一种 Java 遗留系统服务化切分和封装方法

李 翔 怀进鹏 曾 晋 高 鹏
(北京航空航天大学计算机学院 北京 100083)

摘 要 SOA 是一种新型企业应用架构,为复用存在于 Internet 上的软件资源提供了一个最佳实践.面对目前可用的服务资源匮乏,同时大量企业信息系统需要借助服务计算技术重组优化的现状,从遗留系统中切分并封装成可复用的服务是实现 SOA 的关键.目前以人工方式的遗留系统服务化切分和封装过程效率较低且难以保证质量,亟需一种自动化手段辅助开发人员实施这一过程.文中针对 Java 语言的遗留系统,研究自动化的遗留系统服务化切分和封装技术.在综合静态类结构模型和动态对象调用模型的基础上,提出了一个遗留系统的对象依赖频度图表示模型,并基于面向服务的切分目标设计了一种服务模块自动识别和切分的有效方法,并利用 Java 语言的执行码重写技术实现了自动化的服务封装工具,最终通过实际的应用案例验证了文中工作的有效性.

关键词 遗留系统;SOA;软件分析;软件切分;Java
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2009.01804

A Service-Oriented Partitioning and Encapsulation Method for Java Legacy Systems

LI Xiang HUAI Jin-Peng ZENG Jin GAO Peng
(*School of Computer Science & Engineering, Beihang University, Beijing* 100083)

Abstract SOA emerges as a new methodology for software development and system integration through composing existing Web services. However, we face the lack of service resources while a large number of enterprise information systems exist and need to be reorganized with supporting of Web service technologies. Due to identification and partition the service from legacy system by hand are low-quality and time-costing work, an automatic technique is needed. This paper studies the problem of how to automatically partition and encapsulate Java legacy systems. Firstly, for describing object-oriented systems, the authors propose a general model by composing static model for class relationships and dynamic model for object invocations, then, give an algorithm for identifying and partition services. Furthermore, based on java bytecode rewriting technology, the authors introduce a new method for automatically encapsulating the legacy system modules to Web services. Finally, for verifying the work, the authors select a well known java legacy desktop system and successfully transform it to a service-oriented distributed system which can access on internet by multi-users. The case study shows the work provides a promising method and technology.

Keywords legacy system; SOA; software analysis; software partitioning; Java

收稿日期:2009-04-20;最终修改稿收到日期:2009-07-25.本课题得到国家"八六三"高技术研究发展计划项目基金(2007AA010301,2006AA01A106,2009AA01Z419)资助.**李 翔**,男,1977 年生,博士研究生,主要研究方向为服务计算和软件设计与生产.E-mail: lixiang@act.buaa.edu.cn.**怀进鹏**,男,1962 年生,博士,教授,博士生导师,主要研究领域为计算机软件与理论、网络计算技术、信息安全.**曾 晋**,男,1981 年生,博士研究生,主要研究方向为服务计算和软件设计与生产.**高 鹏**,男,1985 年生,硕士研究生,主要研究方向为服务计算和软件设计与生产.

1 引　言

SOA(Service Oriented Architecture,面向服务的体系结构)架构和服务计算技术为企业信息系统的重组、优化和扩展提供了一个最佳实践[1].目前的企业存在大量遗留的信息系统,随着技术和需求的发展,这些信息系统需要不断更新升级.企业信息系统应用 SOA 技术的主要过程是,将企业信息系统中可复用的功能模块封装成 Web 服务,并针对业务需求和环境因素组合这些服务达到快速构建企业应用的目的.目前,企业信息系统服务化封装的过程主要靠人工完成,效率低下,软件质量难以保证.针对这一问题,本文重点研究如何利用软件分析技术和服务化生产技术自动地切分和封装遗留系统,使其快速转化为 SOA 架构的信息系统.

软件构件技术[2]在过去十多年中得到了蓬勃发展,构件的识别、抽取和自动切分技术也受到广泛关注并取得了一定的研究成果.一方面,Web 服务作为 SOA 架构中基本的软件模块也可以看成是一种软件构件,其切分和封装可以借鉴目前已有的技术和方法.然而,Web 服务要求其内部的组成元素是自封闭的,同时具有良好定义的外部接口,并可以通过 Internet 访问,这使得面向服务的软件复用模块抽取技术有别于传统的构件识别和提取技术.另一方面,在过去几年中,面向对象系统特别是 Java 遗留系统的自动分解引起了许多研究者的兴趣,已取得了一些初步的成果①[3-5].然而,SOA 架构下的自动化服务切分和封装的相关研究还较少,同时这一问题带来了多方面的技术挑战:

(1)遗留系统的自身特征.大量遗留系统已经成功运行在线,通常对这些遗留系统的改造、重组和优化过程要求不破坏原有系统的行为,这对遗留系统分解造成困难.目前大量遗留系统缺乏技术文档,设计和开发人员已无法联系,甚至没有源码等,这些问题严重制约了服务化过程;

(2)面向对象技术的复杂性.面向对象方法是目前软件生产过程中最主要的设计和开发技术之一.然而,面向对象技术和方法,如继承、多态和重载等特性,无法直接应用于 SOA 架构;

(3)Java 语言的特性.Java 语言做为目前最广泛使用的语言之一实现了面向对象语言的所有特征,同时也增加了很多独特的特性,如反射、动态类加载、垃圾回收等.这些特性极大地增加了实际 Java 遗留系统的复杂性;

(4)Web 服务技术制约.Web 服务技术有一系列标准规范组成,因此必须考虑分解后的遗留系统必须符合 Web 服务的应用场景和技术规范.Web 服务作为 SOA 架构软件的基本复用单位具有不同于传统软件组件的特性.如何识别和切分遗留系统获取合适的软件模块以形成服务是本文主要面对的问题.

针对这些挑战,本文重点研究 Java 语言遗留系统的服务化切分和封装问题.我们首先分析面向对象软件成分之间的依赖关系和运行过程的行为,并且结合软件复用构件提取技术、切分技术和代码重写技术,最终提出了一种动态的对象级别的遗留系统自动服务化切分和封装方法.本文的主要贡献包括:

(1)分析并总结软件分解的过程和方法,在此基础上针对 SOA 应用场景,提出了一个遗留系统服务化分解和封装过程的框架;

(2)构造了一种描述软件行为和依赖关系的图模型,给出了一个针对面向对象遗留系统的服务化分解算法;

(3)提出了基于 Java 执行码重写技术进行自动化服务化封装的方法,并据此实现了一个服务化封装工具;

(4)最后通过实际的应用案例和实验分析验证了我们方法的有效性.

