基于反射的动态软件体系结构实现

唐 姗 赵文耘

(复旦大学计算机科学与工程系,上海 200433)

摘 要: 文章结合元模型、反射等技术首先从宏观的角度提出了一个支持分布式应用环境的动态软件体系结构模型,并描述了如何对系统进行动态演化,然后从微观的角度给出了软件体系结构的形式化描述。基于此框架,作者在 Web 环境下实现的一个可动态演化的网上宠物商店系统证明了该方法的有效性。

关键词: 软件体系结构,动态演化,反射,元模型

中图法分类号: TP31 文献标识码: A 文章编号: 1000-7180(2006)09-0032-03

Implementation of Dynamic Software Architecture Based on Reflection

TANG Shan, ZHAO Wen-yun

(Department of Computer Science and Engineering, Fudan University, Shanghai 200433)

Abstract: Combining the techniques of meta-modeling and reflecting, this paper from a macroscopical perspective firstly proposes a dynamical software architecture model supporting distributional application environment, and describes how to perform dynamic evolvement to the software system, then from the microcosmic perspective, gives out a formal description to the software architecture. The authors implement this model to an online dynamical pet shop system, and prove its availability.

Key words: Software architecture, Dynamic evolve, Reflection, Meta-model

1 引言

当前大量的分布式应用系统都构架在 Web 的基础上。这样一个开放的、动态的环境要求软件系统的内部组成结构常常需要通过检测需求、环境的变更等因素,做出相应的自我调整,以适应动态的开放环境和多变的用户需求。于是,如何把运行时的适应性机制加到复杂的大规模软件系统中,以满足系统能够动态演化等诸多方面的要求,创建灵活的体系结构是当前软件工程领域的一个研究热点。本文首先提出一个能够支持动态重配置的基本框架,并描述了系统进行动态演化的具体操作流程,然后进一步给出了组成体系结构的构件、连接件的形式化定义,并对它们的动态行为进行了探讨。最后给出基于该模型在 Web 环境下实现的一个可动态演化的网上宠物商店系统的解说实例,以证明该方法的有效性。

2 相关技术

元模型技术[1](Meta modeling)是以元模型定义模型中的类及类之间的关系,控制程序运行时的表现。反射(Reflection)[2]是以自述方式表示系统的状态和行为。通常一个反射系统由两个逻辑空间组成:基空间和元空间,其中基空间是被反射的系统运行的空间,元空间是对它进行反射的空间,通过"具体化",系统从基空间映射到元空间,通过"反射",系统从元空间映射到基空间。因此,利用反射技术在运行时重新配置系统的行为可以实现系统运行时对环境和需求变化的自适应。

3 总体框架设计

该框架的核心思想是:用元数据来描述更改系统的配置信息(元数据是有关业务实体的属性、行为、实体间关系、业务规则等的配置信息),并在运行时利用反射技术来解释执行以存放于领域本体中的元数据建立的对象模型。图 1 为总体框架的示意图。该框架分为用户自定义接口、中心服务器以及分布在各地的应用系统三大部分:

(1) 用户自定义接口是供用户定制系统的一些图形化支撑工具。

收稿日期: 2006-04-25

基金项目: 国家自然科学基金项目(60473061)

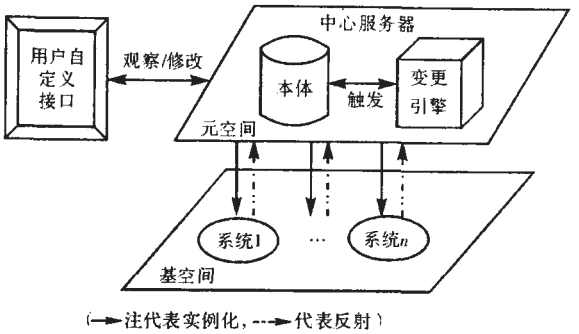

图1 总体框架图

(2) 中心服务器主要有由本体描述的元模型和执行引擎两部分。本体以一种标准的方式描述了整个系统当中的所有资源，包括业务对象（各个构件）、业务规则（本体可依据这些规则自动进行推理）、连接关系(构件之间的交互关系)等;执行引擎负责在应用系统运行时解释执行元模型。抽象地说可以把它看作是一个 Mealy 机。在交互消息的驱动下,将系统从一种状态转换成另一种状态。

(3) 分布在各地的应用系统为分布式环境中通过网络与中心服务器相连并受其控制的客户端机器。就应用系统所反射的类型,文献[3]将反射分为结构反射和行为反射。这两种反射类型具体的元对象协议描述分别如表 1、表 2 所示。

