基于 XML Schema 的知识描述与模式验证①

Knowledge Representation And Pattern Validation Based On XML Schema

许桂艳[1,2]　张　建[1]　李　淼[1]　耿　英[1,2]　徐大庆[1,2]
(1 中国科学院 合肥智能机械研究所 安徽 合肥 230031)
(2 中国科学技术大学 自动化系 安徽 合肥 230027)

摘　要： 设计一种基于 XML 的面向对象的知识表示方法,并完成其 XML Schema 设计。在 XML Schema 的约束规范下,开发相应的知识获取系统、验证查错系统和推理系统,从而构成一个完整专家系统开发平台。本文主要设计并实现其中的验证查错系统。通过 JDOM 对知识获取系统获得的 XML 知识库进行模式验证与错误处理,保证知识获取得到的知识的可用性。

关键词： XML Schema 模式 JDOM 验证

1 引言

XML 以其稳健、可扩展、可维护且易于理解等优点已经逐步称为新软件系统的基础支柱。以 XML 为代表和基础的 XML 技术体系正日益成为数字化、网络化信息环境中对信息进行组织、处理和交换的基础[1]。

本文设计了一套基于 XML 的知识表示方法,用于专家系统开发平台的知识约束,并通过 XML Schema 进行描述。在该约束规范下进行知识获取,得到 XML 知识库,然后对该 XML 知识库进行模式验证与错误处理,保证获取的知识库的可用性,最后通过 XML 的自标记系统信息进行推理应用,将专家的知识通过信息化的方式进行传播和共享。

2 系统结构

2.1 基于 XML 的专家系统开发平台的体系机构

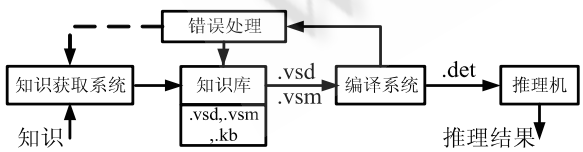

图 2　DET 运行流程图

本实验室早期开发的 DET (Development Tools of Expert System,专家系统开发平台) 主要包括知识获取系统、编译系统和推理系统三部分。开发了自定义的知识描述语言,使用自定义的文件存储形式 (kb、vsd 和 vsm 文件) ,通过编译系统将这些文件编译为二进制码 (det 文件) ,最后通过推理系统解释运行。DET 的基本运行流程如图 1 所示。

由于采取的是“编译 + 解释”的方案,其中涉及到复杂的内存管理和指令操作,而且使用自定义的文件存储形式,所以对于软件平台的开发和维护人员来讲,对软件进行维护和升级都比较困难。

为此本文提出摒弃编译方案,将专家知识按照一定的知识表示方法通过知识获取系统存储到 XML 文件中,并对 XML 知识库文档进行验证和查错,然后直接根据 XML 的节点信息进行推理。新 DET 开发平台的基本运行流程如图 2:

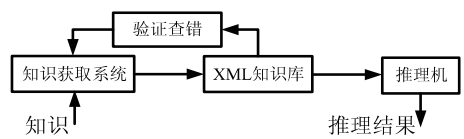

图 2　新 DET 运行流程图

由图 1 和图 2 不难看出,新 DET 平台在功能保持不变的情况下,结构比原平台要简练得多,而且使用通

① 基金项目:中科院知识创新工程重要方向项目 (KGCX2 - SW - 511)

用的 XML 存储方式，不用涉及复杂的内存管理和二进制指令操作，可以节省大量的维护工作，便于系统的升级和更新。

本文主要论述验证与查错系统的设计实现。由于验证与查错建立在知识表示的基础之上，所以本文将先简单介绍基于 XML Schema 的知识表示。

2.2 系统实现

本文的设计实现在 Windows XP 平台上，利用 Altova XMLSpy 开发工具开发用于约束知识表示的 Schema；在 JDK1.6 环境下，利用 Eclipse3.3 + JDOM1.0 完成基于 Schema 的 XML 验证和错误处理。软件开发过程中，尽力做到界面与数据的分离及对知识表示格式的校验。

3 基于 XML Schema 的知识表示

W3C XML Schema 语言是一种非常有力的模式语言。该模式语言可以声明特定元素或者属性，包含整数、日期、字符串、双整型、名称以及正整数等。除语言定义的 42 种简单类型外，还可以自定义类型以扩展该语言的数据类型。

