(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101192141 B
(45)授权公告日 2010.05.12

(21)申请号 200610114649.1

(22)申请日 2006.11.20

(73)专利权人 北京书生国际信息技术有限公司
地址 100083 北京市海淀区学院路 35 号世宁大厦 13 层

(72)发明人 王东临 邹开红

(74)专利代理机构 北京银龙知识产权代理有限公司 11243
代理人 许静

(51)Int.Cl.
G06F 9/44(2006.01)
G06F 17/30(2006.01)

(56)对比文件
CN 1783090 A,2006.06.07, 全文.
US 2004/0015840 A1,2004.01.22, 说明书第 22 至 35 段、第 94 至 98 段、第 112 至 118 段、附图 5.
WO 2005/008484 A1,2004.07.09, 全文.
US 6480865 B1,2002.11.12, 说明书第 5 栏第 35 行至第 11 栏第 25 行.

审查员 孙娟

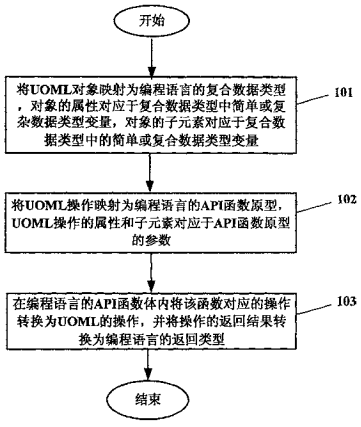
权利要求书 1 页 说明书 6 页 附图 4 页

(54)发明名称

一种将 UOML 封装成应用程序编程接口的方法

(57)摘要

本发明公开了一种将 UOML 封装成 API 的方法,包括如下步骤:将 UOML 对象映射为编程语言的复合数据类型,UOML 对象的属性对应于复合数据类型中简单数据类型变量或复杂数据类型变量,UOML 对象的子元素对应于复合数据类型中的简单或复合数据类型变量;将 UOML 操作映射为编程语言的 API 函数原型,UOML 操作的对象对应于 API 函数原型的参数;根据上述步骤中定义的映射,在编程语言的 API 函数体内将该函数对应的操作转换为 UOML 的操作,并将操作的返回结果转换为编程语言的返回类型。通过上述的方法实现 UOML 针对不同编程语言的 API 函数,从而不同编程语言的应用开发者可以直接通过调用对应的 API 函数来实现文档库系统上的应用,提高了开发效率。



CN 101192141 B

1. 一种将 UOML 封装成编程语言的应用程序编程接口 API 的方法，其特征在于，包括如下步骤：

将 UOML 操作的对象映射为编程语言的复合数据类型，所述 UOML 操作的对象的属性和/或子元素对应于复合数据类型中简单数据类型变量或复杂数据类型变量；

将 UOML 操作映射为编程语言的 API 函数原型，其中，UOML 操作的属性和子元素对应于 API 函数原型的参数；

当调用一个 API 函数时，根据 API 函数原型和 UOML 操作的映射关系，API 函数生成 UOML 操作命令，并将生成的 UOML 操作命令发送给文档库系统；

在文档库系统执行所述 UOML 操作命令后，所述 API 函数将来自所述文档库系统的返回结果转换为符合编程语言返回类型的返回值。

2. 如权利要求 1 所述的方法，其特征在于，当所述编程语言支持函数的多态性定义时，将多个对象的 UOML 操作映射为编程语言的一个 API 函数原型，并通过所述 API 函数原型的参数的类型来区分对不同对象的操作。

3. 如权利要求 1 所述的方法，其特征在于，所述编程语言为面向对象的编程语言，所述的编程语言中的复合数据类型为类，所述的编程语言的 API 函数原型为类的成员函数，所述 UOML 操作的属性和子元素对应于类成员函数的参数。

4. 如权利要求 3 所述的方法，其特征在于，所述的面向对象的编程语言包括 C++、Object-C、Delphi、Java、Python 或 Ruby。

5. 如权利要求 1 所述的方法，其特征在于，所述编程语言为非面向对象的编程语言，所述的编程语言中的复合数据类型为结构；所述的编程语言的 API 函数原型为全局函数原型，所述 UOML 操作的属性和子元素对应于所述全局函数原型的参数。

6. 如权利要求 5 所述的方法，其特征在于，所述的非面向对象的编程语言包括 C、TCL、Pascal 或 Perl。

7. 如权利要求 3 或 5 所述的方法，其特征在于，所述编程语言具有和 C 或者 C++ 的接口，所述 API 函数通过与 C 或者 C++ 的接口调用 C++ 或 C 语言封装的 API 函数来生成所述 UOML 操作命令。