表 1　与系统拓扑结构的更新相关的元对象协议

协议名称	GetTopology	AddConnector	DeleteConnector
协议功能	获取软件系统的拓扑结构	增加一个连接件	删除一个连接件
协议名称	AddComponent	DeleteComponent	ReplaceComponent
协议功能	增加一个构件	删除一个构件	替换一个构件

4　动态演化流程

关于如何演化,业界已做了大量的研究实践工作,文献[4, 5]归纳出软件系统进行动态演化的流程是如下四步:启动更新、确定更新对象、执行更新操作、对更新进行评估。尽管上述流程基本上实现了对系统的动态更新，但它存在一个很大的缺陷:没有考虑到如何保护系统的完整性在演化后不受到破坏。本文通过同步和状态管理两个手段来保护系统的完整性。本文所提出的具体动态执行过程如下所述:(1) 用户发出更新请求，启动系统更新;(2) 更新请求解析器捕获用户请求,并进行分析,确定更新类型及更新范围;(3) 状态管理器根据提交过来的文件做相应处理:① 若是对单个构件的局部更新,则去获取该构件的当前状态信息,如果该构件已处理完所有请求,即该构件处于空闲状态,则挂起该构件的运行实例,由相关连接件缓存其它构件在更新过程中发送过来的请求信息。如果该构件正在处理请求,则等待其处理完,再挂起其运行实例。这时构件进入更新就绪状态;② 若修改操作牵涉到系统中多个构件,即全局更新,则要让所有构件都进入更新就绪状态;完成上述准备工作后,状态管理器就可以正式向重配置执行系统递交更新请求了;(4) 重配置执行系统计算执行获得了重配置许可证的更新请求。执行过程中,为了保证系统的一致性,我们在访问数据对象的过程中引入了事务和加、减锁的机制来实现重配置计算。① 如果系统运行计算成功,将更新信息写入构件的元模型的描述文件中并对此次更新进行评估;② 如果系统运行过程中出错,则撤销此次请求,回滚整个过程中的所有操作。

表 2　与系统的行为更新相关的元对象协议

协议名称	GetComState	GetConnState	GetComDescription
协议功能	获得当前构件的状态	获得当前连接件的状态	获得构件的接口描述
协议名称	GetConnDescription	UpdateComFunction	UpdateConnFunction
协议功能	获得连接件的接口描述	修改构件的功能实现机制	修改连接件的功能实现机制
协议名称	SuspendComInstance	SuspendConnInstance	ActiveComInstance
协议功能	挂起构件的运行实例	挂起连接件的运行实例	激活构件的运行实例
协议名称	ActiveConnInstance	TerminateComInstance	TerminateConnInstance
协议功能	激活连接件的运行实例	中止构件的运行实例	中止连接件的运行实例

5　微观探讨

构件是体系结构的计算、数据部件,实现了一组功能,大致对应于程序中的一个编译单元。它由构件接口集合和构件实现模块组成。接口集合即 $\{Port_1, Port_2, \cdots, Port_n\}$,而每一个接口 $Port_i$ 是一个八元组 $\{ID, Publ_i, Exte_i, Priv_i, Beha_i, Msgs_i, Cons_i, NonFunc_i\}$。其中:ID 是构件的标识;$Publ_i$ 是构件第 i 个接口所能提供给外界的功能集合;$Exte_i$ 是构件第 i 个接口向外界请求的功能集合;$Priv_i$ 是构件第 i 个

接口的私有属性集合; $Beha_i$ 是构件第 i 个接口的行为语义描述; $Msgs_i$ 是构件第 i 个接口所产生的消息集合; $Cons_i$ 是对构件第 i 个接口的行为约束，通常写成 Cons (init, pre- cond, post- cond), init, pre- cond 和 post- cond 分别表示初始条件、前置条件和后置条件的集合; $Non\text{-}Func_i$ 是构件第 i 个接口非功能说明，通常与质量属性相关，包括构件的安全性、可靠性说明等。

构件演化的定义：设 A 和 B 是论域 U 中的两个构件，如果 A 和 B 满足下列条件，则称 B 是由 A 演化而来的，记为 Evolve (B, A)。(1) Publ(B)⊇Publ(A); (2) Extn(B)⊆Extn(A); (3) Priv(B)⊆Priv(A); (4) Beha(B)⇒Beha(A); (5) Msgs(B)=Msgs(A); (6) (Cons(B)=Cons(A)) or (Cons(B)⇒Cons(A)); (7) (Non- Func(B)=Non- Func(A)) or (Non- Func(B)⇒Non- Func(A))。