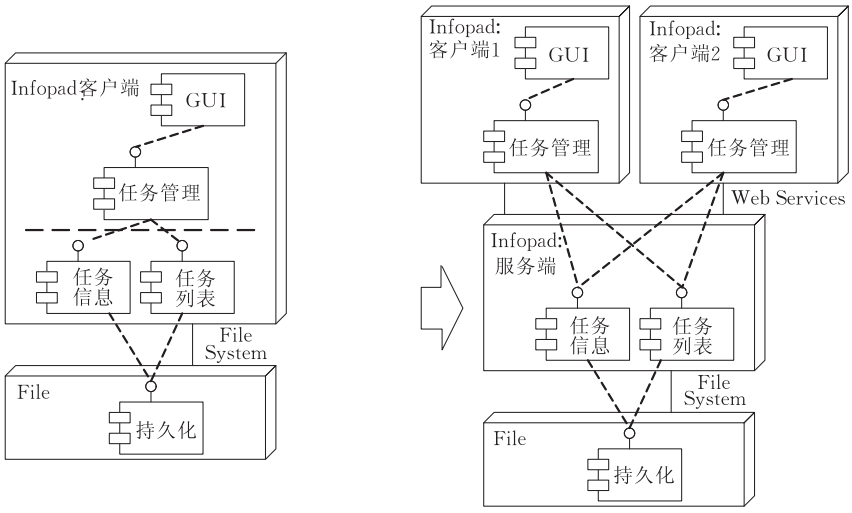
文章第 2 节我们将通过一个简单案例说明遗留系统服务化分解和封装的应用问题,相关的研究工作的分析和比较在第 3 节中给出;第 4 节我们通过分析 Java 遗留系统的依赖关系和行为描述,提出了一个服务化切分和封装框架;作为本文的核心,第 5 节和第 6 节给出了面向对象软件的形式化模型和分解算法,并且介绍了基于执行码重写技术的自动服务化封装技术以及相应工具原型的设计和实现;第 7 节通过应用案例分析和实验分析,对我们方法的有效性进行了验证评估;最后,在第 8 节中对本文工作进行总结和展望.

2 应用案例

本节我们通过一个简单案例说明遗留系统的服

① Dahm Markus. Doorastha: A step towards distribution transparency//Proceedings of JIT. See http://www.inf.fu-berlin.de/~dahm/doorastha/

务化切分和封装的应用场景和意义.个人任务管理软件(To-do List Software)是一类桌面型软件,其功能是管理个人计划任务和便签.图 1(a)给出了这类软件的基本结构.企业用户希望利用 SOA 的理念重构这类软件,使其核心模块可以通过 Web 服务远程访问,并可以接入多个客户端,从而被用作小组任务管理软件.更进一步,如有需求,可以在暴露的 Web 服务接口上扩展其它功能,如任务统计和审核等.重构后的系统结构如图 1(b)所示.



(a) 原系统体系结构 (b) 切分重构后的体系结构

图 1 个人任务管理软件及其 SOA 重构

这一示例虽然简单,它展示的 SOA 重构技术具有可观的应用前景.然而,面对庞大复杂的遗留系统,我们必须解决以下问题:如何保证遗留系统重组前后的行为一致性;如何选择复用模块的粒度;如何切分和重组遗留系统才更为合理;如何将切分出模块自动封装成服务,并保持原系统正常运行等.

3 相关工作

近几年,随着 SOA 架构的广泛应用和 Web 服务技术的发展,大量遗留系统开始采用服务化技术进行重组和改进.文献[6]中 Bisbal 等人评述了遗留系统迁移和集成的问题和研究现状,指出在对遗留系统分析理解的基础上,对遗留系统的切分和封装过程是其服务化迁移的关键步骤.特别是如能自动化完成这一过程将极大地提高生产效率和质量.下面我们就这一过程的相关研究工作从两方面进行分析比较.

首先,对遗留系统的服务化切分可以借鉴传统软件中的构件提取技术,特别是基于软件元素间的全局相关性对系统进行划分的方法[7-11].其中文献[7]主要针对软件系统的结构特性,通过对软件图模型聚类的方法得到软件的局部结构度量指标,以此指导软件构件的提取.文献[8]中利用图的谱分割算法对软件的图模型进行切分,从而可以高效地分解软件系统并从中获得软件构件.然而,这一方法每次只能将软件二分,因此只适用于较为简单的软件系统.文献[9]利用控制树(dominance tree)的算法对大型系统的调用图进行分析,从而可以获得软件系统中构件的层次模型.文献[10]综述了软件构件识别技术,提出了一种构件相似性聚类的新评价指标,并重点关注如何将构件识别技术应用于软件的理解和演化过程.文献[11]主要研究软件构件的正向识别,即从需求模型出发,从中识别出业务构件,进而实现软构件并构造软件系统.以上这些方法的核心思想是分析软件元素之间的关联关系,按高聚合低耦合的原则对软件系统进行划分,从而识别出可以复用的软件模块.文献[12]给出了这类方法的综述.与这些工作比较,我们的工作更多的是考虑服务化切分场景的特性,并详细探讨了这些特性在模型和算法中是如何体现以及如何解决的.

其次,面向对象软件的自动分解和转换技术也是本文工作的一个重要相关工作.面向对象软件切分可以分为如下几类:动态和静态、在线和离线、类级别和对象级别.针对 Java 语言开发的软件,目前已有多个针对于不同领域应用的切分工具(如上一页脚注中提到的 Doorastha 系统及文献 [3-5].其中,Doorastha 系统提供了一个编译器,它可以利用 Java 语言的标签

技术,将 Java 源码编译成为基于 RMI 技术的分布式系统. Pangaea[3] 使用 Doorastha 作为它的后端,并在其基础上扩展了支持对象级别的切分. Addistant[4] 和 J-Orchestra[5] 较为相似,都是类级别的 Java 遗留系统切分工具,可以根据开发人员的配置自切分遗留系统,其底层支撑均使用 RMI 技术. 与上述工作相比,本文的工作主要针对面向服务的遗留系统切分应用场景,采用动态、离线、对象级别的方法解决 Java 遗留系统的服务化自动切分问题.

