软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(5):942−960 [doi: 10.3724/SP.J.1001.2013.04371]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于模型的构件软件修改影响分析*

陶传奇[1,2], 李必信[1,2], Jerry GAO[3], 孙小兵[4]

[1](东南大学 计算机科学与工程学院,江苏 南京 211189)

[2](东南大学 软件工程研究所,江苏 南京 211189)

[3](Department of Computer Engineering, College of Engineering, San Jose State University, San Jose, USA)

[4](扬州大学 信息工程学院,江苏 扬州 225127)

通讯作者: 李必信, E-mail: bx.li@seu.edu.cn

摘 要: 基于构件的软件构建方法目前被广泛使用在软件开发中,用于减少软件开发的工程成本和加快软件开发进度.面向构件的系统主要由第三方提供的可重用构件或者内建的可重用构件组成,因此,系统的质量好坏和维护的难易程度依赖于构件的品质.一个软件修改会给其他构件甚至整个系统带来影响,而修改影响分析是控制和消除这类影响的有效手段.然而,现有的研究很少涉及构件软件的修改影响分析,尤其缺少对系统层面的修改影响分析研究.提出了一种基于模型的系统化修改影响分析方法,该方法的基本思路是:首先提出构件及系统层面的修改影响分析模型,然后根据分析模型分别从构件和系统两个层面对构件软件修改前后的版本进行修改识别,并且利用"防火墙"方法进行影响分析.理论分析和实验结果表明,该方法是可行的,也是有效的.

关键词: 修改影响分析;修改影响分析模型;基于构件的软件;软件维护

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 陶传奇,李必信,Gao J,孙小兵.基于模型的构件软件修改影响分析.软件学报,2013,24(5):942−960. http://www.jos.org.cn/1000-9825/4371.htm

英文引用格式: Tao CQ, Li BX, Gao J, Sun XB. Model-Based change impact analysis for component-based software. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):942−960 (in Chinese). http://www.jos.org.cn/1000-9825/4371.htm

Model-Based Change Impact Analysis for Component-Based Software

TAO Chuan-Qi[1,2], LI Bi-Xin[1,2], Jerry GAO[3], SUN Xiao-Bing[4]

[1](School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

[2](Institute of Software Engineering, Southeast University, Nanjing 211189, China)

[3](Department of Computer Engineering, College of Engineering, San Jose State University, San Jose, USA)

[4](School of Information Engineering, Yangzhou University, Yanzhou 225127, China)

Corresponding author: LI Bi-Xin, E-mail: bx.li@seu.edu.cn

Abstract: Component-Based software construction is a widely used approach in software development, to reduce the engineering effort and speed up the development cycle. Component-Based software systems consist of various components such as third-party components and in-house built components. Due to software changes, a component-based system is usually affected at both the component level and system level. Thus, a change impact analysis is needed to ensure the software quality and support maintenance. Existing research seldom addresses the issue of change impact analysis on component-based software, especially at a system level. This paper proposes a systematic approach to change impact analysis from the components to the system. Firstly, the change impact analysis models are proposed, and the change types are classified. Then, a change identification and an impact analysis are performed using a firewall approach based on the

* 基金项目: 国家自然科学基金(60773105, 60973149); 国家高技术研究发展计划(863)(2008AA01Z113); 高等学校博士学科点专项科研基金(20100092110022); 中国科学院软件研究所计算机科学国家重点实验室开放基金(SYSKF1110)

收稿时间: 2012-03-22; 修改时间: 2012-08-20; 定稿时间: 2013-01-07

proposed models at both levels. The paper reports the case studies are based on a realistic component-based system. The study results show that the approach is feasible and effective.

Key words: change and impact analysis; change impact analysis models; component-based software; software maintenance

在现代软件系统的生命周期内,软件缺陷修复、功能增强、性能改进、需求增加、运行环境改变等均要求软件系统具有较强的演化能力,从而要求软件工程人员能够快速适应改变,减少软件维护的代价.软件演化和软件维护有着密切联系,在软件维护期间,可以通过具体的维护活动使系统不断向前演化.软件系统交付使用之后对其进行修改的过程称为软件维护,目的是为了纠正错误或缺陷、提高软件系统性能或其他属性,以及使该软件系统适应变化后的环境等(IEEE Standard for Software Maintenance,1993,1998 修订).现代的软件系统很多是由可重用的构件所构成,这些构件可以由第三方提供或者是内建(built-in)的.在构件系统中,系统的质量主要依赖于系统的组成构件.在软件维护期间,构件发布者对构件的任何修改都可能会对构件本身带来影响,还可能影响构件所在的整个软件系统.因此,从构件发布者的角度,我们需要对构件修改进行影响分析,这是构件软件质量保证的一种重要手段.有效的修改影响分析可以减少软件维护的成本,同时,测试员只需要对受修改影响的构件功能或者关系进行测试,从而也减少了回归测试的成本.

在构件软件演化周期中,当构件由于更新或者升级而被修改以后,我们对构件和系统层面都需要进行修改影响分析,同时也是为了后面的单元、集成以及系统重测做好前期的准备工作.对于构件软件系统的修改影响分析,主要存在着以下难点:

- 构件的用户往往不了解构件的程序源代码和详细的设计文档;
- 构件的修改信息对于用户和应用开发者来说一般是不可见的;
- 目前还没有自动化的支持构件修改影响分析的工具;
- 缺乏成本有效性的方法对构件进行修改影响分析.

基于模型的方法已经被广泛应用在修改影响分析和回归测试等软件维护活动中,实践证明是有效且可行的[1−4].在构件系统中,模型可以用来表示构件内部以及构件之间的依赖、交互及架构关系.因此,我们需要对构件系统进行合适的建模以进行修改影响分析.从不同的角度来看,可以对由构件组成的系统建立不同的模型,比如,从外部用户的角度,构件的主要功能是提供 API,于是可以建立 API 模型;从系统层面看,构件之间存在着交互关系,还有组合、配置等多种架构关系,可以建立相应的系统模型.因此,建立合适的构件以及系统模型是修改影响分析的重要前提.另外,模型还可以帮助定义重测标准和制定重集成策略,方便测试以及回归测试的自动化,对于整个软件维护阶段都具有重要意义.

目前,针对构件软件的修改影响分析有如下一些开放性的研究问题:

(1) 如何识别不同的构件修改类型,以及对其他构件和整个系统造成的影响;

(2) 有什么样的模型可以支持构件的修改影响分析;

(3) 如何利用修改影响分析结果去对构件以及系统进行回归测试;

(4) 如何进行系统化的构件软件修改影响分析.

针对这些问题,本文提出了一种基于模型的系统化修改影响分析方法.其主要贡献如下:

(1) 定义了一些针对构件和系统的模型来支持构件软件修改影响分析;

(2) 提出了一种系统化的从构件到整个系统的修改影响分析手段;

(3) 通过实际的构件系统进行实验验证,来表明方法的可行性和有效性.

本文第 1 节分析和定义相关的修改影响分析模型.第 2 节提出一种从构件到系统的系统化修改影响分析方法,并且给出度量评估手段.第 3 节是对本方法的实验研究.第 4 节介绍当前传统软件和构件软件中修改影响分析的相关工作.第 5 节给出结论并对未来的工作进行展望.

1 修改影响分析模型

本节我们将定义方法中使用到的一些模型.我们首先定义一个构件模型,然后分别从构件和系统两个层面分别定义一些修改影响分析的模型,用来辅助构件开发者对构件及系统进行修改影响分析.

1.1 构件模型

构件可以看成是一个黑盒子.构件对外的接口主要有 API 功能(参数)和端口,其中,API 是构件对外提供的应用功能,端口表示构件内部调用外部其他构件功能的函数.图 1 是一种构件模型.

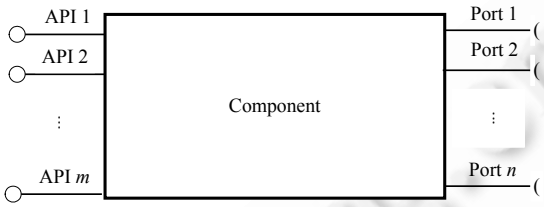


Fig.1 A component model

图 1 构件模型

构件模型可以定义如下:

定义 1. 构件模型:

1) 任何构件 C 的 API 功能可以定义为一个集合 F_{API},该集合包含构件提供给用户的所有 API 功能.可以定义如下:

$$F_{API}=\{f_1,f_2,\ldots,f_m\},$$

其中,$f_1,f_2,\ldots,f_m$ 表示 API 功能.

2) *Port* 表示构件内部调用其他构件功能的函数集合,可以定义为构件的端口:

$$Port=\{p_1,p_2,\ldots,p_n\},$$

其中,$p_1,p_2,\ldots,p_n$ 表示端口函数.

1.2 构件层面模型

这里,我们假定构件提供者(发布者)拥有构件内部的功能依赖图以及功能数据依赖图这些模型信息.功能依赖图(function dependency graph,简称 FDG)表示构件内部功能函数以及它们之间的直接调用关系,而数据功能依赖图(data function dependency graph,简称 DFDG)则描述了构件内部功能函数通过数据定义-使用所形成的依赖关系.Gao 等人已经将这些构件层面模型应用到构件的回归测试中[1].