构件相等的定义：设 A、B 是论域 U 中的两个构件，如果 A, B 满足下列条件，则称 A, B 相等，记作A=B。(1) Publ(A)=Publ(B); (2) Extn(A)=Extn(B); (3) Priv(A)=Priv(B); (4) Beha(A)⇔Beha(B); (5) Msgs(A)=Msgs(B); (6) Cons(A)⇔Cons(B); (7) Non- Func(A)⇔Non- Func(B)。

连接件定义了构件间的交互关系，它在体系结构中主要是进行数据及控制传递、接口适应性修改及数据转换、访问协调与同步、通信拦截与构件动态连接。连接件是一个六元组〈ID, Role, Beha, Msgs, Cons, Non- Func〉。其中：ID 是连接件的标识; Role 为连接件与构件的交互点的集合，每个 Role=〈Id, Action, Event, LConstrains〉组成。其中：Id 是 Role 的标识; Action 是 Role 活动的集合，每个活动由事件的连接（谓词）组成; Event 是 Role 产生的事件集合; LConstrains 是 Role 的约束集合。Msgs 是连接器中各 Role 中事件产生的消息的集合。Beha 是连接器行为的语义描述。Cons 是连接器约束的集合，Non- Func 是连接器的非功能说明，包括连接器的安全性、可靠性说明等。

6 系统实现

演化的层次分为语义级和实例级两个层次。目前我们均已在网上宠物商店这个实例系统上成功实现了这两个层次上的动态演化。鉴于篇幅有限，这里只举一个在实例级上的简单演化的例子。假如在系统运行的某个时刻，有个处理订单的构件 OrderDeliver_5 由于负载过重而消亡了，此时系统必须能自动找到一个与 OrderDeliver_5 功能等价的构件来代替它继续工作，以维持系统的正常运行。这里我们假设找到了 OrderDeliver_3 来替代 OrderDeliver_5 继续为 Customer_4 提供服务。演化前系统的拓扑结构如图 2 所示。

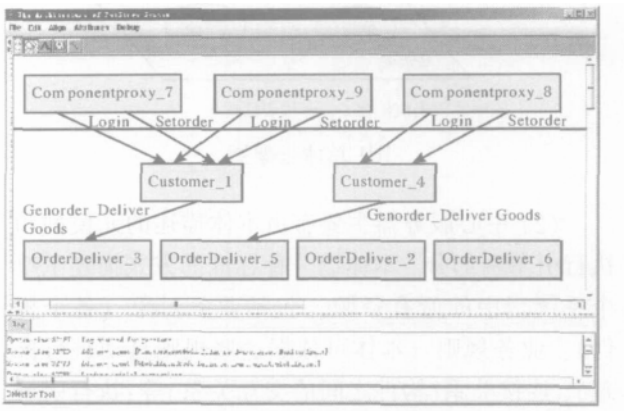

图2 系统演化前的拓扑结构

具体的演化过程为：(1) 先获得 Customer_4 的当前状态：GetComState(Customer_4); (2) 若该构件处于活动状态，先将其运行实例挂起 SuspendComInstance (Customer_4); (3) 获得 Customer_4 的接口描述 GetComDescription(Customer_4)，以让系统知道它所要请求的功能; (4) 调用执行引擎去搜索系统中满足该请求接口条件的构件，通过计算系统找到了 OrderDeliver_3 满足条件; (5) 于是通过新增一个连接件将两构件关联起来，AddConnector (Customer_4, OrderDeliver_3); 6 > 激活 Customer_4 的运行实例，ActiveComInstance(Customer_4)，此时系统就能和 OrderDeliver_5 消亡前一样正常运行了。演化后系统的拓扑结构如图 3 所示。

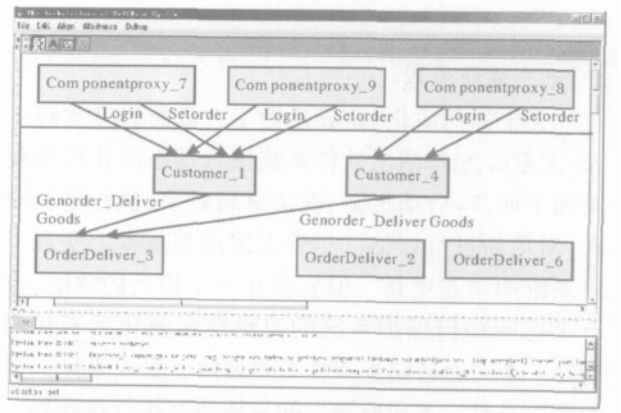

图3 系统演化后的拓扑结构

7 结束语

本文对支持分布式环境下的动态软件体系结

(下转第 37 页)