4　服务化切分和封装过程框架

已有大量工作研究了面向对象系统分解和构件抽取技术[12]. 针对面向服务的遗留系统的系统分解和服务抽取与已有方法类似,但也存在差异. 其过程主要包括以下步骤:

① 遗留系统的分析. 遗留系统中存在大量潜在的可复用模块,必须通过对遗留系统的分析理解才能准确地识别出这些模块. 本文工作中主要包括分析获取面向对象遗留系统的静态类结构信息和运行期对象动态调用信息,并在此基础上通过一个统一模型综合两种信息,从而获得遗留系统的完整模型. 除此,还需对模型进行一定程度的裁减,这是为了降低模型的复杂度,从而加快后续处理的速度;

② 遗留系统切分. 在遗留系统模型的基础上,将遗留系统分解为客户端及可复用的服务模块. 与传统的构件提取技术不同,服务的识别提取必须考虑后续服务化封装过程的技术限制. 比如,服务是可以跨广域网访问的,因此必须考虑服务调用的时间开销对整体系统的性能影响;

③ 服务化封装. 经过识别后的服务需要经过封装形成 Web 服务. 传统封装技术(比如 Axis 项目的工具 Java2WSDL 和 WSDL2Java)只能封装 Web 服务的服务端,还需要在此基础上重构或开发新的客户端,以便使原有系统中的方法调用成为跨越 Web 的服务调用. 然而遗留系统中往往缺乏充足的必要信息,手工完成客户端的改造费时费力. 我们针对 Java 语言遗留系统,利用 Java 执行码重写技术,根据原系统调用过程自动生成服务及其调用代理(客户端),并保持原系统的行为和语义;

④ 软件模块重组和扩展. 经过前三步后系统在保持原有功能和行为特征的基础上暴露出了开放的 Web 服务接口,这使得系统的重组和扩展变的相对简单. 开发人员可以利用 SOA 的生产方法,利用业务流程重新组合服务,并在此基础上快速调整系统结构或为系统增加新功能.

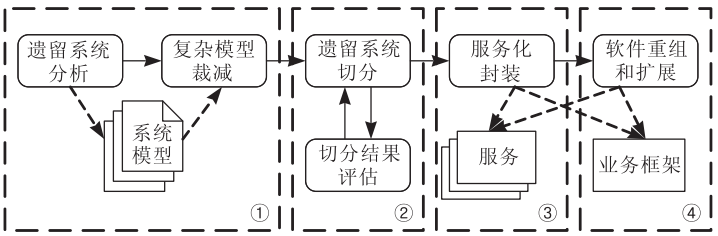


图 2　遗留系统服务化切分和封装过程框架

5　服务模块识别和切分

本文中服务识别和切分的主要针对采用面向对象方法开发的遗留系统,并特定于 Java 程序开发语言. 其基本思路是通过分析遗留系统中类之间以及对象之间的关系,通过图模型进行刻画,并利用基于图模块度的切分算法获取遗留系统服务的候选集,并在此基础上综合服务的接口和性能评价,得到合理的服务划分,该过程如图所示. 以下章节我们具体介绍该过程的实现.

5.1　遗留系统模型

一般来说,软件系统的结构可以简单地看作由基本软件组件和它们之间的关系组成,这种结构非常适合用图模型进行表示. 基于这种考虑,我们将基于面向对象方法开发的遗留系统表示为两种图模型. 首先,我们定义遗留系统的类关系模型.

定义 1.　类关系图(Class Relationship Graph, CRG)是一个有向图,以三元组〈VC, EC, I〉表示,其中 VC 是点集合,每个点代表遗留系统中的一个类; EC 是边的集合,每条边代表两个类之间的关系; I 是将点和边映射到标签的映射函数.

CRG 定义类之间的关系和结构,其中每个点代表一个类,类的描述信息以一个三元组〈$name$, F, M〉表示,其中 $name$ 是类的名称, F 是类的属性集合, M 是类的方法集合. 边代表了类之间的关系,类

之间关系也用一个三元组刻画〈R,FA,MI〉,其中 R 表示两个类之间关系的集合,本文中主要采用 UML 中定义的关系,包括继承、聚合、关联等.为了方便后续分析工作,我们特别抽取了属性访问信息的集合 FA 和方法调用信息的集合 MI.

然而类级别的切分过于粗糙,难以应用于大多数遗留系统,对于本文所述的遗留系统服务化场景的切分必须采用对象级别的切分技术.因此,有必要进一步提供对象级别的模型,这里我们定义了遗留系统的对象调用图.

定义 2. 对象调用图(Object Invocation Graph,OIG)是一个有向有权图,以三元组〈V_O,E_O,W_E〉表示,其中 V_O是点集合,每个点代表遗留系统运行实例中的一个对象;E_O是边的集合,每条边 $e(u,v)$代表对象 u 调用对象 v 的方法或访问 v 的属性;W_E是边的权值函数,权值大小代表了两个对象之间调用的频繁程度.

OIG 描述了遗留系统运行期的动态行为,其中每个点代表一个运行期对象,对象信息通过一个二元组〈ID,C〉表示,其中 ID 是对象的唯一表示,C 是对象所属的类,通过 C 我们可以在 CRG 中找到该对象所属类的详细信息;对象 A、B 之间的一条边代表在运行期 A 调用过 B 的方法,边上的权值代表了调用次数.由于 OIG 是有向图,A 调用 B 和 B 调用 A 用不同方向的边表示.

通过利用已有工具对遗留系统分析,可以获得遗留系统的 CRG 图,并根据遗留系统的一次执行实例,对执行序列进行记录,从而能够获得遗留系统的 OIG 图.在此基础上,我们对遗留系统进一步分析,抽取出合适的模块并将其封装为服务.这一过程可以转换为图的分解问题,即将图 G 的节点分解为不同子集.最基本的切分是将遗留系统分解为两部分,在服务化封装后,一部分形成客户端,另一部分形成服务.复杂的切分可以一次抽取出多个模块,从而最终形成一个核心客户端调用多个服务的系统结构.

在实践中,除了形式化地切分出服务外,更大的挑战来自于遗留系统的实现技术细节.为了自动化地进行切分和封装,这些细节也必须如实地反映到模型中,并在算法设计过程中一起考虑,否则计算出的切面难以具有实际的应用价值.

(1)外部依赖.实际遗留系统部分运行时对象依赖于特定的资源(计算、存储资源及 Java 虚机),如文件、数据库的持久化对象,或桌面软件的 GUI 对象等,这些对象必须和相应资源分配在一起.所有依赖于某一资源的对象必须作为整体切分,不能将它们中的个别对象单独切分出来;

(2)单向边.服务调用通常是单向的,即每次调用都有客户端向服务端发起,一般不能反过来调用或者发生双向调用.这要求计算获得的切面中所有边必须方向相同,即指向切面的同一侧;

(3)接口.服务的接口不同于传统组件,服务接口只能暴露方法,而不能访问对象的属性.因此,切面计算时须避免选择直接访问属性的对象被分开;

(4)性能问题.一般来说,Web 服务的访问都是远程,特别是通过 Internet 的分布式访问.将系统中的本地调用改为通过 Web 服务调用会对系统的性能造成较大损失.而多数遗留系统的内部访问调用发生在单机环境下,两者的性能开销相差甚远.在系统切分的过程中需要考虑服务化造成的额外性能损失,应避免服务化后因为性能原因导致的系统瘫痪.

