构造基于构件依赖关系矩阵的元数据模型*)

马良荔[1] 刘杰生[2] 李 娟[1]

(海军工程大学计算机工程系 武汉 430033)[1] (武汉 7423 信箱 武汉 430074)[2]

摘 要 构件集成到应用环境中实施集成测试时,会与其它构件产生相应的依赖关系。本文在对这些依赖关系进行概述的基础上,给出了构件直接依赖图、构件间接依赖图和构件依赖图的定义,并依据这些定义,提出了构件依赖矩阵(C_mDM)的概念,用以描述构件 C_m 与构件系统中其它构件之间的依赖关系,接着进一步定义了复杂依赖关系矩阵(C_mDDM),用以对依赖关系矩阵(C_mDM)进行更为详细的描述。对依赖矩阵方法在回归测试、构件变更处理和软件重用的应用方法进行了形式化的描述,进一步将基于依赖关系矩阵的方法应用于学校内部开发的构件 RegisterStuGrade 中,并另外选取了三个与之有关的构件,对其依赖关系进行分析,建立相应的构件依赖关系矩阵和复杂依赖关系矩阵,并与 Orso 方法、Spec 方法(基于规范说明的方法)所生成的测试用例对于构件 RegisterStuGrade 源代码的覆盖情况进行了对比,从而证明了本方法的有效性。

关键词 构件,构件集成测试,构件依赖关系,构件元数据

Construct Metadata Model Based on Component Dependency Relationship Matrix

MA Liang-li[1] LIU Jie-sheng[2] LI Juan[1]

(Department of Computer Engineering, Naval University of Engineering, Wuhan 430033, China)[1]

(Mail box 7423 , Wuhan, Hubei, 430074, China)[2]

Abstract When a software component is integrated into application environment and implemented integrated testing, it may create the dependency relationship with other components. Based on summarizing these dependency relationships, definitions of component direct dependency graph, component indirect dependency graph and component dependency graph are given. According to these definitions, the concept of component dependency matrix(C_mDM) is presented to describe the dependency relations between component C_m and other components in component system. And component detailed dependency matrix(C_mDDM) is further presented to describe C_mDM in detail. Based on these, the application methods of C_mDM to regression testing, change processing and software reusing are formally described. Lastly, the above methods are applied to component RegisterStuGrade developed by ourselves before, and other three relevant components are chose. After analyzing the dependency relationships among them, component dependency matrix and component detailed dependency matrix are constructed. And our method is compared with Orso method and Spec method(the method only based on component specification)on the cover percentage to source code of component RegisterStuGrade, presenting the comparison results. The results illustrate that our method is more effectively.

Keywords Component, Component-based software integration testing, Component dependency relationship, component Metadata

1 引言

目前依赖分析是构件软件系统中一个重要的研究方向[1],Stafford 和 Wolf[2]对构件软件系统体系结构及其构件之间的接口进行了依赖分析,在使用软件体系结构描述语言对构件之间接口和行为关系进行刻画的基础上,提出了体系结构层次上的依赖分析技术,文中将之称为链(chaining),用以将体系结构中间接相关的元素连接起来,并将该技术应用于称为 Aladdin 的工具中,Aladdin 是用于 Ada 和 C++程序的实现级上的依赖分析工具。文中首先给出了中间表示,接着在该表示基础上进行了语言级的分析。Vieira 和 Richardson[3-5]使用了构件依赖模型来管理构件系统中的依赖关系。构件依赖模型为一个图集,表示系统构件之间的特别联系,这些联系是通过构件提供的服务发生关联。Murphy 和 Nottkin[6]利用源结构和调用图抽取系统模型,从而进行源程序结构分析。Ferrante 和 Ottenstein[7]引入了程序依赖图的概念进行编译优化;Sangal 和 Jordan[8]使用依赖模型管理复杂的软件体系结构。Zhao[9]使用依赖分析对描述的软件体系结构进行描述,按照该方法,可以将软件体系结构用体系结构描述语言(ADL)如 Wright, ACME, RAIPE 等进行形式化,并在此基础上进行相关的依赖分析。

本文在上述研究的基础上,讨论构件在集成到新的应用环境时可能对其它构件产生的影响,以便构件使用方实施集成测试,对这些影响进行分析形成依赖关系,建立相关的依赖矩阵,并以元数据的形式提供给构件使用方以帮助提高构件的可测试性。

