基于 XML 的业务指令分解系统研究与设计①

王如跃 雷 电 (上海大学 机电工程与自动化学院 上海 200072)

摘 要： 以证券行业为研究背景，提出了用 XML 构建证券行业中的业务知识库，并以此来自动产生业务指令的新方法。研究了知识库中数据结构的定义、分解规则的创建以及如何用 XML 描述各种取值方式和 XSLT 变换器的实现等问题。实践表明：该方法不仅能简化软件的开发，而且能高效的对业务通知单进行指令分解。同时，以此方法为基础可开发出适于多种应用领域的交互性好、易扩展的知识获取平台及相应的推理程序。

关键词： 知识库；XML；分解规则；XSLT；模板

Research and Design of Instruction Decomposition System Based on XML

WANG Ru-Yue, LEI Dian

(School of Electromechanical & Automation, Shanghai University, Shanghai 200072, China)

Abstract: This paper, with the securities industry as the research background, proposes a new method that uses XML to build a business knowledge base in the securities industry to automatically generate instruction. It studies the definition of data structures, how to create decomposition rule and use XML to describe the way of the values, how to create converter based on XSLT and so on. Practice shows that this method does not only simplify software development, but is also highly effective to carry out instructions decomposition. Based on this approach, knowledge platform and its corresponding reasoning process can be developed which is interactive and easy to expand for a variety of applications.

Keywords: knowledge base; XML; decomposition rules; XSLT; template

知识表示是为了描述外界事物所作的一组约定，是知识的符号化，是将关于事物的事实、关系、过程等进行编码并成为一种合适的数据结构[1]。传统的知识表示方法有很多，如：一阶谓词逻辑、产生式、框架、表格等。但在表示能力，可理解性，可操作性，可扩充性方面仍存在不足。目前，基于 XML 的面向对象知识表示方法成为研究热点。

1 基于XML的业务指令分解系统

证券交易所管理部作为交易系统业务指令的产生部门，承担着繁重的交易系统业务维护工作。根据各业务部门提供的业务通知单，分解成对交易主机的操作指令后对主机操作。目前交易管理部的证券指令分解工作仍采用纯手工的处理，不但效率低，且需要处理的通知单类型越来越多为安全运行埋下了隐患。有一套可定制的业务指令分解系统具有重要的作用，这不但能提高工作效率，而且能降低出错率。

针对以上背景，本文提出了用 XML 构建证券行业中的业务通知单知识库并自动产生分解指令的新方法。首先，采用 XML 语言的自描述功能，将业务通知单数据以 XML 的格式保存，然后分别利用 XML Schema 和 XSLT 建立了通用的业务指令分解规则架构和变换器，从而完成了从业务通知单数据到目标指令的成功分解。

① 基金项目:上海市重点学科建设项目资助(T0103)
收稿时间:2009-08-10;收到修改稿时间:2009-09-13

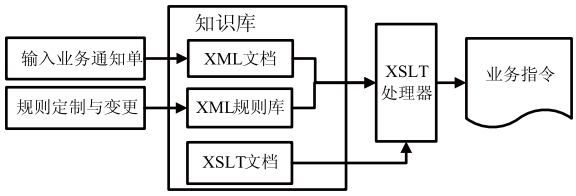

图 1　基于 XML 业务指令分解系统流程图

2　基于XML知识库的构建

2.1 基于 XML 的业务通知单知识表示

应用 XML 存储数据，首先要创建一个 XSD 文件，文件中要包含所有需要存储的数据字段。同时，在创建 XSD 的时候要考虑这些字段的组织结构，要使得这些字段的结构更为合理，使得后续的操作更为方便。本文根据系统的实际需求，并通过对目前业务通知单信息的分析研究，得到描述业务通知单的 XML 文档的数据结构如图 2 所示。

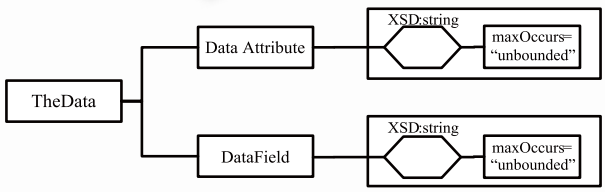

图 2　业务通知单数据结构

其中， XSD:string 表示元素的数据类型，max Occurs 表示数据出现的最大次数，1 表示只能出现一次，unbounded 表示可重复出现无数次。

业务通知单知识库的构成可用 BNF 范式表述如下[2]：

TheData∷=<DA><DF>

其中，DA (Data Attribute)代表知识属性，是对所描述知识本体特征的表述，包含规则匹配信息；DF(DataField)代表知识表达，是知识本体的具体表示。

