体系结构描述语言 XADL 和组合失配检测*

张 波, 冯玉琳, 黄 涛

(中国科学院 软件研究所 计算机科学重点实验室,北京 100080);
(中国科学院 软件研究所 软件工程技术研究开发中心,北京 100080)
E-mail: {zb,tao}@otcaix.iscas.ac.cn; feng@ios.ac.cn
http://www.iscas.ac.cn

摘要: 在各种构件模型和构件标准定义中,接口描述信息的不足和隐含式的构件交互协议容易造成构件复用的失配问题,不利于构件的复用、验证和管理.提出了基于 XML 消息的体系结构描述语言 XADL(XML-message based architecture description language),支持对构件交互协议和构件组合关系的描述,在此基础上给出了组合失配的定义和检测算法.XADL 不仅丰富了构件的接口语义,能够有效地避免组合失配问题,并且便于实现系统的运行监控、性能分析和动态调整.

关 键 词: 构件;接口;有限自动机;软件体系结构;组合失配
中图法分类号: TP311 文献标识码: A

基于构件的软件开发技术日益成为软件开发领域的主流技术.该技术以可复用的软件构件作为预制块,支持组装式软件复用,是提高软件生产效率和产品质量、缩短产品交付时间的现实有效的途径.若干构件模型和构件互操作标准相继建立.CORBA,DCOM/COM+,JavaBeans 构件模型三足鼎立,形成了实现级构件模型工业标准的竞争与互操作并存的格局.

但是,这些构件标准都侧重于构件之间互操作的实现,以 CORBA 为例,主要解决 CORBA 对象的平台独立性、语言独立性和位置独立性问题.而对于构件接口语义、构件之间的依赖关系以及交互协议等问题,目前的构件标准尚缺乏一定的支持,主要体现在以下两个方面:

- 接口描述信息不足

建立在构件模型的基础上,构件描述语言通过描述构件接口的语法和语义向外界提供构件的结构和行为信息,使构件的使用者不必关心其内部细节.针对接口描述信息的丰富程度的不同,我们可以将接口描述分为 4 个等级[1]:语法级、行为级、同步级和质量级.在 CORBA 等构件模型中都提供了描述构件接口的接口描述语言(IDL),这些 IDL 都能描述构件接口的语法,并且具备编译和浏览工具的支持[2].但是它们在描述构件接口语义和构件间复杂的交互协议方面缺乏进一步的支持.仅仅语法级的构件描述信息对于构件的复用、验证和管理来说是远远不够的.

- 隐含式的构件交互协议

在 CORBA 等构件模型中,构件之间通过远程方法调用来实现互操作[3],即构件之间只通过对象服务请求进行交互.也就是说,以中间件方式实现的为构件提供通信和协作等服务的基础设施只能支持一种或几种固定的构件交互模式[4],因此构件之间复杂的交互协议只能由构件自身来实现,即交互协议隐藏在构件内部.这种隐含

* 收稿日期: 2000-11-21; 修改日期: 2001-05-25
基金项目: 国家自然科学基金资助项目(69833030);国家 863 高科技发展计划资助项目(863-306-ZD02-01-1)
作者简介: 张波(1972－),男,山东兖州人,博士,助理研究员,主要研究领域为分布对象计算,软件体系结构;冯玉琳(1942－),男,江苏泰县人,博士,研究员,博士生导师,主要研究领域为软件工程,形式化方法,分布对象计算;黄涛(1965－),男,江苏淮安人,博士,研究员,博士生导师,主要研究领域为分布对象计算,对象语义理论,软件工程方法学.

式的交互协议一方面易于造成构件复用时的失配问题[5,6];另一方面,使得对软件体系结构的设计在最终实现中得不到显式的体现,不利于系统的性能分析、系统监控和动态调整.

针对上述问题,本文提出了一种基于 XML 消息的体系结构描述语言 XADL(XML-message based architecture description language),支持对构件交互协议和构件组合关系的描述.对构件接口的描述主要包括两部分:(1) 构件端口和端口上允许接收和发送的 XML 消息的类型;(2) 协议部分,描述这些消息所应满足的次序上的约束关系.构件通过端口之间的连接进行组合,对于两个连接的端口来说,其中一个端口上的发送消息即为另一个端口上的接收消息,反之亦然.由于每个构件都定义了消息所应满足的协议,因此,我们可以对组成系统的多个构件之间是否存在组合失配的问题进行检测.