5.2 切分目标

如上所述,我们通过将遗留系统切分为可复用的服务和执行业务逻辑的客户端,从而使得遗留系统得以优化和扩展.这一问题可以转化为图论中的图切分问题.图切分是图论中一个经典问题,已有工作中提出了多种切分算法,如最大流/最小割算法[13]、聚类算法[14]、k-cut 图切分[15]、基于贪婪策略的快速合并算法[16]等.对于一般图来说,图切分是一个 NP 问题[17],在多项式时间内无法获得最优解.因此需要针对切分的目标,采用各种启发式算法进行切分.

一般来说,不同的图切分算法会对应不同的切分目标,如图的模块度、流量等.本文工作针对遗留系统服务化切分问题,即何种软件模块适合作为服务.通过借鉴传统的软件度量方法[18-19],我们选取并定义了遗留系统模型的服务化切分指标,并将其分解为模块度 MQ,接口评价 IQ 和性能代价 PC 的综合,以下我们将分别解释这些指标的实际意义.

一般来说,具有良好设计的面向对象软件为了更好地满足封装性的设计需求,其模块接口与内部实现相比则相对较为简单.因此从统计意义上来说,具有功能相对完整且可复用的软件模块通常应具有内部高聚合、外部松耦合的特征.我们采用服务模块度(Module Quality,MQ)对这一特征进行评价,其计算方式如下:$MQ=\sum_i (e_{ii}-a_i^2)$,其中 e_{ij} 表示图中连接分片 i 和分片 j 之间点的边数在所有边数中

占的比例,同理 e_{ii} 表示图中分片 i 内部各点之间的边数在所有边数的比例,$a_i=\sum_j e_{ij}$ 表示与第 i 个分片中点相连的边在所有边中所占的比例.因此,模块度 MQ 表示系统模块内部边的比例减去任意连接的边比例的期望值.

考虑到切分出的软件模块将被封装成为服务的形态,我们必须考虑服务的特殊性.与一般的软件组件不同,服务应该具有自包含的特性和良好定义的接口.对前者我们在切分过程中需要寻找切面无外部依赖的模块,对后者我们通过服务接口评价加以度量.首先,服务接口封装了服务的内部实现,描述了服务的外在行为,对服务的复用起到约束和指导作用.目前服务接口通常通过自动机的形式加以描述[20],其中自动机的每个变迁对应服务的一条调用.一般来说,对于具有相对独立完整功能的服务模块,其接口不应太复杂,换言之自动机的状态数目越少越好.然而,通过执行记录计算模块接口自动机的过程复杂度过高,将其应用于软件切分过程是不现实的.考虑到自动机的状态数目受限于接口中的操作数目,因此选取操作数目较少的切面作为服务接口将有效地限制服务接口的复杂度.其次,考虑服务还要求具有较好的互操作性和在互联网上传输消息的性能,其操作中的参数也不应复杂.综上所述,我们主要通过计算接口中的操作数目和参数数目获取服务接口评价指标.我们希望这两个值越小越好,即服务包含丰富业务逻辑的同时其接口越简单越好,因此 IQ 定义为接口中参数数目的倒数.

除此之外,服务不同于传统软件组件的是,服务一般是通过 Internet 远程访问的,并且其传输过程中的报文采用文本描述,这些特征使服务具有良好的互操作性,但同时也极大地降低了服务调用的性能.系统的服务化切分将遗留系统中的本地方法调用转换为服务调用,会在一定程度上降低系统性能.特别是如果将一个频繁发生的调用变为服务调用,势必会大幅损害系统的整体性能,甚至导致系统不可用.因此我们引入一个评价这种转换造成性能损失的性能代价指标 PC.设每次服务调用的通信时间为 $t_{ws}=t_{resp}-t_{comp}$,其中 t_{resp} 为响应时间,t_{comp} 为本地计算时间,为方便计算,假设 t_{ws} 为一常量,则 $PC=1\Big/\Big(\sum_{o\in cut}\sum_{t\in o}t_{ws}\times f+1\Big)$,$f$ 为调用频度,实际计算中一般令 f 为运行记录中的平均调用次数.

综合以上指标,我们给出整体的服务化切分质量(Parition Quality,PQ)的计算公式:$PQ=\alpha\times MQ+\beta\times IQ+(1-\alpha-\beta)\times PC$.其中 α 和 β 都是小于 1 的系数,其具体取值视遗留系统的规模、结构和运行环境进行选择.比如较大规模的系统应对模块度较为敏感,则 α 值应取较大.而运行在广域网环境中的系统应较多考虑性能代价,则 $1-\alpha-\beta$ 应较大,等等.

5.3　切分算法

通过前两节的分析,我们将遗留系统切分问题转化为图的切分问题,并在遗留系统建模的基础上提出了一组切分结果的评价指标.本节中我们将讨论如何在遗留系统的 CRG 和 OIG 模型中寻找符合目标的切分.为了方便进一步的分析,我们将 CRG 和 OIG 合并形成一个独立模型 ODFG.

定义 3.　对象依赖频度图(Object Dependent Frequency Graph, ODFG)是一个四元组,$\langle V,E,I,WE\rangle$,其中 V 是点集合,每个点代表遗留系统运行实例中的一个对象;E 是边的集合,每条边代表两个对象之间的依赖关系;I 是将点和边映射到标签的映射函数;WE 是边的权值函数,权值大小代表了两个对象之间的依赖程度.

ODFG 经过部分合并后就形成了模块依赖频度图(Module Dependent Frequency Graph, MDFG),每个模块是多个对象的集合.MDFG 的定义和 ODFG 类似,这里不再赘述.基于 ODFG 和 MDFG,遗留系统的切分问题可以形式化地描述为图论中有向加权图的切分问题.特别是,如 6.2 节所述,本文中所求切面必须是单向的,即图切分后的两个切片之间的所有边都指向同一切片.同时,我们希望寻找到的切面不仅具有良好的模块度,还应具有良好的接口,并不会造成过大的性能损失.我们借鉴经典的图分级聚类方法[16],通过计算获取图的割树,在此基础上进一步得到更优的分割.据此,我们提出了面向服务的系统切分算法.

算法 1.　面向服务的系统切分算法.

输入:对象依赖频率图 ODFG

输出:最优分割 P_{best}

//合并无法分割的对象

1. ODFG′=ReduceODFGByMerge(ODFG);

//求最大强连通集以获得模块依赖频度图 MDFG

2. MDFG=StrongConnected(ODFG′);

//以模块度为目标求 MDFG 的分割树 PTree

3. PTree=GreedyPartitioning(MDFG);

//取分割树的上部 d 层切面

4. PCs=enumerate(Ptree,d);

//过滤出模块度大于阈值 m 的切面

5. PCs'=FiltePartitionings(m);

//以接口评估和性能为目标求最佳切面

6. P_{best}=arg max PartitionQuality(pi), $pi \in PCs'$;

7. return P_{best}.

