基于 XML 的构架描述语言 XBA 的研究

赵文耘,张 志
(复旦大学计算机与信息技术系,上海 200433)

摘 要: 本文通过对现有构架描述语言进行分析,指出其存在的不足,并提出了一种基于 XML 的构架描述语言 - XBA.它把 XML 应用于软件构架的描述,具备了较多的先进特性.通过对组成构架的基本元素进行描述,同时利用 XML 的可扩展性,XBA 可对现有的各种 ADL 进行描述及定义.

关键词: 软件构架; 构架描述语言; 可扩展标记语言; 构件; 连接器
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112 (2002) 12A-2036-04

A Research on XML Based Architecture Description Language-XBA

ZHAO Wen-yun, ZHANG Zhi
(*Dept. of Computer & Information Technology, Fudan University, Shanghai* 200433, *China*)

Abstract: Through analyzing current ADLs, this paper points out their shortcomings and presents an XML based ADL-XBA. It applies XML to the description of software architecture, and possesses many advanced characteristics. Through describing the basic elements that constitute architectures and making use of the extensibility of XML, XBA can describe and define the current various ADLs.

Key words: software architecture; ADL; XML; component; connector

1 引言

基于构件的软件开发方法正越来越受到人们的接受和重视.作为新一代的软件开发方法,它将软件人员的注意力吸引到构件的复用上,而基于构件的软件开发需要在较高的层次对软件系统进行把握,要对应用系统的构架进行描述,并在构架基础上实现可复用构件的组装.软件构架是一个具有较高抽象级别的系统视图,对构件组装具有十分重大的意义.构架描述语言(Architecture Description Language, ADL)是进行构架设计的主要技术及描述手段.有代表性的 ADL 包括 ACME[2], Wright[3], C2[4], Darwin[5]等.但这些 ADL 都是为满足某种特殊的设计目标与领域特征而开发的,有各自不同的侧重点和特色,对软件构架的研究起到了重要作用[6].它们在各自领域内是非常有效的,但由于它们的语法都是固定的并且差别较大,所以难以增加新的特性,而且彼此之间的交互非常困难,难以做到构架表示信息的共享.在关于到底什么是 ADL 这个问题上,在众多的 ADL 之间也还没有达成一致,它们对应该对一个构架的哪些方面建模的理解也不尽相同.例如,Rapid 作为一种通用系统描述语言,它对构件的接口和它们的外部可视行为进行了建模,而 Wright 对构架连接的语义进行了形式化描述[7].

为了支持我们的构件组装工具的开发,我们提出了一种基于 XML 的构架描述语言 XBA(XML Based ADL),它采用 XML Schema 作为定义机制.

2 XML 的特点

eXtensible Markup Language(XML)一经出现,就由于它的可扩展性、自描述性以及结构化等优点,受到了计算机界的广泛关注,并迅速成为了互联网上数据交换和处理的标准.同时由于 XML 出色的描述能力,它在互联网之外的许多领域也成为信息描述的标准.

XML 为开发软件构架的可扩展建模语言提供了理想的平台.XML 作为构架的标准描述语言有很多优点.使用 XML 作为构架的标准描述语言,就可以使得构架模型可以得到更为广泛的共享,许多应用都可以搜索、引用、处理和保存这些构架模型.根据不同领域的要求,对基于 XML 的 ADL 进行必要的扩展,就能快速满足新的需要.

3 一种基于 XML 的构架描述语言:XBA

3.1 XBA 的结构

在现在已经出现的各种 ADL 中,虽然每一种都有各自的特性,但它们还是有很多相似点的.例如几乎所有的 ADL 都是由构件(component)和连接器(connector)组成的,都定义了构件和连接器之间的约束即配置(configuration).作为一种可扩展的 ADL,XBA 包括了构架描述中的三种基本抽象元素:构件,连接器和配置.我们将通过一个实例来对 XBA 进行说明.