本文第 1 节介绍了基于 XML 消息的体系结构描述语言 XADL,包括对构件接口的描述和对构件之间组合关系的描述,并给出了一些例子.第 2 节给出了组合失配的定义和组合失配检测算法.第 3 节是本文的工作和其他相关工作的比较及结论.

1 基于 XML 消息的体系结构描述语言 XADL

体系结构描述语言 XADL 的基础是一种基于消息的体系结构模型,即构件通过端口接收请求消息和发送响应消息;构件之间通过端口的连接进行消息交换,从而组合为复合构件或应用系统.因此,XADL 中包含了对构件接口的描述和对系统组合的描述,第 1.1 节和 1.2 节分别介绍了这两部分内容,第 1.3 节给出了完整的例子.

1.1 构件接口描述

在 XADL 中,构件通过端口接收和发送消息,端口分为输入端口和输出端口,分别用来接收请求消息和发送响应消息.我们用关键字 InPorts 和 OutPorts 定义端口.在构件组合过程中,只有在不同输入、输出方向的端口之间才能进行连接.

端口除了具有方向之外,还必须指明其所允许的消息类型.一个端口上可以允许多个不同的消息类型.为了使系统具有良好的可扩展性,XADL 采用 XML 来描述消息,因此在接口描述中,我们采用文档类型定义(DTD)表示消息类型,并把 DTD 作为定义消息类型的关键字.

实际上,端口及其所允许的消息类型定义了接口的基调(signature),描述了构件与外界所能交换的消息的类型和方向.我们把某个端口上某个 XML 消息的出现称为一个事件,而把由端口和其所允许的消息类型所组成的二元组定义为事件类型:

事件:e=⟨port,XML Message⟩,

事件类型:E=⟨port,XML DTD⟩.

换句话说,事件类型的集合定义了接口的基调.与传统的 IDL 不同的是,在 XADL 的接口描述中,除了定义接口基调外,还将对事件之间的次序约束关系进行描述.我们采用有限状态自动机来进行次序约束关系的刻画.有限状态自动机由一组状态和一组迁移构成.自动机在某个状态下会响应某个类型的事件,并转移到另外的状态:

⟨state⟩:⟨event type⟩→⟨state⟩.

自动机的定义由关键字 Constraints 引导,包含状态的定义和迁移的定义,分别以 States 和 Transitions 为关键字,在自动机的状态中,有一个起始状态和一个或多个终结状态,分别用 init 和 final 来标注.

1.2 系统组合描述

在 XADL 中,不仅支持对构件接口的描述,还支持对构件组合关系的描述.构件之间通过端口的连接进行系统组合,只有方向相反和消息类型相同的端口才能进行连接.在参与某个连接的端口中,不允许存在属于同一个构件的两个端口.XADL 支持一对多的连接,对于输出端口来说,其上的消息将被同时输出到所有与其相连的输入端口上;而对于输入端口来说,同一时刻最多只能有一个与其相连的输出端口上的消息被输入到该端口上.对于具有连接关系的输入端口 A 和输出端口 B 来说,以下的约束成立:事件⟨A,X⟩和事件⟨B,X⟩同时发生,其中 X 是满足两个端口上的消息类型的 XML 消息.我们称这种约束为连接约束.

构件接口描述定义了构件的类型,而在系统组合描述中,我们首先要定义构件实例.构件实例的定义以

Instances 为关键字,一个系统中可以包含同一构件类型的多个实例.构件端口之间的连接关系用关键字 to 表示,具有以下的形式:

C1.p1 to C2.p4 或 C1.p1 to C2.p3, C3.p4.

其中 C1,C2 和 C3 分别为构件实例,而 p1,p3 和 p4 分别为各个构件的端口.