算法 1 的基本思路是先以模块度作为首要评价指标,不断合并系统图模型中的节点,合并后的点形成超图,再以新图进行合并,直到系统合并为一个点.从对象到整个系统的合并过程可以用一棵树表示,称之为割树.图 3 给出了一个图及其按照模块度切分后所得割树的示例.

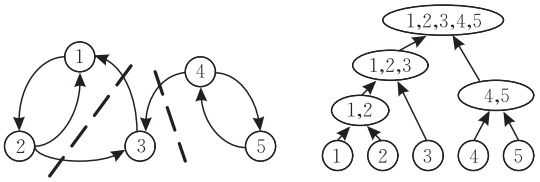


图 3 图的分割及其割树表示

在得到割树后,枚举割树前 d 层的所有割,并按照接口评估和性能指标从中找出满意的分割.算法计算最复杂的步骤在于第 3、4 步,即获取分割候选集的过程.如果不加任何优化措施,对于一个中等规模的遗留系统算法也难以在可接受时间内完成.我们通过多种手段加快这一过程.首先,我们通过剪枝合并一些分解的节点,以减少节点的数目,从而缩小合并时的搜索空间.这主要包括表达对象关系的边上具有特殊的 Java 性质或是带有复杂参数类型的调用边,具体细节详见第 6 节讨论.算法第 1 步即完成此功能;其次,受制于 Web 服务本身性质,切面不能有双向的调用,需要保障所有的调用都是指向一个分片,因此我们将具有双向边的点集合并成一个点.这一问题可以转换为求图的强连通子集,算法第 2 步实现此功能.合并后的图构成一个超图,即模块依赖频率图 MDFG,其中每个节点对应原图 ODFG 的强连通子集;再次,在计算切面的过程中,我们借鉴文献[16]中提出的算法,通过贪婪策略的启发式算法加速割树的计算过程,即算法第 3 步中的 GreedyPartitioning()方法.文献[16]中的原算法只能用于无向图,在算法 2 中我们扩展了这一算法将其用于有向图,这可能会破坏切面的单向性.为此我们限定算法 2 的输入必须是有向无环图(DAG),这在算法 1 的第 2 步计算强连通分支并构成超图的过程中得到保障,这样算法 2 总能找到单向调用的切分.之后,我们在获得割树的基础上,过滤掉深度大于阈值 d 的分割.这是因为深度越大表示切分的粒度越细,过细的切分对于 Web 服务来说是没有意义的.最后,从最终的分割候选集中选出接口评价和性能代价最优的分割.以上这些措施可以极大地提高系统切分时的计算速度.

算法 2. 贪婪分割算法 GreedyPartitioning.

输入:模块依赖频率图 MDFG $\langle V_M, E_M \rangle$

输出:分割树 PTree

//初始化 PTree 叶结点,令每个模块为一个独立分片

1. $P=\varnothing$;
2. for each $m \in V_M$ do {
3. $p=\{m\}$; $P=P \cup \{p\}$;
}

//不断合并分片,直到所有分片合并成一个分片

4. while $|P|>1$ do {
5. $PC=\varnothing$; //新分割候选集

//尝试合并相连分片对构成新的分割

6. for each $\langle p_1, p_2 \rangle$, $p_1 \in P$, $p_2 \in P$, $e(p_1, p_2) \in E_M$ do {
7. p=compose(p_1, p_2); $p.children=\{p_1, p_2\}$; $P'=(P-\{p_1, p_2\}) \cup \{p\}$; $PC=PC \cup P'$;

//计算合并后的模块度增量

8. $\Delta mq(P')=mq(P')-mq(P)$;
}

//选择新分割候选集中最大 mq 增量的分割

9. P=arg max $\Delta mq(P_i)$, $P_i \in PC$;
}

10. $PTree.root=P$;
11. return PTree.

算法 2 通过贪婪策略加速求解割树的计算过程.其主要思想是每次合并分片时,总是沿着模块度增大最多或是减小最少的方向进行.

5.4 复杂度分析

算法 1 包含算法 2,因此我们先分析算法 2 的复杂度.令 n 和 m 分别为 ODFG 图模型的节点数目和边数目(MDFG 由 ODFG 合并节点得到,因此其规模小于等于 ODFG,最坏情况下等于 ODFG 的规模).算法 2 的复杂度取决于第 4～9 步的两层循环.其内层循环(第 6～8 步)需要判断对每条边进行判断,其后合并不同分片中的点,这一步的复杂度是 $O(n+m)$.算法的外层循环对于每个分片执行一次合并,由于初始分片数目等于节点数目,因此共执行了 n 次.算法的总的复杂度为 $O((n+m)n)$,对于稀疏图来说为 $O(n^2)$.实际情况中,大多数的遗留系统的 ODFG 图模型为稀疏图.算法 1 中,步骤 1 的复杂度是 $O(n)$,步骤 2 计算图的强连通分支,采用了文献[20]中的经典算法,复杂度是 $O(n+m)$.步骤 4～6 复杂度和分片数目 k 有关($k \ll n$),都是常量

$O(k)$,可以忽略.可见,算法 1 的整体复杂度取决其第 3 步求解模块依赖频度图的割树的过程,即算法 2.因此,算法 1 的复杂度也为 $O(n^2)$,这对于切分大规模的遗留系统来说是足够快的.

6　Java 遗留系统自动切分和封装技术

在确定了切面的基础上,本节提出了一种自动化服务切分和封装技术.其核心思想是针对 Java 语言执行码重写工具,将原有系统中的本地方法调用自动改写为远程的服务调用.目前已有多种不同原理的 Java 语言执行码操作工具包括 Javassist①、BCEL②、ObjectWeb ASM③ 等.本文工作中我们选用 ObjectWeb ASM.我们的封装工具可以在无需 Java 源码辅助的情况下基本全自动完成对 Java 遗留系统的自动化切分和封装过程.

对于遗留系统可以切分出多个服务的情况,假定服务间并没有访问,即所有调用都由统一的客户端发起,则多个服务的情况可以归结为对客户端的多次切分.因此,不失一般性,这里重点讨论从遗留系统中切分出一个服务端和一个客户端的情况.多个服务的情况与此类似.

以下通过一个简单示例说明基本原理,之后我们再详细解释这一过程实现的难点和我们的解决办法.考虑图 4 左边的遗留系统,其中 A 类的方法 a 中调用了 B 类的方法 b.通过自动服务化重组,我们可以将 B 类封装为 Web 服务并部署在分布式环境中,同时我们利用 Java 执行码重写技术修改 A 类的方法 a 使其通过调用本地的一个代理 B′类,而 B′类实际上是通过 Web 服务调用原来的 B 类的方法 b.这样的处理使得原来的遗留系统成为了分布式系统,并且由于 B 类改变为服务的形式,也可以被新增的扩展功能调用,从而实现了遗留系统的重组和升值.目前实现这一过程的做法大多依赖于程序员手工完成切分和封装过程,效率低下,质量不高.自动完成切分和重组的过程能极大地提高了软件的复用度和软件重组过程的开发效率和质量.实现时服务化切分利用 Java 语言的执行码操作工具来完成修改 B 的实现方法从而达到以上目的,这种解决方法还较简单,只能进行类级别的切分.

