高技术通讯 2012 年 第22 卷 第7 期:675 ~ 682

doi:10.3772/j.issn.1002-0470.2012.07.002

基于资源访问规则与行为截获的软件封装方法①

朱克楠② 尹宝林

(北京航空航天大学计算机学院软件开发环境国家重点实验室 北京 100191)

摘 要 在分析软件系统接口框架体系和软件资源访问接口特征之后,给出了一个通用资源定义模型及其描述语言和一个资源访问规则映射机制及其描述语言。在此基础之上,提出了一种面向二进制的基于资源访问抽象描述和软件行为截获的黑盒遗留软件封装方法,而且设计并实现了一个自动软件封装器原型 TinyWrapper。对该方法在典型的软件重用场景中的作用进行了评估,结果表明,方法可以成功处理场景中的所有复用需求,同时验证了该方法的有效性,证明了该方法容易部署且对改造后的遗留软件产生的负载低。

关键词 遗留软件,软件封装,资源特征,规则映射,软件行为分析,行为截获

0 引 言

软件封装是解决遗留软件复用问题的关键技术。近20 年最主要的软件复用技术研究方向为软件构件技术[1-3],然而许多软件复用问题与当前已存在的系统紧密相连,这些已存在系统通常被称为遗留系统[4]。遗留系统在某些形态上表现出其弱点[5],需要不断演化。但正如杨芙清等人[6]提到的,在很多情况下,遗留系统往往会发生各类文档的丢失,给系统的再工程带来巨大代价。解决这一问题最简单的方法就是放弃遗留系统,针对新需求,采用新技术进行重新开发。但由于像 Ning 等人[7]所指出的,遗留系统中往往包含着重要的业务逻辑,存储着企业长期、有价值的业务知识信息,因此遗留系统的现代化改造成为软件工程领域的研究热点。本文基于软件系统的行为和资源特征模型,提出了一种黑盒软件封装技术,该技术通过截获遗留系统的资源访问行为,形式化描述资源和转换规则,映射资源使用接口,从而实现在不改变遗留系统内部实现的前提下,对遗留系统的数据访问接口进行迁移,达到系统改造的目的。

1 相关研究

根据软件改造过程中需要对系统的理解层次不同,对遗留系统的改造可以分为白盒改造和黑盒改造[8],前者需要了解系统的内部结构,后者只需要了解系统的外部接口。从风险、成本、效率、难易程度等因素角度出发,Bisbal 等人在文献[9]中介绍了遗留系统的封装、维护、迁移和重构等现代化改造方法,并认为封装技术通过最小化遗留系统修改降低了改造风险和成本。IBM 的 Thomas Dietrich 在 1988 年的 OOPSLA 会议中首次提出了包装器的概念。Parsa,Gerardo,David,Zhang 等人分别在文献[10-13]中提出了不同的系统封装方法,这些方法在有效性、适应性和性能等方面存在一定问题。

Parsa 和 Ghods[10] 通过评估、转换、重构造和 Web 服务生成4 个步骤,对遗留方法进行封装,生成外部环境可以使用的 Web 服务。该方法以源代码的静态分析为基础,因此需要遗留系统的源码和实现设计文档,对应用造成较大限制。

Canfora 等[11]提出了一种基于有限自动机的封装器,封装器模拟用户与遗留系统进行交互,从而将用户交互的遗留系统转换为 request/response 接口的 Web 服务。此方法是基于模拟用户操作的方式对系统人机交互接口层面的封装,针对的是基于表单的用户接口,同时当处理人机交互接口复杂的系统时,有限自动机生成和解析的复杂度大大增加,从而限制了该方法的应用。

Millard 等[12]提出了 3 种遗留系统封装设计模

① 863 计划(2009AA043303)和中央高校基本科研业务费专项资金(YWF-11-03-Q-037)资助项目。

② 男,1984 年生,博士;研究方向:软件工程,软件复用,工作流技术,信息安全;联系人,E-mail: zhukenan@gmail.com

(收稿日期:2011-05-13)

式,分别为最小共同特点模式、最普遍模式和协商接口模式。该 3 种设计模式适用于解决具有通用接口的相似遗留系统的封装问题,但无法给出一个具有普遍适用性的遗留系统封装解决方案,不具有广泛的应用性。

