基于 XML Schema 技术的设计模式定义方法

古 辉 张炜星
(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 在程序理解和逆向工程中,对软件设计模式的识别有助于软件技术人员从软件结构上理解系统的设计意图和实现功能。通常,采用 UML 类图表示的软件设计描述难以从特征上对设计模式进行准确的识别。提出一种基于 XML Schema 技术定义 XML 文档结构的设计模式定义语言——DPDLXS。通过运用 DPDLXS 语言描述特定设计模式的实例表明,该设计模式定义语言能够准确表述设计模式的特征,可以为设计模式的识别提供技术支持。

关键词 程序理解,DPDLXS,设计模式定义,设计模式识别
中图法分类号 TP311 **文献标识码** A

Method of Design Patterns Definition Based on XML Schema Technology

GU Hui ZHANG Wei-xing
(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract The identification of software design patterns can help software technical personnel understand the system's design intent and function from the software structural in program comprehension and reverse engineering. Generally, the software design information in the form of UML class diagram representation is hard to identify design patterns accurately from the pattern feature. This paper proposed a design patterns definition language can based on XML Schema—DPDLXS. The representation of specific design patterns instance by using DPDLXS language shows that the language can portray the feature of design patterns accurately, and provide a technical support for the identification of design patterns.

Keywords Program comprehension, DPDLXS, Design patterns definition, Design patterns identification

1 引言

设计模式在软件系统设计中用来解决一个重复发生的问题,由此增强应用程序的可复用性、可维护性、可理解性、可演化性和健壮性[1]。在程序理解和逆向工程中,对设计模式的重定义和识别可以帮助软件技术人员从结构上分析和理解系统的设计意图和实现功能。自从设计模式被广泛应用于软件工程中以来,国外的研究人员就开始从软件逆向工程中程序理解的角度出发,将设计模式从已有的源代码中识别出来。

近几年来,国内外对软件逆向工程中设计模式识别的研究也越来越重视。文献[2]通过分析特定设计模式中每个类之间的关系,建立对应的属性模型并且以关系演算语言将设计模式表示为逻辑语言表达式,通过计算表达式中逻辑语句的可满足性来检查是否运用了该设计模式。文献[3]将设计模式和源代码都表示成一种称为抽象对象语言(AOL)的中间表示方式,创建 AOL 设计模式知识库。然后通过解析 AOL 知识库得到抽象语法树(AST),从而抽取到设计模式类度量信息。文献[4-6]研究了一种特定的设计模式描述方法,定义了源码信息模型及其简化方法,同时还给出了设计模式模型和源代码模型的匹配方法。文献[1]提出了一种由可复用特征类型组成的可变模式定义方法。该定义方法用来描述特定的设计模式和源代码信息,通过适合特定特征的搜索技术检测特征类型来识别设计模式。其中,多重搜索技术包括在源代码模型组成的知识库中进行的 SQL 查询、识别源代码中类之间聚合和合成等关系的特定源代码解析、用来抽取表达式中注释信息的正则表达式匹配。以上文献都没有给出用来表示设计模式的标准定义,只是简单地描述每个类角色的属性和方法的数目,难以准确表述特定设计模式的特征信息。

设计模式的特征约束不仅表现在类属性和方法等静态结构信息上,同时还强调了类之间关系以及方法之间的调用等动态信息。因此,对设计模式识别初期的信息抽取还要增加对软件设计动态信息的抽取。文献[7]在进行设计模式的识别时,综合了模式静态结构信息和模式方法调用的动态信息,将设计模式的识别转化为在 XML 文档中进行 XML 查询,但文中没有进一步描述如何用 XML 表示设计模式。文献[8]提出了一种描述类之间继承、关联、聚合、合成等结构信息以及方法之间的调用、创建和重载信息的方法,并提出一种基于 XML 的设计模式标记语言(DPML)。类似地,文献[9]将所有的设计模式描述为一种基于 XML 的语言——DPXML。这种 DPML 设计模式描述语言[8]是基于 DTD 文档结构定义语言定义的,相对于 XML Schema 来说在建立复杂的可重用内容模型等方面的功能不够完善;这种 DPML 设计模式描述

到稿日期:2013-03-05 返修日期:2013-05-20

古 辉(1956—),男,教授,主要研究方向为软件理解与文档自动生成技术、嵌入式应用技术等,E-mail:gh@zjut.edu.cn;张炜星(1988—),男,硕士生,主要研究方向为程序理解技术与软件逆向工程。

