中国科学 E 辑: 信息科学
2007 年 第 37 卷 第 10 期: 1235~1247
http://www.scichina.com

《中国科学》杂志社
SCIENCE IN CHINA PRESS

基于中间件的软件系统特征交互问题在线检测与消除

黄 罡* 刘譞哲 梅 宏

(高可信软件技术教育部重点实验室, 北京大学信息科学技术学院, 北京 100871)

摘要 作为主流的 Internet 软件系统运行支撑平台, 中间件必须支持越来越丰富的交互模式. 交互的多样和复杂以及 Internet 开放、动态、多变等特点导致非预期交互产生的几率逐渐增加. 这些非预期的交互可能导致服务降级, 功能损失, 甚至系统崩溃等严重后果. 文中从电信领域特征交互问题的角度研究 Internet 软件中的非预期交互问题, 并提出了一个基于中间件的在线检测与消除方法. 首先, 对中间件使能的系统交互进行分类, 与电信系统进行多角度的比较, 并分析了 4 个实例, 从而论证了 Internet 软件特征交互问题的存在并考察其与电信系统特征交互问题的异同. 随后, 提出了以运行时软件体系结构为核心的在线检测和消除方法. 该方法在 J2EE 环境中得以实现并成功解决了 4 个实例.

关键词 特征交互 中间件 软件体系结构 反射

0 引言

目前主流的中间件技术, 如 CORBA (common object request broker architecture), COM (component object model), J2EE (Java 2 platform enterprise edition), SOA (service oriented architecture)等, 均提供了一组服务以简化分布式系统的开发、部署和管理. 随着 Internet 技术的持续发展及其应用的日益普及, 作为主流的分布式系统支撑平台, 中间件得到了极大的繁荣: 首先, 中间件封装了计算资源管理等传统分布式操作系统的主要功能; 其次, 尽管中间件强调提供较为通用的支持机制, 但越来越多特定于应用领域的机制也由中间件提供, 如金融、电信和零售等; 最后, 中间件开始对分布式系统的开发和部署予以支持, 如构件模型和部署工具等.

不论中间件自身功能如何发展演变, 其主要目的还是为了支持越来越复杂、多样的交互操作. 传统的基于中间件的系统大多运行于静态、封闭的环境之中, 中间件仅支持简单、基本的

收稿日期: 2006-09-12; 接受日期: 2007-07-11
国家重点基础研究发展规划(批准号: 2002CB31200003)、国家自然科学基金(批准号: 60233010, 90612011, 90412011, 60403030)及 IBM 大学合作项目资助
* 联系人, E-mail: huanggang@sei.pku.edu.cn

交互，略为复杂的交互由应用开发人员设计实现，即开发人员规划和控制所有的交互情形，因此，非预期的交互几乎不会发生，且其副作用也非常有限. 但是，随着中间件的日益繁荣，分布式系统之间的交互更加频繁和复杂，基于中间件系统的交互也变得不可预期且难于控制. 非预期交互出现的几率和频率不断增加，而这些交互所带来的消极影响，如服务质量降级、功能缺失，甚至整个系统崩溃也不可避免.

基于中间件的分布系统中出现的非预期交互和电信系统中出现的特征交互问题(feature interaction problem, FIP)[1,2]非常类似. 尽管在软件领域中，这些交互问题已经得到较为全面深入的研究，产生了不同的术语或概念，如基于构件系统中的端对端集成测试问题 [3]，基于策略系统中的策略冲突问题等 [4]，但我们认为，从特征交互的角度研究这类问题必要且可行. 自从1989 年Bowen等人正式提出FIP以来，许多研究者对电信领域的FIP进行了认真深入地研究，如问题和解决方案的分类法、交互问题的起因分析、生命周期中各个阶段的检测和解决方法等. 这些丰富的研究成果和经验可以帮助我们解决电信领域之外所出现的类似的交互问题 [5,6]. 更为重要的是，“三网合一”(Internet、电信网和电视网)正成为一种趋势，因此从一个统一的角度来研究交互问题显得尤为重要.

本文将从特征交互的角度来研究中间件系统中的交互问题，并提出一种在线检测和消除方法. 本文其余部分按如下章节组织: 第 1 节简要回顾中间件繁荣导致的分布系统交互复杂化和多样化; 第 2 节从体系结构、开发、演化和资源管理 4 个角度，考察了基于中间件的系统和电信系统的异同，并通过 4 个实例说明了基于中间件系统中特征交互问题的存在; 第 3 节提出一种基于反射式中间件的在线检测和消除方法，并在 J2EE 应用服务器上予以实现; 第 4 节用该方法检测并消除了一组特征交互问题实例; 第 5 节介绍相关工作; 第 6 节总结本文的主要贡献并展望下一步的工作.

1 中间件使能的交互

如图 1 所示，中间件通过提供服务的方式来调用底层系统软件的功能，如操作系统、网络协议和数据库管理系统，来使能由不同编程语言开发并在不同机器上运行的应用之间的交互. 根据服务器数目的不同和消息顺序的不同，中间件使能的交互可以分为以下几种:

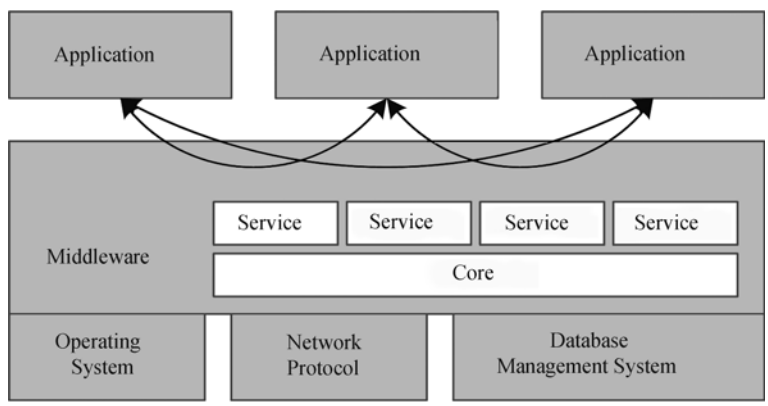

图1 基于中间件的系统结构

1) “无序”单服务器模式: 即客户端同步或异步地发送请求消息到单个服务器. 这是分布

式系统最为常见的交互模式, 也是中间件支持的最基本的交互模式. 目前主流的中间件均支持此种交互模式, 如 CORBA 的 IIOP 和 AMI, J2EE 的 RMI 和 JMS, COM 的 RPC, 以及 SOAP 等等.