1.3 系统层面模型

在系统层面,我们从构件之间交互关系的角度提出一种构件交互图模型;从系统架构角度提出一种组合配置树模型.下面具体定义这两个模型.

1.3.1 构件交互图

构件交互图(component interaction graph,简称 CIG)表示在一个构件系统中,构件之间通过调用、消息传递、协议机制等形成的交互关系.定义 2 给出了构件交互图的定义.

定义 2(构件交互图). 构件交互图是一个有向图 $CIG=(N,E,R)$,其中,N 表示构件的节点集合;$E=E_{MSG}\cup E_{USA}$,表示图中边的集合;$R=\{MSG,USA\}$表示构件之间的交互关系;$E_{USA}\subset N\times N\times R$ 表示构件之间使用关系的有向边集合;$E_{MSG}\subset N\times N\times R$ 表示构件之间消息传递关系的有向边集合.

图 2 是一个电梯系统各个构件之间的交互关系图,图中的节点表示系统的各个构件,边表示构件之间的交互关系.我们考虑两种主要的交互关系:使用(usage)和消息传递(message).该电梯系统是由楼层控制面板(floor

panel)、用户控制面板(user panel)、电梯车载(car)、车载控制(car controller)、电梯门(door)等构件所组成.比如在该系统中,user panel 使用了 user panel queue,所以它们之间有着使用关系.另外,car 需要和 user panel 之间相互通信发送消息,以确保电梯系统的正常运行,因此它们之间存在消息传递关系.

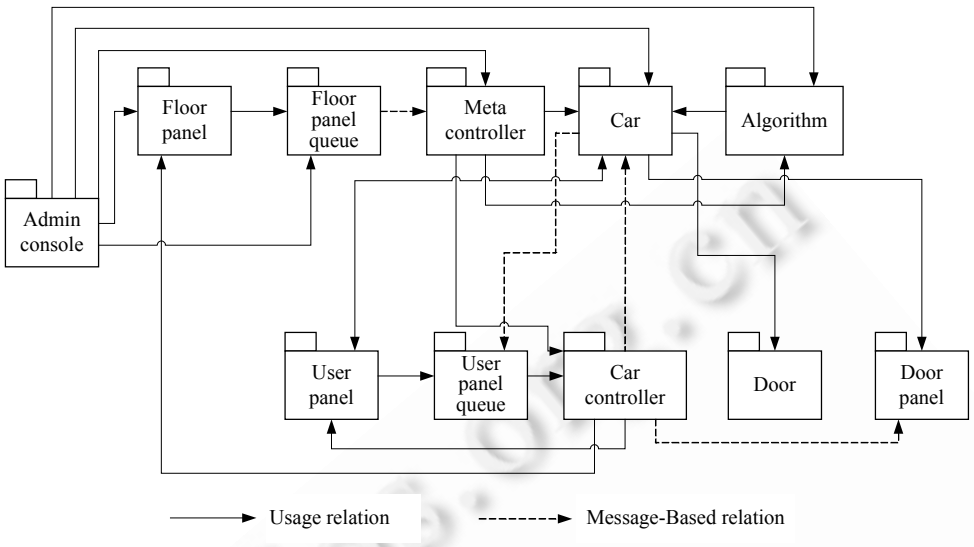


Fig.2 Component interaction graph

图 2 构件交互图

1.3.2 组合配置树

组合就是将一些可重用的构件以相同或不同的形式组合成更加复杂的构件.另外,现在的构件系统一般都是可配置的.可配置的软件可以让用户根据需要定制自己的不同应用,比如不同的配置环境、不同的配置架构以及不同的配置功能.因此在系统架构的层次上,构件之间存在着组合(composition)和配置(configuration)等关系.因此,配置也是构件系统的一种重要架构关系,而当前的构件重测或修改影响分析模型都没有考虑这一点.对于提供架构可配置服务的构件软件,对配置关系进行修改影响分析可以支持重测新版本中不同的可配置架构或组织,测试可配置的架构是否得到成功的构造,从而确保提供服务功能的质量.我们的实验研究对象也是具有配置功能的构件系统.下面给出组合配置树的具体定义.

定义 3(组合配置树). 组合配置树(composition and configuration tree,简称 CCT)用来描述系统层面的架构关系,可以形式化地定义为一个三元组 $CCT=(N,E,R)$,其中,

- N 表示树节点的集合;
- R 是节点间语义关系的集合,这里,R 有组合关系和配置关系,也就是 $R=\{R_{comp},R_{conf}\}$,其中,配置关系通常包括 *EOR*(异选)、*AND*(并选)、*Switch*(多选一)和 *Multiplex*(多选多)等关系,$R_{conf}=\{EOR,AND,Switch,Multiplex\}$;
- E 是节点之间边的集合,$E=E_{comp}\cup E_{conf}$,E_{comp} 表示组合关系边,E_{conf} 表示配置关系边:

$$E_{conf}=E_{AND}\cup E_{EOR}\cup E_{Switch}\cup E_{Multiplex}.$$

例如在图 3 中,电梯系统主要由 5 个主要构件所组成,而其中一些主要构件也是由下层的一些其他构件组合而成.例如,car 是组成的系统的重要构件,car 这个复杂构件本身也是由 user panel,door 等构件所构成.配置是构件系统的重要属性之一.比如,电梯门(door)可以有单门(single door)和双门(double door)的两种配置,用户面板也可以有文本(text)和符号(symbol)两种显示配置.组合配置树中的节点表示(复杂)构件.节点之间的边表示组合或者配置关系,其中,可配置构件下面不同的节点符号表示不同的配置关系.图中三角形节点(比如 color)表示多选一的配置关系,六边形表示异选(比如 display)的配置关系.

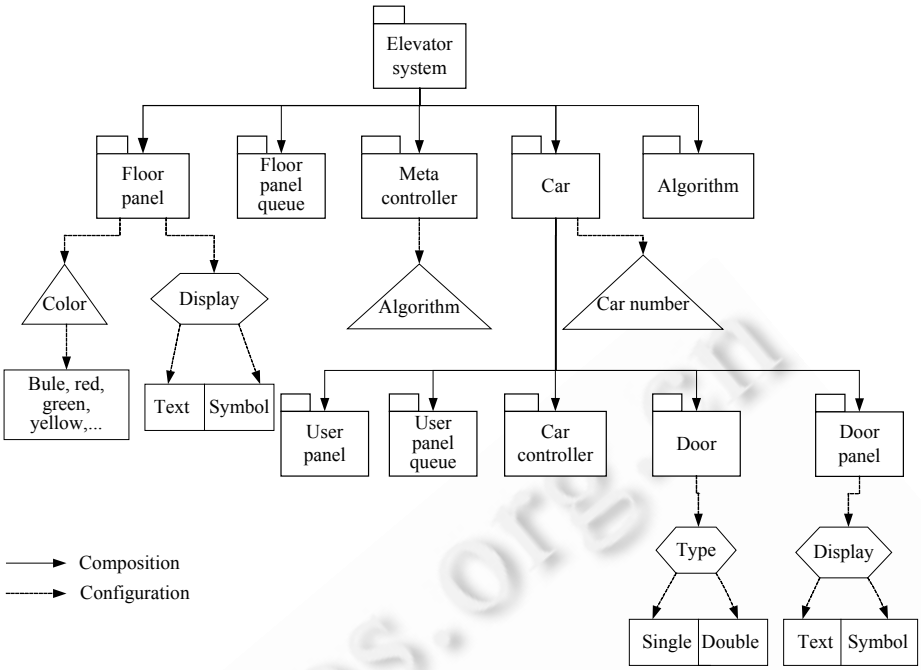


Fig.3 Composition and configuration tree

图 3 组合配置树

2 系统化的修改影响分析方法

2.1 修改识别

由于构件软件缺乏对外可见、可追踪和可管理的修改信息,所以从构件外部的角度来说,需要从某种机制中使得修改信息能够可见.构件的修改可以发生在不同的层面上,因此也导致了很多不同类型的修改,比如 API 修改、构件交互的修改以及系统架构的修改等.在模型上面识别这些不同的修改类型,是影响分析的前提条件.

我们总结了常见的构件和系统修改类型以及相应的模型修改,见表 1.通过比较修改前后的构件模型,可以比较容易地识别出构件在不同层面上的修改.比如说,通过比较修改前后的构件功能依赖图 FDG,我们可以识别出图中添加、删除和改动的节点或者边,从而识别出发生修改的构件功能或者依赖关系.