2 构件之间的依赖关系

一般来说,构件在集成到新的应用环境时,会对其它构件产生影响。新集成的构件会对系统中已有构件产生一定的影

*)湖北省自然科学基金资助(编号 2005ABA266)。**马良荔** 副教授,博士,研究方向为软件工程、软件测试。

响，会使用系统已有的构件或被已有的构件使用，这都会产生显式的直接和间接的依赖关系。另外，还有与系统环境有关的隐式依赖关系，目前，有四种依赖关系：显式依赖关系、显式间接依赖关系、隐式直接依赖关系和隐式间接依赖关系[10]。

系统通过交互、合作和协作提供系统功能，交互、合作和协作会在构件之间产生依赖关系。构件之间的依赖关系可定义为构件之间的相互关联性以及接口之间的约束，用以支持特定的功能或配置。系统功能并不是固定在某一个特定的构件上，通常情况下，一组构件相互依赖以提供复杂的系统功能。由于系统能实现的各种组合功能是由不同的构件完成的，因此对构件的任何修改会使得系统的组合功能发生变化。另外，替换一个新构件也会使得构件之间的依赖关系发生变化[11]。

传统的依赖分析都建立在数据流和控制流基础上[12]，本文的依赖分析在对构件之间的关系以及与集成环境之间的关系建立相应的构件依赖图，进而实施集成测试，一般来说，这些依赖关系有[11]：

1)数据依赖(Data Dependency)：是不同构件在数据集成时产生的，指在某一个构件中定义的数据在另一个构件中使用；

2)控制依赖(Control Dependency)：是不同构件在控制集成时产生的，不是显式依赖，是构件进行远程调用或一般传送时产生的；

3)时间依赖(Time Dependency)：是指一个构件的行为在构件系统中另一个构件行为的后面或前面；

4)状态依赖(State Dependency)：是指如果系统或系统的某些部分没有处在特定的状态下，则构件的某个行为就不会产生；

5)因果依赖(Cause and Effect Dependency)：是指构件的某个行为隐含在另一个构件中；

6)输入/输出依赖(Input/Output Dependency)：输入依赖是指某个构件需要来自另一个构件的相应信息；输出依赖是指某个构件向另一个构件提供相应的信息；

7)上下文依赖(Context Dependency)：是指构件运行所需的特定环境；

8)接口依赖(Interface Dependency)：是在构件集成时构件之间的接口产生的依赖关系，基于接口的依赖关系是构件系统中的主要依赖关系，当某个构件需要另一个构件完成特定的任务时，首先发送某个消息通过其接口触发某个事件，该事件会激活另一个构件，这之间的关系称为接口依赖。

本文将上述这些依赖关系以矩阵形式表示，并以元数据形式嵌入在提供给构件使用方的构件中，以提高构件的可测试性。

3 构件依赖关系模型

本文引入了依赖关系矩阵的概念用以描述某个构件与系统中其它构件之间的关系，目的在于提高构件的可测试性。

定义 1 构件系统中带依赖标记的构件直接依赖关系图(Component Direct Dependency Relationshi PGraph，简称 CDDRG)可表示为一个三元组 $G_{CDDRG}=(N_c,E_c,A_c)$，为一个直接图，其中

N_c 是顶点集合，每个顶点表示系统中的一个构件；

E_c 是依赖边的集合，每条边表示构件之间的直接依赖关系；

A_c 是边上的标记集合，每个标记表示依赖边之间的依赖类型，依赖类型分为 8 种，分别是数据依赖、控制依赖、时间依赖、状态依赖、因果依赖、输入/输出依赖、上下文依赖和接口依赖，在 G_{CDDRG} 中分别记为 $D,C_r,T,S,C/E,I/O,C_t$ 和 I，其中输入/输出(I/O)依赖可进一步标记为输入依赖为 I_n 和输出依赖为 O_u，可表示为 $A_c=(D_t,D_d)$，其中 D_t 为依赖类型，按照前面的 8 种依赖类型分别进行标记，D_d 为与前面类型相关的依赖描述；假设顶点 N_a 表示构件 a，顶点 N_b 表示构件 b，如果构件 a 依赖于构件 b，则从顶点 N_a 向顶点 N_b 画一条有向边。