2) “有序”单服务器模式: 业务逻辑经常表现为以固定时序关系编排的一组请求. 中间件通常以会话(session)的方式支持这种交互模式, 例如 CORBA 的任务和会话管理服务, 以及 J2EE 中的会话 EJB. 但是, 这种时序关系大多隐含在业务逻辑中, 容易造成开发者的误解或者交互双方的失配. 因此, WSCI 以标准化的方式显式规约了两个 Web 服务之间的消息顺序, 并通过 Web 服务中间件实施对交互顺序的约束.

3) “无序”多服务器模式: 在此交互模式下, 一条请求消息被发送到多个服务器进行处理. 中间件一般通过事件服务来支持此交互模式, 主要形式有发布/订阅机制和集群机制, 如 CORBA 事件服务和 Java 消息服务. 这些请求消息之间不存在顺序关系. 在某些情形下, 当且仅当所有给定消息都已产生时, 这些消息才作为一个整体传送到服务器, 例如 CORBA 的通知服务, 这些消息之间存在特殊的关联, 但并不属于时序关系.

4) “有序”多服务器模式: 一般而言, 一个软件系统的使用可以通过场景来描述. 一个应用场景包括多个构件以及它们之间的交互. 这种复杂的交互关系一般使用设计制品(如 UML 的协作图)而不是中间件来描述. 但是, 当需要在运行时刻按需集成应用时, 中间件必须对该交互模式提供相应支持. 例如, BPEL 允许制定一组执行工作流, 并通过中间件在运行时刻根据指定顺序来调用服务.

值得一提的是, 如果考虑交互的语义, 如功能性需求或非功能性需求, 上述分类将会变得更加复杂. 主流的中间件大多同时支持事务、安全、持久性以及其他常见的非功能属性, 而这些属性可能隐含着某种交互或对交互产生重要的影响. 例如, 事务服务可以用于一个(或多个)服务器的一个(或多个)方法; 访问控制服务则通常只部署在单服务器上, 但在某些情形下也会在特定安全域内控制多个服务器; 而持久性服务则隐式地支持多个应用之间与数据有关的各种交互.

目前主流的中间件产品支持上述所有的交互模式, 这意味着不论大型还是小型的分布式系统中, 复杂、多样、中间件使能的交互大量存在. 以 PKUAS(一个符合 J2EE 规范的应用服务器)为例, 如图 2 所示, PKUAS 不仅提供 Internet 网络协议, 如 TCP/IP, HTTP, HTTPS 和 SOAP, 也提供 Intranet 网络协议, 如 IIOP, RMI-IIOP 和 RMI-IIOP-SSL 等. 为了优化运行于同一虚拟机的多个构件之间的交互, PKUAS 提供了 EJBLocal (所有参数均为引用传递且无需编解码) 和 RMI-IntraVM (除接口外所有参数均为值传递且无需编解码). PKUAS 提供的 Java 消息服务支持异步组交互, JDBC 支持构件与后端数据源之间的交互, JCA 支持 J2EE 构件和其他企业级应用之间的交互(如企业资源管理系统和客户管理系统等). 此外, PKUAS 还提供多种服务以处理非功能属性, 包括 JTS (Java transaction server, 支持事务)和 JAAS (Java authentication and authorization server, 支持认证授权)等.

2　基于中间件的软件系统中的特征交互问题

与其他领域相比, 电信领域的某些特性使其特征交互问题尤为突出和重要. 而基于中间件的软件系统也存在类似的特性, 这意味着特征交互问题也将随着中间件的繁荣而日益显著. 本章将从体系结构、开发、演化和资源管理 4 个角度来考察中间件系统和电信系统之间的异

同, 并给出 4 个中间件系统特征交互问题实例, 这些实例将用第 4 章提出的在线方法来检测和消除.

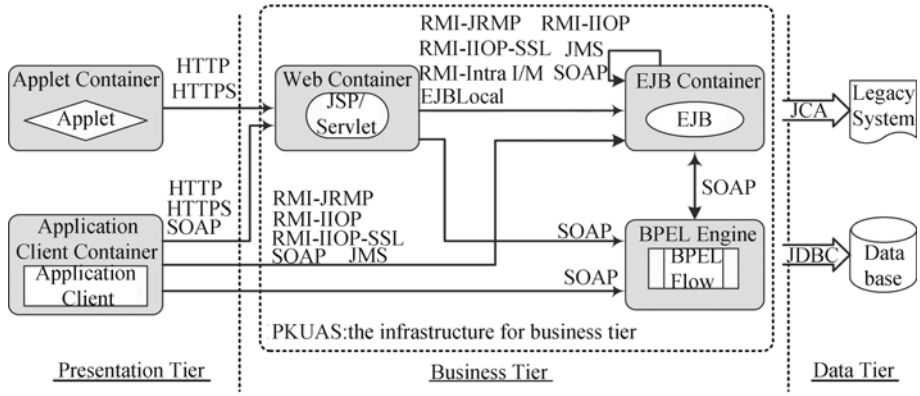

图2 PKUAS 中支持的交互模式

2.1 概念证明

如图 1 所示, 从体系结构上看, 基于中间件的系统是一种层次化的结构. 其中, 中间件核心负责管理运行于中间件上的各个构件, 完成消息传递等功能. 这种结构和传统的公共交换电话网络(PSTN)非常类似: 中间件服务和应用是系统的基本功能单元, 电信系统中则以特征为基本功能单元. 更为重要的是, 与电信系统类似, 中间件系统也具备开放、持续增长、不能停机和遍布全球等特点. 这些特点被认为是特征交互问题的基本成因.