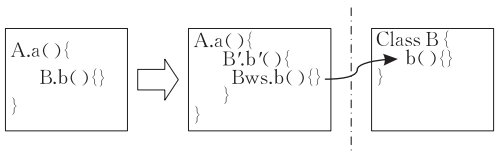


图 4　基本切分方法

如上所述,我们可以容易地将对象方法调用修改为服务调用.然而,在实际的遗留系统切分过程中,我们必须要解决由面向对象技术的本身特征和 Java 语言特有特征引发的一些技术问题.表 1 中大致描述了一些具体问题和相应的解决办法,下面我们就其中部分核心问题进一步进行解释.

表 1　针对 Java 语言特征的遗留系统切分问题及解决办法

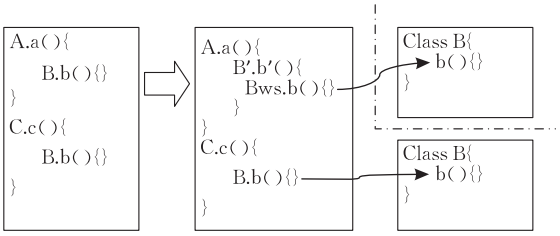
Java 语言特征	问题描述	转化机制
继承	父类和子类被切分到切面两侧	在客户端和服务端同时加载父类
多态	子类的对象类型不确定,在运行期才能确定	类级切分无法支持,通过对象级切分支持,见 6.1 节
重载	服务端类具有同名方法,Web 服务一般不支持	以方法名+参数名的方式重写方法名称
组合	属性访问问题	计算切面时过滤并合并所有存在属性访问的对象,使其不可分
引用	对象标识问题	见 6.2 节
Java 核心类	JDK 的部分类在 Java 虚机实例中必须加载	客户端和服务端同时加载
静态方法和属性	服务端暴露了静态方法和属性	利用 Java 语言每个类具有类对象的特征,将类对象作为普通对象处理
构造方法	对象标识问题	见 6.2 节
复杂类型	Java 语言支持的参数类型远比服务支持类型多	支持所有 Java 原生类型、聚集类型和简单 Bean 类型,其它类型须手工扩展
环境变量	系统执行依赖于部分客户端所在虚机的环境变量	程序启动后,第一次服务调用时对客户端和服务端的环境变量进行同步
同步方法调用	方法具有原子性,在运行期只能被线程串行访问	默认以同步方式实现所有服务调用

6.1　对象级切分

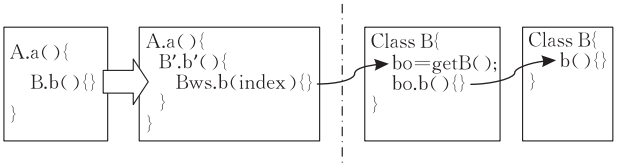
如上所述,类级别的切分主要是一种静态切分,不考虑系统运行期信息.类级别切分局限性较大,一般难以切分在系统中多处出现的核心数据结构或工

① Javassist. http://www.csg.is.titech.ac.jp/~chiba/javassist/
② Apache BCEL. http://jakarta.apache.org/bcel/
③ OW2 ASM. http://asm.ow2.org/

具类，因此难以应用于较大规模的系统. 针对这一问题我们采用对象级的切分，即在运行期，将同一个类的不同对象实例分别放置于服务端和客户端. 如果这种对象刚好处于服务端的切面部分，则所有在客户端并访问了该对象的方法需要重写，而访问本地对象的方法则不作修改. 我们改进上节提出的基本方法来解决这一问题. 改进的方法在修改 B 后，将其重命名为 BProxy，并在部分调用类中修改调用实现，如本例中 A.a()内部调用 B.b()位置，修改为调用 BProxy.b(). 而客户端 B 类不做修改，如本例中 C.c()内部仍然调用 B.b(). 这样同一个类的不同对象实例被切分到客户端和服务端，从而实现对象级别的切分. 过程如图 5(a)所示.



(a) 对象级的系统切分方法



(b) 支持对象标识的系统切分方法

图 5 Java 遗留系统的改进切分方法

6.2 对象标识问题

对象级的切分导致了对象标识问题，即在服务调用过程中如何保证跨越虚机之间的对象访问定位到正确的对象上. 对于类级别切分出的服务不考虑对象引用，也不带状态信息，适用于每次服务调用都是独立的情况. 而对象级的切分就复杂得多，因为在切分过程中，服务端和客户端的对象引用被完全割裂. 针对这一问题，我们进一步改进基本切分方法，为对象定义了全局唯一的标识. 对于切分后的遗留系统，服务端修改原有类为对象实例类，并替换原有类的名称. 同时以原有类名增加一个对象池类，维护所有该类对象的标识. 当服务调用时将对象标识传递给服务端，先从对象池中检索出目标对象实例，然后再完成调用. 具体过程如图 5(b)所示.

目前，自动化的遗留系统封装技术还存在一定的限制，这些限制导致了部分遗留系统服务化转换过程中划分服务的切面不能任意选择，比如遗留系统的本地方法调用、反射调用、动态类加载方法和序列化机制等以及带有自定义复杂类型的方法是无法作为自动切分切面的. 我们通过对 ODFG 模型的剪枝，在服务抽取之前对模型进行过滤和合并，从而避免造成错误的切分.

7 案例分析

本节通过一个实际的应用案例来说明我们方法的有效性，并讨论目前工作的局限和可能的解决办法.

7.1 Infopad

Infopad① 是 SourceForge 上的一个实际的开源项目[14]，从发布以来已有 550 次下载. Infopad 是一款 Java 开发的桌面型任务列表管理软件(To-do List Software)，其功能主要是管理个人任务和便签. Infopad 提供了开放源码，其中共包括 1069 行代码、29 个类、73 个方法. 这一案例虽然简单，但却是较为典型的服务化切分和封装案例，它完全涵盖了本文第 4 节所给框架的所有步骤. 除此之外，这一案例的自动化程度是非常高的，除了开始获取执行记录时需要手工完成运行配置以外，几乎不需要任何人为干预，这也是我们选取这一应用程序作为应用案例的主要原因. Infopad 的系统运行界面如图 6 所示.