例如，假设有如下示例图。

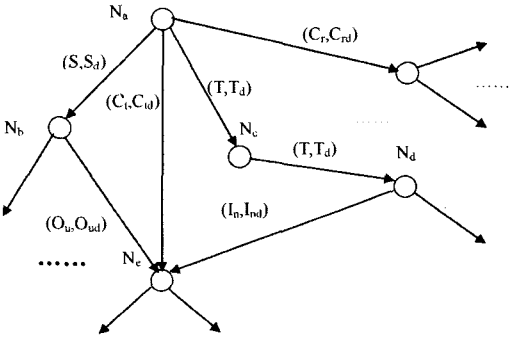

图 1 带依赖标记的构件直接依赖关系图(CDDRG)

图 1 中，顶点 N_a 和顶点 N_b 之间为状态依赖关系，其具体的依赖内容在 S_d 所指向的描述中，即构件 a 必须要构件 b 在某种状态 x 下才能执行行为 y；顶点 N_a 和顶点 N_c 之间为时间依赖关系，其具体的依赖内容在 T_d 所指向的描述中，即只有构件 c 发生一系列行为 α 后构件 a 才能执行一系列行为 β，或者是只有构件 a 发生一系列行为 α 后构件 c 才能执行一系列行为 β，其中具体的时间关系在 T_d 给出。后面的顶点分析与此类似。

要发现上述定义 1 中构件系统的依赖关系，则必须明确构件之间的所有直接和间接依赖关系，直接和间接关系中包括显式和隐式两种层次上的关系。

定义 2 构件系统的构件依赖关系图(Component Dependency Relationship Graph，简称 CDRG)可表示为一个二元组 $G_{CDRG}=(G_{CDDRG},G_{CIDRG})$，是指构件依赖关系图分为两个部分，分别表示构件之间的所有直接依赖关系和间接依赖关系，其中直接依赖关系用 G_{CDDRG}(见定义 1)表示，间接依赖关系用 G_{CIDRG} 表示。

定义 3 构件间接依赖关系图 G_{CIDRG} 为 G_{CDDRG} 的扩展，表示系统中使用 Warshall 传递闭包算法计算的所有间接依赖关系。即如果在 G_{CDDRG} 中，假定顶点 N_a 表示构件 a，顶点 N_b 表示构件 b，顶点 N_c 表示构件 c，且顶点 N_a 到顶点 N_b 存在一条有向边 ab，顶点 N_b 到顶点 N_c 也存在一条有向边 bc，且边 ab 和边 bc 上的标记类型相同，则构件 a 间接依赖于构件 c，在 G_{CIDRG} 中从顶点 N_a 到顶点 N_c 画一条有向边，用虚线来表示，边上的标记按原有的标记进行标识；如果边 ab 和边 bc 上的标记类型不同，则要根据具体情况进一步分析确定依赖类型。例如，可以将上面图 1 的示例扩展为构件依赖关系图 CDRG，如图 2 所示。

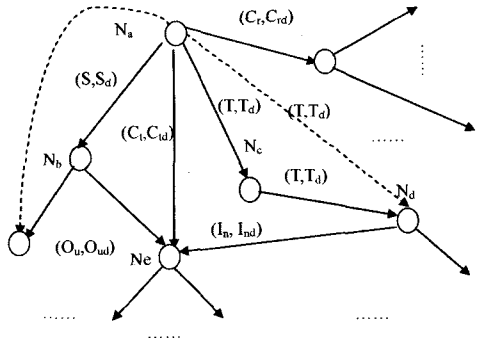

图 2　带依赖标记的构件依赖关系图(CDRG)

4　构件依赖关系矩阵

在上述对构件依赖图进行定义的基础上，进一步给出构件依赖矩阵的定义。

定义 4　假设构件系统由 n 个构件组成，n 个构件分别为 $C_1,C_2,\cdots,C_n$(这里的 C_i 为构件使用方标识的特定表示方式)，$C_m(1\leqslant m\leqslant n)$ 为构件系统中的一个构件，则构件 C_m 的构件依赖关系矩阵 $C_mDM_{8\times n}$ 可定义为(为不失一般性，按照前面的描述将构件 C_m 与构件系统中其它构件之间的依赖关系仍然刻画为 8 种依赖关系)：

$$C_mDM_{8\times n}=\begin{bmatrix} d_{D1} & d_{D2} & \cdots & d_{Dn} \\ d_{C1} & d_{C2} & \cdots & d_{Cn} \\ d_{T1} & d_{T2} & \cdots & d_{Tn} \\ d_{S1} & d_{S2} & \cdots & d_{Sn} \\ d_{CE1} & d_{CE2} & \cdots & d_{CEn} \\ d_{IO1} & d_{IO2} & \cdots & d_{IOn} \\ d_{CO1} & d_{CO2} & \cdots & d_{COn} \\ d_{I1} & d_{I2} & \cdots & d_{In} \end{bmatrix}$$