从开发过程上看, 基于中间件的系统开发也与电信系统类似. 在电信系统中, 由于业务需要, 新的服务不断加载到电信系统中, 使其服务数量飞速增加; 同时, 出于市场竞争的压力, 成千上万的服务提供商不断开发新的服务且仅关注自己所提供的服务. 基于中间件的系统同样如此. 例如, 面向服务的体系(SOA)允许组织或个人为网络上的所有用户提供服务. 因此, 基于中间件的系统及其提供的服务都在一种并行、独立、增量式的模式下进行开发. 由于需求的不完全和不一致、假设冲突、设计不当、实现失当和测试不足等原因, 很容易出现特征交互问题. 在基于中间件的系统中, 特征交互问题的表象与后果较易感知, 但其根本的产生原因则是深层次的, 难以确定.

从维护和演化看, 电信系统要求每周 24 h×7 d不间断的服务, 中间件的典型应用领域, 如金融、军事、银行和保险系统等, 同样要求不间断的服务. 因此, 一旦发生特征交互问题, 电信系统和中间件系统都不可能停机重启以实施给定的解决方案. 目前, 在线维护与演化是下一代中间件的研究热点之一 [7], 这种新能力将成为本文特征交互问题检测与消除方法的基础.

从资源管理看, 基于中间件的系统中的资源竞争较电信系统更为严重. 许多电信系统的特征交互问题都源于资源冲突, 如忙时呼叫等待(CWB)和无条件呼叫转移(CFU)之间的冲突, 就是因为二者对同一信道的占用. 但是, 电信系统中的资源一般都是物理资源(如交换机), 而基于中间件的系统中的资源则是虚拟的(如事务服务、日志服务等). 虚拟资源往往可以从不同角度和层次上来定义, 因此虚拟资源的冲突就更加难以检测和消除. 特别地, 一个基于中间件的系统可能同时共享物理资源(如 CPU、内存和网络带宽)和虚拟资源(如线程、内存堆和连接数等).

2.2　实例证明 1: 截取器导致的特征交互问题

截取器是构造松耦合系统的常见设计模式之一. PKUAS使用截取器来动态增加、删除和替换服务, 包括日志服务、消息服务、事务服务、安全服务和持久性服务等[8], 如图 3 所示.

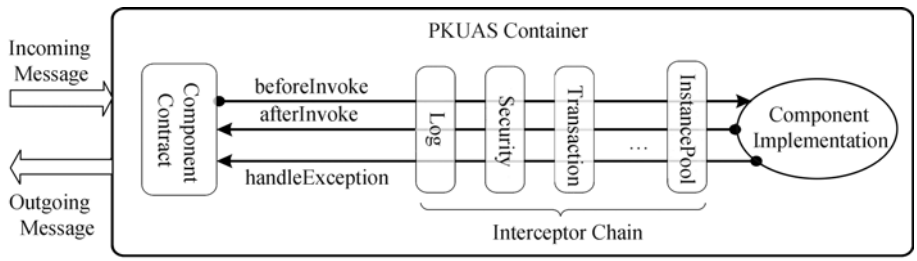

图3　PKUAS 中的截取器链

在运行时刻, 截取器的串行处理可能导致特征交互问题. 以优先级截取器(PI)和并发截取器(CI)为例, 如果两种截取器都安装于同一服务器, 且 CI 先于 PI 加载, CI 将从一个有限大小的线程池中为所有接收到的请求分配线程. 若当前线程池无可用线程分配, 当一个优先级高的请求到达时, CI 只有等到已分配的线程被释放时才能为这个请求分配线程. 此时, PI 只有等到 CI 为请求分配线程时才能正常工作, 从而产生优先级翻转. 因此, CI 先于 PI 加载, PI 的功能将被破坏. 类似的特征交互问题也存在于验证截取器(AI)和日志截取器(LI)之间. 若 AI 先于 LI 被加载, 所有验证未通过的请求都会被 AI 直接拒绝, 导致 LI 记录所有到达请求的功能无法实现.

上述特征交互问题出现的原因在于: 不同截取器提供商之间存在需求假设的不一致. 此类问题在分布式特征组装(distributed feature composition, DFC)中进行了深入研究[9], 本文将介绍如何在线检测和消除此类问题.

2.3　实例证明 2: 实体构件之间的特征交互问题

实体构件具有持久性属性, 这些属性通常存储在关系型数据库中, 由对象/关系映射中间件(如J2EE中的实体EJB容器和Hibernate)进行管理. 当一个后台数据同时与多个前台应用构件发生关联, 而这些构件由不同组织或开发者开发和部署时, 容易出现由于这种隐式的数据依赖导致的特征交互问题[10].

例如, JPS 和 JST 是两个分别开发的 J2EE 蓝图程序. 当它们被部署在同一个企业中时, JPS 中的 ContactInfoEJB 和 JST 中的 AccountBean 将被映射到同一个数据库表中(暂取名为 CustomerTB), 如图 4 所示. 这两个 EJB 的主键和数据库表的主键分别为 UserID, UserName 和 URID. CustomerTB 表中的一项记录可以被 JPS 删除, 但该删除操作是 JST 所不能接受的. 在传统的数据库为中心的系统中, 往往由数据库管理员来进行辨识和控制这些全局化的数据依赖. 但是在基于中间件的分布系统中, 对象/关系映射把应用的语义从数据库系统中剥离出来, 导致数据库管理员无法获知全局数据视图, 难以对数据依赖进行全局控制.

实体构件之间的特征交互问题出现的原因在于: 缺乏实体构件的全局视图和对它们之间隐含的数据依赖关系的辨识. 我们在文献[10]中研究了此类问题. 本文将从在线的角度讨论解决方案.