Table 1 Component and system changes, and corresponding model changes

表 1 构件和系统的常见修改类型以及相应的模型修改

Change type	Specific changes	Model changes
AID/DID/CID	Add/Delete/Change an internal data	Add/Delete/Change a node in DFDG
AIF/DIF/CIF	Add/Delete/Change an internal function	Add/Delete/Change a node in FDG
AAD/DAD/CAD	Add/Delete/Change an API data	Add/Delete/Change a node in DFDG
AAF/DAF/CAF	Add/Delete/Change an API function	Add/Delete/Change a node in FDG
AP/DP/CP	Add/Delete/Change a port	Add/Delete/Change a node in DFDG/FDG
AM/DM/CM	Add/Delete/Change a message	Add/Delete/Change a node/link in CIG
ACP/DCP/CDP	Add/Delete/Change a composition	Add/Delete/Change a node/link in CCT
ACF/DCF/CCF	Add/Delete/Change a configuration	Add/Delete/Change a node/link in CCT

2.2 影响分析

通过修改识别,我们得到了构件修改前后版本的修改,以及在模型上面体现的修改点信息.现在需要利用识别出的这些修改信息进行影响分析.White 等人提出一种“防火墙(firewall)”技术,用于集成测试的回归测试选择[5].“防火墙”就是将受被修改模块影响的程序模块分离出来,封装需要重测的模块集[5−7],这里的防火墙可以看

成是被修改模块以及受影响模块与程序其他部分的边界.他们使用控制流分析模块之间的修改影响关系,并且使用调用图建立控制依赖关系.“防火墙”方法可以有效识别出修改影响,而且可以在模型上直观体现需要重测的程序中受影响的元素.该方法的核心是需要定义修改分析或者重测的模型.前面我们已经定义过了构件软件在构件层面和系统层面的模型,包括 FDG,DFDG,CIG 以及 CCT.在这些模型中存在着各种依赖关系,通过前面模型上的修改信息,我们可以利用图的可达性(reach-ability)结合具体的模型,定义一些针对构件软件的修改“防火墙”以及算法.我们提出的系统化构件软件修改影响分析主要包含以下内容:

1) 识别受修改的构件内部函数对构件其他函数的影响;
2) 识别受修改的构件内部数据对构件内部函数的影响;
3) 识别受修改影响的构件 API;
4) 识别受修改影响的构件交互关系;
5) 识别受修改影响的其他构件 API 及端口;
6) 识别受修改影响的架构关系(组合,配置).

其中,第 1 项~第 3 项表示构件层面的修改影响,第 4 项~第 6 项则表示系统层面的影响.构件层面的修改不仅会影响构件本身,而且会影响到整个系统.在如下情况下,系统层面会受到构件修改的影响:

- 构件内部函数和数据的修改影响到构件 API;
- 构件内部函数和数据的修改影响到构件交互关系,包括调用关系和消息传递关系;
- 构件内部函数和数据的修改影响到构件之间的组合和配置关系;

比如,某个构件的内部修改影响到了其中的一个 API,而调用该 API 的其他构件就会受到影响.另外,由该构件组成的架构关系可能也会受到影响.对于受影响的交互关系及构件,需要重新集成及测试;对于受影响的架构关系,也需要进行回归测试.下面我们将详细介绍在构件和系统层面的修改影响防火墙.

2.2.1 构件功能修改防火墙

在构件层面,Gao 等人在早期定义过构件功能依赖防火墙 CFFW 和构件数据依赖防火墙 CDFW[1].CFFW 定义的是,根据构件功能之间的调用关系,受添加、删除、改动所影响的构件功能函数集合;CDFW 定义的是,根据构件功能之间的数据定义和使用关系,受添加、删除、改动所影响的构件功能函数集合.图 4 是一个构件功能修改防火墙的示例.假定图中的数据 *D*1 被修改,*D*1 在函数 *F*1 中被定义,并且在 *F*2 中被使用.函数 *F*6 调用 *F*1,所以 *F*6 在修改防火墙中;另外,通过数据“定义-使用”关系,函数 *F*2 也受到影响.最后,我们得到受修改影响的防火墙包括函数 *F*1,*F*2,*F*3,*F*4,*F*5,*F*6,图中用粗框表示.

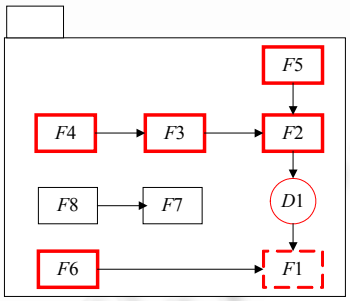


Fig.4 A sample component function and data graph firewall

图 4 构件功能和数据图防火墙示例

对于表 1 中的修改类型 AID/DID/CID,AIF/DIF/CIF,我们可以利用 $CDFW_{add}$,$CDFW_{delete}$ 和 $CDFW_{change}$ 来计算.为了计算受修改影响的构件 API,可以使用 API 功能防火墙 CAW_{API}[1].对于修改类型 AAD/DAD/CAD,AAF/DAF/CAF 以及 AP/DP/CP,由于修改的是构件内部的功能函数和数据,所以可以利用构件功能或者数据依赖防火墙计算受修改影响的构件功能函数.

2.2.2 构件交互图防火墙

在构件系统中,一个构件的修改可能会对其他构件造成影响,这种影响可以通过消息、使用等关系传递.因此,我们提出一种构件交互图修改防火墙(component interaction graph firewall,简称 CIGF).CIGF 根据构件交互图 CIG 计算出在交互关系上受修改影响的构件集合.下面的算法 1 用来在构件交互图上面计算构件修改的防火墙 CIGF.假定 C_0 是被修改的构件,CIG 和 CIG′分别是修改前后的构件交互图.对于改动的构件,需要在 CIG 和 CIG′的交集图上对修改构件做图的可达性计算;对于新添加或被删除的构件,在修改后的交互图 CIG′上进行计算.例如在图 5 中,被修改的构件是 floor panel,通过交互图防火墙的计算,我们可以得到受修改影响的其他构件(粗线框标识).

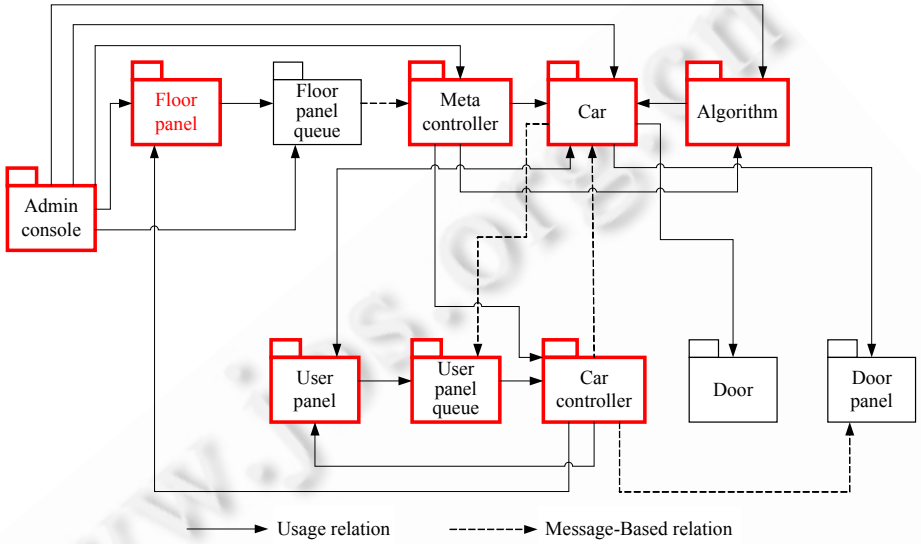


Fig.5 A sample component interaction graph firewall

图 5 构件交互图防火墙示例

算法 1. Component Interaction Graph Firewall.

Declare: C_0: Changed component;

CIG: Component interaction graph;

CIG′: Modified component interaction graph;

$CIGF(C_0,C_i,CIG,CIG')$: Component interaction graph firewall

$CIGF(C_0,C_i,CIG,CIG')$

{

if $C_i=C_0$ **then**

add C_i in $CIGF$

end if

return;

add C_i in $CIGF$;

switch (change type of C_0)

{**case** 'changed':

for each $\langle C_k,C_i\rangle \in R_{CIG}\cap(N\times N)\cap(N'\times N')$ do

$C_i \rightarrow CIGF$;

$CIGF(C_0,C_i,CIG,CIG')$

end for;
break;
case ‘added’:
for each $\langle C_k, C_i\rangle \in R'_{CIG}$ do
$C_i \rightarrow CIGF$;
$CIGF(C_0, C_i, CIG, CIG')$
end for;
break;
case ‘deleted’:
for each $\langle C_k, C_i\rangle \in R'_{CIG}$ do
$C_i \rightarrow CIGF$;
$CIGF(C_0, C_i, CIG, CIG')$
end for;
}
}

表 1 中的修改类型 AM/DM/CM 实际上也是受构件内部的函数、API 或者端口的修改影响,可以通过算法 1 得到受修改影响的其他构件.我们知道,对于构件用户、测试工程师或者 QA 工程师来说,他们感兴趣的是构件 API 修改和影响以及对系统其他构件的 API 或者端口的影响,因此,我们需要对上面的防火墙进行精化处理,使得其中只包括受修改影响的 API 和端口.这里,我们提出一种精化的构件交互图防火墙算法(算法 2).首先计算受修改影响的构件 API 防火墙,然后基于构件交互图防火墙,计算其中受影响的构件 API 和端口函数,从而得到精化的防火墙.

算法 2. Refined Component Interaction Graph Firewall.