文献[13]针对图形交互界面遗留系统提出了一种基于黑盒策略的封装方法,该方法采用窗口嵌入、远程 GUI 控制等技术实现了遗留系统 GUI 的迁移,使用 LUA[14] 脚本语言实现遗留系统的服务化封装,同时提出了在面向服务架构中部署包含用户和遗留系统间互操作的 Web 服务解决方案。此方法完全基于黑盒策略完成封装,避免了对遗留系统内部结构的可知性要求,但封装只支持独立窗口粒度的整合,同时缺乏对接口数据迁移的考虑,因此当遗留系统的数据源发生变化时,该方法无法通过封装实现遗留系统针对新框架的迁移。

本文采用面向二进制文件的黑盒方法解决遗留软件复用问题。首先系统地分析了软件接口的通用架构和软件资源接口特征,采用动态二进制分析监测典型软件系统的行为,跟踪资源使用行为,并在了解软件在二进制代码级别对系统资源使用的情况的基础上,给出了一个通用资源定义模型及通用资源描述语言(common resource description language, CRDL),和资源映射机制及资源映射描述语言(resource mapping description language, RMDL),进而提出了一个面向二进制、基于软件行为分析和资源映射的黑盒封装方法。与以前方法相比,该方法面向可执行码,为完全黑盒策略,同时还引入了动态软件行为分析方法。由此,在对遗留软件进行可重用构件提取和整体功能迁移时不再需要遗留软件的源码和开发文档,这些对文献[10,12]提出的方法来说都是必需的。因此,该方法容易部署,有广泛的适应性,同时能最小化额外开销。

2 软件系统接口框架与资源接口特征

在文献[15]总结出的 6 种软件系统体系结构定义的启发下,本文以为使用资源的使用规则、用户的操作接口以及内部计算逻辑可以唯一定义一个软件系统。用户操作接口通常包括命令行参数、字符交互界面和图形交互界面。内部计算逻辑作为一个软件系统的核心,通常具有一定领域和业务的相关性。资源接口为软件系统与计算机软、硬件资源交互的媒介,用于实现资源访问规则,为内部逻辑计算和业务处理提供操作数据源。一个软件系统的资源接口通常包括数据库访问接口、文件系统访问接口、网络接口、管道、共享内存访问接口及其他进程线程通信接口。

遗留软件系统存在被重用的条件之一是其业务处理和内部计算逻辑基本或完全满足新应用的需求。另一方面,遗留软件的用户操作接口经过长期的磨合,具有稳定特性,而资源接口属于遗留软件中稳定性最差的部分,例如,由于企业业务规模的扩大,需要将原有软件系统作为新系统的一个组成部分纳入新的软件架构中,由此导致的数据源变更、数据库系统的更新和更换、数据仓库的布局变化、本地进程或远程进程间通信内容和格式的变化以及网络通信协议或内容的变化等,均会导致遗留系统资源接口的变更需求。因此,对遗留软件资源接口的封装成为实现遗留系统改造进而将其迁移到新的软件框架或资源环境中的关键。

软件的资源接口由行为和数据访问规则两部分组成。资源接口行为根据软件的功能、业务逻辑、实现语言以及所在平台的不同而不同,具有极大的离散性,但是从系统调用层面考虑,接口行为不外乎打开、读入、写出和关闭 4 种操作,这为对千变万化的软件资源接口行为进行统一处理提供了可行性保证。资源访问规则用于描述接口行为所遵循的操作规范,作为接口行为的依据,资源访问规则可以是系统的一部分,也可以在系统外部描述。访问规则包括资源位置、资源类型、是否具有结构化特征以及资源解析的语法和语义规范等。

对软件资源访问接口的封装,即在不修改原系统接口实现的基础上,改变原系统资源访问行为。由于操作行为是由访问规则直接驱动的,对于完全实现了可配置化资源访问规则的软件系统,可以通过直接修改资源访问规则配置文件实现软件资源接口的封装。但在现实情况下,资源访问规则往往直接嵌入系统代码或半嵌入半配置的实现访问规则。例如一个雇员工资管理系统,需要访问雇员属性和收入信息,如果系统设计阶段决定采用数据库存储雇员信息,通常情况下,开发人员会将数据库访问代码直接嵌入系统实现,而不会选择根据配置文件决定数据的来源(数据库、文本文件或网络等)。因此对于绝大多数软件系统,无法通过静态的方式对资源访问接口进行封装。