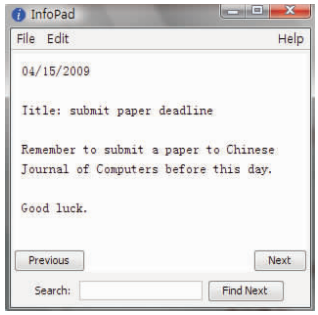


图 6 Infopad 界面

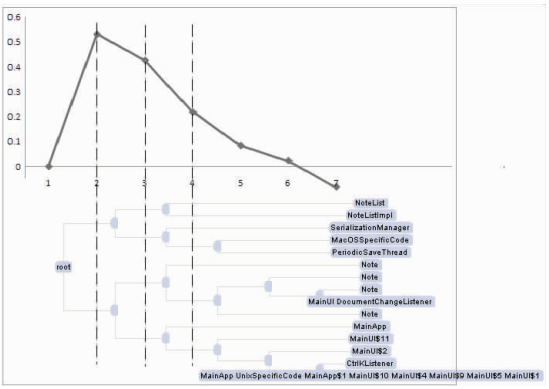
首先，我们利用 OW2 ASM 工具获取了反映 Infopad 静态结构的类关系图，并利用 Eclipse TPTP 框架中的 Probekit 工具② 记录了 Infopad 的多次执行过程，从而获取了它的对象调用图. 在此基础上，我们利用算法 1 计算了其 MDFG 模型表示(见图 7(b))及按照模块度切分的割树(见图 7(a)，纵坐标为模块度). 通过计算并选取前三层切面，评价指标如表 2 所示.

① Infopad. http://infopad.sourceforge.net/
② Eclipse TPTP Probekit. http://www.eclipse.org/tptp/platform/documents/probekit

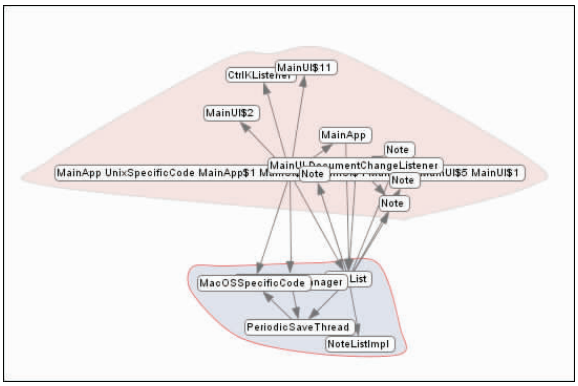
表 2　Infopad 切面评估

评价指标		模块度 *MQ*	接口质量 *IQ*	性能代价 *PC*
度量值	切分 1	0.5321	0.0238	0.0154
	切分 2	0.4772	0.0526	0.0400
	切分 3	0.2193	0.0250	0.0141

一般来说,*MQ* 值大于 0.3 被认为系统具有较好的结构.在本案例中,如果优先考虑接口和性能应选取切分 2,如果考虑更优的模块度,应选择切分 1.由于我们的试验环境是局域网环境,我们选择切分 1 作为切面,并在此基础上通过自动化执行码重写工具将 Infopad 分解成了分布式的应用系统.图 7(b)是利用分析工具显示的切面 1 的软件截图,其中多个 Note 类的对象属于客户端,但包含了从服务端指向的边,这是由于 Note 本身是 Java Bean 类型,可以做为参数传递,因此可以由服务器端通过返回值传递到客户端.



(a) Infopad的ODFG模型按模块度计算得到的割树



(b) Infopad的MDFG模型及一个合理的切面

图 7　计算 Infopad 的切面

分解过程中,我们利用第 6 节介绍的方法对 Infopad中切面两侧的对象进行重写,并增加个别辅助类.由于 Infopad 没有涉及 Java 语言的高级特性,同时也没有复杂类型,因此整个封装过程可以完全自动化的完成.通过分析切分后服务的 WSDL,我们可以看到切分后的服务接口如下:

服务名称:

com.idolstarastronomer.infopad.NoteListJ2WSProxy

操作:

remove1
j2wsinit1
insert1
scheduleSave1
setCurrent1
getNext1
save1
getRecentlyModifiedNote1
continueSearch1
beginSearch1
getPrevious1
getCurrent1

其中,服务名称 NoteListJ2WSProxy 是封装工具增加的代理类,它包含了 Infopad 切面服务端一侧暴露的类中的所有方法.另外,封装工具还会增加一个额外的初始化方法 j2wsinit1,用以初始化服务代理类.可以注意到所有操作名称之后都被添加了数字 1,这是为了解决 Java 的方法重载特性引发的操作名称冲突问题.重载的方法将被按顺序映射为带有数字的操作名.分解后的 Infopad 可以通过多个 Web 服务客户端操作统一的任务列表,因此 Infopad 被扩展成了一个小组的任务管理软件.应用案例的成功服务化重组表明我们的方法是有效的.

7.2　讨　论

本节就本文提出方法和工具的自动化程度和适用范围进行讨论.通过从实际应用案例的实践过程中,我们可以看到本文提出方法的自动化程度非常高,只在以下步骤需要手动工作:

1.获取遗留系统的形式化模型一般不能完全自动化,这是因为系统模型包括静态和动态两类信息.前者可以利用 Java 执行码分析工具全自动地获得,而后者一般不行.动态信息需要在 Java 程序的运行过程中获取,因此依赖于程序的运行配置或者测试用例,就目前的技术来说,这两者都需要程序分析人员手工完成.

2.切面的选择可能需要一定的手工调整,由于软件的模块切分过程是基于统计方法的,因此计算机难以准确地理解软件的功能和行为,存在概率切分出不适当的甚至是错误的模块.这种情况下,需要开发人员在理解系统功能和结构的基础上,对切面两侧的对象进行适当调整.

3.在软件模块自动封装过程中,偶尔也需要手动工作.举例来说,被切分的软件包含特殊的高级 Java 特性(如反

射、Java 序列化、动态类加载器等等),或者包含无法自动映射为 XML 类型的复杂类型时,就需要一定的手动工作.第 3 种情况只在切分特殊用途的程序时出现,其解决方法并不复杂.

除了以上工作,本文方法的其它步骤完全实现了自动化,可以极大地提高了服务化软件的生产效率.

本文方法存在一定适用范围.具体来说,以下 3 方面的软件不适用本文方法:

(1)首先,本文方法是基于软件自身的统计信息对软件实施切分,这要求软件本身具有良好的设计,其结构具有较好的封装性、可扩展性和模块化.对于设计混乱、不清晰或者结构高度复杂、内部关联繁多的软件,应用本文方法的效果不理想.

(2)其次,本文的切分目标是将软件分解为可复用的服务,因此,所有不适用于面向服务架构的软件都不适用于本文方法.例如,高实时性要求的软件、具有复杂操作的人机交互软件(比如 CAD 软件)或者数据密集型计算类软件等等,都不适用于面向服务架构,因此也不适用本文方法.

(3)再次,本文的方法和工具目前仍处于研究过程中,部分高级 Java 语言特性限制了切面的选择和自动化封装.采用了这类高级 Java 语言特性的程序目前还无法应用本文方法,比如 Eclipse 框架及其插件,其中大量使用了 Java 语言的动态加载机制.这部分限制将会随着我们工作的深入逐步缩小.