Declare: C_0: Changed component;
CIG: Component interaction graph;
CIG′: Modified component interaction graph;
$CIGF_{API}(C_0, C_i, CIG, CIG')$: Refined component interaction graph firewall

$CIGF_{API}(C_0, C_i, CIG, CIG')$
{
switch (change type of C_0)
{**case** ‘changed’:
$CFFW_{change}[C_i, f_i]$;
$CDFW_{change}[C_i, f_i]$;
$C.F=ECAW[C_i, f_i]$;
break;
case ‘added’:
$CFFW_{add}[C_i, f_i]$;
$CDFW_{add}[C_i, f_i]$;
$C.F=ECAW[C_i, f_i]$;
break;
case ‘deleted’:
$CFFW_{delete}[C_i, f_i]$;
$CDFW_{delete}[C_i, f_i]$;
$C.F=ECAW[C_i, f_i]$;

}
mark each function in $C.F$ visited;
put $C.F$ in $CIGF$;
C_j=$CIG[C_i].rlink$; //$\langle C_j,C_i\rangle \in R_{CIG}$
while ($C_i.f_i$ in $C.F$) **do**
{
if $\langle C_j.F_j,C_i.f_i\rangle \in P(C_j) \wedge C_j.F_j$ not visited //端口 $P(C_j)$
then
$CIGF_{API}(C_0,C_i,CIG,CIG')$;
}
}

如图 6 所示,假设构件 $C1$ 中的函数 $F1$(深灰色标识)被修改,根据算法 2,可以得到 $C1$ 中函数 $F2$,$F3$ 和 $F4$ 受修改影响,其中,$F3$ 和 $F4$ 是受影响的 API,于是,我们可以通过构件交互图防火墙以及上述算法,我们得到了受修改影响的精化结果,也就是各个构件中受影响的 API 以及端口.所以,精化的构件交互图修改防火墙中包含如下函数:$F1$,API_F3,API_F4,$F5$,API_F6,$F9$,API_F11(浅灰色标识).其中,受影响函数 $F2$ 和 $F8$(虚线标识)则不在精化的防火墙中.

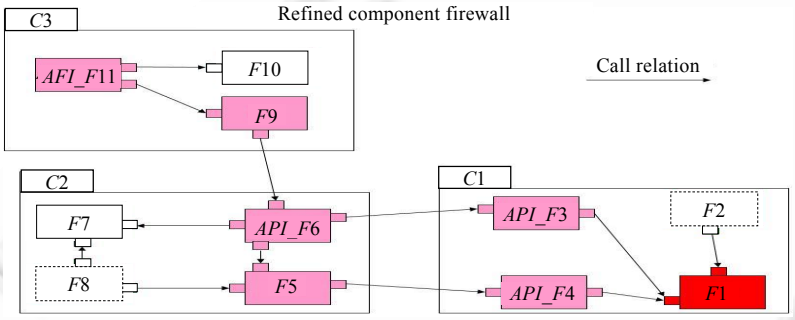


Fig.6 A sample refined component interaction graph firewall

图 6 精化的构件交互图防火墙示例

2.2.3 构件组合配置修改防火墙

在系统层面上,一个构件、配置或者组合关系的修改可能会对系统架构造成影响,这种影响通过组合和配置关系传递,可能影响到其他构件.所以,对于构件开发者来说,架构关系的修改影响分析对系统的重集成和重配置测试很重要.而以往的研究中没有提到针对构件架构修改的影响分析.这里,我们提出一种构件组合配置修改防火墙(composition and configuration tree firewall,简称 CCTF).CCTF 根据前面提出的构件组合配置树 CCT,计算出在架构上面受修改影响的构件集合.例如,在电梯系统中添加一个显示构件 indicator 后,导致了系统架构上面的影响,如图 7 所示,受影响的构件在系统配置树上用粗线框标识.

假设构件 C_i 发生修改,CCT=(N,E,R)和 CCT'=(N',E',R')分别表示修改前后的组合配置树,那么我们给出如下 3 个公式来分别计算架构层面上的改动、添加和删除的防火墙:

$$CCTF_{change}[C_i]=\{C_j \mid (C_j,C_i \in C) \wedge (\langle C_j,C_i\rangle \in R^*_{CCTr})\} \quad (1)$$

其中,C_i 表示改动的构件,C 表示构件的集合.其中,R_{CCT} 表示 CCT 中构件之间的组合配置依赖关系,R_{CCT}=$\{R_{comp},R_{conf}\}$,R_{CCTr}=$R_{CCT} \cap (N\times N) \cap (N'\times N')$,×表示笛卡尔乘积,$R^*_{CCTr}$ 表示该关系的传递闭包.

$$CCTF_{add}[C_i]=\{C_j \mid (\exists C_j)(\exists C_k)(\langle C_k,C_i\rangle \in E'-E) \wedge (C_k \in N) \wedge (\langle C_j,C_k\rangle \in R'^*_{CCT})\} \quad (2)$$

其中,C_i 表示新添加的构件,R'_{CCT} 表示 CCT 中的组合配置关系,R'^*_{CCT} 表示该关系的传递闭包.

$$CCTF_{delete}[C_i]=\{C_j\mid(\exists C_j)(\exists C_k)(\langle C_k,C_i\rangle\in E-E')\wedge(C_k\in N)\wedge(\langle C_j,C_k\rangle\in R'^{*}_{CCT}))\} \quad (3)$$

其中,C_i表示删除的构件,R'_{CCT}表示CCT中的组合配置关系,R'^{*}_{CCT}表示该关系的传递闭包.

表1中的修改类型ACP/DCP/CDP和ACF/DCF/CCF可以看成是构件的修改在组合配置关系上的影响,因此,我们可以利用公式(1)~公式(3)分别计算在系统架构层面上受添加、删除和改动构件所影响的系统其他构件.

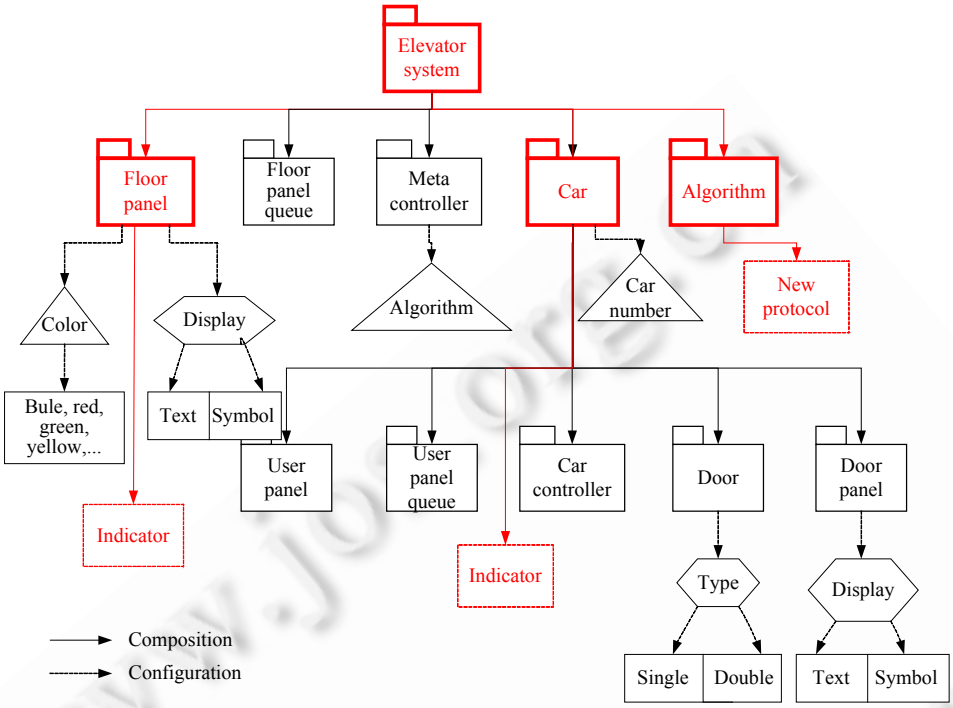


Fig.7 A sample composition and configuration tree firewall

图7 组合配置树防火墙示例

2.3 修改影响度量和评估

前面我们提出了一种针对构件软件的修改识别和影响分析的方法.为了对本方法的有效性进行度量评估,我们将提出度量指标,分别度量修改识别和影响分析,并且将在后面的实验中加以应用.

通过修改前后分析模型的比较可以识别出修改点,所以需要对识别结果进行度量.这里,我们提出两种度量指标:正确率(true_ratio)和错误率(false_ratio).正确率(true_ratio)表示本方法识别出的实际修改和真正实际修改的比率,错误率(false_ratio)表示识别出的非实际修改和识别出修改的比率.具体的度量指标定义如下:

$$True\_ratio=\frac{C_r}{C_t}\times 100\%,\ Faulse\_ratio=\frac{C_f}{C_f+C_r}\times 100\%.$$

其中,C_r表示识别出的正确修改,C_f表示识别出的错误修改,C_t表示实际的修改.

对于影响分析,我们采用精确性(precision)和包含性(inclusiveness)来度量.在这里,精确性表示本方法识别出的正确修改影响和实际受修改影响之间的比重,包含性表示使用本方法识别出的正确修改影响和本方法识别出的所有修改影响之间的比重.我们给出两个度量的形式化定义如下:

$$Precision=\frac{I_r}{I_t}\times 100\%,\ Inclusiveness=\frac{I_r}{I_a}\times 100\%.$$

该公式中,I_t表示本方法所识别出的所有受修改的影响,I_r表示本方法所识别出来的正确受修改的影响,I_a表示实际正确的受修改的影响.比如说,某构件中受修改影响的实际影响是10个API函数,而我们的方法识别出了

12 个,其中有 8 个是正确的,那么该方法的包含性是 S=8/10=80%,而精确性 P=8/12=66.7%.在实验中,我们将实际的修改作为正确的修改影响估算.