本系统的界面采用了 HTML 语言编写的网页形式，不同的业务通知单(如发行、上市、停复牌等)定制不同的输入界面，界面样式类似各业务部门提交的通知单。当业务人员在业务通知单界面中输入数据并提交保存后，就得到如下结构的 XML 文档：

```
<TheData　Attribute = “　”>
<DataFld_name>value</DataFld_name>
……
</TheData>
```

2.2 XML 转换规则

2.2.1 规则架构

在知识库中，有了元数据，还必须定义出规则，才能准确的完成推理[3]。分解规则将定义：指令中的各指令项源于业务通知单的哪项内容；如何建立业务通知单同指令的映射关系；如何将业务通知单分解成多条指令。推理机是基于规则进行推理的，为了使其具有良好的通用性和扩展性，同时也为了降低推理程序的设计难度和增加推理程序的通用性，规则必须要有一套统一的架构，从而使推理程序不随着规则的改变而重新设计。图 3 为本系统转换规则架构图：

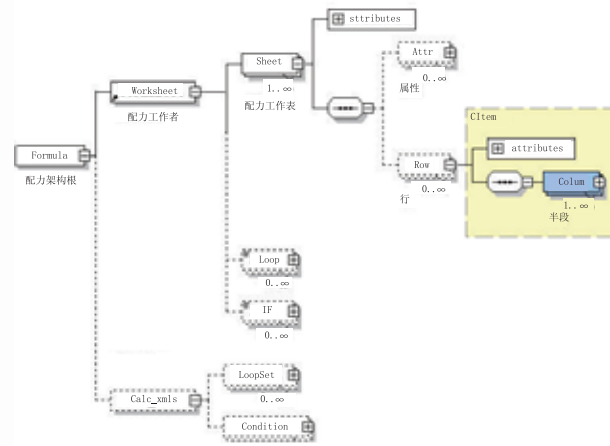

图 3　转换规则架构图

架构图中，Formula 是 XML 规则文档的根元素，它包括 WorkSheet 和 Calc_xmls 两个子元素。WorkSheet 内定义了配方工作表，即具体的指令格式，它又包括三个子元素，Sheet 内部定义了常规指令转换规则，Loop 内部定义了需要循环的指令转换规则，而 IF 内部则定义了基于条件的指令转换规则；Calc_xmls 内部则定义了基于条件的指令规则和循环指令规则中需要的参数，详见下文。

2.2.2 具体的规则描述

(1) 规则中的取值描述

指令分解中最重要的就是取值问题，在根据规则进行指令分解的过程中，许多数据来自业务通知单。在图 3 所示的架构中，Colum 元素再往下就是 XML

取值表达式。取值通常有三种情况：1)取的值是固定的，即常量。2)取的值直接来源于输入的业务通知单或数据库的数据，即变量。3)取的值需要获得业务通知单和数据库的数据然后进行运算得到，即函数值。下面是三种取值方式在规则中的具体描述方式：

① 取的值为常量

```
<const value="常量"/>
```

② 取的值为变量

```
<dataFld name="字段名 " />
```

③ 取的值为函数值

```
<callFuc name="函数名 " >
  <with_param>
    <参数表达式/>
  </with_param>
  ……
</callFuc>
```

当然，在函数取值方式中，参数的取值又会涉及到这三种取值方式，即取的值为常量、变量以及函数值，它的表示方法就是把以上三种取值表示法嵌套进去。

(2) 常规指令转换规则描述

```
<Sheet name="指令集名称 ">
  <Row>
    <Colum name="">
      <参数表达式/>
    </Colum>
    ……
  </Row>
  ……
</Sheet>
```

Sheet 元素代表一个指令集，子元素 Row 表示一行数据，即一条指令，一个指令集可以包含多条指令，Colum 元素内部就是指令中需要的具体参数，参数表达式就是把上面所说的三种取值方式嵌套进去。