3 基于行为和资源访问规则的软件封装

对于一个遗留软件,如何通过一定方式对资源访问规则进行转换,从而控制资源访问行为,是遗留软件封装的关键。本文在给出一个通用资源形式化定义模型和资源访问规则映射描述语言的基础上,提出了一种基于动态行为截获和资源访问规则映射的遗留软件封装技术。

3.1 封装器体系结构

本文封装方法的核心为一个自动化软件封装器(TinyWrapper),该封装器由资源描述解释器、映射规则引擎和程序行为截获引擎3个模块组成。其中资源描述解释器用于读入外部资源描述配置,生成资源在解释器内部表示的数据结构;映射规则引擎用于读入外部映射规则描述配置,并在内部生成转换规则的数据表示,其用到的资源信息来自于资源描述解释器;程序行为截获引擎根据资源描述解释器提供的资源描述和映射规则引擎提供的映射规则信息对遗留系统进行行为截获,在变换规则驱动下访问新资源,最后将映射结果返回给遗留系统。图1给出了该封装器的体系结构。

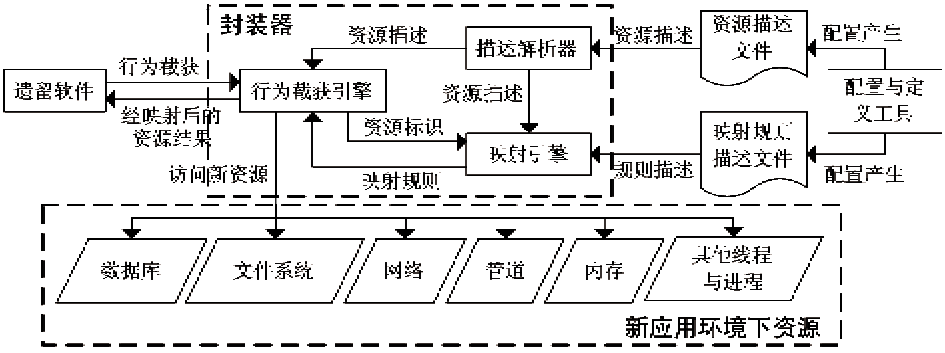

图1 封装器体系结构

3.2 资源访问行为截获

软件系统根据功能、业务逻辑、数据来源等因素的不同,具有不同的资源访问行为,即便具有完全一致的功能和业务处理逻辑的系统,由于设计人员和开发人员的经验、能力水平及个人习惯的不同,最终实现的软件行为仍然千差万别,离散度极高。但是无论软件的上层行为多么复杂和多样,底层系统调用级别的操作是一致的、收敛的,且具有一定规范性。对于所有资源的访问,操作系统将提供一个统一且完备的操作集合,对于不同操作系统平台,该组资源访问操作集合以不同的形式给出。

3.3 资源访问规则转换

遗留系统封装过程中,在完成了资源访问行为截获的基础上,需要通过改变资源访问规则来改变软件资源行为,即对资源的位置、类型、结构化特征以及资源使用的语法和语义规范进行迁移。本节通过提出一种通用的形式化资源定义模型和资源映射规则描述语言,实现资源访问规则的转换。

3.3.1 通用资源形式化定义模型

资源接口可以定义为一个四元组:

$$ResourceInterface:=<Direction,Type,Location,Format> \quad (1)$$

其中 *Direction* 表示接口的方向,可能取值为二元集合 $\{input,output\}$;*Type* 表示接口类型,可能取值为四元集合 $\{pipe,file,network,db\}$;*Location* 表示资源的位置和资源访问的验证信息,根据 *Type* 取值不同 *Location* 定义不同。对于管道资源,*Location* 为管道名称;文件资源 *Location* 为目标文件在本地文件系统中的绝对路径;网络资源 *Location* 被定义为一个二元组,第一个元素为目标IP地址或域名信息,第二个元素为目标服务端口;数据库资源 *Location* 为一个六元组,其中第一、二元素为数据库服务器的主机信息,即IP地址/域名和端口号,第三、四元素为用于链接数据库服务器的验证信息,包括用户名和密码,第五元素为目标数据库名称,第六元素为访问目标数据库表名称。