收稿日期:2002-06-06;修回日期:2002-10-15
基金项目:国家 863 计划基金(No.2001AA113070)

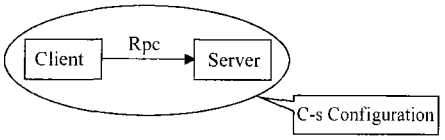

图 1　Client-Server 结构

图 1 给出了一个简单的客户机-服务器结构.　在图 1 中,Client 和 Server 是两个构件,Rpc 为连接器,构件和连接器的实例构成了这个 C-S 系统的 configuration.

3.2　构件(Component)

一个构件描述了一个局部的、独立的计算.在 XBA 里,对一个构件的描述有两个重要的部分:接口(interface)和计算(computation).

一个接口由一组端口(port)组成.每一个端口代表这个构件可能参与的交互.计算部分描述了这个构件实际所做的动作.计算实现了端口所描述的交互并且显示了它们是怎样被连接在一起构成一个一致的整体.

端口定义了一个构件的接口,它指出了构件的两个方面的特征.首先,它指出了构件对外界提供的服务.其次,构件对它所交互的系统的要求.在有些 ADL 中,端口的以上的两种作用分别被服务端口和请求端口所实现,但在 XBA 中,我们并不显式区分服务端口和请求端口,如果要显式说明这两种端口,可以从我们的 Schema 扩展,加入一个属性或元素来说明端口的类型.下面是 component 类型的 XML Schema 定义:

```
< complexType name = "componentType" >
< sequence >
  < element name = "Port" type = "portType" minOccurs = "0"
    maxOccurs = "unbounded" / >
  < element name = "Computation" type = "computationType"
    minOccurs = "0" / >
< /sequence >
  < attribute name = "Name" type = "string" / >
< /complexType >
< complexType name = "portType" >
  < element name = "Description" type = "string" minOccurs =
    "0" / >
  < attribute name = "Name" type = "string" / >
< /complexType >
< complexType name = "computationType" >
  < element name = "Description" type = "string" / >
  < attribute name = "Name" type = "string" / >
< /complexType >
```

3.3　连接器(Connector)

一个连接器代表了一组构件间的交互.使用连接器的一个重要的好处是它通过结构化一个构件与系统其它部分交互的方式,增加了构件的独立性.一个连接器实际上提供了构件必须满足的一系列要求和一个信息隐藏的边界,这个边界阐明了构件对外部环境的要求.构件说明只需指明构件将要做什么,而连接器说明描述了在一个实际的语境中这个构件怎样与其它部分合作.连接器 Rpc 的 XBA 描述如下:

```
< Connector name = "Rpc" >
  < Role name = "Source" >
    < Description >
      get request from client
    < /Description >
  < /Role >
  < Role name = "Sink" >
    < Description >
      give request to server
    < /Description >
  < /Role >
  < Glue >
    get request from client and pass it to server through some protocol
  < /Glue >
< /Connector >
```

一个连接器由一组角色(Role)和胶合(Glue)组成.每一个角色说明了一个交互中的一个参与者的行为.在连接器里 Glue 描述了参与者怎样一起合作来构成一个交互.Glue 说明了各个构件的计算怎样合起来组成一个规模更大的计算.就像构件中的计算部分一样,Glue 表示了连接器的动作.

3.4　Configurations

为了描述整个系统构架,构件和连接器必须合并为一个 configuration.一个 configuration 就是通过连接器连接起来的一组构件实例(Instance).Client-Server 构架的 XBA 描述如下:

```
< Configuration name = "Client-Server" >
  < Component name = "Client" / >
  < Component name = "Server" / >
  < Connector name = "Rpc" / >
  < Instances >
    < ComponentInstance >
      < ComponentName > MyClient < /ComponentName >
      < ComponentTypeName > Client < /ComponentTypeName >
    < /ComponentInstance >
    < ! --etc.-- >
    < ConnectorInstance >
      < ConnectorName > MyRpc < /ConnectorName >
      < ConnectorTypeName > Rpc < /ConnectorTypeName >
    < /ConnectorInstance >
  < /Instances >
  < Attachments >
    < Attachment >
      < From > MyClient.Request < /From >
      < To > MyRpc.Source < /To >
    < /Attachment >
    < ! --etc.-- >
  < /Attachments >
< /Configuration >
```