基于 XML Schema 的知识库主要包括 16 种复杂类型和 8 种简单类型。其中知识模型类型（KnowledgeBaseType）是主要的复杂类型之一。知识模型类型中可以包括变量库类型（VariableBaseType）、函数库类型（FuncBaseType）、规则库类型（RuleBaseType）和子模型库类型（KnowledgeBaseType），但并不要求同时包含。由于子模型库类型与知识模型类型属于同种类型，所以通过知识模型类型与子模型库的嵌套，既可以实现知识的树状结构，也可以实现循环嵌套的收敛。知识模型类型（KnowledgeBaseType）的结构如图 3 所示：

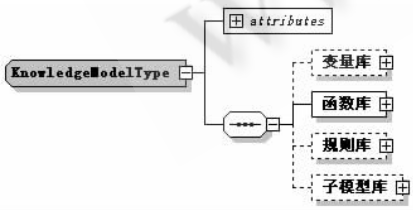

图 3　知识模型类型结构

为增强知识表示的功能性与智能性，在变量库中设计了枚举、多选、模糊变量类型等自定义类型。其中模糊变量类型（FuzzyVarType）的结构如图 4 所示：

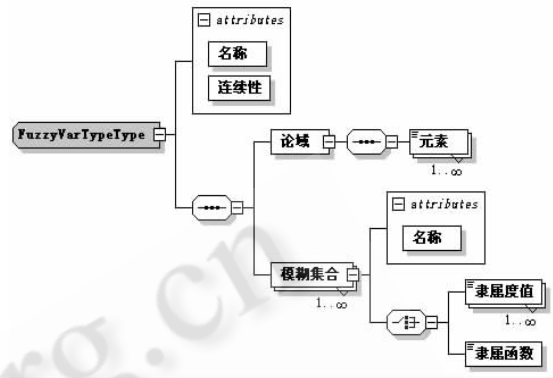

图 4　模糊变量类型结构

规则库是专家系统非常重要的组成部分，规则库类型（RuleBaseType）的结构图如图 5 所示：

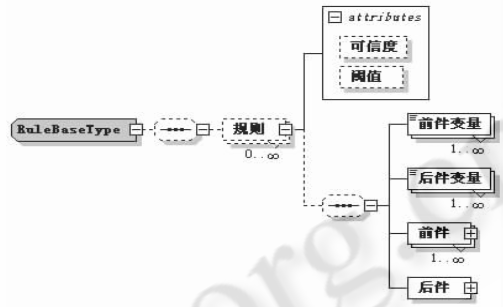

图 5　规则库类型结构

限于篇幅，变量库和规则库中的具体结构就不再一一详述，函数库主要用于内部推理运行，在此也不做介绍[2]。

4 模式验证

本文中的 XML 文档验证主要用于两个用途：（1）验证文档有效性，保证文档符合 XML 语法和 Schema 中描述的约束规范；（2）与文档有效性无关的语义错误处理。基于 Schema 的 XML 文档验证模型如图 6 所示：

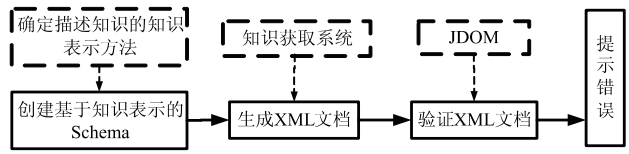

图 6　基于 Schema 的 XML 文档验证模型

4.1 验证 API 选择

本文中介绍三种最为常用的 API：DOM、SAX、JDOM。通过对这三种 API 各自特点的比较，结合本文中的实际应用情况，选择合适的 API。

4.1.1 DOM

DOM（Document Object Model，文档对象模型）是处理 XML 数据的传统方法，是用与平台和语言无关的方式表示 XML 文档的官方 W3C 标准。由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的，在使用 DOM 时，数据以树状结构的形式被加载到内存中。DOM 适合于当今流行的各种语言。

DOM 的优点在于：由于树在内存中是持久的，因此可以修改它，便于应用程序对数据和结构的更改，它还可以在树中上下导航，使用起来比 SAX 方式也要简单得多。

DOM 的缺点在于：在内存中构造 DOM 树需要大量的内存开销，特别是大型文件可能完全占用系统内存容量。

4.1.2 SAX

SAX（Simple API for XML，用于 XML 的简单 API）为 XML 处理程序定义的一组公共接口。SAX 是一个公共的基于事件的 XML 文档解析标准，SAX 提供一种对 XML 文档进行顺序访问的模式，这是一种快速读写 XML 数据的方式。当使用 SAX 解析器对 XML 文档进行解析时，会触发一系列事件，并激活相应的事件处理函数，从而完成对 XML 文档的访问，所以 SAX 接口也被称作事件驱动接口。拥有 XML 处理程序的几乎所有语言都支持 SAX。