本文方法除具有以上特征的软件难以适用外,对于目前存在的大多数 Java 遗留系统都可以适用.

8 结论和未来工作

本文的主要贡献是提出了一套完整的遗留系统服务化切分和封装方法,针对 Java 遗留系统构造了一种描述软件行为和依赖关系的图模型,并在此基础上给出了面向服务的系统分解算法.本文工作并未止步于理论研究,而是进一步给出了基于 Java 执行码重写技术的遗留系统的自动切分和封装技术,并基于此开发了工具原型,最终将这一工作成功应用到了一个知名遗留软件 Infopad 的切分和重构中,通过实际的应用案例验证了本文工作的有效性.

本文的工作还有待进一步完善.首先,少数 Java 语言的特性目前技术实现层面还不能支持,如反射、Java 序列化机制等,这一点限制了切分工具的适用范围.其次,基于贪婪算法的割树求解算法在应用到其它一些较大规模的遗留系统中时效果还不够理想,有待进一步研究效果更好的启发式算法.

参 考 文 献

[1] Zhang L J, Zhang J, Cai H. Services Computing. Beijing: Tsinghua University Press, 2007

[2] Yang Fu-Qing, Lu Jian, Mei Hong. Technical framework for internetware: An architecture centric approach. Science in China Series F: Information Sciences, 2008, 51(6): 610-622(in Chinese)
(杨芙清,吕建,梅宏. 网构软件技术体系:一种以体系结构为中心的途径. 中国科学(E 辑:信息科学), 2008, 38(6): 818-828)

[3] Andre Spiegel. Automatic distribution of object-oriented programs[Ph. D. dissertation]. Freie University, Berlin, 2002

[4] Tatsubori Michiaki, Sasaki Toshiyuki, Chiba Shigeru, Itano Kozo. A bytecode translator for distributed execution of legacy Java software//Lecture Notes in Computer Science 2072. Springer Verlag, Budapest, Hungary, 2001: 236-255

[5] Tilevich Eli, Smaragdakis Yannis. J-orchestra: Automatic java application partitioning//Proceedings of the 16th European Conference on Object-Oriented Programming. London, UK, 2002: 178-204.

[6] Bisbal J, Lawless D, Wu Bing, Grimson J. Legacy information systems: Issues and directions. Software, IEEE, 1999, 16(5): 103-111

[7] Chiricota Y, Jourdan F, Melancon G. Software components capture using graph clustering//Proceedings of the International Workshop on Program Comprehension (IWPC). Portland, Oregon, USA, 2003: 217-226

[8] Shokoufandeh A, Mancoridis S, Maycock M. Applying spectral methods to software clustering//Proceedings of the Working Conference on Reverse Engineering (WCRE). Richmond, Virginia, USA, 2002: 3-10

[9] Girard J F, Koschke R. Finding components in a hierarchy of modules: A step towards architectural understanding//Proceedings of the 13th International Conference on Software Maintenance (ICSM'97). Bari, Italy, 1997: 58-65

[10] Rainer K. Atomic architectural component recovery for program understanding and evolution [Ph. D. dissertation]. University Stuttgart, Stuttgart, Germany, 2001

[11] Lee J K, Jung S J, Kim S D et al. Component identification method with coupling and cohesion//Proceedings of the 8th Asia2Pacific Software Engineering Conference. Macau, China, 2001: 79-86

[12] Wang Z, Xu X, Zhan D. A survey of business component identification methods and related techniques. International Journal of Information Technology, 2005, 2(4): 229-238

[13] Konstantinos T. Maximum flow techniques for network clustering [Ph. D. dissertation]. Princeton University, Princeton, NJ, USA, 2002

[14] Mancoridis S, Mitchell B, Chen Y, Gansner E R. Bunch: A clustering tool for the recovery and maintenance of software system structures//Proceedings of the International Conference on Software Maintenance (ICSM). Oxford, England, UK, 1999: 50-62

[15] Christopher J. Computing program modularizations using the k-cut method//Proceedings of the 6th Working Conference on Reverse Engineering. Los Alamitos, CA, 1999: 628

[16] Newman M E J. Fast algorithm for detecting community structure in networks. Physical Review E, 2004, 69: 066133

[17] Mancoridis S, Mitchell B S, Rorres C et al. Using automatic clustering to produce high-level system organizations of source code//Proceedings of the 6th International Workshop on Program Comprehension. Los Alamitos, CA, 1998: 452521

[18] Goulão M, Abreu F B. Software components evaluation: An overview//5a CAPSI. Lisbon, Portugal, 2004

[19] Washizaki H, Yamamoto H, Fukazawa Y. A metrics suite for measuring reusability of software components//Proceedings of the International Software Metrics Symposium (METRICS). Sydney, Australia, 2003: 211-223

[20] Aho Alfred V, Hopcroft John E, Ullman Jeffrey D. Data Structures and Algorithms. USA: Addison-Wesley, 1983

[21] Berardi D, Calvanese D, Giacomo G D, Lenzerini M, Mecella M. Automatic service composition based on behavioral descriptions. International Journal of Cooperative Information Systems, 2005, 14(4): 333-376

LI Xiang, born in 1977, Ph. D. candidate. His main research interests include service computing and software design and production.

HUAI Jin-Peng, born in 1962, Ph. D., professor, Ph. D. supervisor. His main research interests include software theory, network computing and network security.

ZENG Jin, born in 1981, Ph. D. candidate. His main research interests include service computing and software design and production.

GAO Peng, born in 1984, M. S. candidate. His main research interests focus on service computing.

Background

The group's research interests are currently focused on the technologies for the service-oriented software development, evolution and management. Currently, SOA and service technology provide the best practice in the reorganization, optimization and extension of enterprise information systems. The core idea of SOA is a methodology for software development and system integration through composing existing Web services on the internet. However, we face the lack of service resources while a large number of enterprise information systems exist and need to be reorganized with supporting of Web service technologies. Due to identification and partition the service from legacy system by hand are low-quality and time-costing work, an automatic technique is needed. In the last decade, Component oriented software engineering (COSE) has a rapid development. The technology for the identification, extraction and portioning of components have been widely studied and have achieved some results. As the basic element of SOA, Web services can be thought as a type of software component. So these technologies for the components can be used in service-oriented software partitioning and encapsulation. However, Web services are usually self-containing, with well defined interface and accessible on the internet, which make Web services different from the traditional software components. SOA provides the design and production of software a lot of new challenges. This paper studies the problem of how to automatically partition and encapsulate Java legacy systems. This work is supported by the National Natural Science Foundation of China under grant (Nos. 2007AA010301, 2006AA01A106, 2009AA01Z419).