语言没有体现出设计模式信息从静态结构信息到类之间的关系以及方法间的调用等动态信息的层次结构，只是以单个类为单元描述设计模式的信息，对设计模式的识别没有起到有效地支持。文献[10]在 DPML 的基础上增加半动态和动态分析得到的特征(DPMLd)来定义和表示设计模式，对文献[8]有所改进。

综上所述，上述用来提供设计模式识别的设计模式表示方法存在以下两个方面的不足：(1)没有给出准确描述特定设计模式特征的表示方法，对设计模式表示方法的规则定义不够完善；(2)没有强调对设计模式动态信息的表述，对设计模式的识别准确率产生影响。因此，本文提出一种基于 XML Schema 技术定义 XML 文档结构的设计模式定义语言——Design Pattern Definition Language based on XML Schema (DPDLXS)。通过利用 DPDLXS 语言能够准确表述设计模式特征，并且按设计模式静态结构信息和类之间关系以及方法间的调用等动态信息的层次结构来表示设计模式特征信息。

2 XML Schema

可扩展标记语言 (Extensible Markup Language，XML) 可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。而 XML Schema 技术是一种表达 XML 文档规则的语言。XML 文档结构定义语言主要有两种，分别是 Document Type Definitions(DTD)和 XML Schema Definitions(XSD)[11]。DTD 定义了 XML 文档的合法构建模块，使用一系列合法的元素来定义文档的结构；而 XML Schema 可以定义出现在 XML 文档中的元素、属性，定义哪个元素是子元素，定义子元素的次序、数目，定义元素和属性的数据类型、默认值以及固定值。基于 XSD 的 XML 文档结构定义模型主要的元素和属性等的定义规则如下：[12]

(1)〈schema〉定义规则：〈schema〉元素是 XML Schema 的一个根元素，用来声明名称空间信息和文档中声明的默认值。XML Schema 的名称空间有 3 种形式，分别是：〈Schema xmlns=…〉、〈xs: Schema xmlns: xs=…〉和〈xsd: Schema xmlns: xsd=…〉。其中，本文所定义的 XML 文档结构模型采用 xs 前缀的名称空间形式。

(2)〈element〉元素定义规则：〈element〉元素可以分为全局声明类型和局部声明类型，全局元素是作为〈schema〉根元素的直接孩子而局部元素不是〈schema〉根元素的直接孩子。我们可以对存在相同子元素的复合元素建立可重用的全局类型元素，该全局类型元素可以定义为一个简单元素，这样可以减小元素定义的复杂度。通过设置元素中 minOccurs 属性和 maxOccurs 属性值可以控制元素出现的次数。

(3)〈attribute〉属性定义规则：〈attribute〉属性定义和元素定义的规则类似，也可以分为全局类型和局部类型，同时也可以进行可重用的全局类型属性定义。我们可以通过设置属性中的 use 属性值来确定属性的出现方式：required(必需的)、optional(可选的)或 prohibited(禁用的)，其中默认为可选。

3 设计模式定义语言—DPDLXS

DPDLXS 语言的结构定义标准遵循基于 XSD 的 XML 文档结构定义模型规则，并结合了设计模式特征表述的要求，以表述设计模式静态结构信息和类之间关系以及方法间的调用等动态信息的层次结构。该语言将设计模式的信息表示分为 3 个部分，其中包括描述设计模式中参与角色的类静态结构信息的〈Roles〉、描述类之间关系等动态信息的〈Relations〉和描述自定义类型的〈TypeRep〉。

3.1 设计模式类角色静态结构信息定义标准

每个设计模式都是系统整体设计结构中的一个微结构，其中包括充当该设计模式角色的类结构信息以及类角色之间的关系。因此，一个设计模式的静态结构信息是由一个一个类角色信息组成的。我们用〈Role〉元素来代表设计模式中的一个类角色信息，分别用〈Operation〉和〈Attribute〉子元素来代表类角色中的操作和属性。