8. 如权利要求 7 所述的方法，其特征在于，所述的编程语言包括 Java、TCL、Perl、Python 或 Ruby。

一种将 UOML 封装成应用程序编程接口的方法

技术领域

[0001] 本发明涉及电子文档处理技术，特别涉及一种将 UOML 封装成应用程序编程接口的方法。

背景技术

[0002] UOML 规范是一系列以“动作 + 对象”格式定义的、通过 XML 语言描述的文档库系统命令。本申请人在中国专利申请号为 CN200510131641.1 的专利申请说明书中，对其进行了详细的说明。由于 XML 具有跨平台、跨语言的能力，所以，UOML 规范的提出，解决了文档库系统命令本身的跨平台、跨语言交换的问题。但是，在实际应用过程中，对文档库的操作一般通过编程语言实现的代码控制，因此，在代码中需要实现对 UOML XML 文本的解析和处理。如果每个应用开发人员在自己的代码中实现对 UOML XML 文本的解析和处理，工作量大且开发效率低下。

发明内容

[0003] 本发明要解决的一个技术问题是提供一种方法，通过该方法可以将 UOML 封装成不同编程语言的应用程序编程接口 (API)，以提高文档库系统应用开发者的开发效率。

[0004] 本发明提供的将 UOML 封装成编程语言的 API 的方法，包括如下步骤 ：

[0005] 将 UOML 操作的对象映射为编程语言的复合数据类型 ；所述 UOML 操作的对象的属性和 / 或子元素对应于复合数据类型中简单数据类型变量或复杂数据类型变量 ；

[0006] 将 UOML 操作映射为编程语言的 API 函数原型，其中，UOML 操作的属性和子元素对应于 API 函数原型的参数 ；

[0007] 当调用一个 API 函数时，根据 API 函数原型和 UOML 操作的映射关系，API 函数生成 UOML 操作命令，并将生成的 UOML 操作命令发送给文档库系统 ；

[0008] 在文档库系统执行所述 UOML 操作命令后，所述 API 函数将来自所述文档库系统的返回结果转换为符合编程语言返回类型的返回值。

[0009] 上述的方法，其中，当所述编程语言支持函数的多态性定义时，将多个对象的 UOML 操作映射为编程语言的一个 API 函数原型，并通过所述 API 函数原型的参数的类型来区分对不同对象的操作。

[0010] 对于面向对象的编程语言，所述的编程语言中的复合数据类型为类，所述的编程语言的 API 函数原型为类的成员函数，所述 UOML 操作的属性和子元素对应于类成员函数的参数。面向对象的编程语言包括 C++、Object-C、Delphi、Java、Python 或 Ruby。

[0011] 对于非面向对象的编程语言，所述的编程语言中的复合数据类型为结构 ；所述的编程语言的 API 函数原型为全局函数原型，所述 UOML 操作的属性和子元素对应于所述全局函数原型的参数。非面向对象的编程语言包括 C、TCL、Pascal 或 Perl。

[0012] 进一步，对于具有和 C++ 和 C 语言接口的编程语言，所述 API 函数通过与 C 或者 C++ 的接口调用 C++，C 语言封装的 API 函数来生成所述 UOML 操作命令。

[0013] 通过本发明提供的方法，可以实现将 UOML 封装成编程语言的 API。开发人员在一编程语言上开发对文档库系统的应用时，可以直接调用生成的封装好的该编程语言的 API，从而省去了大量的解析 UOML 的工作量，提高了开发的效率。

附图说明

[0014] 图 1 所示为本发明的将 UOML 封装成编程语言 API 的方法流程图；

[0015] 图 2 所示为本发明的将 UOML 封装成面向对象语言 API 的方法流程图；

[0016] 图 3 所示为本发明的将 UOML 封装成非面向对象语言 API 的方法流程图；

[0017] 图 4 所示为本发明的编程语言的 API 封装层次示意图。

具体实施方式

[0018] 本发明的基本思想是，利用 UOML 规范的“动作 + 对象”的格式特点，以及 UOML 对象和编程语言中的类或结构的内在对应关系，提供一种可以将 UOML 封装为不同的编程语言的应用程序编程接口 (API) 的方法，从而利用不同编程语言对文档库系统进行应用开发时可以直接调用该语言为 UOML 提供的 API，提高开发人员的开发效率。