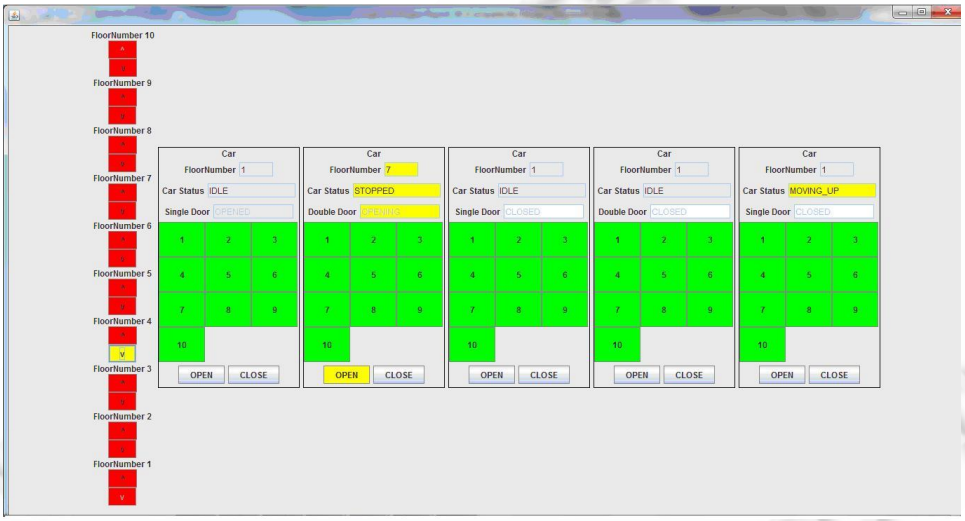
3 实验分析

为了验证所提出方法的可行性和有效性,我们对一个基于构件的电梯模拟系统进行了完整的实验分析.

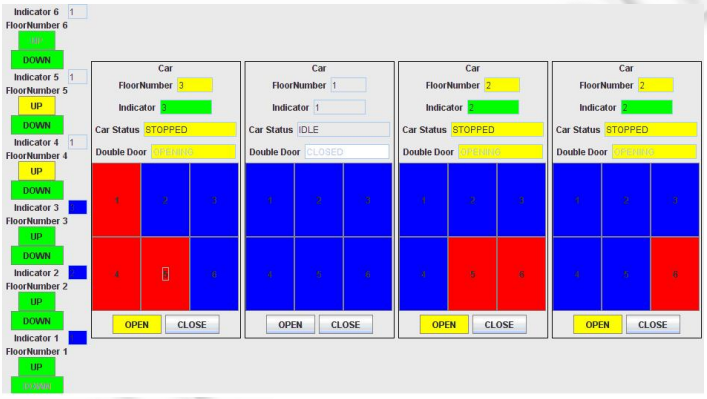
3.1 实验对象

我们选取了一个基于构件设计原则开发的电梯模拟系统.该系统由管理控制、车载、车载控制、电梯门、用户面板、楼层面板等构件组合而成.表 2 简要介绍了系统中每个构件的代码情况.

表 2 给出的是电梯系统的版本 1,也就是修改之前的版本.在实验中有如下一些修改需求,得到版本 2:添加一个新的显示功能 indicator,使得控制面板和车载里面用户面板,可以显示电梯当前到达的楼层.图 8(a)和图 8(b)是修改前后电梯模拟系统的用户界面,我们可以看到,在新版本图 8(b)中,楼层控制面板 floor panel(左边)和电梯车载 car 里(右边)分别添加了 indicator 功能.



(a)



(b)

Fig.8 Screenshots of GUI from Version 1 and Version 2 of the elevator system

图 8 电梯系统版本 1 和版本 2 的图形用户界面示例

Table 2 Summary of elevator simulation system

表 2 电梯模拟系统概况

Component name	No. of classes	No. of functions	Size (Loc)	No. of source code files
Admin console	7	32	912	4
Algorithm	6	7	190	6
Car	12	86	579	11
Car controller	4	8	111	4
Door	9	35	309	7
Door panel	7	36	224	8
Floor panel	7	28	376	7
Floor panel queue	5	16	197	4
Meta controller	5	13	970	5
User panel	11	61	647	11
User panel queue	8	26	238	8

我们这里的修改是新版本发布引起的修改,也就是系统需要增加或者升级某些功能属性(添加 indicator 的显示功能)而导致对某些构件及功能发生修改.因此,电梯系统中的多个构件及架构可能会发生修改.作为构件开发者,我们需要对构件及系统层面进行修改影响分析.

3.2 实验描述和结果

我们首先对实验对象修改前后的版本进行建模,包括版本 1 和版本 2 的功能依赖图、数据依赖图模型、API 模型、交互图模型和组合配置模型;然后在构件和系统模型上识别出相应的修改点;接着,利用本文中提出的修改影响分析方法对修改点进行防火墙分析和计算,找到受修改影响的构件内部功能和数据、API、交互关系、架构关系等.针对版本的修改需求,我们选取了 3 个实验小组,分别进行了独立修改.下面我们介绍和讨论该实验的结果.

表 3 中是实验小组记录的构件层面修改信息,我们分别记录了被修改的每个构件以及它们对应的修改类型.从表中可知,电梯模拟系统中共有 7 个构件在新版本中被修改,说明修改涉及到大部分的构件.其中 3 个实验组都有共同的修改构件 admin console,car,floor panel,user panel.除此之外,每一组还做了不同的修改,第 1 组修改了构件 metacontroller 和 car controller;第 2 组修改了 car controller 和 algorithm;第 3 组修改了 metacontroller 和 algorithm.从修改类型来看,第 2 组添加了 8 个 API 函数,修改了 4 个 API 函数,相比较其他两组,该组对 API 作了较大改动.第 1 组添加了 8 个内部数据,修改了 23 个内部函数;第 3 组添加了 13 个内部函数,修改了 19 个内部函数.因此,与第 2 组相比,第 1 组和第 3 组对内部数据和函数做了较大的修改.

Table 3 Component change report

表 3 构件修改报告

Component	API funtion			Port			Internal data			Internal function		
	A	D	C	A	D	C	A	D	C	A	D	C
Admin console	(–,–,–)	–	–	(–,1,2)	–	–	(2,1,–)	–	(1,1,1)	(13,4,–)	–	(1,1,2)
Car	(4,3,2)	–	(–,1,–)	–	–	–	(3,–,2)	–	(–,1,–)	(3,1,4)	–	(–,–,6)
Floor panel	(4,2,-)	–	(–,–,1)	–	–	–	(2,1,1)	–	(–,1,2)	(4,4,1)	–	(3,–,3)
Meta controller	(–,–,1)	–	(–,–,–)	–	–	–	(1,–,–)	–	(1,–,1)	(1,–,1)	(1,–,–)	(1,–,2)
User panel	(–,2,–)	–	(–,2,–)	–	–	–	(–,2,1)	–	–	(2,3,2)	–	(1,–,5)
Car controller	(–,1,–)	–	-	–	–	–	–	–	–	–	–	–
Algorithm	–	–	(–,1,–)	–	–	–	–	–	–	(–,1,1)	–	(–,–,1)
Floor indicator	(–,–,3)	–	–	–	–	–	(–,–,2)	–	–	(–,–,3)	–	–

注:括号内数据(x,y,z)依次来源于第 1 组、第 2 组和第 3 组.A,D,C 表示添加(add)、删除(delete)和改动(change).

我们在表 3 中发现,新版本的修改涉及到了所有的构件层面修改类型,包括内部数据和函数、API、端口.其中,内部数据和函数是主要的修改类型.这些修改类型涉及的修改方式主要是添加(add)和改动(change),这是因为在版本 2 中我们添加了新的属性.

表 4 中列出了系统层面的修改情况.在系统层面,主要存在交互、组合、配置关系的修改.由于构件 API 和端口是构件对外交互的重要接口,因此我们将它们的修改也归入到交互关系中.表中 API 和 Port 栏的数据表示

所有构件中总的修改量.第 1 组和第 3 组都添加了 4 个组合关系.另外,第 1 组添加了 14 条信息,第 2 组只添加了 1 个组合关系和 1 个配置关系.从表中可以看出,第 2 组修改了较多的 API,但修改了较少的组合及配置关系.

Table 4 System change report

表 4 系统修改报告

Group No.	Interaction									Composition			Configuration		
	API			Message			Port								
	A	D	C	A	D	C	A	D	C	A	D	C	A	D	C
G1	8	–	–	14	–	–	–	–	–	4	–	–	3	–	–
G2	7	–	4	–	–	–	1	–	–	1	–	–	1	–	–
G3	3	–	–	–	–	–	2	–	–	4	–	–	2	–	–

注:G1,G2 和 G3 分别表示第 1 组、第 2 组和第 3 组.

