼちゃサラダ
いちじくのレアチーズタルト
鳥とキャベツのレモン煮
</slot>
<concept label="c:request_cooking_menu">
(COOKING_MENU)
</concept>
【入力】
海老とトマトのパスタ
【言語理解】
c:request_cooking_menu
【シナリオ制御】
a:search_ingredient
言語理解
シナリオ制御
アクション実行
言い換え情報ファイル
エビとトマトのパスタ:海老とトマトのパスタ
かぼちゃサラダ:カボチャサラダ,南瓜サラダ
search_ingredient() {
// 入力された料理タイトルを取得
// 本例では、cooking_menu = "海老とトマトのパスタ"となる
string cooking_menu = KEYWORD.get_name("COOKING_MENU");
// KEYWORD.get_paraphrased_name("COOKING_MENU") を用いると、"エビとトマトのパスタ"が得られる。
// 料理タイトルの材料をDB検索
db_search_ingredient(cooking_menu);
}

以下では、スロットの値（変数 KEYWORD に格納されている値）をアクションコード部で取得する方法について説明する。 スロットの値には、以下の 2 通りがある：

正規化前の値：入力データに含まれる文字列 **正規化後の値：**言い換え（正規化）が適用された後の文字列

2 つの値の違いについて、以下の言語理解 XML と言い換え情報ファイル（DMBuilder 上の名称は「正規化情報ファイル」）とを用いた例で説明する。 言

語理 XML の例:

```
<slot label=" INGREDIENTS">
かぼちゃのサラダ
</slot>

<concept label="c:ask-how-to-cook">
(INGREDIENTS)の作り方
</concept>
```

言い換え情報ファイルの例:

```
かぼちゃのサラダ:カボチャのサラダ,南瓜のサラダ
```

この言語理解 XML および言い換え情報ファイルの下では、「かぼちゃのサラダの作り方」「カボチャのサラダの作り方」「南瓜のサラダの作り方」という入力データはいずれも c:ask-how-to-cook というコンセプトにマッチする。それぞれの入力データにおけるスロットの値は下表の通りである。

入力データ	スロットの値	
	正規化前	正規化後
かぼちゃのサラダの作り方	かぼちゃのサラダ	かぼちゃのサラダ
カボチャのサラダの作り方	カボチャのサラダ	かぼちゃのサラダ
南瓜のサラダの作り方	南瓜のサラダ	かぼちゃのサラダ

言い換え情報ファイルに記述されていない文字列がスロットに格納された場合は、正規化後の値は正規化前の値と同一である。

次に、スロットの値を取得する方法について説明する。変数 KEYWORD には、ラベル名（スロット名）から値を取得する関数が、正規化前および正規化後それぞれについて用意されている。

正規化前の値を取得： KEYWORD.get_name(ラベル名)　　または　KEYWORD[ラベル名]
正規化後の値を取得： KEYWORD.get_paraphrased_name(ラベル名)

ラベル名（スロット名）は文字列であり、丸かっこの有無は無視する。従って以下の 2 つは同一の値が得られる。

```
KEYWORD.get_paraphrased_name(“INGREDIENTS”);
KEYWORD.get_paraphrased_name(“(INGREDIENTS)”);
```

2-3. concept

WFSTDM への入力を、シナリオ WFST(シナリオ制御)への入力となるコンセプトに変換することを目的としたクラスタグ。　ラベル（label 属性で指定する文字列）は、「c:」をプレフィクスとした半角文字列とすること。

ラベルがシナリオ WFST への入力となる(前頁の図を参照)。

2-4. repeat 属性

concept "c:greeting"で「やあ」、「こんにちは」、「やあこんにちは」を表現要素とする場合、以下の定義となる。

```
<concept label="c:greeting">
やあ こんにち
は やあこんに
ちは
</concept>
```

「やあこんにちは」は、「やあ」と「こんにちは」の組み合わせとなっている。このような場合、repeat 属性を使用すれば、上記の定義を以下のようにすることができる。

```
<concept label="c:greeting" repeat="true">
やあ こん
にちは
</concept>
```

repeat 属性を使用することで、表現要素の組み合わせを新たな表現要素としてネットワークを作成することが可能となる。その一方、「やあやあやあ」や「こんにちはやあ」といった入力にもマッチするようになるため、効果と副作用とを考慮した上で使用すること。

repeat 属性は concept タグのみにおいて使用することができ、slot タグや word-class タグで使用するとエラーが発生する。また、repeat 属性の値は “true” か “false” の二値であり、”false” は repeat 属性自体を省略した場合と等価である。