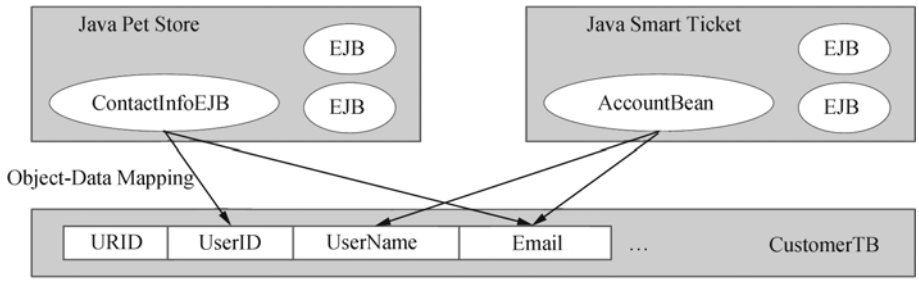

图4 实体构件间的数据依赖

2.4 实例证明 3: 自治计算能力之间的特征交互问题

中间件的繁荣使得中间件愈加复杂和难以被人管理. 因此, 给中间件赋予自治计算的能力成为下一代中间件研究的热点 [11]. 例如, 中间件可在无人工干预的情形下在运行时刻自动管理自身. 自治中间件的一般技术途径为: 给中间件安装一组可完成自治计算功能的实体, 如运行时刻数据采集、统计、规划和决策管理, 以实现中间件的自治管理功能. 显然, 这些自治计算实体将占用系统的物理资源和虚拟资源. 当工作负载超过一定限度, 自治计算实体和中间件中的其他构件或应用将对有限的资源产生竞争, 系统的性能也将明显降低. 从特征交互的角度看, 自治计算实体可被看做是系统的附加特征, 而资源竞争导致了非预期的交互.

例如, 在PKUAS上可以安装响应时间监测器(可用于自动协调响应时间和吞吐量 [11])和日志导出设施(可用于统计、审计等)两个自治计算实体. 如图 5 所示, 在J2EE性能基准程序中, 当工作负载为 8 txRate(即同时有 80 个并发客户端), 响应时间将超过 2 s, 这意味着事务服务失败. 而当日志导出设施被卸载, 响应时间将少于 2 s. 这说明该自治计算能力造成了业务功能的部分失效.

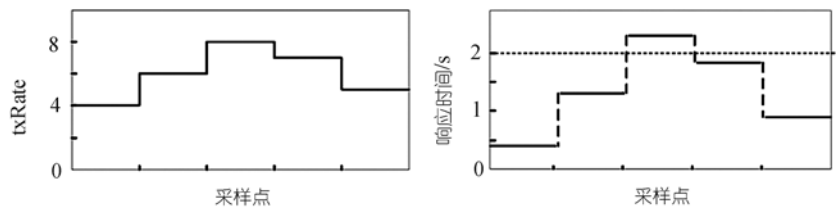

图5 ECPerf 中吞吐量和响应时间的变化

2.5 实例证明 4: 构件实例间的特征交互问题

中间件通常以容器的方式管理构件的生命周期, 如加载实现类, 创建、释放、挂起和激活类的实例等. 和电信系统中由一个特征的不同实例引起的特征交互问题类似, 构件实例之间也会因为资源竞争(如线程、堆和数据库连接)而引发交互问题.

构件的动态更新是一种常见的中间件机制, 如图 6 所示. 当对构件进行更新时, 中间件把所有接收到的请求缓存在一个缓冲池中, 初始化新的构件版本, 将旧版本的状态复制到新版本, 然后把缓冲池中的请求重定向到新的构件版本中去, 最后释放旧版本. 整个过程在语法层次上的正确性可以通过形式化的方法加以保证 [12]. 但是, 中间件不可能保证语义层次上的正确性, 尤其是新构件版本同旧版本客户端之间可能产生特征交互问题. 例如, PKUAS可以动态

更新一个处于会话状态或者事务中的构件, 如前所述, 会话和事务往往隐含了某种构件之间的交互关系. 当构件被替换掉, 对应的会话和事务也可能失效. 对于长时间会话和长事务来说, 由动态构件更新所导致的非预期交互的后果极为严重.

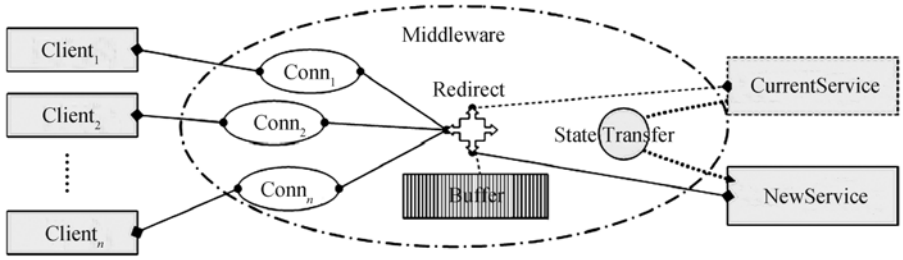

图6　构件的动态更新

3　基于反射的在线方法

作为处理特征交互问题的最后防线, 在线方法无需修改当前运行系统, 较好地支持了系统的开放性和动态性 [1,2,6]. 由于分布系统中出现的特征交互问题同中间件这种运行基础设施密切相关, 因此, 在线方法是检测和消除基于中间件的分布系统特征交互问题的有效途径.

3.1　方法框架

我们提出一种基于反射式中间件和软件体系结构的在线方法, 如图 7 所示, 它由 5 个主要活动组成:

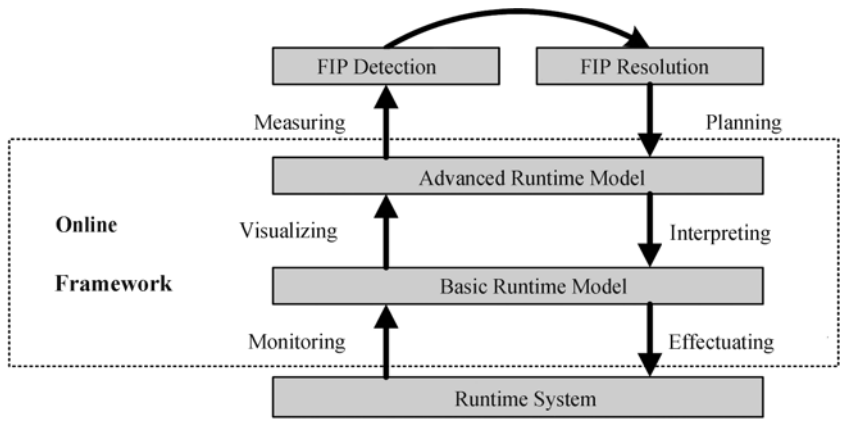

图7　特征交互问题的在线检测和解决

1) 运行时刻系统监测: 运行系统特征交互问题的检测首先需要从系统中采集运行信息, 一般而言, 采集信息越丰富, 在线检测越有效. 与电信系统中内嵌在线检测技术类似 [6], 反射式中间件也内嵌了丰富的监测机制, 可以对所有基于中间件的分布系统进行动态监测.