[0019] 如图 1 所示，本发明的将 UOML 封装成编程语言 API 的方法包括如下步骤：

[0020] 步骤 101，将 UOML 对象映射为编程语言的复合数据类型，UOML 对象的属性对应于复合数据类型中简单数据类型变量或复杂数据类型变量，UOML 对象的子元素对应于复合数据类型中简单或复合数据类型变量。

[0021] 首先在编程语言中提供 UOML 对象的对应表示。利用编程语言的复合数据类型的用户自定义功能，将每个 UOML 对象映射到一个定义的复合数据类型。复合数据类型可以是面向对象的语言中的类，或者是非面向对象的语言中的结构。其中，UOML 对象中的属性在复合数据类型中可以通过简单数据类型的变量或者复杂数据类型的变量来表示，如整型 (INT)、字符类型 (CHAR) 或者浮点类型 (FLOAT) 的变量，或者相应的数组等。UOML 对象的子元素通常属于 UOML 中的对象，在编程语言中定义了相应的复合数据类型，在这种情况下，在复合数据类型中，UOML 对象的子元素通过该子元素对应的复合数据类型的变量来表示。UOML 对象的一些不是 UOML 对象的子元素也可以通过简单数据类型的变量来表示。需要指出的是，上述的简单类型的变量或者复杂数据类型的变量以及复合数据类型的变量包含指针变量的情形。

[0022] 步骤 102，将 UOML 操作映射为编程语言的 API 函数原型，UOML 操作的属性和子元素对应于 API 函数原型的参数。

[0023] 由于 UOML 规范中的操作可能支持多个对象，在将 UOML 操作转换为编程语言的 API 函数原型时，需要考虑该编程语言是否支持函数的多态性定义。如果编程语言不支持函数的多态性定义，需要为每个操作支持的对象定义一个 API 函数模型，如果支持，则可以只定义一个 API 函数原型，通过函数原型中参数的类型来区分对不同对象的操作。UOML 操作的属性和子元素对应于 API 函数原型的参数。对于面向对象的语言，可以将 API 函数定义为类的成员函数，在这种情况下，如果操作的属性或者子元素同时是该类的成员变量，则在成员函数原型中可以不包含这些参数，而在函数体中直接引用这些变量。当然，也可以和非面向对象的语言一样，把操作所有的属性和子元素都作为函数原型的参数。

[0024] 步骤 103，根据函数原型对应的操作和函数原型的参数，在编程语言的 API 函数体内实现函数功能向 UOML 操作的转换和执行，并将操作的返回结果转换为编程语言的返回类型。

[0025] 在编程语言的 API 函数体内，根据函数原型和 UOML 操作的映射关系，以及函数参数和 UOML 操作的属性和子元素的对应关系，生成 UOML 的操作命令。将生成的操作命令发送给文档库系统，文档库系统执行后会返回执行结果。由于文档库系统返回的执行结果是 UOML 格式，所以需要将其转换为编程语言中对应的返回类型。

[0026] 通常编程语言可以分为面向对象的语言和非面向对象的语言。由于编程语言本身的特点不同，所以对不同的编程语言，将 UOML 封装成编程语言 API 的具体实现也有一些不同。

[0027] 下面以 C++ 为例，介绍 UOML 封装成面向对象的语言的 API 的方法。

[0028] 如图 2 所示，为本发明提供的一种将 UOML 封装成面向对象语言 API 的方法流程图。包括如下步骤：

[0029] 步骤 201，将 UOML 对象映射为面向对象语言的类，对象的属性对应于类中简单数据类型变量或复杂数据类型变量，对象的子元素对应于类中的类变量或简单数据类型变量。

[0030] 以 UOML 文档对象为例，首先在 C++ 语言中定义一个对应的类 UOML_Doc。该类可以从 C++ 的所有类的基类 CObject 派生，也可以没有派生的基类。还有一种实现是，首先为所有的 UOML 对象定义一个基类：

[0031]
```
class UOML_Obj
```

[0032]
```
{
```

[0033]
```
public：
```

[0034]
```
  virtual ～UOML_Obj()；
```

[0035]
```
  void Init(UOML_Obj_Type type)；
```

[0036]
```
  UOML_Obj Clone(bool bMaintainRef = true)；
```

[0037]
```
}；
```

[0038] 然后在基类 UOML_Obj 上派生 UOML_Doc。

[0039] 为 UOML 文档对象定义对应类 UOML_Doc 之后，将 UOML 文档对象的子元素映射为 UOML_Doc 类的对应的成员变量。UOML 文档对象包含 metadata（元数据），pageset（各页面）fontinfo（嵌入字库）等子元素。这些子元素分别有对应的类：metadata 用 UOML_Meta 类表示，pageset 用 UOML_Page 类表示，fontinfo 用 UOML_FontInfo 类表示。所以 UOML 文档对象的这些子元素在 UOML_Doc 类定义中有对应的成员变量：UOML_Metametadata；UOML_Page*pageset；UOML_FontInfo fontinfo。其中，pageset 子元素还可以通过 UOML_Page 的数组来表示。由于 UOML 文档对象不包含属性，所以没有属性向成员变量的映射。