SAX 的优点在于：分析能够立即开始，而不是等待所有的数据被处理；应用程序在读取数据时检查数据，不需要将数据加载到内存，对于大型文档来说这是个巨大的优点。

SAX 的缺点在于：应用程序没有以任何方式存储数据，使用 SAX 来更改数据或在数据流中往后移都比较困难。

4.1.3 JDOM

JDOM 是一种用 JAVA 语言读、写、操作 XML 的 API 函数，与 SAX 和 DOM 标准兼容，这些 API 函数被最大限度的优化。相对 SAX 或 DOM，用 JDOM 来处理 XML 文档要简单得多，2002 年的 JavaOne 会议上 Jason Hunter 有一篇精彩的演讲介绍 JDOM：JDOM Makes XML Easy[3]。JDOM 自身不包含解析器，它通常使用 SAX2 解析器来解析和验证 XML 文档。它可以将 JDOM 表示输出成 SAX2 事件流、DOM 模型或 XML 文本文档。

JDOM 与 DOM 非常类似，JDOM 核心的要求是以 JAVA 为中心，只适合于 JAVA 语言，它遵循 DOM 接口的主要规则，除去了 DOM 中与 JAVA 习惯的不同。

JDOM 与 DOM 的不同在于：JDOM 使用具体类而不使用接口，从而简化了 API，但也限制了灵活性；大量使用 Collections 类，使得 JAVA 开发者使用更方便。

4.1.4 选择 JDOM

不管选择那种 API，努力坚持它的被许多文件证明了的部分，尽可能应用那些标准 API 中的方法和类。如果确实遇到性能问题或者 bug，上面的做法可以使你能更容易地切换到更好的实现机制。

综上所述，针对 JAVA 编程实现的特点以及实际的应用情况，本文的验证模型采用 JDOM 对 XML 知识库进行验证。

4.2 文档有效性验证实现

XML 的验证主要用于 XML 文档的有效性，保证 XML 知识库符合 XML 语法和 Schema 约束。由于 JDOM 自身不包含解析器，本文采用 Xerces2 – j 提供的 SAXParser 对 XML 进行验证[4]。本文中 SAXParser 验证 XML 文档的步骤如下：

（1）导入 SAXParser 类和 DefaultHandler 类：

```
import org. apache. xerces. parsers. SAXParser;
import org. xml. sax. helpers. DefaultHandler;
```

（2）创建 SAXParser 对象 sp：

```
SAXParser sp = new SAXParser( );
```

（3）设置验证（valadition）的特征为 true 来报告不符合 XML 语法的验证错误：

```
sp. setFeature( " http://xml. org/sax/features/validation" , true);
```

（4）设置验证（valadition）/模式（Schema）的特征为 true 来报告不符合 Schema 约束规范的验证错误：

```
sp. setFeature ( " http://apache. org/xml/features/validation/schema" ,true);
```

（5）将 validation/schema – full – checking 的特征设置为 true，使能够在整体上对模式和语法约束进行

检查：

```
sp. setFeature ( " http://apache. org/xml/features/validation/schema – full – checking" ,true) ;
```

(6)为解析器指定验证模式：

```
sp. setProperty( " http://apache. org/xml/properties/schema/external – schemaLocation" ,SchemaUrl) ;
```

(7)创建一个继承 DefaultHandler 的类 Validator：

```
private class Validator extends DefaultHandler {
  public boolean validationError = false;
    // 将 validationError 初始化为 false,如果发现错误则变为 true.
  public SAXParseException saxParseException = null;
    //创建用于接收错误的 SAXParseException 对象.
  public void error( SAXParseException exception)
   throws SAXException {
    //接收可恢复的错误的通知.
    validationError = true;
    saxParseException = exception;
   }
  public void fatalError( SAXParseException exception)
   throws SAXException {
    //接收不可恢复的错误的通知.
    validationError = true;
    saxParseException = exception;
   }
  public void warning( SAXParseException exception)
   throws SAXException {
    //不接收警告的通知.
   }
}
```