2) 语法和语义可视化: 在上一步中所采集的原始信息往往缺乏足够的语义或知识来检测特征交互问题, 因此, 需要给这些信息加入丰富的语义信息, 组织成一个全局化且易理解的视图.

3) 检测并消除特征交互问题: 基于上面两个步骤所得到的运行时刻信息, 由人或者一些

智能程序来测量和分析运行系统的情形, 检测特征交互问题或者可能导致特征交互问题的潜在危险, 并尽可能消除或预防这些问题. 为了避免失配或误解, 特征交互解决方案所使用的模型应与问题检测中使用的一致. 本文提出的在线方法中, 软件体系结构是最主要的语义模型, 解决方案主要体现为体系结构层次上的动态调整, 如构件或连接子的增、删、改, 修改约束条件等.

4) 解决方案的转换: 解决方案往往表现为高层模型的调整, 这些变化必须转换为底层运行系统的基本模型, 从而保证给定的调整得以正确实施. 这种解决方案自顶向下的转换可以视为检测模型自底向上的构造的逆向过程.

5) 运行系统的实际调整: 最终, 解决方案通过运行时刻系统的调整来实施. 需要注意的是, 某些将要发生的改变可能会被阻止, 以预防产生潜在的特征交互问题. 例如, 可通过禁止截取器的增加或删除以杜绝截取器变化导致的特征交互问题.

一般而言, 无需人工干预、通用的特征交互问题检测和消除难以甚至无法实现. 本文则提出了一种在线检测与消除框架, 以部分或完全自动实施上述步骤, 确保管理员将精力集中于语义分析、模型推理等必须由人主导的活动.

在线框架存在两个基本的技术挑战: 其一, 如何监测和调整运行系统? 反射式中间件是一种理想的解决途径, 它可以检测并动态调整中间件平台及部署于其上的应用的状态和行为 [7]; 其二, 如何可视化运行时刻信息, 并加入丰富的语法信息帮助理解? 软件体系结构是一种可行途径, 它描述了一个软件系统的总体结构, 使设计人员可以从更高层次来设计和分析系统的功能与质量 [13]. 更进一步地, 中间件使能的交互可以由软件体系结构捕捉 [14], 解决特征交互问题时对系统的调整也可由软件体系结构预测和规约 [15]. 上述两个技术挑战将由运行时软件体系结构同时解决 [16].

3.2 运行时软件体系结构

运行时软件体系结构(runtime software architecture, RSA)可以从软件体系结构的角度对运行系统进行描述和操控. RSA 的体系结构元素与运行系统的内部状态及行为之间存在因果关联, 即 RSA 的任意改变会立即导致运行系统进行相应的调整, 反之亦然. 这种因果关联由反射式中间件维护. 如图 8 所示, 在 PKUAS 反射框架中, RSA 的体系结构元素均实现为元层实体(meta entities).

RSA 分为平台和应用两种: 在平台 RSA 中, 构件和连接子表示中间件平台的细节信息, 应用只能通过其他方式(如构件的属性)来表示. 例如, J2EE 应用服务器是由若干容器和服务组成, J2EE 应用则是由一些 EJB 和 Servlet 组成, 在平台 RSA 中, 容器和服务都表示为构件, 它们之间的交互和依赖则表示为连接子, EJB 构件和 Servlet 则表示为容器的属性. 与之相反, 应用 RSA 把中间件应用表示为构件和连接子, 而中间件平台则表示为一些属性或者约束. 例如, J2EE 的安全和事务服务表示为对 EJB 构件安全和事务的约束.

RSA 框架和中间件往往由同一个开发商设计实现, 因此平台 RSA 可以提供精确的语法信息和足够语义信息来描述中间件平台的状态和行为. 另一方面, 尽管应用 RSA 可以通过对整个系统进行追踪和体系结构恢复, 却很难获得足够的语义信息, 这些语义信息需要解析运行系统的设计制品, 如软件体系结构描述.

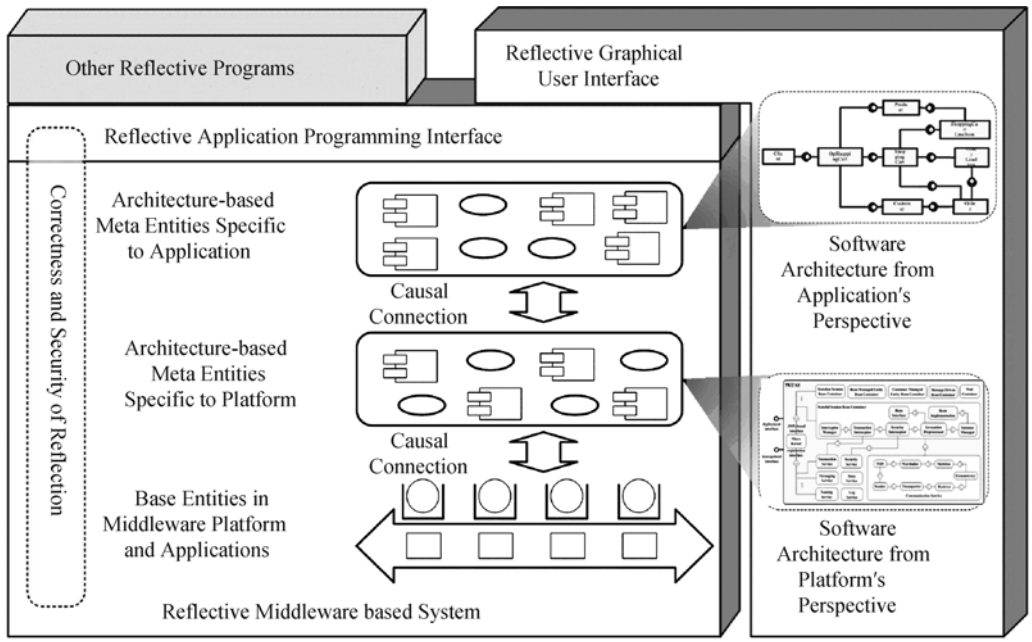

图8　PKUAS 中基于运行时刻软件体系结构的反射框架

RSA带来的性能损耗在多个实际应用中被证明是可接受的, 如重负载和复杂事务系统中RSA降低 3%的性能 [11], 低负载和简单事务系统的损耗则为 8%[17]. 关于RSA的更多细节及其PKUAS实现, 如编程模型、可视化用户界面和安全体系等, 在文献 [16, 17]中有所讨论. 本文仅讨论如何使用RSA来检测和消除特征交互问题.