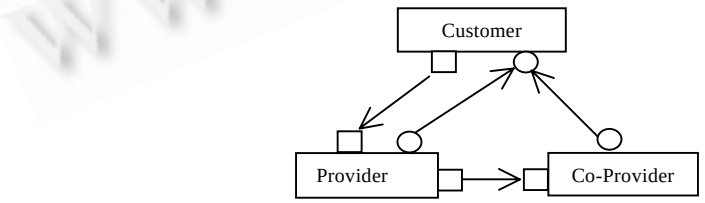
1.3 例 子

这里给出一个用 XADL 定义系统的例子,系统的名称为 InfoQuery,如图 1 所示.在本系统中,存在一个客户构件 customer,两个提供者构件 provider 和 co-provider.customer 向 provider 发出查询某产品价格和数量的请求,若 provider 能够提供此类信息,则返回给 customer;否则将请求转发给其合作伙伴 co-provider,由它对请求进行处理并将结果返回给 customer.其结构关系如图 2 所示.

```
System InfoQuery
  DTD dQuery
    ⟨!ELEMENT product_ID(#PCDATA)⟩
  End dQuery;
  DTD dInfo
    ⟨!ELEMENT Info(price, quantity)⟩
    ⟨!ELEMENT price(#PCDATA)⟩
    ⟨!ELEMENT quantity(#PCDATA)⟩
  End dInfo;
  Component TCustomer
    InputPort pReply;
    OutputPort pRequest;
    EventType
      eRequest=⟨pRequset, dQuery⟩,
      eReply=⟨pReply, dInfo⟩,
    End;
    Constraints
      States S0 (init) (final),S1;
      Transitions
        S0:eRequest→S1;
        S1:eReply→S0;
      End;
    End;
  End TCustomer;
  Component TProvider
    InputPort pReceive;
    OutputPort pAnswer,pFwd;
    EventType
      eReceive=⟨pReceive,dQuery⟩;
      eFwd=⟨pFwd,dQuery⟩;
      eAnswer=⟨pAnswer,dInfo⟩;
    End;
    Constraints
      States S0 (init) (final),S1;
      Transitions
        S0:eReceive→S1;
        S1:eAnswer→S0;
        S1:eFwd→S0;
      End;
    End;
  End TProvider;
  Component TCo-provider
    InputPort pReceive;
    OutputPort pAnswer;
    EventType
      eReceive=⟨pReceive,dQuery⟩;
      eAnswer=⟨pAnswer,dInfo⟩;
    End;
    Constraints
      States S0 (init) (final),S1;
      Transitions
        S0:eReceive→S1;
        S1:eAnswer→S0;
      End;
    End;
  End TCo-provider;
  Instances
    TCustomer:customer
    TProvider:provider
    TCo-provider:co-provider
  End;
  Connections
      customer.pRequest to provider.pReceive;
      provider.pFwd to co-provider.pReceive;
      customer.pReply to provider.pAnswer,
                         co-provider.pAnswer;
  End;
End InfoQuery.
```

Fig.1 The definition of system InfoQuery in XADL

图 1 采用 XADL 对系统 InfoQuery 的定义



□ The ports bundling with message type of dQuery① ○ The ports bundling with the message type of dInfo②

①消息类型为 dQuery 类型的端口,②消息类型为 dInfo 类型的端口.

Fig.2 The structure of system InfoQuery

图 2 InfoQuery 的系统结构

2 组合失配检测

在系统的定义中,由于连接约束的存在,构件交互协议与构件组合关系之间有时会产生矛盾.举一个直观的例子:在构件 C1 中,要求事件$\langle A,X\rangle$先于事件$\langle A,Y\rangle$,在构件 C2 中,要求事件$\langle B,Y\rangle$先于事件$\langle B,X\rangle$,而端口 A 和端口 B 的连接将导致无法同时满足构件 C1 和 C2 中规定的事件之间的次序关系.在本节中将对系统的这类性质进行讨论.第 2.1 节对交互协议的同步和异步语义进行了讨论,在此基础上,第 2.2 节给出了组合失配的定义,第 2.3 节给出了组合失配的检测算法.

2.1 交互协议语义

对于基于消息的构件交互协议来说,有两种不同的语义:同步语义和异步语义.在异步语义下,只要构件处在可以发送消息的状态,不论消息的接收方是否处在可以接收消息的状态,都可以发送相应的消息.在这种模式下,每个构件都有一个消息队列来保存其接收到但还未来得及处理的消息.虽然异步语义易于实现,但是系统的一些性质比如死锁等是不可预测的[7].