因为在一个系统中对一个给定的构件或连接器可能会使

用多次,所以我们把前面在介绍构件和连接器时所举的例子称为是构件或连接器的类型,也就是说,它们代表了构件或连接器的属性,而不是使用中的实例.为了区分一个 configuration 中出现的每一个构件和连接器的不同实例,XBA 要求每一个实例都明确的命名,并且这个命名应唯一.

Attachments 通过描述哪些构件参与哪些交互定义了一个 configuration 的布局或称为拓扑(Topology),这是通过把构件的端口和连接器的角色联系在一起来完成的.构件实现了计算,而端口说明了一个特别的交互.端口与一个角色联系在一起,那个角色说明了为了成为连接器所描述的交互的合法参与者那个端口所必须遵守的规则.如果每一个构件的端口都遵守角色所描述的规则,那么连接器的 Glue 定义了各个构件的计算怎样合并为一个单一的更大规模的计算.

3.5 层次结构

XBA 支持层次结构描述.一个构件可以由一个嵌套的子系统构架构成.这样,这个构件就相当于这个子系统构架的包装器,它的内部是一个完整的构架,而在外界看来,它相当于一个构件.一个子系统构架就像前面讲过的那样描述为一个 configuration.

图 2 给出了层次结构的一个例子.在系统的最高层,有 client 和 server 两个构件,而 server 是由一个子系统构架实现的,这个子系统构架由 ApplicationServer 和 DatabaseServer 两个构件组成.这样我们必须修改上述 component 的 XML Schema 定义,应该增加一个元素 subConfiguration,类型为 ConfigurationType.

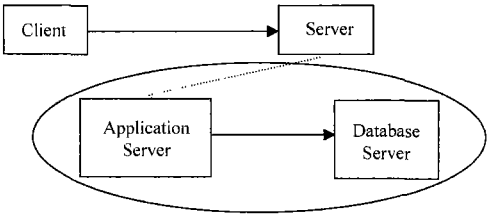

图 2 Client-Server 的层次结构

4 XBA 的可扩展性

通过利用 XML Schema 的扩展性机制,可以方便的通过 XBA 的核心 XML Schema 定义来获得新的可以描述其它 ADL 的 XML Schema.本节中将以 Darwin[5] 为例来说明怎样通过扩展 XBA 来描述其它的 ADL.

Darwin 起源于 1991 年,它吸收了早期 CONIC 语言(另一种构件配置语言)的特点,强调系统中构件的复合型,位置的分布性与生成的动态性.Darwin 倡导一种基于构件的系统结构,其中每一个单元(构件)封装其内部的操作行为,并对外提供定义良好的接口.系统通过创建构件实例并绑定实例间相匹配的端口得以实现.构件接口包含一到多个端口,按服务流向不同可分为服务端口(provide)与请求端口(require).与其它大多数 ADL 不同,Darwin 并没有使用显式的连接器.而是在构件间通过端口绑定直接连接.上面例子中简单的 C/S 结构可用 Darwin 语言描述如下:

```
component Client{
    require r;
}
component Server{
    provide p;
}
componet CS{
  Inst
    MyClient:Client;
    MyServer:Server;
  Bind
    MyClient.r-MyServer.p
}
```

由于 Darwin 的端口分为请求端口和服务端口两种,因此我们必须扩展 XBA 核心 Schema 定义中关于 Port 的定义,给它加一个属性 type 来标识这个端口是哪种类型,它有两种取值"in" 和"out",分别对应请求端口和服务端口.同是由于 Darwin 没有使用 configuration 的概念,而是使用了复合构件,所以我们应该把顶层的复合构件转换为 configuration.