[0040] 同理，对于 UOML 页对象，为其定义 UOML_page 类。将 UOML 页对象的 resolution，size，rotation，log 等属性，在 UOML_Page 类定义中用整型或浮点类型的成员变量表示，而 UOML 页对象的 GS，metadata 等子元素在 UOML_page 类中以相应类的成员变量表示。

[0041] 通过上述的方式，可以为 UOML 的所有对象在面向对象的语言中定义相应的类，包括文档库 (UOML_DOCBASE)、文档集 (UOML_DOCSET)、文档 (UOML_DOC)、页 (UOML_PAGE)、

层(UOML_LAYER)、对象组(UOML_OBJGROUP)、文字(UOML_TEXT)、图像(UOML_IMAGE)、直线(UOML_LINE)、曲线(UOML_BEIZER)、圆弧(UOML_ARC)、路径(UOML_PATH)、源文件(UOML_SRCFILE)、背景色(UOML_BACKCOLOR)、前景颜色(UOML_COLOR)、ROP(UOML_ROP)、字符尺寸(UOML_CHARSIZE)、字体(UOML_TYPEFACE)、角色(UOML_ROLE)、权限(UOML_PRIV)等对象。本技术领域的技术人员可以参照上面的说明来具体实现,在此不一一赘述。

[0042] 步骤202,将UOML操作映射为面向对象语言的类的成员函数,UOML操作的属性和子元素对应于类成员函数的参数。

[0043] 以UOML文档对象为例,作用于UOML文档对象的操作,如打开(UOML_OPEN)、关闭(UOML_CLOSE)等,在UOML_Doc类内定义相应的成员函数:Open()和Close()。操作UOML_OPEN的属性包括文档所在的位置信息,将作为成员函数Open()的参数:Open(char*szDocPath)。而UOML_CLOSE的子元素包括文档对象的句柄,也将作为成员函数Close()的参数。

[0044] 对于UOML的其它操作,如获取(UOML_GET)、设置(UOML_SET)、插入(UOML_INSERT)、删除(UOML_DELETE)、检索查询(UOML_QUERY)等也可以通过上述的方式对应到类的成员函数。

[0045] 步骤203,在成员函数体内将该函数对应的操作转换为UOML的操作,并将操作的返回结果转换为面向对象语言的返回类型。

[0046] 例如在UOML_Doc类的成员函数Open(char*szDocPath)的实体内,把相应的调用转为对文档库调用的UOML XML字符串,并发送到文档库:

[0047] <? xml version =" 1.0" encoding =" UTF-8" ? >

[0048] <UOML_OPEN create =" true" >

[0049] <path val = szDocPath/>

[0050] </UOML_OPEN>

[0051] 文档库收到请求后即执行操作。如果打开文档成功,即返回XML字符串。在Open()函数体内部,根据接到的字符串构建UOML_Doc对象,然后作为函数的返回值返回该对象。

[0052] 上面以C++语言为例对面向对象的语言进行UOML到API的封装方法,可以看出的是,上述方法对其它的一些面向对象的语言,如Object-C、Delphi、Java、Python、Ruby等同样适用。

[0053] 下面以编程语言C为例,介绍UOML封装成非面向对象语言的API的方法。

[0054] 如图3所示,为本发明提供的一种将UOML封装成非面向对象语言API的方法流程图。包括如下步骤:

[0055] 步骤301,将UOML对象映射为非面向对象语言的结构,对象的属性对应于结构中简单数据类型变量或复杂数据类型变量,对象的子元素对应于结构中结构变量或简单数据类型变量。

[0056] 以UOML文档对象为例,首先在C语言中定义一个对应的结构struct_UOML_Doc。

[0057] 为UOML文档对象定义对应结构struct_UOML_Doc之后,将UOML文档对象的子元素映射为结构struct_UOML_Doc的对应的成员变量。UOML文档对象包含metadata(元数据),pageset(各页面)fontinfo(嵌入字库)等子元素。这些子元素分别有对应的结构定义:metadata用结构struct_UOML_Meta表示,pageset用结构struct_UOML_Page表