这里 $d_{Di},d_{Ci},d_{Ti},d_{Si},d_{CEi},d_{IOi},d_{COi},d_{Ii}$ 分别表示构件 C_m 和构件系统中构件 C_i 之间的数据依赖、控制依赖、时间依赖、状态依赖、因果依赖、输入/输出依赖、上下文依赖和接口依赖。

在上述矩阵中，有 8 行和 n 列，第 i 列表示构件系统中的某个构件 C_i，第 i 行表示某个构件 C_m 与系统中其它构件之间的不同依赖关系。这里，列 m 的所有值均为 1(因为上述矩阵讨论的是构件 C_m 与构件系统中其它构件之间的依赖关系，因此列 m 讨论的是构件 C_m 与构件 C_m 自身的关系)。

这里假定 Γ^{α} 表示构件 C 与构件系统中其它构件之间的依赖关系 α，α 可以是数据依赖、控制依赖、时间依赖、状态依赖、因果依赖、输入/输出依赖、上下文依赖和接口依赖。因此当且仅当 C_m 依赖于 C_i 时，$C_m\Gamma^{\alpha}C_i$ 为真。

$$d_{Di}=\begin{cases}1 & \text{如果 } C_m\Gamma^{D}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{D}C_i\end{cases}\qquad d_{Ci}=\begin{cases}1 & \text{如果 } C_m\Gamma^{C}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{C}C_i\end{cases}$$

$$d_{Ti}=\begin{cases}1 & \text{如果 } C_m\Gamma^{T}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{T}C_i\end{cases}\qquad d_{Si}=\begin{cases}1 & \text{如果 } C_m\Gamma^{S}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{S}C_i\end{cases}$$

$$d_{CEi}=\begin{cases}1 & \text{如果 } C_m\Gamma^{CE}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{CE}C_i\end{cases}\qquad d_{IOi}=\begin{cases}1 & \text{如果 } C_m\Gamma^{IO}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{IO}C_i\end{cases}$$

$$d_{COi}=\begin{cases}1 & \text{如果 } C_m\Gamma^{CO}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{CO}C_i\end{cases}\qquad d_{Ii}=\begin{cases}1 & \text{如果 } C_m\Gamma^{I}C_i\\ 0 & \text{如果}\rightarrow C_m\Gamma^{I}C_i\end{cases}$$

上面引入了构件依赖关系矩阵的概念，在此基础上，进一步给出构件复杂依赖矩阵的概念以更有效地支持测试用例的生成，定义如下：

定义 5　假设构件系统由 n 个构件组成，n 个构件分别为 $C_1,C_2,\cdots,C_n$(这里的 C_i 为构件使用方标识的特定表示方式)，$C_m(1\leqslant m\leqslant n)$ 为构件系统中的一个构件，则构件 C_m 的复杂依赖关系矩阵 $C_mDDM_{8\times n}$ 如下(这里还是包含 8 种依赖关系)：

$$C_mDDM_{8\times n}=\begin{bmatrix} dd_{D1} & dd_{D2} & \cdots & dd_{Dn} \\ dd_{C1} & dd_{C2} & \cdots & dd_{Cn} \\ dd_{T1} & dd_{T2} & \cdots & dd_{Tn} \\ dd_{S1} & dd_{S2} & \cdots & dd_{Sn} \\ dd_{CE1} & dd_{CE2} & \cdots & dd_{CEn} \\ dd_{IO1} & dd_{IO2} & \cdots & dd_{IOn} \\ dd_{CO1} & dd_{CO2} & \cdots & dd_{COn} \\ dd_{I1} & dd_{I2} & \cdots & dd_{In} \end{bmatrix}$$