3.1.1 类角色中操作〈Operation〉复合元素定义标准

用来描述设计模式类角色操作或方法信息的〈Operation〉复合元素节点包含了代表方法 ID、方法名、是否抽象、方法参数类型以及返回类型等静态信息的子元素节点，同时包含了代表方法重写、方法调用等动态信息的子元素节点。利用 XML Schema 可重用的全局类型定义方法对〈Operation〉复合元素节点的定义如下：

```
〈xs:element name="Operation"〉
  〈xs:complexType〉
    〈xs:sequence〉
      〈xs:element ref="Override" minOccurs="0"/〉
      〈xs:element ref="Calls" minOccurs="0" maxOccurs="un-
       bounded"/〉
      〈xs:element ref="Create" minOccurs="0"/〉
      〈xs:element ref="hasParameterType" minOccurs="0"/〉
      〈xs:element ref="hasReturnType "/〉
    〈/xs:sequence〉
    〈xs:attribute ref="id" use="required" /〉
    〈xs:attribute ref="name" use="required" /〉
    〈xs:attribute ref="kind" use="required" /〉
    〈xs:attribute ref="isAbstract" use="required" /〉
  〈/xs:complexType〉
〈/xs:element〉
```

3.1.2 类角色中属性〈Attribute〉复合元素定义标准

描述设计模式类角色属性信息的子元素节点主要包括属性 ID、属性名称和属性类型，其定义标准如下：

```
〈xs:element name="Attribute"〉
  〈xs:complexType〉
    〈xs:sequence〉
      〈xs:element ref="hasType" /〉
    〈/xs:sequence〉
    〈xs:attribute ref="id" use="required" /〉
    〈xs:attribute ref="name" use="required" /〉
  〈/xs:complexType〉
〈/xs:element〉
```

除此之外，类角色信息还包括 id、表示类名的 name、表示是否为抽象类的 isAbstract 和表示充当该类角色次数的 occurNum 等属性，限于篇幅，此处不再给出具体标准。

3.2 设计模式类之间关系动态信息定义标准

根据统一建模语言 UML(Unified Modeling Language)提供的类图中的关系，类与类之间的关系可以分为[13]：一般化关系(Generalization)、关联关系(Association)、聚合关系(Aggregation)、合成关系(Composition)和依赖关系(Depen-

dency)。一般化关系表示一个类派生自另一个类，是类与类之间的继承关系；关联关系表示一个类知道另一个类的属性和方法，即一个类拥有另一个类类型的数据成员；聚合关系是一种强的关联关系，在逻辑上两个类之间是整体和个体之间的关系；合成关系是一种比聚合关系更强的关联关系，在逻辑上表现为代表整体的类负责代表部分的类的生命周期；依赖关系表示一个类依赖于另一个类的定义，即一个类方法返回的类型或参数调用的类型是另一个类类型。

每种关系类型〈RelationType〉元素都包含了 id、代表关系源类的 source 和代表关系目标类的 target 属性，其定义标准如下：

```
〈xs:complexType name="RelationType"〉
  〈xs:attribute ref="id" use="required"/〉
  〈xs:attribute ref="source" use="required"/〉
  〈xs:attribute ref="target" use="required"/〉
〈/xs:complexType〉
```

利用可重用的全局类型定义方法，可以将〈RelationType〉子元素引用到每一个类关系中。最后，代表该设计模式中所有类之间关系的〈Relations〉元素定义标准如下：

```
〈xs:element name="Relations"〉
  〈xs:complexType〉
    〈xs:sequence〉
      〈xs:element ref="Generalization" minOccurs="0" maxOccurs="unbounded" /〉
      〈xs:element ref="Association" minOccurs="0" maxOccurs="unbounded" /〉
      〈xs:element ref="Aggregation" minOccurs="0" maxOccurs="unbounded" /〉
      〈xs:element ref="Composition" minOccurs="0" maxOccurs="unbounded" /〉
      〈xs:element ref="Dependency" minOccurs="0" maxOccurs="unbounded" /〉
    〈/xs:sequence〉
  〈/xs:complexType〉
〈/xs:element〉
```

3.3 自定义类型定义标准

在利用 DPDLXS 语言进行设计模式特征描述时，需要限定方法的参数类型或返回类型以及属性的类型，才能准确地表达特定设计模式的特征。为此，我们设计自定义类型来限定前面部分中涉及到的类型，该类型可以是设计模式中出现的类类型也可以是内置数据类型，并且用〈ClassType〉元素来表示类类型，其定义标准如下：

```
〈xs:element name="TypeRep" 〉
  〈xs:complexType〉
    〈xs:sequence〉
      〈xs:element ref="ClassType" minOccurs="0" /〉
      〈/xs:sequence〉
    〈xs:attribute ref="id" use="required"/〉
  〈/xs:complexType〉
〈/xs:element〉
```