况下更新报文发送总数，如图 2 所示。图中横坐标为模拟节点个数，纵坐标为 300 秒内所有模拟节点发送的更新报文总和。

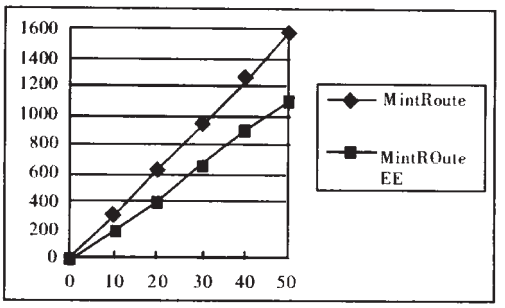

图2 总更新报文数目

总的来看，MintRouteEE 协议通过阀值控制能够有效减少重复路由更新报文的发送，同时更新报文强制发送机制能够解决邻居表过期的问题。在保证路由有效，不降低性能的前提下，MintRouteEE 比 MintRoute 协议更加节能，特别是在网络规模大，节点数目多的条件下，优势更加明显。

5 结束语

目前，TinyOS 系统中设计的地址字段长度只能支持较小的网络规模。未来，日益发展的无线传感器组网技术将促使网内节点规模不断扩大。成千上万个节点间协同工作时，单个节点因能量消耗过快而‘死亡’的情况也将频繁发生。因此节能是一个不可回避的问题。本文基于实际可运行的 Mote 节点和 TinyOS 系统，设计了一种具有能量有效性的 MintRouteEE 路由协议，通过选择性周期发送减少不必要的路由更新，在减少协议开销的同时实现节能；通过等价父节点轮选实现一定区域内报文转发的负载平衡。TOSSIM 上的模拟结果表明，MintRouteEE 比 MintRoute 在节能上有更好的性能。

参考文献：

[1] Jamal N. Al-Karaki, Ahmed E. Kamal. Routing techniques in wireless sensor networks: A Survey. In: Proc of IEEE Wireless Communications, 2004, 6: 6~28

[2] www.tinyos.net

[3] David Gay, Philip Levis, Robert von Behren at el. The nesC Language: A Holistic approach to networked embedded systems. Programming language design and implementation (PLDI), 2003

[4] P Levis, N Lee, M Welsh, D Culler. TOSSIM: Accurate and scalable simulation of entire tinyOS applications. To appear in proceedings of the first ACM conference on embedded networked sensor systems, SenSys, 2003

作者简介：

叶　嘉　男，(1981-)，硕士研究生。研究方向为无线传感器网络路由协议。

彭　伟　男，(1973-)，副研究员，硕士生导师。研究方向为移动自组网、无线传感器网络、路由器技术。

(上接第 34 页)

构的描述、演化过程进行了详细的探讨。并结合实现的系统实例具体介绍了演化效果。目前我们课题组在这个领域已取得了一些研究成果。进一步的工作是集成自然语言处理子系统，以让用户能用自然语言表达他们的需求。另一方面将由手工修改或由系统预定义的一些演化升级到智能化的系统全自动演化。

参考文献：

[1] COSTAFM. Combining meta-information management and reflection in an architecture for configurable and reconfigurable middleware [D]. Lancaster: PhD Thesis, Lancaster Uiversity, 2001

[2] D G Bobrow, R G Gabriel, et al. CLOS in context-The shape of the design space. In: Object Oriented Programming-The CLOS Perspective. Massachusetts: MIT Press, 1993

[3] J Ferber. Computational reflection in class based object oriented languages.In: Proc of the 4th Conf on Object-Oriented Programming:Systems,Languages and Applications(OOPSLA'89), SIGPLAN Notices 24. New York: ACM Press, 1989, 317~326

[4] Jeremy S Bradbury, James R Cordy, Juergen Dingel, A survey of self-management in dynamic software architecture specifications. Newport Beach, CA, USA, WOSS'04, 2004

[5] J Andersson. Issues in dynamic software architectures. In Proc. Of the 4th Int. Software Architecture Workshop (I-SAW-4), 2000: 111~114

作者简介：

唐　姗　女，博士研究生。研究方向为软件体系结构、软件构件技术。

赵文耘　男，教授，博士生导师。研究方向为软件工程、电子商务。