(3) 基于条件的指令转换规则描述

即如何根据业务通知单中的数据将业务数据生成不同的目标指令。用 XML 描述如下：

```
<IF condition="条件 id" >
  ……
</IF>
```

省略号代表的是具体的与业务数据相关的规则集，格式同常规指令转换规则相同。当 XSLT 匹配到 IF 元素时，首先就会调用 condition 模板进行条件判断，只有条件为真才会继续往下执行。

在Calc_xmls元素的Condition子元素内对条件规则进行描述如下：

```
<Condition id=" ">
  <callFunc name="compStr" >
    <with_param data="判断条件 "/>
      <with_param>
        <dataFld name="字段名 "/>
      </with_param>
    </callFunc>
</Condition>
```

compStr 是在 XSLT 变换器中定义个进行字符串比较的脚本函数，两个参数是要进行比较的两个字符串，其中第二个参数来自当前的业务通知单。

(4) 含循环的指令转换规则描述

有的目标指令有时需要根据同一个规则产生多条指令，而具体的条数需要根据通知单中的数据来决定，这就需要运用循环机制来实现。用 XML 描述如下：

```
<Loop>
  ……
</Loop>
```

同条件转换规则类似，省略的规则集格式与常规指令转换规则相同。当 XSLT 转换器执行到 Loop 元素时，首先会到 Calc_xmls 元素的 LoopSet 子元素内读取与循环相关的初始参数，具体的描述如下：

```
<LoopSet >
  <start>
    <参数表达式/>
  </start>
  <end>
    <参数表达式/>
  </end>
</LoopSet>
```

在本应用中，两个参数均来自业务通知单中输入的日期，在 XSLT 转换器中通过比较这两个参数决定循环的次数。

2.3 XSLT 推理程序

推理是知识系统中必不可少的过程，所谓推理是指依据一定的规则从已有的事实推出结论的过程。以

往的知识推理都是基于知识的表示方式和组织方式开发独立的推理程序，当规则发生变换时就需要重新编写推理程序，使系统缺少灵活性和扩展性。本系统根据 XML 的知识表示特点我们建立了相应的基于 XSLT 的推理程序，当规则发生变更时，只要规则符合之前定义的架构，就不需要重新编写推理程序，从而使变换规则具有很强的可扩展性。

XSLT 转换是以 XML 源文档为输入，利用预先编写好的转换模板(XSLT 样式表) 把它转换成目标文档[4]。在此系统中 XML 源文档包括从界面输入所产生的 XML 文档和指令分解规则的 XML 文档。

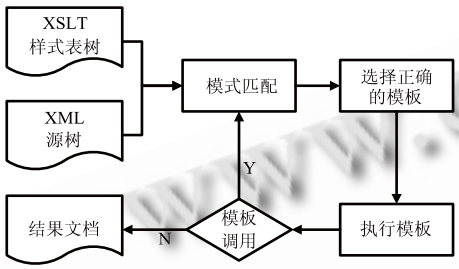

图 4　XSLT 转换流程图

2.3.1 取值变换

在 XSLT 转换样式表中，我们为每一种取值方式定义一个模板，在根据规则进行变换的过程中，当遇到取值时就会去调用相应的模板。

(1) 常量取值的变换实现：

```
<xsl:template match="const">
  <xsl:value-of select="@value"/>
</xsl:template>
```

(2) 变量取值的变换实现：

```
<xsl:template match="dataFld">
  <xsl:apply-templates select="//the Data
/*[name()=current()/@name]" />
</xsl:template>
```

当执行到 dataFld 元素时，就把 dataFld name 对应的字段名的具体值赋给它，这个值就是 theData 目录下的对应字段名的值。

(3) 函数调用的变换实现：

```
<xsl:template match="callFuc">
  <xsl:variable name="size"select="count
(with_param) "/>
  <xsl:variable name="call">
    <xsl:value-of select="@name"/>(
      <xsl:for-each select="with_param">
        <xsl:apply-templates select="."/>
        <xsl:if test="position()&lt;$size">,
</xsl:if>
      </xsl:for-each> );
  </xsl:variable>
<xsl:value-of select="L:runCommonCode(string
( $call))"/>
</xsl:template>
```