2-5. 表現要素

2-5-1. ワイルドカード 表現要素中にワイルドカード「(*)」を指定することができる。これを指定することで、任意の数の任意の形態素に置き換えることができる。以下の例では、「エビとトマトのパスタ(形態素数：1、形態素：トマト)」はもちろん、「エビとトマトクリームのパスタ(形態素数：2、形態素：トマト|クリーム)」といった入力もコンセプト"c:request_cooking_menu"に変換される。

```
<slot label="COOKING_MENU">
エビと(*)のパスタ　かぼちゃ
サラダ　いちじくのレアチー
ズタルト　鳥とキャベツのレ
モン煮
</slot>

<concept　label="c:request_cooking_menu">
(COOKING_MENU)
</concept>
```

なお、concept タグ内の各行の表現要素は、前後に暗黙のワイルドカードが付与されているものとして扱われる。従って、以下の 4 通りのコンセプトは全て等価であり、いずれも「…エビのパスタ…」という入力とマッチする。(「…」は 0 個以上の任意の形態素を表わす。)

```
<concept label="c:without-wildcard">
エビのパスタ
</concept>
```

```
<concept label="c:with-wildcard-both">
(*)エビのパスタ(*)
</concept>
<concept label="c:with-wildcard-left">
(*)エビのパスタ
</concept>
<concept label="c:with-wildcard-right">
エビのパスタ(*)
</concept>
```

一方、表現要素の両端のワイルドカードと repeat 属性とを併用した場合は、ワイルドカードの有無によってマッチする入力データが異なる。例えば以下の 4 通りのコンセプトはいずれも「エビのパスタ」の 1 回以上の繰り返しにマッチするが、c:repeat-without-wildcard は「エビのパスタ」の間に他の形態素が挿入されるのを許容しないのに対し、残りの 3 つはそれを許容する。

```
<concept label="c:repeat-without-wildcard" repeat=”true”>
エビのパスタ
</concept>
<concept label="c: repeat-with-wildcard-both" repeat=”true”>
(*)エビのパスタ(*)
</concept>
<concept label="c: repeat-with-wildcard-left" repeat=”true”>
(*)エビのパスタ
</concept>
```

```
<concept label="c: repeat-with-wildcard-right" repeat=”true”>
エビのパスタ(*)
</concept>
```

2-5-2. Any ワード

表現要素中に Any ワード「(.)」を指定することができる。これを指定することで、任意の形態素 1 語に置き換えることができる。　ワイルドカードは任意の数の形態素に置き換えることが可能であるが、Any ワードは 1 語のみとなる。　以下の例では、入力「エビとトマトのパスタ(形態素数：1、形態素：トマト)」はコンセプト"c:request_cooking_menu"に変換されるが、「エビとトマトクリ ームのパスタ(形態素数：2、形態素：トマト|クリーム)」では変換されない。

```
<slot label="COOKING_MENU">
エビと(.)のパスタ　かぼちゃ
サラダ　いちじくのレアチー
ズタルト　鳥とキャベツのレ
モン煮
</slot>

<concept　label="c:request_cooking_menu">
(COOKING_MENU)
</concept>
```

2-5-3. ユーザーによる形態素分割　ユーザーは半角カンマ","を使用することで、表現要素を自由に形態素分割することができる。

半角カンマで形態素境界を指定した場合、WFSTDM で形態素分割処理は実施されない。この性質を利用すると、表現要素の先頭か末尾にコンマを付与することで、その表現要素に形態素解析が適用されるのを抑止することもできる。　本機能は、研究開発時の作業、デバッグの効率化を主たる目的としたものであるため、サービス(本番)環境での利用を推奨しない。

```
<word-class　label="SPLIT_BY_WFSTDM">
<!-- WFSTDM で形態素分割。形態素分割した結果は「情報通信研究機構」 -->
情報通信研究機構
</word-class>

<word-class label="SPLIT_BY_USER">
<!-- ユーザーによる形態素分割。上記と異なり、「情報|通信|研究|機構」と分割される -->
情報,通信,研究,機構
<!—表現要素の先頭か末尾にカンマを付与すると、表現要素の分割を抑止される　　-->
,情報通信研究所
情報通信研究所,
</word-class>
```

2-5-4. 表現要素内の空白の扱い

Ver.1.1.0 より、表現要素内に空白を含めることができるようになった。[1] 空白が形態素区切りに与える影響は、空白の両側の文字が半角か全角かによって異なる。