示，fontinfo 用结构 struct_UOML_FontInfo 表示。所以 UOML 文档对象的这些子元素在结构 struct_UOML_Doc 定义中有对应的成员变量 :struct_UOML_Meta_metadata ;struct_UOML_Page*pageset ;struct_UOML_FontInfo fontinfo。其中，pageset 子元素还可以通过 struct_UOML_page 的数组来表示。由于 UOML 文档对象不包含属性，所以没有属性向成员变量的映射。

[0058] 同理，对于 UOML 页对象，为其定义结构 struct_UOML_Page。将 UOML 页对象的 resolution，size，rotation，log 等属性，在结构 struct_UOML_Page 定义中用整型或浮点类型的成员变量表示，而 UOML 页对象的 GS，metadata 等子元素在结构 struct_UOML_Page 中以相应的成员变量表示。

[0059] 通过上述的方式，可以为 UOML 的所有对象在非面向对象的语言中定义相应的结构，包括文档库 (UOML_DOCBASE)、文档集 (UOML_DOCSET)、文档 (UOML_DOC)、页 (UOML_PAGE)、层 (UOML_LAYER)、对象组 (UOML_OBJGROUP)、文字 (UOML_TEXT)、图像 (UOML_IMAGE)、直线 (UOML_LINE)、曲线 (UOML_BEIZER)、圆弧 (UOML_ARC)、路径 (UOML_PATH)、源文件 (UOML_SRCFILE)、背景色 (UOML_BACKCOLOR)、前景颜色 (UOML_COLOR)、ROP(UOML_ROP)、字符尺寸 (UOML_CHARSIZE)、字体 (UOML_TYPEFACE)、角色 (UOML_ROLE)、权限 (UOML_PRIV) 等对象。本技术领域的技术人员可以参照上面的说明来具体实现，在此不一一赘述。

[0060] 步骤 302，将 UOML 操作映射为非面向对象语言的全局函数原型，UOML 操作的属性和子元素对应于函数原型的参数。

[0061] 以 UOML 文档对象为例，对于作用于 UOML 文档对象的操作，如打开 (UOML_OPEN)、关闭 (UOML_CLOSE) 等，定义相应的 API 函数原型 :

[0062] struct_UOML_Doc*UOML_Doc_Open(char*szDocPath) 和

[0063] void UOML_Doc_Close(HANDLE*hDocHandle)。

[0064] 操作 UOML_OPEN 的属性包括文档所在的位置信息，将作为 API 函数 UOML_Doc_Open() 的参数 :szDocPath。而 UOML_CLOSE 的子元素包括文档对象的句柄，也将作为 API 函数 UOML_Doc_Close 的参数 :hDocHandle。

[0065] 对于 UOML 的其它操作，如获取 (UOML_GET)、设置 (UOML_SET)、插入 (UOML_INSERT)、删除 (UOML_DELETE)、检索查询 (UOML_QUERY) 等也可以通过上述的方式对应到 API 函数原型。

[0066] 步骤 303，在函数体内将该函数对应的操作转换为 UOML 的操作，并将操作的返回结果转换为非面向对象语言的返回类型。

[0067] 例如在 API 函数 UOML_Doc_Open(char*szDocPath) 的实体内，把相应的调用转为对文档库调用的 UOML XML 字符串，并发送到文档库 :

[0068] <? xml version =" 1.0" encoding =" UTF-8" ? >

[0069] <UOML_OPEN create =" true" >

[0070] <path val = szDocPath/>

[0071] </UOML_OPEN>

[0072] 文档库收到请求后即执行操作。如果打开文档成功，即返回 XML 字符串。在 UOML_Doc_Open() 函数体内部，根据接到的字符串构建 UOML_Doc 对象，然后作为函数的返回值返回该对象。

[0073] 上面以C语言为例对非面向对象的语言进行UOML到API的封装方法，可以看出的是，上述方法对其它的一些非面向对象的语言，如TCL、Pascal、Perl等同样适用。