4　实例研究

本章将介绍如何使用RSA来检测和消除前面提到的 4 个特征交互问题实例. 需要说明的是, 这些实例均源自我们此前的研究与实践 [10~12,18], 本文从特征交互的角度研究这些实例, 进而验证在线方法的可行性和有效性.

4.1　实例研究 1: 截取器导致的特征交互问题

检测截取器的特征交互问题通常可在两个时刻进行. 其一是在特征交互问题发生的时刻. 例如, 系统管理员可能发现很多请求在它们失效前都不能完成, 此时将监测优先级截取器. 由于截取器是中间件平台的一个实体, 系统管理员可以考察平台 RSA, 如图 9 所示, 可以清晰地看到, 并发截取器安装在错误的位置, 导致优先级截取器失效. 另一个时机是在加入新的截取器时. 例如, 如果系统管理员准备在鉴权截取器之后的任何位置加入日志截取器, 可以发现上文所描述的特征交互问题.

一旦特征交互问题被检测到后, 可以通过重新安装截取器来解决(如并发截取器和优先级截取器的交互), 或者重新规划截取器的增加(如验证截取器和日志截取器的交互), 甚至删除一个截取器或拒绝新截取器的加入. 上述对截取器的调整均可通过 RSA 在线实施.

4.2　实例研究 2: 实体构件之间的特征交互问题

J2EE 规范中的两种数据依赖可能引起数据冲突. 一种是两个构件和同一个数据库的同一

张表发生关联，此时一个构件对某一列的修改可能也改变了另一个构件的属性．另一种是构件 A 和 B 拥有持久关联，而构件 B 的持久关联属性同时又是构件 C 的持久属性．此时，构件 C 修改的数据列也将影响构件 B 的属性， 并间接影响构件 A．上述交互究竟是预期的还是非预期的，只能通过应用语义来决定，而相应的解决方案也特定于应用的语义．

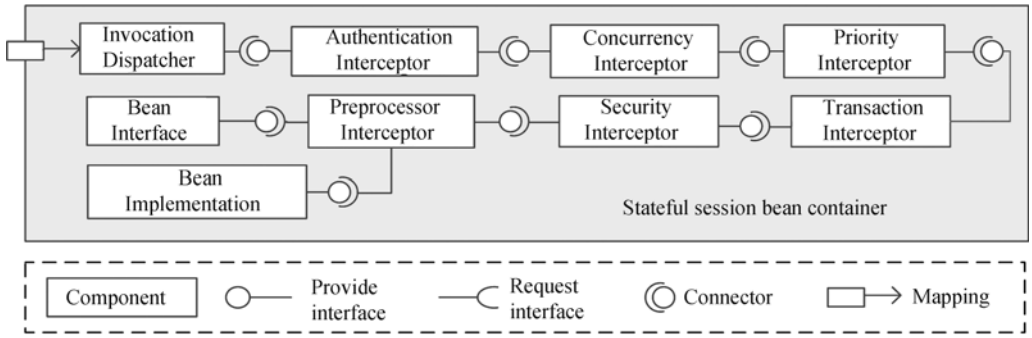

图9 平台 RSA 可视化的截取器视图

对实体构件交互实例，可以通过中间件的相应机制自动检测到数据依赖的存在，并由应用 RSA 表示，如图 10 所示．其中，数据库表示为一个复合构件，数据库表则表示为复合构件的内部构件，EJB 和数据库表之间的关联被表示为连接．当 ContactInfoEJB 和 AccountBean 都同 CustomerTB 发生关联时，将会检测到潜在的特征交互问题．为了解决这个问题，管理员可以通过设置 CustomerTB 的数据完整性以禁止两个 EJB 删除表中的记录，或者加入新的访问控制规则，如任何人都不可调用删除方法．

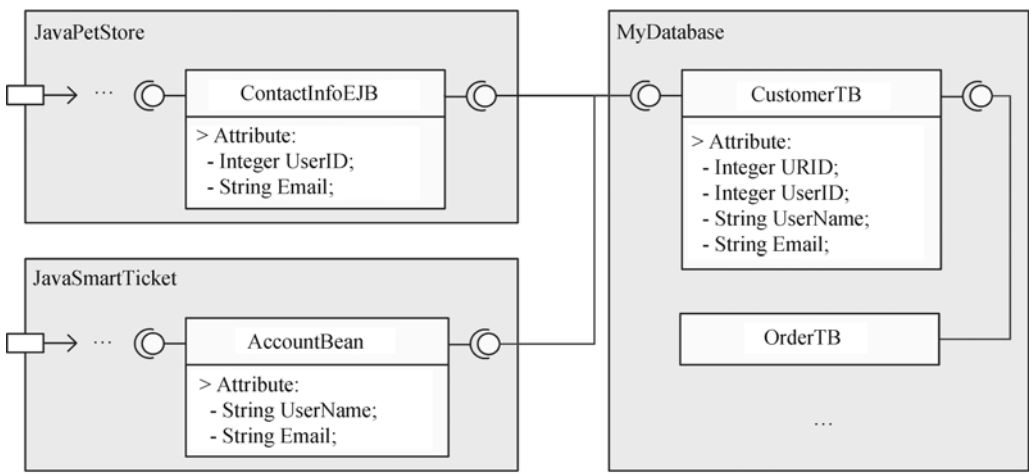

图10 应用 RSA 可视化的实体构件数据依赖视图

4.3 实例研究 3：自治计算能力之间的特征交互问题

中间件的管理往往隐含很多中间件的实现细节，一般方法很难检测到自治计算实体之间的交互，往往需要中间件专家的介入．另一方面，不同的自治计算实体涉及不同的中间件实现细节，因而对不同的自治计算实体之间特征交互问题的检测方法也会有所差别．考虑前面提