这里 $dd_{Di},dd_{Ci},dd_{Ti},dd_{Si},dd_{CEi},dd_{IOi},dd_{COi},dd_{Ii}$ 分别表示构件 C_m 与构件系统中构件 C_i 的数据依赖、控制依赖、时间依赖、状态依赖、因果依赖、输入/输出依赖、上下文依赖和接口依赖，其值为一个二元组。

在上面的矩阵中，也包含 8 行和 n 列，第 i 列表示构件系统中的某个构件 C_i，第 i 行表示某个构件 C_m 与系统中其它构件之间的不同依赖关系。这里，列 m 的所有值均为 1。

这里引入 Γ^{α} 表示依赖关系 α，其定义同定义 4。因此，当且仅当构件 C_m 与构件 C_i 有依赖关系时，$C_m\Gamma^{\alpha}C_i$ 为真，此时列 i 行 α 的值可表示为一个二元组，定义如下：

$(1,ptr_i^r)$

上述二元组中 ptr_i^r 是指向依赖关系列表的指针，从该指针可以访问不同的依赖关系列表，每种依赖关系列表分别以链表的形式表示，链接相关的直接和间接依赖关系，r 表示依赖关系类型如数据依赖等，i 表示构件 C_m 与构件 C_i 有依赖关系，矩阵 $C_mDDM_{8\times n}$ 中列 m 的所有值均为(1,null)。

下面定义构件复杂依赖关系关系矩阵 $C_mDDM_{8\times n}$ 的值如下：

$$dd_{Di}=\begin{cases}(1,ptr_i^D) & \text{如果 } C_m\Gamma^{D}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{D}C_i\end{cases}$$

$$dd_{Ci}=\begin{cases}(1,ptr_i^C) & \text{如果 } C_m\Gamma^{C}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{C}C_i\end{cases}$$

$$dd_{Ti}=\begin{cases}(1,ptr_i^T) & \text{如果 } C_m\Gamma^{T}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{T}C_i\end{cases}$$

$$dd_{Si}=\begin{cases}(1,ptr_i^S) & \text{如果 } C_m\Gamma^{S}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{S}C_i\end{cases}$$

$$dd_{CEi}=\begin{cases}(1,ptr_i^{CE}) & \text{如果 } C_m\Gamma^{CE}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{CE}C_i\end{cases}$$

$$dd_{IOi}=\begin{cases}(1,ptr_i^{IO}) & \text{如果 } C_m\Gamma^{IO}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{IO}C_i\end{cases}$$

$$dd_{COi}=\begin{cases}(1,ptr_i^{CO}) & \text{如果 } C_m\Gamma^{CO}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{CO}C_i\end{cases}$$

$$dd_{Ii}=\begin{cases}(1,ptr_i^{I}) & \text{如果 } C_m\Gamma^{I}C_i\\ 0 & \text{如果 } \rightarrow C_m\Gamma^{I}C_i\end{cases}$$

按照上面的描述，二元组$(1,ptr_i^r)$表示构件 C_m 与其它构件之间不同类型的依赖关系，指针 ptr_i^r 以链表的形式创建，其中包含两个域，分别为 *NameID* 和 *nextptr*，这里 *NameID* 表示相应的标识符，*nextptr* 表示在指向下一个标识符的指针。例如，如果 $C_m\Gamma^{D}C_i$ 为真，则 ptr_i^D 的内容如图 3 所示(假

设数据 $a,b,\cdots,n$ 在构件 C_i 定义，在构件 C_m 使用。这样，构件 C_m 与构件 C_i 具有数据依赖关系）。

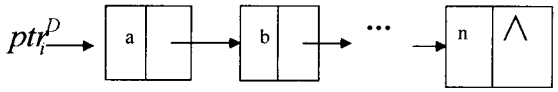

图 3 $(1,ptr_i^D)$ 的结构

在实际应用中，由于构件源代码对于构件使用方未知，如果构件 C_m 与构件 C_i 具有数据依赖关系，构件开发方应提供具体的产生数据依赖的数据名称以及位置，以便更好地实施构件测试。这时可以在图 3 中增加一个域 *location* 用以标识在构件 C_i 中定义而在构件 C_m 中使用的位置。另外还要考虑在构件 C_m 中定义而在构件 C_i 中使用的情况，这种情况通过构件 C_i 与构件系统中其它构件之间的依赖关系矩阵来表现。