当执行到 callFuc 元素时，首先定义一变量 size，并取得具体的被调函数的参数个数，然后把值赋给 size。然后定义一变量 call，取得被调函数的函数名和具体的参数值，然后赋给 call，运行 call 函数，即得到目标函数值。runCommonCode()是我们在变换器中定义的脚本函数，其功能都是运行用户的 call 函数。

2.3.2 条件规则的变换

```
<xsl:template match="IF">
  <xsl:variable name="condition" select=
"msxsl:node-set($calc_xml)/Condition[@id=cu
rrent()/@condition]"/>
  <xsl:if test="$condition = 1">
    <xsl:apply-templates/>
  </xsl:if>
</xsl:template>
```

当 XSLT 变换器执行到条件规则 IF 元素时，就会调用 IF 模板。在 IF 模板内部，首先定义一个变量 condition，并将 calc_xml 元素内对应 id 的 condition 的值赋给它，然后进行判断，如果 condition 的值为 1，继续往下执行，得到指令。

2.3.3 循环规则的变换

```
<xsl:template match="Loop">
  <xsl:variable name="child" select="./*"/>
  <xsl:for-each select="msxsl:node-set($calc_
xml)/LoopSet[@id=current()/@loop]/*">
    <xsl:apply-templates select="$child"/>
  </xsl:for-each>
</xsl:template>
```

当 XSLT 变换器执行到循环规则 Loop 元素时，就会调用 Loop 模板。进入 Loop 模板后，接着就会

调用 LoopSet 模板,它的作用是到变换规则 calc_xml 元素对应的 LoopSet 子元素中获取循环需要用到的初始参数。

```
<xsl:template match="LoopSet">
  <xsl:variable name="start">
    <xsl:apply-templates select="start/*"/>
  </xsl:variable>
  <xsl:variable name="end">
    <xsl:apply-templates select="end/*"/>
  </xsl:variable>
  <xsl:call-template name="forLoop">
    <xsl:with-param name="start" select="$start"/>
    <xsl:with-param name="end" select="$end"/>
  </xsl:call-template>
</xsl:template>
```

在获取到循环参数初始值后，带着这两个参数调用 forLoop 模板：

```
<xsl:template name="forLoop">
  <xsl:param name="start"/>
  <xsl:param name="end"/>
    <xsl:if test="$start != $end">
      <xsl:element name="{$tag}">
        <xsl:value-of select="$start"/>
      </xsl:element>
      <xsl:call-template name="forLoop">
        <xsl:with-param name="start" select=""/>
        <xsl:with-param name="end" select="$end"/>
      </xsl:call-template>
    </xsl:if>
</xsl:template>
```

在 forLoop 模板中，首先会判断 start 和 end 的值是否相等，如果不等，就会继续往下执行，产生一个新的元素 tag，并将 start 的值赋予 tag，以备在产生的循环指令中使用，然后产生新的 start 值，然后再调用 forLoop 模板，直到 start 和 end 的值相等，返回到 Loop 模板，在 Loop 模板，根据 tag 元素的个数决定循环的次数，产生循环指令。

3 结论

本文提出了基于 XML 知识表示的新方法。XML 有表示结构化数据的能力，在知识库表示上具有以下优势[5]：(1)降低用户构建知识库的难度和开发成本；(2)提供面向对象知识表示的支持；(3)统一规则的表示方法使系统具有良好的扩展性；(4)基于 XSLT 的推理程序方便系统的开发。笔者以证券行业中的业务指令分解系统为背景详细介绍了知识表示的具体实现，以此方法为基础，可在更多相关领域开发出自动化程度较高的知识获取及推理工具。

参考文献

1 于卫红,贾传荧.基于 XML 的海上搜索智能支持决策系统知识库.大连海事大学学报, 2005,31(4):30－32.

2 陈颖,熊前兴.XML 在专家系统中的知识表示技术研究[硕士学位论文].武汉:武汉理工大学, 2008.

3 黎建辉,吴威.一种基于 XML 的元数据映射与转换方法.微电子学与计算机, 2008,25(1):34－38.

4 尹章才,李霖.基于 XSLT 的知识表示方法研究.武汉理工大学学报, 2007,31(6):1083－1086.

5 徐大庆,李淼,等.基于 XML 的面向对象知识模式设计.计算机系统应用, 2008,17(3):64－68.