[0074] 如图4所示，为本发明的编程语言的封装层次示意图。过程401表示的是将C/C++语言封装的API在对应函数体内直接转换为UOML XML操作命令，过程402表示的是将其它非C/C++的编程语言封装的API在对应函数体内直接转换为UOML XML操作命令。对于其它非C/C++的编程语言的API封装，除了在对应的API函数体内直接转换为UOML XML操作命令，还可以利用这些编程语言跟C++或C语言的接口，在该编程语言的函数体内，直接调用C++，C语言封装成的对应API函数，正如过程403所示。而实际向UOML XML操作命令的转换在C/C++封装的API函数体内完成。以编程语言Java为例，可以利用Java语言的本地接口规范(Java Native Interface，简称JNI)，在Java封装的API函数体内，绑定(binding)C/C++封装的对应API函数，而不需要进行直接的UOML XML转换。大多数脚本语言都提供了利用C++或C语言作扩展的机制，利用这种机制，就可以实现脚本语言的API对本地应用程序API函数的绑定。上述的脚本语言，除了Java，还可以包括TCL、Perl、Python和Ruby等语言。

[0075] 通过上述提供的方法，可以实现将UOML封装成编程语言的API。开发人员在一编程语言上开发对文档库系统的应用时，可以直接调用生成的该编程语言封装的的API，从而省去了大量的解析UOML的工作量，提高了开发的效率。

[0076] 以上所述，仅为本发明较佳的具体实施方式，但本发明的保护范围并不局限于此，任何熟悉本技术领域的技术人员在本发明揭露的技术范围内，可轻易想到的变化或替换，都应涵盖在本发明的保护范围之内。因此，本发明的保护范围应该以权利要求的保护范围为准。

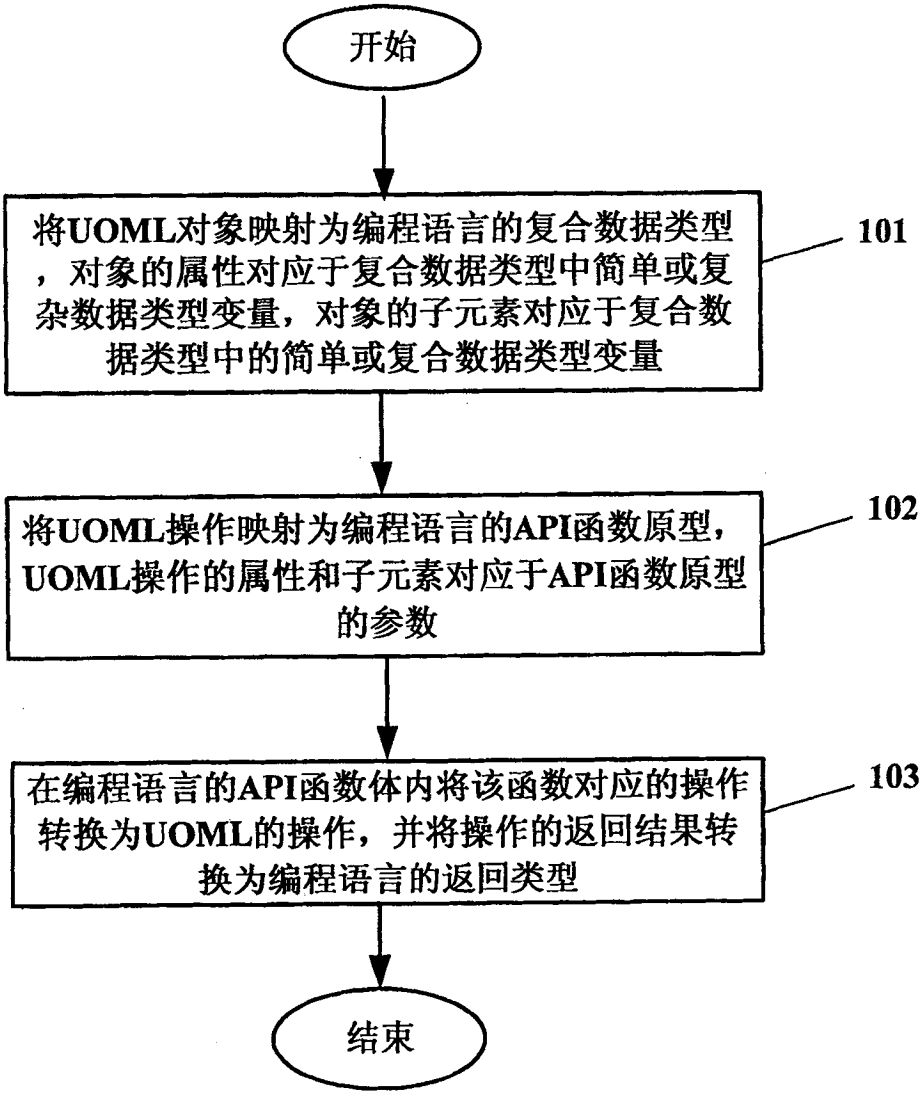

开始
将UOML对象映射为编程语言的复合数据类型，对象的属性对应于复合数据类型中简单或复杂数据类型变量，对象的子元素对应于复合数据类型中的简单或复合数据类型变量
101
将UOML操作映射为编程语言的API函数原型，UOML操作的属性和子元素对应于API函数原型的参数
102
在编程语言的API函数体内将该函数对应的操作转换为UOML的操作，并将操作的返回结果转换为编程语言的返回类型
103
结束