根据以上 3 个部分的定义标准，可以给出 DPDLXS 语言根元素〈DesignPattern〉的定义标准：

```
〈xs:element name="DesignPattern"〉
  〈xs:complexType〉
    〈xs:sequence〉
      〈xs:element ref="Roles" /〉
      〈xs:element ref="Relations" /〉
      〈xs:element ref="TypeRep" minOccurs="0" maxOccurs="unbounded" /〉
    〈/xs:sequence〉
  〈/xs:complexType〉
〈/xs:element〉
```

4 基于 DPDLXS 语言的设计模式表示

通过第 3 节中给出的 DPDLXS 语言的定义标准，结合设计模式中工厂方法模式和代理模式[14]的特征描述，我们将工厂方法模式和代理模式用本文提出的 DPDLXS 语言进行表示。

4.1 工厂方法模式

根据文献[11]对设计模式的概括，可得出工厂方法模式的特征：

(1)一个抽象类 Creator 和抽象类 Product；

(2)一个 ConcreteCreator 类继承自 Creator 类，一个 ConcreteProduct 类继承自 Product；

(3)Creator 类和 ConcreteCreator 类至少有一个共同的方法 FactoryMethod；

(4)每个 FactoryMethod 方法都创建一个 ConcreteProduct 对象，并返回 Product 类型的 ConcreteProduct 对象。

结合工厂方法模式类图(见图 1)，我们给出如下基于 DPDLXS 语言定义的 XML 文档格式的工厂方法模式表示形式：

```
〈DesignPattern id="DP1" name="FactoryMethod"〉
  〈Roles〉
    〈Role id="R1" name="Procudt" isAbstract="true" occurNum="1"〉
    〈/Role〉
    〈Role id="R2" name="ConcreteProcudt" occurNum="*"〉
    〈/Role〉
    〈Role id="R3" name="Creator" isAbstract="true" occurNum="1"〉
      〈Operation id="031" name="FactoryMethod" kind="normal" isAbstract="true"〉
        〈hasReturnType〉T31〈/hasReturnType〉
      〈/Operation〉
    〈/Role〉
    〈Role id="R4" name="ConcreteCreator" occurNum="*"〉
      〈Operation id="041" name="FactoryMethod" kind="normal" isAbstract="true"〉
        〈Override〉031〈/Override〉
        〈Create〉R2〈/Create〉
        〈hasReturnType〉T31〈/hasReturnType〉
      〈/Operation〉
    〈/Role〉
  〈/Roles〉
  〈Relations〉
    〈Generalization id="RE1" source="R2" target="R1"/〉
    〈Generalization id="RE2" source="R4" target="R3"/〉
    〈Dependency id="RE3" source="R4" target="R2"/〉
  〈/Relations〉
  〈TypeRep id="T31"〉
    〈ClassType〉R1〈/ClassType〉
  〈/TypeRep〉
〈/DesignPattern〉
```

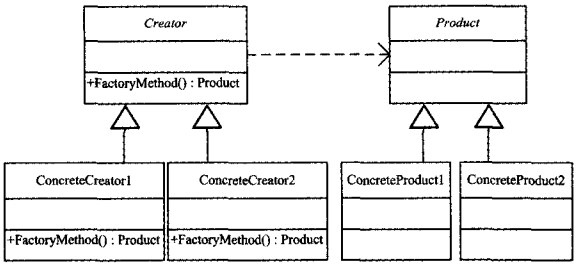

图 1 工厂方法模式类图[13]