Format 用于描述资源的语法和语义格式,其定义如下:

$$Format:=<std\_type,format\_desc> \quad (2)$$

其中 *std_type* 用于指定资源的标准类型,如果资源格式为非标准类型,则 *std_type* 为 NULL,表示目标资源为自定义格式。*format_desc* 用于给出自定义资源类型的格式描述,对于标准类型的资源,*format_desc* 为 NULL。根据 *Type* 取值不同,标准资源格式

取值不同,对于管道资源和数据库资源,无标准格式,因此 *std _ type* 只取 NULL。

3.3.2 自定义数据格式描述语言

对于自定义格式的目标资源,需要一种格式描述语言对其语法进行标定,以便在进行资源接口访问规则转换时,对资源进行映射。上文中 *format _ desc* 元素即为数据格式描述载体,其定义如下:

$$format\_desc := < encoding\_mode, section\_list > \quad (3)$$

$$section := < physical\_range, section\_type, body > \quad (4)$$

$$physical\_range := < start, end, unit > \quad (5)$$

如式(3),*format _ desc* 定义为一个二元组,其中 *encoding _ mode* 为编码模式,说明了目标资源内容以二进制形式或以纯文本形式存储,可能取值为集合 $\{binary, text\}$。*section _ list* 为目标资源的段列表,每个段记录了资源内容中逻辑上相对独立的一块内容。*section* 定义为一个三元组如式(4),其中 *physical _ range* 为该段在资源中的物理位置,定义如式(5),包括起始位置、结束位置和单位跨度,其中 *unit* 可取值为集合 $\{Byte, KByte, MByte\}$。*section _ type* 说明该段数据的类型,包括单个变量名值对、数组和结构化数据,可取值为集合 $\{param, array, structed\}$,其中结构化数据用于描述行数据,即具有表头的二维表结构数据。对于不同的 *section _ type* 值,*body* 具有不同的定义:

$$section\_type = param \Rightarrow body := < name, type > \quad (6)$$

$$section\_type = array \Rightarrow body := < name, type, dimensions\_num, delimiter > \quad (7)$$

$$section\_type = structed \Rightarrow body := < record\_delimiter, field\_delimiter\ fields\_title, fields\_type > \quad (8)$$

对于变量名值对,*body* 被定义为包含名称和类型的二元组。对于数组,*body* 被定义为名称、类型、维数信息和分隔符组成的四元组,其中维数信息为维度 1 至维度 *N* 的元素个数,如式

$$dimensions\_num := < dimension1\_num, \cdots, dimensionN\_num > \quad (9)$$

对于结构化数据,*body* 被定义包含记录分隔符、域分隔符、域标题和域数据类型的四元组,其中域标题和域数据类型分别为字段域 1 至 *N* 的标题列表和类型列表,如式

$$fields\_title := < field1\_title, \cdots, fieldN\_title > \quad (10)$$

$$fields\_type := < field1\_type, \cdots, fieldN\_type > \quad (11)$$

根据上文对格式描述的定义,给出通用格式描述语言(common format description language,CFDL)。通用格式描述语言使用可扩展标记语言(XML)书写,用于记录自定义格式资源的语法和语义规范。使用 XML 描述的资源树中每个资源节点都有一个 ID 号,非叶资源节点 ID 采用两位十六进制表示,叶资源节点 ID 采用四位十六进制表示,其中前两位为其父节点的 ID,后两位为其在兄弟节点中的偏移量。