・半角文字に挟まれた空白は、区切りとして扱われる。
・全角文字に挟まれた空白は、無視される。そこが形態素の区切りになるかどうかは、空白の有無とは無関係であり、形態素解析の結果に依存する。
・半角文字と全角文字とに挟まれた空白は無視される。空白の有無にかかわらず、必ず区切りとして扱われる。
・丸かっこで囲まれた文字列（例えば“(SPOT)”）の両端の空白は無視される。空白の有無にかかわらず、丸かっこで囲まれた文字列の両端は区切りとして扱われる。

[1] Ver. 1.0.0 では、空白は表現要素の区切りとして扱われるため、表現要素内に空白を用いることはできなかった。そのため、英語等において 2 単語以上で構成される表現要素を記述するときは、空白をカンマに置き換える必要があった。

3. 特記事項

(1) ラベルの展開 ラベルを他のクラスタグで使用する場合は、そのラベルの定義は前（前方参照）であっても後ろ（後方参照）であっても良い。

```
<word-class label="GREETING">
おはようございます
こんにちは こんば
んは
</word-class>

<concept label="c:greeting">
<!—前方参照の例-->
(GREETING)
</concept>

<concept label="c:request_cooking_menu">
<!—後方参照の例 -->
(COOKING_MENU)
</concept>

<slot label="COOKING_MENU">
エビとトマトのパスタ かぼちゃサ
ラダ
```

```
いちじくのレアチーズタルト
鳥とキャベツのレモン煮
</slot>
```

(2) 未定義ラベル 未定義のラベルを使用すると、エラーが発生する。
[2]

(3) ラベルの重複定義 同名ラベルの重複定義はエラーである。

```
<word-class label="SAMPLE">
<!--最初の定義-->
</word-class>

<word-class label="SAMPLE">
<!--SAMPLE　が再び定義されているのでエラー-->
</word-class>
```

(4) 空ラベル

空ラベルとは、表現要素を持たないタグによって定義されたラベルのことであり、以下の EMPTY がその例である。表現要素内で空ラベルを使用すると、

[2] Ver.1.0.0 においてはエラーにならず、生成された FST のアークに未展開のラベルが付与されていたが、ver.1.1.0 では仕様を明確化し、未定義ラベルは仕様外とした。

その空ラベルは無視される。また、空ラベルのみからなる行については、その行が存在していないものとして扱われる。

```
<word-class label="EMPTY">
<!—空ラベルの定義 -->
</word-class>

<concept label="c:example">
これは(EMPTY)　　<!-- “これは”と等価 -->
(EMPTY)　<!—この行は存在しないものとして扱われる -->
</concept>
```

空ラベルを含む言語理解 XML から FST を生成しようとすると、FST 自体は生成されるが、警告が表示される。

(5) 自己参照および循環参照　ラベルの自己参照（ラベルの定義の中でそのラベルを使用すること）および相互参照（ラベルを定義する複数のタグにおいて、お互いに他方のラベルを使用す　ること）をしている言語理解 XML はエラーが発生する。[3]

以下は、自己参照の例である。繰り返しを表現したい場合は、自己参照ではなくて concept タグの repeat 属性を使用すること。

```
<word-class　label="SELF-REFERENCE">
(SELF-REFERENCE)は自己参照の例
```

[3] word-class タグ内での自己参照や相互参照に対しては、ラベル定義のネストの深さが上限（10000）を超えた旨のエラーが発生する。一方、slot タグ内での自己参照では、slot ラベルを concept タグ内以外で使用しているというエラーが発生する。

```
</word-class>
```

以下は、相互参照の例である。

```
<word-class label="CLASS-A">
(CLASS-B)は相互参照の例
</word-class>
<word-class label="CLASS-B">
(CLASS-A)は相互参照
</word-class>
```

(6) 言い換え情報ファイルが言語理解 XML に与える影響

言い換え情報ファイル（DMBuilder 上の名称は「正規化情報ファイル」）に記述された値と slot, word-class, concept タグ内の表現要素とが厳密に一致した場合に限り、言語理解 XML 内の表現要素は言い換え情報ファイルに従って展開される。

例えば、言い換え情報ファイルが以下の通りであるとする。このファイルの記述の意味は、「海老」「蝦」「えび」を「エビ」と同一視することである。

```
エビ:海老,蝦,えび
```