4.2 代理模式

代理模式的特征描述如下：

(1)一个 Subject 抽象类有抽象方法 Request；

(2)一个 RealSubject 类和 Proxy 类都继承自 Subject 类；

(3)RealSubject 类、Proxy 类和 Subject 类有共同的方法 Request；

(4)Proxy 类有 RealSubject 类型的数据成员，即 Proxy 类和 RealSubject 类存在关联关系；

(5)Proxy 类的 Request 方法调用 RealSubject 类的 Request 方法 。

结合代理模式类图(见图 2)，我们给出如下基于 DPDLXS 语言定义的 XML 文档格式的代理模式表示形式：

```
〈DesignPattern id="DP2" name="Proxy"〉
    〈Roles〉
        〈Role id="R1" name="Subject" isAbstract="true" occurNum="1"〉
            〈Operation id="011" name="Request" isAbstract="true"〉
                〈hasReturnType〉T11〈/hasReturnType〉
            〈/Operation〉
        〈/Role〉
        〈Role id="R2" name="ProxySubject" occurNum="1"〉
            〈Operation id="021" name="Request" isAbstract="false"〉
                〈Override〉011〈/Override〉
                〈class〉031〈/class〉
                〈hasReturnType〉T11〈/hasReturnType〉
            〈/Operation〉
            〈Attribute id="A21" name="realSubject"〉
                〈hasType〉T31〈/hasType〉
            〈/Attribute〉
        〈/Role〉
        〈Role id="R3" name="RealSubject" occurNum="1"〉
            〈Operation id="031" name="Request" isAbstract="false"〉
                〈Override〉011〈/Override〉
                〈hasReturnType〉T11〈/hasReturnType〉
            〈/Operation〉
        〈/Role〉
    〈/Roles〉
    〈Relations〉
        〈Generalization id="RE1" source="R2" target="R1"/〉
        〈Generalization id="RE1" source="R3" target="R1"/〉
        〈Association id="RE3" source="R2" target="R3"/〉
    〈/Relations〉
    〈TypeRep id="T11"/〉
    〈TypeRep id="T31"/〉
        〈ClassType〉R3〈/ClassType〉
    〈/TypeRep〉
〈/DesignPattern〉
```

代理模式的结构类图如图 2 所示。

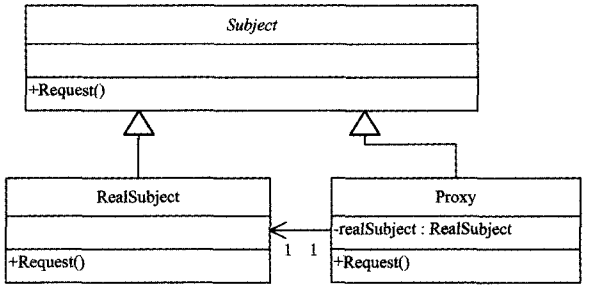

图 2 代理模式类图[13]

结束语 本文提出的基于 XML Schema 技术定义 XML 文档结构的设计模式定义语言——DPDLXS，能够针对设计模式静态结构信息和动态信息进行分层次结构表示，亦可以通过定义语法结构规则对设计模式的特征进行准确描述。将 DPDLXS 语言用于模式表示的实例表明，该语言能够准确描述设计模式的特征，可以用于支持程序理解中设计模式的识别。

参 考 文 献

[1] Rasool G, Mader P. Flexible Design Pattern Detection Based on Feature Types[C]// 26th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2011: 243-252

[2] 苗康，余啸，赵吉，等. 基于关系演算的 Java 模式识别[J]. 计算机应用研究，2010，27(9)：3425-3430

[3] Antoniol G, Fiutem R, Cristoforetti L. Design pattern recovery in object-oriented software[C]// Proceedings of the 6th IEEE International Workshop on Program Comprehension (IWPC 1998). 1998: 153-160

[4] 冯铁，李文锦，张家晨. 面向 Java 语言的设计模式抽取方法的研究[J]. 计算机工程与应用，2005，41(25)：28-33

[5] 冯铁，李文锦，张家晨. 从源码中抽取设计模式技术研究综述[J]. 计算机应用研究，2005(8)：6-9

[6] 李文锦. 基于设计模式的软件设计恢复方法研究[J]. 计算机与现代化，2007(8)：89-92

[7] Bouassida N, Ben-Abdallah H. Structural and Behavioral Detection of Design Patterns[J]. Springer Advances in Software Engineering, 2009, 59: 16-24

[8] Balanyi Z, Ferenc R. Mining Design Patterns from C++ Source Code[C]// Proc. Int'l Conf. Software Maintenance, (ICSM '03). 2003: 305-314

[9] 肖卓宇. 基于设计模式的逆向工程研究[D]. 长沙：长沙理工大学，2008

[10] Dobis M, Majtas L. Mining Design Patterns from Existing Projects Using Static and Run-Time Analysis[J]. Springer Software Engineering Techniques, 2011, 4980: 62-75

[11] W3C. Schema[OL]. http://www.w3.org/standards/xml/schema, 2013-1-20

[12] Hunter D, Rafter J, Fawcett J, 等. XML 入门经典(第 4 版)[M]. 吴文国，译. 北京：清华大学出版社，2009

[13] 阎宏. Java 与模式[M]. 北京：电子工业出版社，2002

[14] Gamma E, Helm R, Johnson R, et al. Design Patterns-Elements of Reusable Object-Oriented Software[M]. New Jersey: Addison-Wesley, 1995