3.3.3 资源访问规则映射描述语言

有了对资源接口的形式化描述,下面可以对资源访问接口转换规则进行描述。转换规则描述采用 XML 书写,负责给出源资源和目标资源映射关系。其中源和目标是相对资源接口的方向而定,当资源接口的方向为输入时,遗留系统所使用的新资源作为转换规则来源,遗留系统原来使用资源作为转换规则目标,当资源接口的方向为输出时,新资源作为转换规则目标,旧资源作为转换规则来源。根据上文提到的通用格式描述语言,规则转换描述中将使用资源节点的 ID 进行资源映射。资源映射描述可以定义为如下三元组:

$$resource\_mapping := < src\_resource, dest\_resource, rules\_set > \quad (12)$$

式中的 *src _ resource* 为转换源资源描述,*dest _ resource* 为转换目标资源描述,其内容为资源描述文件路径。*rules _ set* 为转换规则集合,转换规则定义如下:

$$rule := < function, src\_set, dest\_set > \quad (13)$$

式中,*function* 为变换方法,可表示为 $Function = F_s \cup F_m \cup F_l$,其中 F_s 为字符串处理变换集合,F_m 为数学变换集合,F_l 为逻辑变换集合;*src _ set* 为源资源集合,也是变换方法的参数列表,因此该集合元素个数类型须与变换方法参数个数和类型一致;*dest _ set* 为目标资源集合。变换源或目标可以表示为

$$src \mid dest := < class, [ID], [rule], [type], [value] > \quad (14)$$

式中 *class* 用于描述源或目标的类别,可取值为 $\{RES, RULE, CONST\}$,分别代表资源类别、规则类别和常量类别。式(14)中 *ID*,*rule*,*type*,*value* 分别表示资源单位在资源描述中的标识、嵌套规则、常量类型和常量取值,它们根据 *class* 取值不同可选存

在。当类别为资源时,表示该项为一个资源单位,需要给出资源 ID;类别为规则时,表示该项目为一个嵌套的规则,嵌套规则不显示给出目标,其目标即为父项目;类别为常量时,表示该项目来自于常量,需要给出常量的类型和取值。

4 案例分析

本节将通过具体的使用案例,验证本文方法的有效性、广泛适用性、易部署性以及低性能负载。首先,我们选择了 5 个典型的遗留软件系统,并针对软件规模、关键操作空间和二者的关系特点进行了量化分析;其次,针对选取遗留软件,验证本文方法在可重用构件的抽取和整体系统迁移两方面的有效性;最后,对方法的配置代价和由该方法引入的系统开销进行量化分析。

实验平台为两台配有 Intel 双核 2.4GHz CPU 和 2G 内存容量的 PC 机,其中一台装有 Linux Ubuntu 操作系统,另外一台为 Windows XP 操作系统。选取了 5 个跨平台软件(Windows 和 Linux 版本),分别为:人力资源系统(HRS)、物流管理系统(LMS)、销售管理系统(SMS)、库存管理系统(DMMS)和医药信息系统(MIS)。

4.1 软件规模与关键操作空间

通过对被选实例软件规模的评估和对这些软件关键操作空间监测,我们可以对软件规模与软件对应的关键操作集合规模之间的关系进行分析,同时了解软件规模对本文行为截获方法的影响。首先,我们从代码量、功能点数量、模块数、配置文件的数量和规模、开发人员数量以及开发周期 6 个方面对选取的 5 个遗留软件的规模进行了评估。表 1 列出了对所选软件在 Windows 和 Linux 操作系统上的评估结果。

表 1 所选遗留软件规模评估结果

软件名称	代码行数(K line)		功能点数量		模块数		配置文件的数量及规模(数量·平均行数)		遗留软件的开发人员数量		遗留软件的开发周期(月)	
	W	L	W	L	W	L	W	L	W	L	W	L
HRS	30	25	50	50	4	5	5·30	8·20	5	1	6	12
LMS	50	48	100	100	5	8	3·22	6·18	8	10	3	4
SMS	80	76	120	120	8	12	8·18	10·22	4	6	12	6
DMMS	100	90	80	80	11	10	13·35	8·21	5	5	15	17
MIS	200	180	250	250	18	25	20·50	15·60	8	10	24	24