5 应用分析

前面给出的依赖关系矩阵在应用实际中可以解决构件软件工程所涉及的多个方面，如回归测试，构件变更处理、构件重用等，本文对此进行了相关的分析。

5.1 在构件变更和回归测试中的应用

构件发生变更的情况分为两种情况，一是构件开发方对该构件内部进行的修改，这种修改会对该构件所集成的构件系统中于该构件有依赖关系的构件产生影响，该构件的修改直接影响这些被依赖的构件；二是当该构件有依赖关系的构件实施修改后，对该构件产生的影响。本文将前一种变更称为主动变更；后一种变更称为被动变更。

对这两种变更所产生影响的处理都可以根据前面的定义 4 来进行分析，当某个构件 C 的依赖矩阵中的某个元素 $d_{ij}=1$ 时，如果对构件 C 实施变更，则可通过其依赖关系矩阵对与其有依赖关系的相应构件实施变更；被动变更的处理实际上转化为影响使该构件产生被动变更的特定构件 A 的主动变更，可以通过构件 A 的依赖关系矩阵来处理。在上述分析基础上，给出如下定义 6。

定义 6 假定存在构件系统 $CS=\{C_i \mid 1\leqslant i\leqslant n\}$，且有构件 C_m（$C_m\in CS$，且 $1\leqslant m\leqslant n$）和构件 C_m 的依赖关系矩阵 $C_mDM_{8\times n}$，当对构件 C_m 实施变更后，对于 $\forall d_{jk}\in C_mDM_{8\times n}$，当且仅当 $d_{jk}=1$（其中 $1\leqslant j\leqslant 8$，$1\leqslant k\leqslant n$，且 $k\neq m$）时，有

$$\mathrm{affC}(C_m)=\{C_k \mid C_k\in CS\}$$

其中 affC(C_m) 为受构件 C_m 变更影响的构件集合。

下面给出计算 affC(C_m) 的算法（用伪代码来表示）。

算法 ComaffC(m,n,CDM [8][n]) / * m 为变更的构件 C_m 在构件系统中的构件编号，n 为构件系统中构件的数目，CDM 表示其依赖关系矩阵的二维数组 * /

输入：变更的构件 C 的编号 m，构件 C 的依赖关系矩阵 CDM

输出：受构件 C 变更影响的构件集合 affC[k]（存放构件系统中的构件编号）

对于 CDM [i][j]中的每个 i，每个 j，

if CDM [i][j]=1 then

for (k=0;k<n;k++)

从下标 k 一直向前搜索至 0 看数组 affC 中是否已经放入构件编号 j；

if 没有放入 then

affC[k]=j;

endif;

endif

在上述定义和算法的基础上，对由于构件 C_m 变更而影响的构件进行如下处理：

假定存在 affC(C_m) 和构件 C_m 的复杂依赖关系矩阵 $C_mDDM_{8\times n}$，$\forall dd_{ij}\in C_mDDM_{8\times n}$，且 $j\neq m$，$dd_{ij}=(1,ptrij)$，$ptrij$ 为指向具体依赖关系说明的指针，可以根据该指针所指向链表的位置来进行相应的变更。

当构件系统中的某个构件 C_m 发生变更时，根据 C_m 的依赖关系矩阵 $C_mDM_{8\times n}$ 找出受该构件变更影响的构件集合 affC(C_m)，对该集合中的每个构件根据复杂依赖关系矩阵 $C_mDDM_{8\times n}$ 实施相应的变更。

对构件进行变更处理后，需要确定相应的回归测试用例生成，生成的测试用例应覆盖集合 affC(C_m) 的每个元素。

5.2 在构件重用上的应用

当构件系统中的某个构件 C_m 从一个应用程序 A 重用到另一个应用程序 B 中时，根据构件 C_m 的依赖关系矩阵 $C_mDM_{8\times n}$ 进行相关处理，处理框架按照如下步骤：

1）检查构件 C_m 的依赖关系矩阵 $C_mDM_{8\times n}$：

对于 $\forall d_{ij}\in C_mDM_{8\times n}$，按照该矩阵的 8 行分别进行处理：

$i=1$ 时，if $d_{1j}=1$ then 将构件 C_j 中有数据依赖关系的数据加入到构件 C_m 中相关类的数据中；

$i=2$ 时，if $d_{2j}=1$ then 检查构件 C_m 是否有远程调用功能，如没有，则无须考虑；

$i=3$ 时，if $d_{3j}=1$ then 将构件 C_m 的一些行为产生之前必须产生的构件 C_j 的行为加入到构件 C_m 中，与 C_m 封装在一起；