在同步语义下,构件只有在如下两个条件同时成立的条件下才能发送消息,即构件自身处在可以发送消息的状态,并且消息的接收方处在可以接收消息的状态.这样,消息传递的双方将同时由原来的状态迁移到新的状态.同步语义简化了系统的复杂程度,有利于对系统的某些性质进行研究.当然,系统的实现仍可以采用异步的方式,只要交互的双方能够在消息收发的次序上保持一致,同步语义下得到的结论仍然成立[3].

2.2 组合失配

我们在同步语义下讨论组合失配的问题.假设系统中有 n 个构件实例,我们将它们分别记为 $C_1,C_2,\ldots,C_n$,并记这些构件的协议分别为 $P_1,P_2,\ldots,P_n$.

对于协议 P,定义 States(P)为 P 的状态集合,Transitions(P)为 P 的迁移集合,Init(P)为 P 的初始状态,Final(P)为 P 的终结状态集合.为讨论方便,我们设每个迁移集合都包含一个表示空迁移的元素ε,在空迁移ε下,构件状态不发生变化.

对于某个迁移规则 t,设 t 为 $S:\langle g,D\rangle\rightarrow S'$,定义 PreTrans($t$)为 t 的初始状态(即 S),PostTrans(t)为 t 的目标状态(即 S');对于端口 g 来说,如果 g 是输入端口,则定义 Direction(g)=‘+’,如果 g 是输出端口,则 Direction(g)=‘−’.

定义 **1**. 设 n 元组$\alpha=\langle S_1,S_2,\ldots,S_n\rangle$,如果对每个 $i=1,\ldots,n$,都有 $S_i\in$States(P_i),则称α为系统的一个全局状态;设α是系统的一个全局状态,如果对每个 $i=1,\ldots,n$,都有 $S_i\in$Final(P_i),则称α为系统的终结全局状态.

定义 **2**. 设 $T=\langle t_1,t_2,\ldots,t_n\rangle\in$Transitions($P_1$)×Transitions($P_2$)×…×Transitions($P_n$),并且$\alpha=\langle S_1,S_2,\ldots,S_n\rangle$是系统的全局状态,若 T 同时满足下列条件,则称 T 为全局状态α下的可执行全局迁移:

(1) $\forall 1\leq i\leq n(t_i\neq\varepsilon\Rightarrow S_i=\text{PreTrans}(t_i))$.

(2) 对任意非空的迁移 $t_k=S_k:\langle g,D\rangle\rightarrow S_k'$,如果 Direction($g$)=‘−’,则对于任意与 g 相连的端口 g',设 g'为构件 C_i 的端口,则有 $t_i=S_i:\langle g',D\rangle\rightarrow S_i'$.

(3) 对任意非空的迁移 $t_k=S_k:\langle g,D\rangle\rightarrow S_k'$,如果 Direction($g$)=‘+’,则在与 g 相连的端口中,存在惟一的端口 g',设 g'为构件 C_i 的端口,则有 $t_i=S_i:\langle g',D\rangle\rightarrow S_i'$;对于其他与 g 相连的端口,设 g'为构件 C_j 的端口,则有 $t_j=\varepsilon$.

显然,状态组合α下可执行全局迁移可能不存在,也可能存在一个或多个.

定义 **3**. 对于全局状态$\alpha=\langle S_1,S_2,\ldots,S_n\rangle$和$\alpha$下的可执行全局迁移 T,设$\alpha'=\langle S_1',S_2',\ldots,S_n'\rangle$,若满足下列条件,则称$\alpha'$是$\alpha$关于 T 的后继:

(1) $t_i=\varepsilon\Rightarrow S_i'=S_i$;

(2) $t_i\neq\varepsilon\Rightarrow S_i'=$PostTrans($t_i$).