(8) DefaultHandler 类实现了 ErrorHandler 接口,用来为 Xerces 解析器指定一个 ErrorHandler。Validator 类允许我们把它注册为解析器的一个 ErrorHandler：

```
Validator handler = new Validator( ) ;
sp. setErrorHandler( handler) ;
```

(9)因为 Validator 类实现了 ErrorHandler,我们可以用它来解析 XML 文档：

```
sp. parse( XmlDocumentUrl) ;
```

(10)报告异常：主要是 IOException 和 SAXException 这两种异常。

经过上述步骤可以验证 XML 文档的有效性,验证成功就说明被验证的文档是符合 XML 的语法规范和我们预先定义的 Schema 约束规范的。

4.3 错误处理

上述验证过程主要是用于文档有效性的验证,但是通过了有效性验证还不能保证 XML 知识库是可用的,因为可能还存在与文档有效性无关的语义错误。要确保 XML 文档的可用性,还要进行与文档有效性无关的错误检查与处理。

与文档有效性无关的错误检查发生在模式验证之后,如果模式验证成功,则进入本阶段。本阶段的程序设计思想主要是：将 Schema 中定义的所有复杂类型都设计为一个类,并通过 JAVA 类进行模拟,每个类都包含两个方法：Save( ) 和 Check( )。在读 XML 文档时,调用 Save( ) 方法将文档的所有内容都装载到类中保存起来,然后遍历该类,调用 Check( ) 方法进行查错。

根据 Schema 的描述,Schema 中自定义的复杂类型主要有 16 种。在本文验证的实际操作过程中,为简化操作扩展了两种过渡类型,一共模拟实现了 18 种复杂类型的 JAVA 类,其中(15)和(16)是在验证过程中添加的两个过渡验证类型。18 种自定义的复杂类型如下：

(1) KnowledgeBaseType：知识库类型
(2) KnowledgeModelType：知识模型类型
(3) VariableBaseType：对象库类型
(4) RuleBaseType：规则库类型
(5) FuzzyVarType：模糊变量类型
(6) FunDefineType：函数定义类型
(7) ExpressionType：表达式类型
(8) AssignExpType：赋值表达式类型
(9) ArithmeticExpType：算术表达式类型
(10) LogicExpType：逻辑表达式类型
(11) FunCallType：函数调用类型
(12) FunBodyType：函数体类型
(13) StatementType：Statement 类型

(14) ArrayElementType:数组元素

(15) ChildModelBaseType:子模型库类型

(16) FuncBaseType:函数库类型

(17) MainFunType:主函数类型

(18) LeftValType:左值类型

知识库验证检查出错误时需要进行相应的错误处理。经过验证显示被验证 XML 文档有错误时,则调用静态方法:public static void exception(int code, String errPosition)来进行错误处理,然后根据错误信息的代码,将错误信息以"错误信息代码+错误类型+错误发生的位置"的格式输出到屏幕并保存到文件 err.tmp 中。其中 code 是错误编号,errPosition 是错误发生的位置。

表1 错误编号与错误类型

编号	类 型	编号	类 型
1	未定义的变量	8	常量类型不匹配
2	未定义的函数	9	常量类型错
3	变量类型不匹配	10	变量类型错
4	元素个数与模糊集合个数不匹配	11	类型重复定义
5	变量重复定义	12	参数类型错
6	函数重复定义	13	属性类型错
7	参数不匹配		……

本文前期完成的知识获取系统已经对部分错误进行了拦截,所以在最终验证的时候,错误类型相对较少,与文档有效性无关的错误类型主要包括(见表1)。

通过文档有效性验证和与文档型有效性无关的语义错误处理,可以保证通过验证的 XML 知识库是可用的 XML 知识库,可以交付推理机推理应用。

5 结束语

本文所述的知识表示、知识获取、验证查错和知识推理都已经开发完成,并经过多次测试,但该套系统尚未投入实际使用,在实际应用中,根据使用者操作习惯的不同,可能会出现新的错误类型,可以据此逐步完善验证查错系统。

参考文献

1 Elliotte Rusty Harold 著,徐罡,黄涛译,Effective XML:有效使用 XML 的 50 种方法,北京:电子工业出版社,2005.

2 徐大庆,李淼,袁媛.基于 XML 的面向对象知识表示模式设计.计算机系统应用,2008,17(3):64-68.

3 JDOM Makes XML Easy:http://www.jdom.org/downloads/docs.html.

4 How do I validate against a schema when using JDOM: http://www.jdom.org/docs/faq.html.