表 5 记录了构件层面的影响信息,我们分别列出受影响的 API 函数、端口、内部数据以及内部函数.3 个实验组都在构件 AdminConsole,car,metacontroller 和 user panel 中报告了受修改产生的影响.另外,第 2 组在构件 floorpanel、第 3 组在构件 algorithm 中都发现了修改影响.3 个组都报告了受影响的 API 函数,第 1 组还有 4 个受影响的端口.第 3 组报告的受影响的内部数据最多,而第 2 组受影响的内部函数最多.

Table 5 Component impact report

表 5 构件影响报告

Component name	Affected API functions (G1,G2,G3)	Affected ports (G1,G2,G3)	Affected internal data (G1,G2,G3)	Affected internal functions (G1,G2,G3)
AdminConsole	(–,–,–)	(1,–,–)	(5,2,6)	(1,5,–)
Car	(4,1,2)	(–,–,–)	(2,1,3)	(–,2,–)
FloorPanel	(–,1,–)	(–,–,–)	(–,–,2)	(–,–,–)
MetaController	(–,1,1)	(–,–,–)	(1,–,3)	(1,–,–)
UserPanel	(–,1,–)	(1,–,–)	(–,–,1)	(–,1,–)
CarController	(–,1,–)	(1,–,–)	(–,–,–)	(–,–,–)
Algorithm	(–,–,–)	(–,–,–)	(–,–,1)	(–,–,–)
UserPanelQueue	(–,–,–)	(1,–,–)	(–,–,–)	(–,–,–)

表 6 是系统层面的影响记录,其中,第 1 组报告的受影响的交互关系最多,共有 4 个 API、3 条消息和 4 个端口受影响;第 2 组受影响的 API 最多,共 5 个;第 3 组则共有 5 个受影响的组合和 3 个受影响的配置,是 3 个组中架构受影响最多的.这也表明,第 3 组可能在系统层面受影响比较大.

Table 6 System impact report

表 6 系统影响报告

Group No.	Affected interactions			Affected compositions	Affected configurations
	API	Messages	Ports		
G1	4	3	4	1	1
G2	5	4	–	2	1
G3	2	3	–	5	3

3.3 实验度量

3.3.1 修改识别度量

我们首先度量修改识别.根据第 2.3 节提出的两种度量指标:正确率(true_ratio)和错误率(false_ratio),得到图 9(a)中所示的 3 个小组(G1,G2,G3)的度量结果.从图中可以看出,本方法对于几乎所有修改类型的修改识别正确率都比较高,其中只有第 3 个实验组识别内部数据修改的正确率比较低.这说明我们对构件和系统的建模的安全性比较高,能够识别出实际中的大部分修改类型.另外,该方法的错误率也比较低,如图 9(b)所示,构件层面的 API function,port 以及系统层面的 message 和 configuration 的错误率都是 0,说明我们修改识别的精度比较高.值得注意的是,3 个实验组都报告了识别内部数据修改的错误率,说明我们的模型对于构件内部数据的表示还不够精确.另外,第 3 组还报告了 composition 修改识别的错误率.

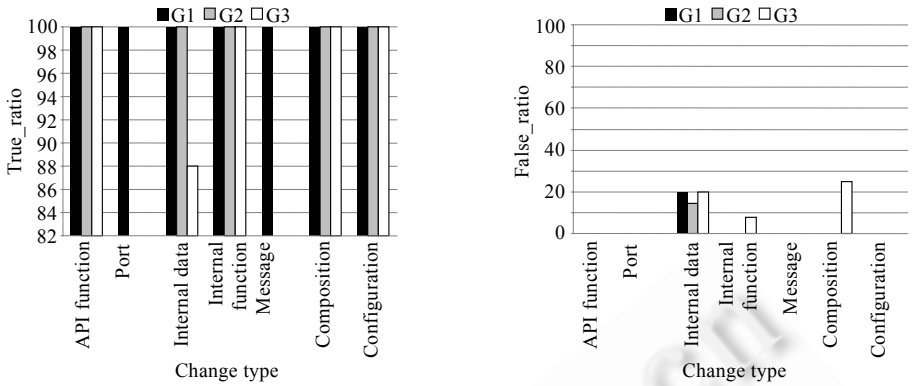


Fig.9 Measurement results of change identification

图 9 修改识别度量结果

3.3.2 影响分析度量

我们采用精确性(precision)和包含性(inclusiveness)来度量影响分析.根据这两个度量指标,我们对系统版本 2 中所得到的修改影响分析结果进行度量,如图 10 所示.我们分别度量了构件和系统层面的影响类型(impact type)的精确性和包含性.从图 10(a)中不难看出,我们的方法能够识别出受修改影响的各种影响类型.在构件层面,第 1 组和第 3 组的 API function 和 port 的精度都在 100%,第 2 组 API 的精度是 80%.在系统层面,我们所有的实验组报告了 100%的精度,所以我们的方法精度很高.另外我们发现,本方法对内部数据的修改影响分析的精确性稍低.这可能与构件内部复杂的依赖关系以及模型的精确性有关.图 10(b)中是包含性的度量结果.在构件层面,API 的包含性最高,内部数据的包含性相比较而言要低些.值得注意的是,第 3 组在系统层面的包含性和其他两组相比低了不少.我们分析,这是由于第 3 组添加了一个新的构件 floor indicator,引入了新的构件和系统依赖关系,从而给修改和影响分析带来更大的复杂性.其中,message 和 composition 都有遗漏的影响分析.总体来说,实验度量结果能够表明我们的方法能够得到比较好(尤其在系统层面)的影响分析精确性和包含性.

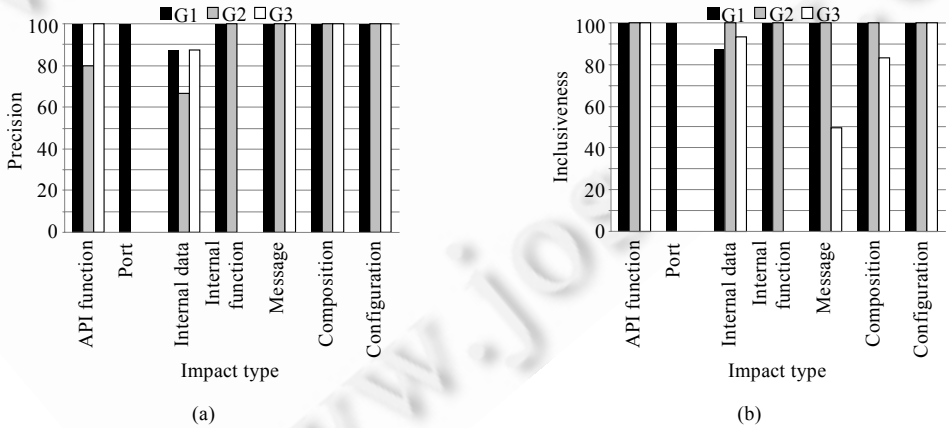


Fig.10 Measurement results of impact analysis

图 10 影响分析度量结果

3.4 实验比较

目前缺乏对构件软件的系统化修改影响分析研究,尤其是系统层面,而已有的工作主要集中在传统软件或

者构件层面.因此,我们将与已有的可以使用在构件内部防火墙的方法做比较.我们选取的比较方法来自White[7]和Zheng[8]. White在防火墙中考虑了数据流和控制流,Zheng的方法则是基于功能调用图.实验采取前面提出的精度(precision)和包含性(inclusiveness)两个度量指标来比较不同的方法.在构件层面,我们对系统版本 2中所得到的修改影响分析结果进行度量.这里的度量对象是构件内部数据、内部函数、API 和 Port.表 7 是我们利用 3 种方法得到的实验结果.

Table 7 Comparison results of different change impact analysis approaches

表 7 不同修改影响分析方法的比较结果

Metrics	White's approach (%) (G1,G2,G3)	Zheng's approach (%) (G1,G2,G3)	Our approach (%) (G1,G2,G3)
Internal data precision	(90,76.3,89.5)	(83,68,86.2)	(87.5,66.7,87.5)
Internal function precision	(100,100,–)	(100,100,–)	(100,100,–)
Api precision	(95,72,100)	(98.4,80,99.5)	(100,80,100)
Port precision	(100,–,–)	(100,–,–)	(100,–,–)
Internal data inclusiveness	(100,100,100)	(76,91,87)	(87.5,100,93.3)
Internal function inclusiveness	(100,100,–)	(100,100,–)	(100,100,–)
Api inclusiveness	(97,99.5,100)	(97,100,96)	(100,100,100)
Port inclusiveness	(100,–,–)	(100,–,–)	(100,–,–)

首先,我们比较 3 种方法的影响分析精度.从表中可以看出,我们的方法在 API 上的精度比较高,3 个实验小组都报告了比其他方法更高的精度结果.White 等人的方法精度相对比较低,因为他们采取的是数据流的分析方法,尽可能地将一些受潜在影响的内部数据都包含到防火墙中.Zheng 等人的方法只使用了构件的功能依赖图,因此,他们对于内部数据的影响分析精度相对比较低.3 种方法对内部函数的影响分析精度都达到了 100%,说明了这些方法对于内部函数建模的精度都比较高.所有方法报告的 Port 精度都很高,其中一个主要原因是由于实验结果中受修改影响的 Port 比较少.