注:W 表示 Windows 系统,L 表示 Linux 系统

如表 1 所示,所选软件中,MIS 规模最大,HRS 规模最小。我们分别在 Windows 和 Linux 系统上运行所选的 5 个遗留软件,选择一些熟悉这些系统的关键用户对这些软件进行日常操作,同时,在系统上运行 TinyWrapper 以监测目标软件在运行过程中所使用的关键操作数量。我们使用关键操作数量衡量软件关键操作空间的大小。图 2 表明,在特定时间段内,随着软件规模的增大,关键操作空间也随之扩大。

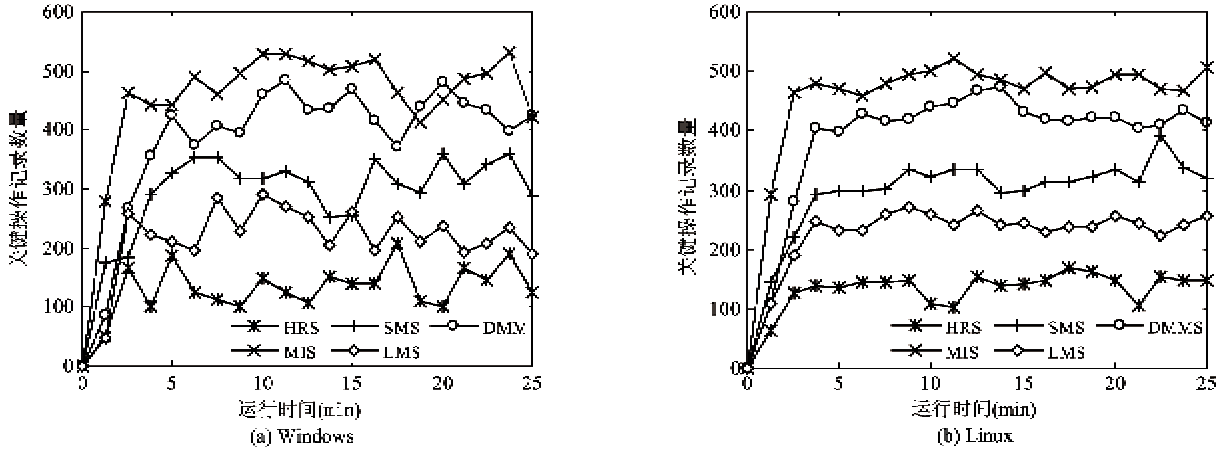

图 2 被选遗留软件在不同操作系统上的关键操作空间

4.2 有效性评估

我们评估了本文方法在遗留软件中可重用构件提取和整体功能迁移两方面的有效性。首先,我们在 5 个选中的遗留软件系统中分别选出一个功能子集,选中的功能子集具有相对独立且完整的特点。表 2 列出了选中的功能子集的详细情况。

表 2 用于可复用构件抽取的功能子集详细情况

软件名称	重用子功能模块名称	网络	文件系统	数据库	进程通信
HRS	雇员信息管理	×	√	×	√
LMS	订单管理	×	√	√	×
SMS	财务报表管理	√	√	√	×
DMMS	库存管理	×	√	√	×
MIS	病例管理	√	√	√	√

对于整体功能迁移,根据所选的 5 个遗留软件系统的资源使用情况,我们分别设计了 5 个功能迁移场景。从两方面考虑对迁移场景进行设计:资源介质的变化情况和资源访问规则的变化情况。在迁移场景中,除了 MIS 之外,其他遗留软件均存在资源介质变化。例如,HRS 和 LMS 分别存在由本地文件系统到数据库和网络资源的迁移;SMS 和 DMMS 分别存在由数据库和网络资源到本地文件系统的迁移。

另一方面,5 个遗留软件均面临资源访问规则的更改。HRS、LMS 和 MIS 所使用的关系数据中的某些域发生变化。其中,HRS 变化前的域集合为变化后域集合的子集,LMS 和 MIS 变化前后的域集合不满足该条件,但是满足如下条件:对于任意旧域集合中的元素 o,必存在一个新域中的子集 N 和映射函数集合的子集 Fun,使得该函数子集作用于新域子集后结果与旧域元素 o 等价,即:

$$\forall o \in OLD\_FILELD\_SET \Rightarrow \exists N, Fun \mid N \subseteq NEW\_FIELD\_SET \wedge Fun \subseteq MAPPING\_FUN\_SET \wedge o = Fun(N) \quad (15)$$