定义 **4**. $h=\alpha_1\rightarrow_{T1}\alpha_2\rightarrow_{T2}\ldots$是系统的某个执行历史,如果:

(1) 每个α_i都是系统的一个全局状态;

(2) $\alpha_1=\langle\text{Init}(P_1),\text{Init}(P_2),\ldots,\text{Init}(P_n)\rangle$;

(3) T_i是状态组合α_i下的一个可执行全局迁移,α_{i+1}是α_i关于 T_i 的后继,其中 $i=1,2,\ldots$

定义 **5**. 全局状态α是系统的可达全局状态,当且仅当α是某个执行历史的最后一个全局状态.设$\psi_r=\{\alpha|\alpha$是可达全局状态$\}$,显然,$\psi_r\subseteq States(P_1)\times States(P_2)\times\ldots\times States(P_n)$,所以$\psi_r$是一个有限集合.

定义 **6**. 设集合$H=\{h:h$是系统的执行历史$\}$,如果存在有限长度的执行历史$h\in H,h=\alpha_1\rightarrow_{T1}\alpha_2\rightarrow_{T2}\ldots\rightarrow_{Tm}\alpha_m$,满足下列条件,则称系统存在组合失配:

(1) α_m不是终结状态组合;

(2) h不是集合H中其他任何一个执行历史的前缀.

从上述定义中我们可以看出,可执行全局迁移的定义保证了系统中消息传递的双方对消息的同步响应,避免了没有接收方的消息的发送和没有发送方的消息的接收;组合失配表明了系统中存在发生死锁的可能性,如果系统不存在组合失配,则能保证系统不会死锁.

2.3 失配检测算法

虽然一个执行历史的长度可能是无穷的,但是其中的全局状态的个数却是有限的.只要我们能构造出可达全局状态集ψ_r,就能逐一地对其中的全局状态进行检查,判断其是否为终结全局状态并且是否存在后继.如果ψ_r中存在没有后继的非终结全局状态,则可以判定系统存在组合失配.由此,我们有如下的组合失配检测算法:

(1) 设ψ_r为空集,然后将$\langle Init(P_1),Init(P_2),\ldots,Init(P_n)\rangle$加入到$\psi_r$中;

(2) 对于ψ_r中所有未被标注上处理标记的元素α,转到(3);

(3) 对任意$T\in Transitions(P_1)\times Transitions(P_2)\times\ldots\times Transitions(P_n)$,如果$T$为$\alpha$下的可执行全局迁移,则将$\alpha$关于$T$的后继$\alpha'$加入到$\psi_r$中,并给$\alpha$标注上存在后继的标记;

(4) 给α标注上处理标记,若ψ_r中存在未被标注上处理标记的元素,转到(2);

(5) 遍历ψ_r中所有未被标注上存在后继标记的元素,若存在非终结全局状态,则系统存在组合失配.

3 相关工作和结论

本文的工作针对如下两个方面进行了研究:一是对软件体系结构的研究;二是构件技术中关于如何扩充构件接口描述能力的研究.

20 世纪 90 年代以来,软件体系结构逐渐发展成为软件工程中一个新的研究领域.Stanford 大学的 Rapide[8] 小组对基于事件的软件体系结构进行了深入研究,提出了用于刻画事件依赖关系的事件模式理论,并用它来描述构件和体系结构.但是,一方面由于事件模式的表述方式比较复杂,另一方面由于其主要目的在于通过模拟对体系结构进行分析,所以 Rapide 中没有对组合失配问题进行探讨.本文在事件的基础上采用表述方式相对简单的有限自动机来描述构件之间的交互协议,从而实现了对组合失配的检测.事实上,事件模式中事件之间的依赖关系也可以用自动机来描述,限于篇幅,这里不再给出证明.

传统的语法级的构件接口描述方法不能提供对构件复用的有力支持.针对构件接口的扩充有很多建议,其中 IBM 的 Yellin 和 Strom 提出了采用有限自动机对构件交互协议进行描述和分析的方法[3].但是该方法仅支持对两个构件之间交互过程的分析,并且缺乏对构件组合关系的描述,因此无法被运用到多个构件进行交互的情形中.而在实际系统中,构件的个数通常是很多的.我们在构件交互协议的基础上,引入了对构件组合关系的描述,进而提出了多个构件之间交互的组合失配检测算法,因此具有比较强的实用性.