然后,我们比较 3 种方法的包含性.表中可以看出,White 等人的方法对于内部数据的影响分析的包含性达到了 100%,说明了他们的方法在内部数据影响分析上的安全性较高.而我们的方法在内部数据的包含性上面相对低点,介于这两种方法之间.这也说明了我们的建模方法对于内部数据的表示以及相应的防火墙计算方法还有待改进.3 种方法对内部函数的影响分析包含性也都达到了 100%.但是,对于 API 的包含性,我们的方法比另外两种方法都高.这也体现了我们提出的精化的防火墙技术对于识别受修改影响的部件 API 的有效性.

3.5 实验局限性

本实验存在着一些局限性:首先,由于模型的局限性,基于防火墙的修改影响方法不能保证实际所有修改影响的构件或系统关系都在防火墙内部.因此,在防火墙外部也可能存在着修改带来的影响或程序错误;另外,我们没有考虑图形用户界面(GUI)、外部其他软件和环境的修改.我们的实验对象电梯模拟系统主要是为了学术使用而设计,所以对于复杂的工程环境来说,实验对象的规模可能仍然不够大.

3.6 实验讨论和启发

通过本实验,我们有如下一些实验启发和心得:

构件层面的修改不仅会带来构件内部的影响,而且会影响系统中的其他构件,还有构件间的交互关系,甚至架构.因此,系统化的修改影响分析手段对于基于构件的软件系统维护和质量保证是很有必要的.

在构件层面,修改越多并不一定带来更多的影响.比如,第 1 组比第 2 组修改更多的内部数据和函数,但第 2 组比第 1 组的影响更多.这是因为第 2 组修改了较多的 API 函数,并且还修改了端口,从而造成了更多的影响.修改类型对影响起着重要的作用.比如,构件 API 和端口的修改通常会影响构件之间的交互关系,从而带来更多的系统层面的影响.不同构件的修改也会造成不同的影响,比如说构件 floor panel 的修改会引起很多其他构件受到影响.因此,交互关系或者架构关系较多的核心构件以及复杂构件的修改,往往会造成更多的构件及系统层面影响.

在实验中,我们选取了 3 个实验小组,分别对同一个系统版本修改需求,做出了不同的具体修改实施.我们发

现,这 3 个实验组的修改影响分析结果有着比较明显的差异.尤其是第 3 组添加了一个新的构件 floor indicator,这也导致了较多的构件修改量,带来了更多的修改影响.因此在新版本的修改中,我们应该尽量避免添加新的构件,降低受修改影响的复杂性,从而减少软件维护的成本.

4 相关工作

修改影响分析是软件维护中的重要阶段,也是回归测试之前的必要活动,在软件工程中得到了广泛的应用.目前的相关研究大部分集中在传统软件中,其中提出了不少修改影响分析技术,而针对构件软件的修改影响分析研究较少.下面我们简要介绍传统软件和构件软件的修改影响分析研究概况.

4.1 传统软件的修改影响分析

修改影响分析在传统的面向过程和面向对象程序中得到了比较系统的研究[3,9−22],研究者们提出了传统依赖分析[19,20]、软件历史库挖掘[16]、耦合度量[18]等技术,而传统的依赖分析是其中最主要的技术[4,23,24].在面向过程程序中,一般是基于程序依赖关系建立修改影响分析模型,比如调用图[14]、控制调用图[3]、依赖图[15,25−28]等.这些技术可以分为静态和动态的.静态技术依赖于分析程序的静态信息来进行相关的修改分析,精确性相对较差,而且依赖图的规模通常也比较大[11−13,21];动态修改影响分析收集程序实际运行时的信息来构造依赖关系,然后通过这些依赖关系进行影响分析,影响的结果规模也比传统的静态分析要小,这样可以有效地减轻维护人员的负担[10,15,17].动态分析需要收集程序运行时的一些相关的动态信息,这给影响分析增加了较大的额外成本.

Kung 和 Gao 等人对面向对象程序(主要是 C++)的修改影响分析及相关回归测试做了比较系统和深入的研究[9,29−31],他们主要也是基于面向对象程序模型建立了对象关系图、模块分支图、对象状态图等,分别表示面向对象的一些属性关系,比如继承、聚合、多态,然后根据这些模型进行数据、方法、类及类库的修改识别,然后在类上利用防火墙方法进行影响分析.他们的方法已经在实验中得到了应用,并且被证实是可行有效的.

4.2 构件软件的修改影响分析

目前还没有专门针对构件软件的修改影响分析的系统研究.一些构件软件的回归测试研究中涉及到了修改影响分析,比如,Zheng 等人提出了针对商用货架产品 COTS(commercial-off-the-shelf)的构件修改识别的集成黑盒方法 I-BACCI(integrated- black-box approach for component change identification)[8,32].对于第三方的构件来说,只能从构件规约、用户接口和参考手册中获得软件内部信息.他们的研究前提条件是构件的二进制代码和文档可见.其中,COTS 构件的应用接口代码——粘合代码(glue code)是修改分析和测试选择的操作代码.他们的方法是基于构件的功能调用图(call graph).Orso 等人对基于构件的软件提出了两种回归测试用例选择方法,一种是基于代码的,另一种是基于规约的[33].这两种方法都要基于第三方提供的构件元数据(metadata).但是,该方法还需要提供很多其他信息,比如版本信息、修改数据和覆盖分析等.Gao 等人研究了针对 API 构件的回归测试用例选择方法[1],其中,修改识别分析了基于 API 的数据修改和功能修改类型.然后,基于防火墙技术提出了构件功能防火墙、API 功能防火墙及数据依赖防火墙的概念,主要针对构件层面 API 修改进行影响分析,但该方法没有强调系统层面的交互及架构修改影响.Mao 等人[34,35]提出了一种基于方法调用图的内建式构件软件回归测试方法,但没有明确考虑构件的修改影响分析问题.以上研究主要针对构件内部层面.

Wu 等人提出了一种基于 UML 的构件回归测试手段,他们的修改影响分析方法基于 UML 类图、协作图和状态图.其中,类图有类层结构的修改影响,协作图中有成员函数和交互序列的修改影响[36,37].他们是从系统行为(behavior)方面考虑修改影响对回归测试的影响.

当前工作还有一些关于软件体系结构的修改影响分析研究.Feng 等人将修改影响分析应用到基于构件的软件体系结构设计中[38],在构件之间的动态交互关系上定义了一些影响分析规则,然后在构件的界面和方法中使用切片技术分析修改影响.Hassan 等人提出了一种针对知识系统的体系结构修改影响分析方法[39],他们基于形式化模型 ASCM,考虑在软件体系结构和源代码之间的影响分析.Kotonya 等人使用体系结构描述语言 ADL[40]定义了 COTS 构件的体系结构框架,并且从修改管理角度提出了一种和构件开发文档相匹配的修改影

响分析技术.上述研究没有提出完整的从构件到系统的修改影响分析技术,而且没有考虑 API 修改对系统的体系机构的影响;而我们的方法初步解决了上述问题,不仅从构件层面进行修改影响分析,而且提出了配置、交互、组合等架构层面的影响分析手段,并且通过实验验证了方法的有效性.

5 结束语与展望

软件修改是软件的固有特征.在软件演化过程中,需要对软件进行不断的维护与修改.在每次修改实施前,我们需要对这次修改进行分析.构件软件的修改影响分析是保证构件系统质量和进行维护的重要手段.构件本身的不可见和难追踪管理给修改影响分析带来不少难度.本文从构件 API 出发,到系统交互和架构层面,提出了一种系统化的基于模型的修改影响分析方法.该方法包含了模型的修改识别、基于防火墙技术的影响分析.通过实验分析验证了我们方法的可行性和有效性.在未来的研究工作中,我们将继续完善构件及系统的修改影响分析模型,从而提高修改影响分析的精度和安全性.同时,我们将利用分析结果应用到构件软件的回归测试过程中,支持构件和系统的重测活动.另外,我们还将开发自动化的构件修改影响分析以及回归测试工具.