SMS 和 DMMS 将分别面临域字段缩减和域字段添加变化,MIS 的数据库管理系统由 SQL Server2000 变化至 ORACLE 11G。可重用构件提取和整体功能迁移的有效性评估结果表明,本文方法成功解决了所提出的所有重用场景,而文献[10,12,13]中提到的方法均无法解决这些问题。这些方法失效的原因主要为:文献[10]中提到的方法基于遗留系统源代码静态分析,需要提供遗留软件的源代码,而在我们的重用场景中,遗留软件的代码是无法得到的。文献[12]和[13]中的方法无法解决数据迁移问题,因此无法完成本文重用场景。

4.3 配置代价和性能负载评估

本节我们将对 TinyWrapper 的性能负载和针对不同应用的配置代价进行量化评估。配置代价可以使用配置文件的行数与目标系统的代码量之比进行衡量。因此我们对本文方法针对 5 个被选软件的 Windows 版和 Linux 版在对系统进行整体迁移和可重用构件提取时的配置文件行数进行统计,结果如图 3 所示。

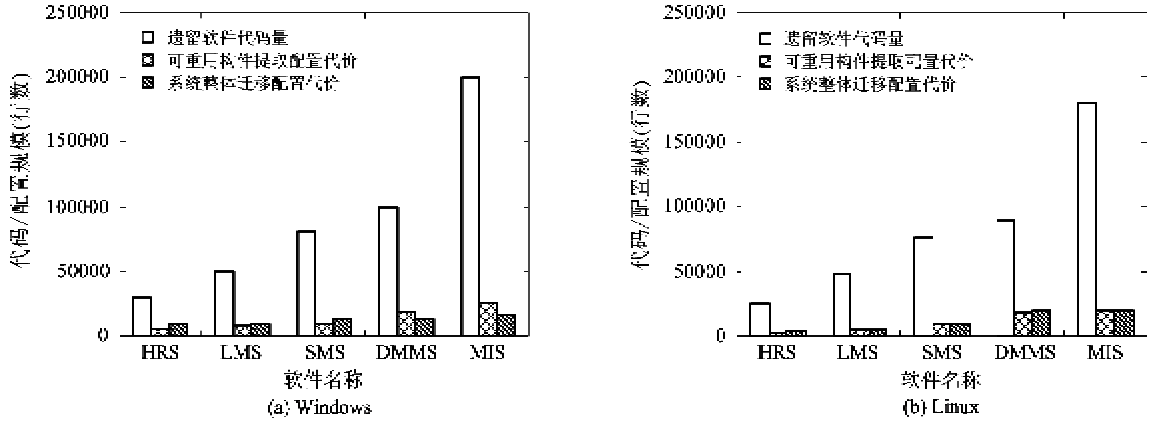

图 3 所选遗留软件在不同操作系统上针对可用构件抽取和整体迁移的配置代价

结果表明,随着遗留软件规模的扩大,使用本文方法进行可重用构件提取和整体系统迁移的配置代价均呈现微弱的上升。平均配置代价小于系统代码规模的8%,而且随着系统代码规模的增加,该比例呈现下降趋势。

为了衡量 TinyWrapper 对系统性能的影响情况,我们分别对5个被选遗留软件的 Windows 版本在迁移前后进行了性能测试。首先,为每个遗留软件选择一个典型的业务流程,针对这些流程,设计和实现了5个用户模拟脚本,以完成流程的模拟操作。5个业务流程分别是:(1)雇员建立流程(雇员规模为20000人);(2)物流订单建立流程(订单规模为10000单);(3)季度结算流程(交易数量为15000笔,平均毛利为18元);(4)入库登记流程(货物种类为21,平均数量为1000箱);(5)病例创建流程(病人数量为25000人)。表3给出了性能测试结果。

表3 性能测试结果

软件名称	业务流程	迁移前(s)	迁移后(s)	效率影响
HRS	雇员建立流程	8.63	9.17	6.2%
LMS	物流订单建立流程	7.45	8.01	7.5%
SMS	季度结算流程	19.43	20.77	6.8%
DMMS	入库登记流程	9.55	10.11	5.8%
MIS	病例创建流程	12.99	13.76	5.9%