值得说明的是,在本文中,虽然 XML 只是用来描述构件之间交互信息的类型,但是 XML 的采用对构件接口描述能力的扩充和系统实现的开放性具有十分重要的意义.比如,我们可以很方便地描述请求消息和响应消息的 XML 元素之间所应满足的逻辑关系,以此来扩充构件接口对行为约束的描述能力.此外,由于构件之间传递的都是 XML 消息,与 CORBA 等其他系统的私有的消息编码体制相比,无疑具有良好的开放性,因而便于实现系统的运行监控、性能分析和动态调整等维护工作.限于篇幅,本文对此没有进行进一步的讨论,这部分工作可参见文献[9].

目前的构件模型标准虽然实现了构件的互操作,但也存在接口描述信息不足和隐含式的构件交互协议等问题,因而不利于构件的复用、验证和管理,容易造成构件复用时的失配问题.本文提出了基于 XML 消息的体系

结构描述语言 XADL,采用有限自动机刻画构件的交互协议,并利用构件端口的连接来描述构件之间的组合关系.在此基础上给出了组合失配的定义和检测算法,不仅提高了构件接口的描述能力,而且有效地避免了组合失配的问题.同时,采用 XML 来描述构件之间交互信息的类型,不仅便于对构件接口描述能力的扩充,也使得最终实现的系统具有良好的开放性.

References:

[1] Beugnard, A., Jezequel, J., Plouzeau, N., *et al*. Making components contract aware. IEEE Computer, 1999,32(7):38~45.

[2] Orfali, R., Harkey, D., Edwards, J. The Essential Distributed Objects Survival Guide. John Wiley & Sons, Inc., 1996.

[3] Yellin, D.M., Strom, R.E. Protocol specification and component adaptors. ACM Transactions on Programming Languages and Systems, 1997,19(2):292~333.

[4] Mehta, N.R., Medvidovic, N., Phadke, S. Towards a taxonomy of software connector. In: Ghezzi, C., Jazayeri, M., Wolf, A., eds. Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000). New York, NY: ACM Press, 2000. 179~185.

[5] Garlan, D., Allen, R., Ockerbloom, J. Architectural mismatch: why reuse is so hard. IEEE Software, 1995,12(6):17~26.

[6] Gacek, C., Boehm, B.W. Composing components: how does one detect potential architectural mismatches? In: Workshop on Compositional Software Architectures. 1998. http://www.objs.com/workshops/ws9801/papers/paper050.html.

[7] Brand, D., Zafiropulo, P. On communicating finite-state machines. Journal of the ACM, 1983,30(2):323~342.

[8] Luckham, D.C., Kenney, J.J., Augustin, L.M., *et al*. Specification and analysis of system architecture using rapide. IEEE Transactions on Software Engineering, 1995,21(4):336~355.

[9] Zhang, Bo, Ding, Ke, Li, Jing. An XML-message based architecture description language and architectural mismatch checking. In: Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC 2001). 2001. http://dlib.computer.org/conferen/compsac/1372/pdf/13720561.pdf.

The Architecture Description Language XADL and Architectural Mismatch Checking*

ZHANG Bo, FENG Yu-lin, HUANG Tao

(*Key Laboratory for Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*);

(*Software Engineering Technology Center, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*)

E-mail: {zb,tao}@otcaix.iscas.ac.cn; feng@ios.ac.cn

http://www.iscas.ac.cn

Abstract: By the currently developed component models and standards, the component interfaces convey only limited semantics and the interaction protocol among components is hidden within the implementations, which may result in the problem of architectural mismatches, also make these components in difficulty to reuse, validate and manage. In this paper, an XML-message based architecture description language (XADL) is outlined, which supports the description of explicit interaction protocols and the composing relations of components. Based on the XADL, the notion of architectural mismatch is introduced and shows how it can be checked. The XADL enhances the descriptive capacity of interfaces, prevents systems from potential architectural mismatches, and facilities the implementation of system monitor, performance analysis and dynamic tuning.

Key words: component; interface; finite-state automata; software architecture; architectural mismatch

* Received November 21, 2000; accepted May 25, 2001

Supported by the National Natural Science Foundation of China under Grant No.69833030; the National High Technology Development 863 Program of China under Grant No.863-306-ZD02-01-1