到的自治计算实体交互实例, 文献 [11]中所实现的自动权衡机制说明线程池对吞吐量和响应时间的影响较大. 如图 11 所示, 并发服务管理应用对线程池共享, 而此时监测服务和日志服务占用 2 个线程, 应用所能共享的线程数由 10 降到 8, 吞吐量和响应时间受到严重影响. 但是, 仅基于上面的分析, 我们并不能认为线程池资源的竞争就是产生特征交互问题的原因. 此时可以停止日志服务或监测服务中的一个, 若响应时间始终低于 2 s, 则说明日志服务、监测服务和并发服务之间的确存在由于线程竞争导致的特征交互问题.

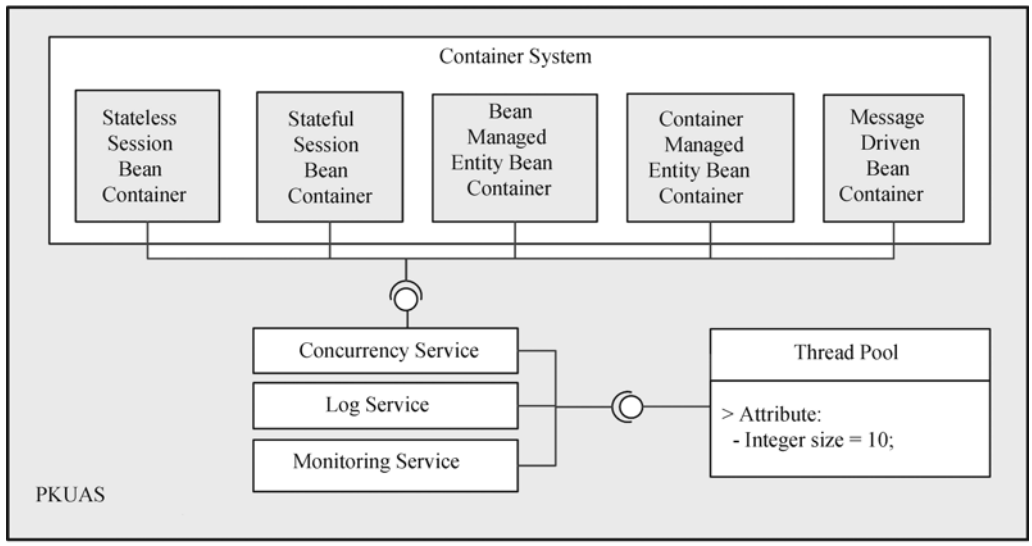

图11　平台 RSA 可视化的线程竞争视图

自治计算实体间的特征交互问题可通过分配更多的资源或者停止次要的计算单元来消除. 在上面的实例中, 当工作负载较高时, 无论是监测服务还是日志服务都可以被停止, 当工作负载降低时再重新启动. 这些调整可以手工完成, 也可以实现为元层实体并安装到反射框架中.

4.4　实例研究 4: 构件实例间的特征交互问题

在线构件更新最为严峻的挑战是构件实例的状态也需要更新. PKUAS中, 一个EJB实例的生命周期分作 5 种状态 [17]. 载入状态表示一个构件的实现类被载入PKUAS但并没有进行实例化; 实例状态表示一个构件实例被创建但并无任何客户关联; 会话状态表示当前实例同一个客户端关联, 不能处理来自其他客户的调用; 当一个会话实例隶属于一个尚未完成的事务时, 则处于事务状态, 相关的事务上下文需要在多次调用之间保持; 服务状态表示实例正在处理一个调用. 处于载入状态和实例状态的构件实例更新仅需直接删除旧版本, 装载新版本实现类, 创建新版本实例即可. 而对会话态和事务态构件实例的更新, 还需将旧版本的状态复制到新版本. 会话上下文和事务上下文由容器自动维护, 无须状态复制, 而构件的内部状态由于新旧版本的属性和参数不尽相同, 无法直接复制.

开发者通常应该了解旧版本中哪些状态应该传送到新版本中对应的状态中去. 当更新一个 EJB 时, 可以通过应用 RSA 考察其实例, 如图 12 所示. 开发者决定哪些实例可以更新, 中间件执行具体的更新操作. 更为复杂的是, 开发者可以在构件更新机制中加入元层实体, 以指明如何复制会话态和事务态构件实例的内部状态, 从而实现自动更新.

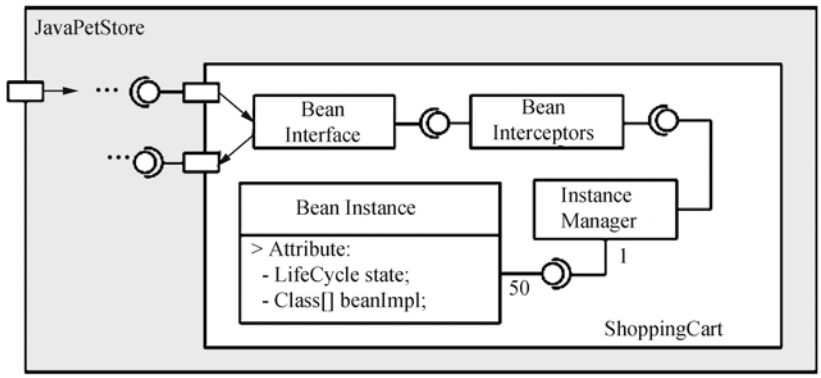

图12 应用 RSA 可视化的构件实例视图

5 相关工作

特征交互问题并不仅限于电信领域. Blair等人 [1)]最先将其引入到中间件领域, 关注如何辨识单个的构件并对它们进行组装, 提出了一种在设计阶段和运行阶段特征交互问题检测方法, 并设计了一种基于AOP的消除方案. 他们的核心思想是进行“关注点分离”, 用一个 2 层模型来对构件核心功能和Aspect进行封装. 同时, 引入反射式中间件以实现系统的自适应和重配置. 他们开发的反射式COM中间件OpenORB与PKUAS非常类似, 他们与我们在中间件特征交互问题的观点也相近. 但是, 他们仅对中间件特征交互问题进行初步探讨, 而我们则深入研究了更多的特征交互问题实例, 并形成了一个较为通用的在线检测和消除方法.