References:

[1] Gao J, Gopinathan D, Mai Q, He JS. A systematic regression testing method and tool for software components. In: Proc. of the Int'l Computer Software and Applications Conf. 2006. 455−456. [doi: 10.1109/COMPSAC.2006.17]

[2] Tao CQ, Li BX, Sun XB. A hierarchical model for regression test selection and cost analysis of Java programs. In: Proc. of the Asia Pacific Software Engineering Conf. 2010. 290−299. [doi: 10.1109/APSEC.2010.41]

[3] Badri L, Badri.M. Supporting predictive change impact analysis: a control call graph based technique. In: Proc. of the Asia-Pacific Software Engineering Conf. 2005. 167−175. [doi: 10.1109/APSEC.2005.100]

[4] Lin L, Prowell SJ, Poore JH. The impact of requirements changes on specifications and state machines. Journal of Software Practice and Experience, 2009,39(6):573−610. [doi: 10.1002/spe.v39:6]

[5] White LJ, Leung HKN. A firewall concept for both control-flow and data-flow in regression integration testing. In: Proc. of the Int'l Conf. on Software Maintenance. 1992. 262−271. [doi: 10.1109/ICSM.1992.242535]

[6] White LJ, Jaber K, Robinson B. Utilization of extended firewall for object-oriented regression testing. In: Proc. of the Int'l Conf. on Software Maintenance. 2005. 695−698. [doi: 10.1109/ICSM.2005.101]

[7] White LJ, Jaber K, Robinson B, Rajlich V. Extended firewall for regression testing: An experience report. Journal of Software Maintenance and Evolution: Research and Practice, 2008,20(6):419−433. [doi: 10.1002/smr.v20:6]

[8] Zheng J, Robinson B, Laurie W, Karen S. Applying regression test selection for COTS-based applications. In: Proc. of the Int'l Conf. on Software Engineering. 2006. 512−522. [doi: 10.1145/781131.781152]

[9] Gao J, Chen C, Toyoshima Y, Kung D, Hsia P. Identifying polymorphism change and impact in object-orientated software maintenance. Journal of Software Maintenance: Research and Practice, 1996,8(6):357−387. [doi: 10.1002/(SICI)1096-908X(199611)8:6<357::AID-SMR139>3.0.CO;2-S]

[10] Apiwattanapong T, Orso A, Harrold MJ. Efficient and precise dynamic impact analysis using execute after sequences. In: Proc. of the Int'l Conf. on Software Engineering. 2005. 432−441. [doi: 10.1109/ICSE.2005.1553586]

[11] Canfora G, Cerulo L. Fine grained indexing of software repositories to support impact analysis. In: Proc. of the Int'l Workshop on Mining Software Repositories. 2006. 105−111. [doi: 10.1145/1137983.1138009]

[12] Park S, Bae DH. An approach to analyzing the software process change impact using process slicing and simulation. Journal of Systems and Software, 2011,84(4):528−543. [doi: 10.1016/j.jss.2010.11.919]

[13] Jashki MA, Zafarani R, Bagheri E. Towards a more efficient static software change impact analysis methods. In: Proc. of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. 2008. 84−90. [doi: 10.1145/1512475.1512493]

[14] Sun XB, Li BX, Tao CQ, Wen WZ, Zhang S. Change impact analysis based on a taxonomy of change types. In: Proc. of the Int'l Conf. on Computer Software and Applications. 2010. 373−382. [doi: 10.1109/COMPSAC.2010.45]

[15] Law J, Rothernel G. Whole program path-based dynamic impact analysis. In: Proc. of the Int'l Conf. on Software Engineering. 2003. 308−318. [doi: 10.1109/ICSE.2003.1201210]

[16] Canfora P, Cerulo L. Impact analysis by mining software and change request repositories. In: Proc. of the Int'l Software Metrics Symp. 2005. 29−38. [doi: 10.1109/METRICS.2005.28]

[17] Huang LL, Song YT. Precise dynamic impact analysis with dependency analysis for oo programs. In: Proc. of the Int'l Conf. on Advanced Software Engineering and Its Applications. 2008. 217 −220. [doi: 10.1109/SERA.2007.109]

[18] Beszedes A, Gergely T, Farago S, Gyimothy T, Fischer F. The dynamic function coupling metric and its use in software evolution. In: Proc. of the European Conf. on Software Maintenance and Reengineering. 2007. 103−112. [doi: 10.1109/CSMR.2007.47]

[19] Briand LC, Labiche Y, Soccar G. Automating impact analysis and regression test selection based on UML designs. In: Proc. of the Int'l Conf. on Software Maintenance. 2002. 252−261. [doi: 10.1109/ICSM.2002.1167775]

[20] Diaz J, Perez J, Garbajosa J, Wolf AL. Change impact analysis in product-line architectures. In: Proc. of the 5th European Conf. on Software Architecture. 2011. 114−129. [doi: 10.1007/978-3-642-23798-0_12]

[21] Acharya M, Robinson B. Practical change impact analysis based on static program slicing for industrial software systems. In: Proc. of the Int'l Conf. on Software Engineering. 2011. 746−755. [doi: 10.1145/1985793.1985898]

[22] Sun XB, Li BX, Tao CQ. Using LoCMD to support software change analysis. Ruan Jian Xue Bao/Journal of Software, 2012,23(6): 1368−1381 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4072.html [doi: 10.3724/SP.J.1001.2012.04072]

[23] Engstrom E, Runeson P, Skoglund M. A systematic review on regression test selection techniques. Information and Software Technology, 2010,52(1):14−30. [doi: 10.1016/j.infsof.2009.07.001]

[24] Harrold MJ, Jones JA, Li TY, Liang DL. Regression test selection for Java software. In: Proc. of the ACM Conf. on OO Programming, Sysetems, Languages, and Applications. 2001. 312−326. [doi: 10.1145/504282.504305]

[25] Tao CQ, Li BX, Sun XB. An approach to regression test selection based on hierarchical slicing technique. In: Proc. of the Int'l Computer Software and Applications Conf. 2010. 347−352. [doi: 10.1109/COMPSACW.2010.67]

[26] Rothermel G, Harrold MJ. Analyzing regression test selection techniques. IEEE Trans. on Software Engineering, 1996,22(8): 529−551. [doi: 10.1109/32.536955]

[27] Rothermel G, Harrold MJ. A safe, efficient regression test selection technique. ACM Trans. on Software Engineering and Methodology, 1997,6(2):173−210. [doi: 10.1145/248233.248262]

[28] Rothermel G, Harrold MJ. Empirical studies of a safe regression test selection technique. IEEE Trans. on Software Engineering, 1998,24(6):401−419. [doi: 10.1109/32.689399]

[29] Kung D, Gao J, Hsia P, Toyoshima Y, Chen C. On regression testing of object-oriented programs. Journal of Systems and Software, 1996,32(1):21−40. [doi: 10.1016/0164-1212(95)00047-X]

[30] Kung D, Gao J, Hsia P, Toyoshima Y, Chen C. Firewall regression testing and software maintenance of object-oriented systems. Journal of Object-Oriented Programming, 1994,6(2):33−41.

[31] Kung D, Gao J, Wen Y, Toyoshima Y. Change impact identification in object-oriented software maintenance. In: Proc. of the Int'l Conf. on Software Maintenance. 1994. 202−211. [doi: 10.1109/ICSM.1994.336774]

[32] Zheng J. In regression testing selection when source code is not available. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2005. 752−755. [doi: 10.1145/1101908.1101997]

[33] Orso A, Harrold MJ, Rosenblum D, Rothermel G, Soffa ML, Do H. Using component metacontent to support the regression testing of component-based software. In: Proc. of the Int'l Conf. on Software Maintenance. 2001. 716−725. [doi: 10.1109/ICSM.2001. 972790]

[34] Mao CY, Lu YS. Strategies of regression test case selection for component-based software. Journal of Computer Research and Development, 2006,43(10):1767−1774 (in Chinese with English abstract). [doi: 10.1360/crad20061014]

[35] Mao CY. Regression testing for component-based software via built-in test design. In: Proc. of the ACM Symp. on Applied Computing. 2007. 1416−1421. [doi: 10.1145/1244002.1244307]

[36] Wu Y, Chen MH, Offutt J. UML-Based integration testing for component-based software. In: Proc. of the Int'l Conf. on COTS-Based Software. 2003. 251−260. [doi: 10.1007/3-540-36465-X_24]

[37] Wu Y, Pan D, Chen MH. Techniques for testing component-based software. In: Proc. of the Int'l Conf. on Engineering of Complex Computer Systems. 2001. 222−232. [doi: 10.1109/ICECCS.2001.930181]

[38] Feng T, Maletic JI. Applying dynamic change impact analysis in component-based architecture design. In: Proc. of the Int'l Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing. 2006. 43−48. [doi: 10.1109/SNPD-SAWN.2006.21]

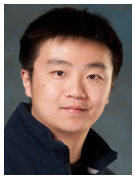
[39] Hassan MO, Deruelle L, Basson H. A knowledge-based system for change impact analysis on software architecture. In: Proc. of the Research Challenges in Information Science. 2010. 545−556. [doi: 10.1109/RCIS.2010.5507308]

[40] Kotonya G, Hutchinson J. Managing change in COTS-based systems. In: Proc. of the Int'l Conf. on Software Maintenance. 2005. 69−78. [doi: 10.1109/ICSM.2005.61]

附中文参考文献:

[22] 孙小兵,李必信,陶传奇.基于 LoCMD 的软件修改分析技术.软件学报,2012,23(6):1368−1381. http://www.jos.org.cn/1000-9825/4072.html [doi: 10.3724/SP.J.1001.2012.04072]

[34] 毛澄英,卢炎生.构件软件回归测试用例选择策略.计算机研究与发展,2006,43(10):1767−1774. [doi: 10.1360/crad20061014]



陶传奇(1984－),男,安徽枞阳人,博士生,主要研究领域为回归测试,软件维护及演化.
E-mail: chuanqi118@gmail.com

Jerry GAO(1960－),男,博士,教授,博士生导师,主要研究领域为构件测试,云测试.
E-mail: jerry.gao@sjsu.edu

李必信(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为程序切片技术及其应用,软件演化与维护,软件建模、分析、测试与验证.
E-mail: bx.li@seu.edu.cn

孙小兵(1985－),男,博士,讲师,CCF 会员,主要研究领域为程序分析,软件维护及演化,修改分析.
E-mail: xbsun@yzu.edu.cn