对每个被选系统的流程均运行6次,表3中列出的为除最高运行时间和最低运行时间的其他4次的平均值。结果表明,采用本文方法进行遗留软件整体功能迁移所带来的平均额外负载为6.44%,没有超出可接受的范围。

5 结 论

遗留系统的封装技术是解决遗留系统改造与集成的主流方法。当前方法存在的不足在于对资源形式化定义中缺乏对某些资源类型的支持,例如,网络资源和文件资源中对其他标准格式协议和文件类型的支持,对实时数据库资源的支持等。下一步工作完善资源形式化定义描述,增加标准类型支持的种类,扩大方法的适用性。

参考文献

[1] Heineman G T, Councill W T. Component-based Software Engineering: Putting the Pieces Together. Boston: Addison-Wesley Longman Publishing Co, 2001. 110-120

[2] Ravichandran T, Rothenberger M A. Software reuse strategies and component markets. *Communications of the ACM-Program Compaction*, 2003, 46(8): 109-114

[3] 杨芙清, 梅宏. 软件复用与软件构件技术. 电子学报, 1999, 2: 68-75

[4] Seacord R C, Plakosh D, Lewis G A. Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices. Boston: Addison-Wesley Longman Publishing Co, 2003. 30-50

[5] 詹剑锋, 程虎. 基于 Mobile Agent 技术的遗留系统再工程方法. 软件学报, 2002, 13(12): 2343-2348

[6] 杨芙清, 王千祥, 梅宏等. 基于复用的软件生产技术. 中国科学(E辑), 2001, 31(4): 363-371

[7] Ning J Q, Engberts A, Kozaczynski W. Recovering reusable components from legacy systems by program segmentation. In: Proceedings of Working Conference on Reverse Engineering, Chicago, USA, 1993. 64-72

[8] Weiderman N H, Bergey J K, Smith D B, et al. Approaches to Legacy System Evolution: [Technical Report], CMU/SEI-97-TR-014, Carnegie Mellon University, 1997.

[9] Bisbal J, Lawless D, Wu B, et al. Legacy Information Systems: Issues and Directions. *IEEE Software*, 1999, 16(5): 103-111

[10] Parsa S, Ghods L. A new approach to wrap legacy programs into web services. In: Proceedings of the 11th International Conference on Computer and Information Technology, Khulna, Bangladesh, 2008. 442-447

[11] Canfora G, Fasolino A R, Frattolillo G, et al. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. *Journal of Systems and Software*, 2008, 81(4): 463-480

[12] Millard D E, Howard Y, Chennupati S, et al. Design patterns for wrapping similar legacy systems with common service interfaces. In: Proceedings of the European Conference on Web Services, Zurich, Switzerland, 2006.

191-200

[13] Zhang B, Bao L, Zhou R. A black-box strategy to migrate GUI-based legacy systems to web services. In: Proceedings of the IEEE International Symposium on Service-Oriented System Engineering, Jhongli, Taiwan, China, 2008. 25-31

[14] Ierusalimschy R, Celes W, Figueiredo L H. A Summary of all Things Lua Scripting Language. http://www.lua.org/about.html

[15] 孙昌爱，金茂忠，刘超. 软件体系结构研究综述. 软件学报，2002，13(7)：1228-1237

A software wrapping approach based on resource access rules and behavior interception

Zhu Kenan, Yin Baolin

(State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, BeiHang University, Beijing 100191)

Abstract

After analyzing the software interface architecture and resource access interface characteristics, this study proposed a common resource definition model described by a resource description language and a resource mapping mechanism expressed by a resource mapping description language. On the basis of this, a binary-centric and black-box wrapping approach based on resource access abstract description and software behavior interception was presented. Then, an automatic software wrapper prototype called TinyWrapper was designed and implemented. Moreover, the functions of the approach in a typical software reuse scenario were evaluated. The results show that the approach is effective and it can deal with all reuse requirements in the scenario, additionally, the approach is easy to deploy. The performance overhead caused by the approach is low.

Key words: legacy software, software wrapping, resource characteristics, rule mapping, software behavior analysis, behavior interception