分布式特征组装(DFC)中的特征交互问题 [9]与PKUAS截取器特征交互问题类似. Jackson 和 Zave将电信系统视作由不同特征所组成的复合系统, 并使用DFC来管理特征交互问题. DFC是一种“管道-过滤器”体系结构风格, 其核心是一个虚拟体系结构, 所有特征在这个体系结构中都有对应的构件类型. 在DFC中, 每个构件实现为两种构件类型, 每一个来自外部的呼叫都由动态组装而成的一组构件处理, 以保证特征构件间彼此独立, 进而有效控制特征交互问题的产生.

作为“下一代的中间件”, 面向服务的体系结构(SOA)主要支持Internet上的巨型系统的开发和管理. SOA通过标准的方式(如XML)将松耦合的构件以服务的形式组装在一起. SOA中也会出现特征交互问题. 由于Web服务已经呈现出比现有中间件服务更为多样和复杂的交互模式, Web服务中的特征交互问题引起了研究者的关注. Weiss[19]分析了Web服务的非功能属性, 并提出一个面向目标的方法来管理Web服务间的特征交互, 同时也考虑到业务环境变化所引起的特征交互问题, 并对动态适应方案做了初步的研究. 这些工作启发我们将反射机制引入到SOA中 [20], 并将本文所提出的在线方法扩展到SOA特征交互问题的检测与消除.

6 结束语

大规模、增量分布开发、分散式系统大多面临特征交互问题的困扰, 此类问题往往难以检测和消除. 本文研究了基于中间件的软件系统的特征交互问题. 通过对比中间件系统和电信系统的相似性以及 4 个具体实例, 说明了基于中间件的软件系统中存在大量的特征交互问题.

1) Blair L, Blair G, Pang J. Feature interaction outside a telecom domain. In: Proceedings of International Workshop of Feature Interaction in Composed Systems, 2001. 15—20. http://www.comp.lancs.ac.uk/~efstrati/publications/content/FICS01.pdf

针对中间件系统的特点, 提出了一种在线检测与消除方法, 利用运行时软件体系结构分析特征交互问题并规划解决方案, 利用反射式中间件实时监测和调整运行系统, 在 J2EE 中实现了相应的支撑框架, 并成功检测和消除了 4 个特征交互问题实例.

参　考　文　献

1 Kech D, Kuehn P J. The feature and service interaction problems in telecommunications systems: A survey. IEEE Trans Softw Eng, 1998, 24(10): 779—796 [DOI]

2 Chi C X, Hao R B, Huang G, et al. Feature Interaction: A Survey. Bell Labs Technical Report, March 17, 2003

3 Mei H. End-to-end integration testing in CBSD. In: Proceedings of Annual International Computer Software and Applications Conference (COMPSAC), Washington: IEEE CS, 2001. 284—285

4 Wang D, Hao R B, Lee D. Fault detection in rule-based software systems. Elsevier Information and Software Technology, 2003, 45(12): 865—871 [DOI]

5 Pulvermüller E, Speck A, Coplien J O, et al. Feature interaction in composed systems. In: Proceedings of the Workshops on Object-Oriented Technology. Berlin: Springer, 2001. 86—97

6 Calder M, Magill E, Kolberg M, et al. Feature interaction: A critical review and considered forecast. Int J Telecommun Comput Netw, 2003, 41(1): 115—141

7 Agha G. Adaptive middleware: Introduction. Commun ACM, 2002, 45(6): 30—32 [DOI]

8 Mei H, Huang G. PKUAS: An architecture-based reflective component operating platform. In: Proceedings of 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS). Washington: IEEE CS, 2004. 163—169

9 Jackson M, Zave P. Distributed feature composition: A virtual architecture for telecommunication services. IEEE Trans Softw Eng, 1998, 24(10): 831—847 [DOI]

10 Teng T, Huang G, Li R, et al. Feature interactions induced by data dependencies among entity components, In: Proceedings of 8th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI). Amsterdam: IOS Press, 2005. 252—269

11 Huang G, Liu T, Mei H, et al. Towards autonomic computing middleware via reflection. In: Proceedings of 28th Annual International Computer Software and Applications Conference (COMPSAC). Washington: IEEE CS, 2004. 122—127

12 Shen J, Sun X, Huang G, et al. Towards a unified formal model for supporting mechanisms of dynamic component update. In: Proceedings of 5th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE). New York: ACM, 2005. 80—89

13 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. New Jersey: Prentice Hall, 1996

14 Zhu Y, Huang G, Mei H. Modeling diverse and complex interactions enabled by middleware as connectors in software architectures. In: Proceedings of 10th IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS). Washington: IEEE CS, 2005. 37—46

15 Zhu Y, Huang G, Mei H. Quality attribute scenario based architectural modeling for self-adaptation supported by architecture-based reflective middleware. In: Proceedings of Asia Pacific Software Engineering Conference (APSEC). Washington: IEEE CS, 2004. 2—9

16 Huang G, Mei H, Yang F Q. Runtime software architecture based on reflective middleware. Sci China Ser F-Inf Sci, 2004, 47(5): 555—576

17 Huang G, Mei H, Yang F Q. Runtime recovery and manipulation of software architecture of component-based systems. Int J Autom Softw Eng, 2006, 13(2): 251—278

18 Liu X, Huang G, Zhang W, et al. Feature interaction problems in middleware services. In: Proceedings of 8th International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI). Amsterdam: IOS Press, 2005. 313—319

19 Weiss M. Feature interactions in web services. In: Proceedings of Feature Interactions in Telecommunications and Software Systems (ICFI). Amsterdam: IOS Press, 2003. 57—68

20 Huang G, Liu X, Mei H. SOAR: Towards dependable service-oriented architecture via reflective middleware. Int J Simul Process Model, 2007, 3(1/2): 55—65 [DOI]