图 1

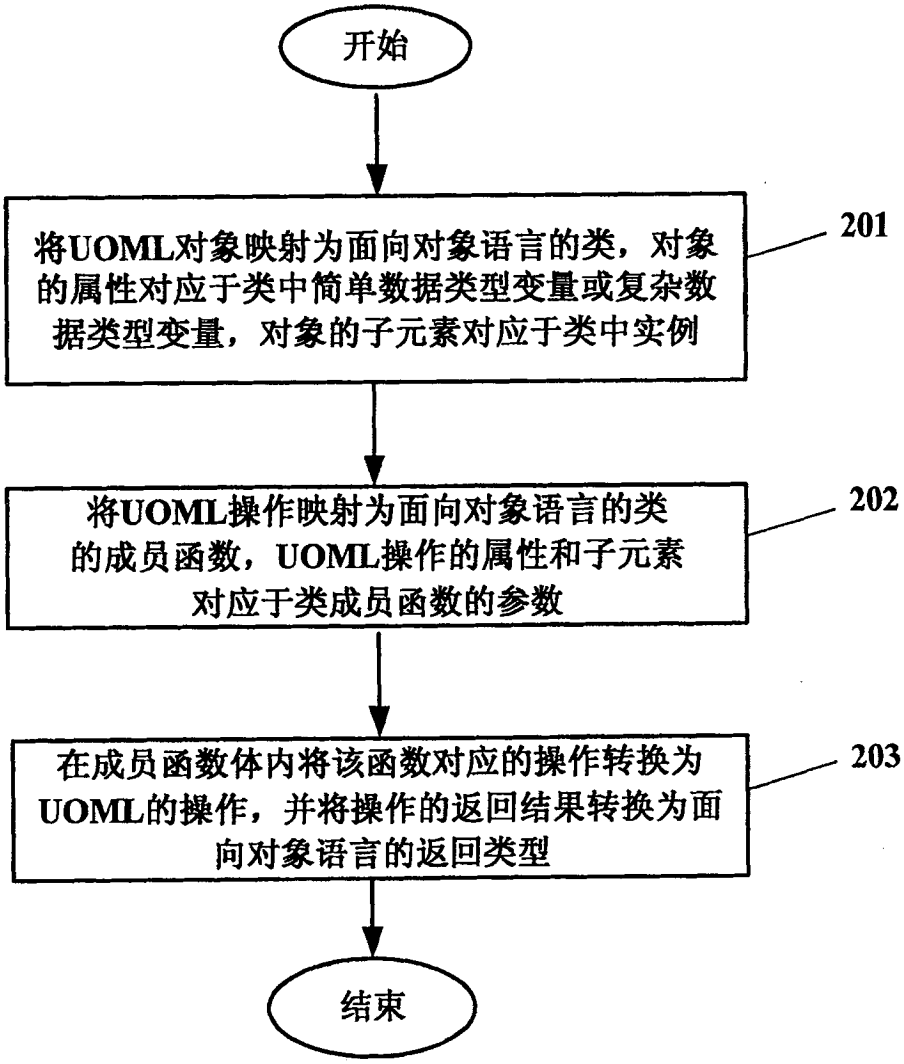

开始
将UOML对象映射为面向对象语言的类，对象的属性对应于类中简单数据类型变量或复杂数据类型变量，对象的子元素对应于类中实例
201
将UOML操作映射为面向对象语言的类的成员函数，UOML操作的属性和子元素对应于类成员函数的参数
202
在成员函数体内将该函数对应的操作转换为UOML的操作，并将操作的返回结果转换为面向对象语言的返回类型
203
结束