一方、言語理解 XML において、以下のようなタグが存在するとする。（この例では slot タグだが、word-class タグや concept タグの表現要素についても下記と同様の展開が行なわれる。）

```
<slot label="INGREDIENTS">
エビ
```

```
エビとトマトのパスタ
</slot >
```

このタグの 2 つの表現要素のうち、「エビ」については、言い換え情報ファイル内の「エビ」と厳密に一致するため、「エビ／海老／蝦／えび」の 4 通りに展開される（これを「言い換えによる展開」と呼ぶ)。一方、「エビとトマトのパスタ」については、「エビ」は含むものの厳密な一致ではないため、展開は発生しない。したがって、上記の言語理解 XML は、以下のように記述されているのと等価である。

```
<slot label="INGREDIENTS">
エビ
海老
蝦
えび
エビとトマトのパスタ
</slot >
```

「エビとトマトのパスタ」についても展開させたい場合は、言い換え情報ファイルに以下のような記述を追加する。

```
エビとトマトのパスタ:海老とトマトのパスタ,蝦とトマトのパスタ,えびとトマトのパスタ
```

あるいは、言い換え情報ファイルはそのままとする一方、以下のように、言語理解 XML において「エビ」をラベルとして定義する。

```
<word-class label=”EBI”>
エビ
</word-class>
```

```
<slot label="INGREDIENTS">
(EBI)
(EBI)とトマトのパスタ
</slot>
```

他方、ラベルの展開によって言い換え情報ファイルと同じ値が生成されても、展開は発生しない。　例えば、以下のような内容の言い換え情報ファイルを考える。

```
カボチャサラダ:南瓜サラダ,かぼちゃサラダ
```

そして以下のような言語理解 XML を考える。

```
<word-class label="PUMPKIN">
カボチャ
</word-class>

<slot label="INGREDIENTS">
(PUMPKIN)のサラダ
</slot>
```

この例では、PUMPKIN というラベルが値に展開されることで、INGREDIENTS ラベルの値として「カボチャのサラダ」が生成される。しかし、表現要素自体は「(PUMPKIN)のサラダ」であって「カボチャのサラダ」ではないため、言い換え情報ファイルの値とは一致せず、言い換えによる展開は発生しない。

(7) 同名スロットの複数回使用

concept タグにおいて、一つの表現要素の中に同名のスロットラベル（slot タグで定義されたラベル）を複数回記述した場合、ビルド時のエラーは発生しないが、2 番目以降のスロットについては値を取得することができない。
例えば以下の言語理解 XML を考える。

```
<slot label=”INGREDIENT”>
エビ
トマト
</slot >

<concept label=”c:ask-ingredient”>
(INGREDIENT)と(INGREDIENT)
</ concept >
```

コンセプト c:ask-ingredient は、「エビとトマト」という入力データにマッチし、「エビ」と「トマト」はそれぞれ別の INGREDIENT スロットに格納される。しかし、KEYWORD.get_name(“INGREDIENT”) によって取得できる値は、先に出現した「エビ」のみであり、「トマト」は取得されない。

同様の現象は、concept タグ内の表現要素内で slot のラベルを記述しながら repeat 属性を有効にした場合も発生する。例えば以下の言語理解 XML で定義されるコンセプト c:ask-ingredient は、「エビとトマトと」という入力文にマッチし、「エビ」「トマト」はそれぞれ異なる INGREDIENT スロットに格納されるが、KEYWORD.get_name(“INGREDIENT”) によって取得できる値は「エビ」のみである。

```
<slot label=”INGREDIENT”>
エビ
トマト
</slot >
```

```
<concept label=”c:ask-ingredient” repeat=”true”>
(INGREDIENT)と
</ concept >
```

同じ値を取り得るスロットを concept タグ内の表現要素内で複数回記述したいときは、以下のように、それぞれ別の名前のスロット（INGREDIENT1 と INGREDIENT2）を用意するとよい。

```
<word-class　label=”INGREDIENT_ELEMENTS”>
エビ
トマト
</ word-class >

<slot label=”INGREDIENT1”>
(INGREDIENT_ELEMENTS)
</slot >

<slot label=”INGREDIENT2”>
(INGREDIENT_ELEMENTS)
</slot >

<concept label=”c:ask-ingredient”>
(INGREDIENT1) (INGREDIENT1)
と(INGREDIENT2)
```

```
</ concept >
```

(8) 文字コードおよび改行コード

言語理解 XML ファイルの文字コードは BOM なしの UTF-8 で記述すること。一方、改行コードは LF および CR+LF のどちらも可能である。