由于 Darwin 的特点,我们必须扩展 XBA 的核心 Schema 的定义,来描述请求端口和服务端口.如果我们的 XBA 核心 Schema 的定义放在 xbacore.xsd,而新的描述 Darwin 的 Schema 放在 darwin.xsd,则 darwin.xsd 定义如下:

```
< schema targetNamespace = "http://www.example.com/ADLs"
xmlns = "http://www.w3.org/2001/XMLSchema"
xmlns:adl = " http://www.example.com/ADLs" >
< redefine schemaLocation = "http://www.example.com/xbacore.
    xsd" >
< complexType name = "Component" >
  < complexContent >
    < extension base = "adl:Component" >
      < attribute name = "type" type = "PortType"/ >
    < /extension >
  < /complexContent >
< /complexType >
< ! --etc.-- >
< /redefine >
< simpleType name = "PortType" >
< restriction base = "string" >
  < enumeration value = "in"/ >
  < enumeration value = "out"/ >
< /restriction >
< /simpleType >
< ! --etc.-- >
< /schema >
```

5 结语

本文提出了一种基于 XML 的构架描述语言 XBA.它利用 XML 的可扩展、自描述以及结构化等优点,围绕着构架的

种基本抽象元素:component、connector 和 configuration 来展开,实现了一种切实可行的描述系统构架的方法.它具有以下一些优点:

(1)XBA 具有开放式的语义结构,继承了 XML 的基于 Schema 的可扩展性机制,在使用了适当的扩展机制之后,XBA 可以表示多种构架风格.而且可以利用 XML Schema 的 include 和 import 等机制来复用已经定义好的 XML Schema,实现 ADL 的模块化定义.

(2)利用 XML 的链接机制(XLink),可以让我们实现构架的协作开发.我们可以先把构架的开发分解,由不同的开发者分别开发,然后再利用 XML 的链接机制把它们集成起来.

(3)易于实现不同 ADL 开发环境之间的模型共享.可能会存在多种基于 XML 的 ADL 开发工具,尽管它们所使用的对 ADL 的 XML 描述会有不同,但通过 XSLT 技术,可以很方便的在对同一构架的不同 XML 描述之间进行转换.

(4)XML 有希望成为构件库中构件存贮的标准,因此采用 XBA 有利于构架描述与构件库的结合.

在下一步的工作中,我们将对 XBA 作进一步的细化,以使它更加完善,并将 XBA 并入可视化的构件组装工具中.

参考文献:

[1] David C Fallside. XML Schema Part 0: Primer(W3C Recommendation, 2 May 2001)[S].

[2] David Garlan, Robert T Monroe, David Wile. Acme: An architecture description interchange language[A]. Proceedings of CASCON '97[C]. CSACON, 1997.

[3] R Allen. A Formal Approach to Software Architecture[D]. CMU, 1997.

[4] N Medvidovic, D S Rosenblum, R N Taylor. A language and environment for architecture-based software development and evolution[A]. Proc. 21st International Conference on Software Engineering[C]. ICSE, 1999.

[5] J Magee, N Dulay, S Eisenbach, J Kramer. Specifying distributed software architecture[A]. Proc. Fifth European Software Eng. Conf[C]. ESEC, 1995.

[6] 郑建丹,张广泉.软件体系结构描述语言 ADL[J].重庆师范学院学报(自然科学版),2000,(12).

[7] Nenad Medvidovic, Richard N Taylor. A classification and comparison framework for software architecture description languages[J]. IEEE Transaction on Software Engineering, 2000, 26(1).

作者简介:

赵文耘　男,1964 年 9 月出生于江苏省长熟市,复旦大学计算机与信息技术系教授,现在担任中国计算机学会软件工程专业委员会副主任,目前的主要研究领域是软件工程,软件构件技术,电子商务技术,已发表 10 多篇研究论文,获得过 10 多次国家级和省部级奖励.

张　志　男,1976 年出生于河北省石家庄市,2000 年 7 月毕业于复旦大学,获理学学士学位,现为复旦大学计算机与信息技术系硕士研究生,研究领域为软件工程,软件构件技术.