图 2

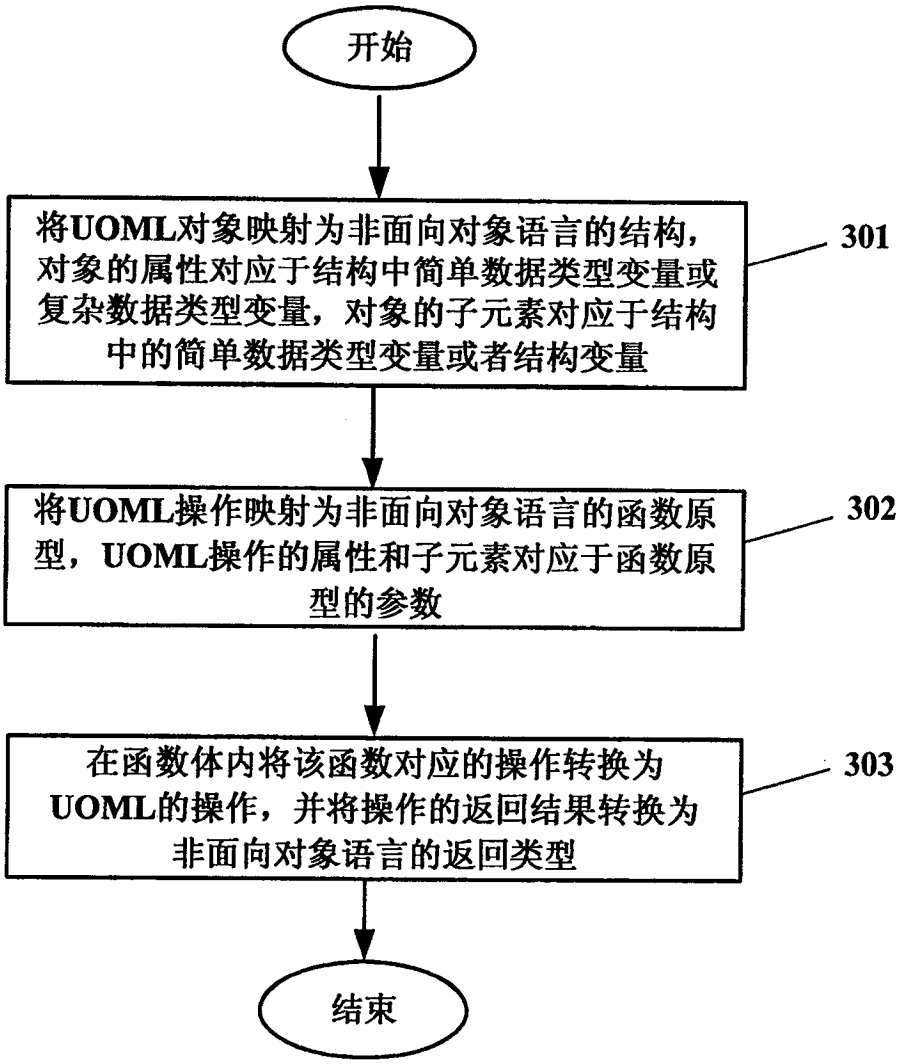

开始
将UOML对象映射为非面向对象语言的结构，对象的属性对应于结构中简单数据类型变量或复杂数据类型变量，对象的子元素对应于结构中的简单数据类型变量或者结构变量
301
将UOML操作映射为非面向对象语言的函数原型，UOML操作的属性和子元素对应于函数原型的参数
302
在函数体内将该函数对应的操作转换为UOML的操作，并将操作的返回结果转换为非面向对象语言的返回类型
303
结束

图 3

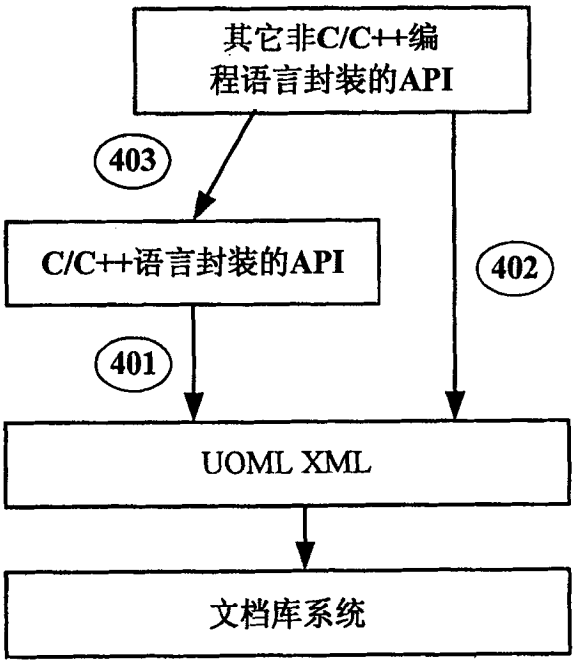


图 4