$i=4$ 时，if $d_{4j}=1$ then 将构件 C_m 的一些行为产生之前必须产生的构件 C_j 的行为加入到构件 C_m 中作为相关类的方法；

$i=5$ 时，if $d_{5j}=1$ then 将与构件 C_m 有因果关系的构件 Cj 的行为加入到构件 C_m 中；

$i=6$ 时，if $d_{6j}=1$ then 将应用程序 A 相关构件输入到构件 C_m 的输入接口进行分析，看应用程序 A 是否能提供这样的输入，如不能，则将其输入加入到 C_m 中，作为其中的数据或方法；

$i=7$ 时，if $d_{7j}=1$ then 将应用程序 B 的上下文与原来构件 C_m 集成的应用程序 A 的上下文进行比较，检验是否能有效集成构件 C_m；

$i=8$ 时，if $d_{6j}=1$ then 将应用程序 B 与构件 C_m 之间的接口进行分析，检验是否能有效地集成；

2）将新生成的构件 C_m' 提供给应用程序 B。

根据上述处理框架得出的新构件 C_m'，该构件可集成到新的应用程序 B 中，根据依赖关系矩阵和伏在依赖关系矩阵可清楚看出与构件 C_m 有依赖关系的构件名以及相应的依赖类型。

6 实例研究

在给出上述形式化研究和应用分析后，将这些研究应用在内部开发的构件 RegisterStuGrade 中，该构件用在基于构件开发技术开发的学校信息管理系统中，用于管理学生成绩，由于该构件是内部开发的构件，因此源代码可知。

将构件 RegisterStuGrade 的源代码进行分析，并运用上

述理论研究模型实施构件集成测试，以验证上述方法的有效性。由于整个学校信息管理系统涉及的构件有58个，分析复杂度高，因此只选择了其中与其关系紧密的三个构件，分别是构件RegisterStuInf(完成学生信息的登记工作)、构件ManaTecInf(完成教师授课信息的管理信息)、构件ManaCouInf(完成课程信息的管理工作)。

为了验证本文提出的方法的有效性，首先根据上述四个构件的源代码，构件RegisterStuGrade记为C_1，构件RegisterStuInf记为C_2，构件ManaTecInf记为C_3，构件ManaCouInf记为C_4，找出这四个构件之间的依赖关系，形成4个矩阵，以C1为分析对象，与构件系统中其它三个构件的依赖关系矩阵记为$C_1DM_{8\times4}$，矩阵表示如下所示：

$$C_1DM_{8\times4}=\begin{bmatrix}1&1&0&1\\1&0&1&0\\1&0&0&1\\1&1&0&1\\1&1&0&1\\1&1&1&1\\1&0&0&1\\1&1&1&1\end{bmatrix}$$

使用矩阵$C_1DM_{8\times4}$进而生成复杂依赖关系矩阵$C_1DDM_{8\times4}$：

$$C_1DDM_{8\times4}=\begin{bmatrix}(1,\Lambda)&(1,ptr12)&0&(1,ptr14)\\(1,\Lambda)&0&(1,ptr23)&0\\(1,\Lambda)&0&0&(1,ptr34)\\(1,\Lambda)&(1,ptr42)&0&(1,ptr44)\\(1,\Lambda)&(1,ptr52)&0&(1,ptr54)\\(1,\Lambda)&(1,ptr62)&(1,ptr63)&(1,ptr64)\\(1,\Lambda)&0&0&(1,ptr74)\\(1,\Lambda)&(1,ptr82)&(1,ptr83)&(1,ptr84)\end{bmatrix}$$

从上述矩阵可以看出所选用的四个构件之间的依赖关系。构件RegisterStuGrade在与其它三个构件集成在一起实施集成测试时，根据这些依赖关系，生成了49个测试用例。这里将该方法记为Matrix方法，为了说明本方法的有效性，与不使用依赖矩阵，在构件RegisterStuGrade源代码未知的情况，仅仅依据构件的相关说明，生成了35个测试用例，将该方法称为Spec方法；在此基础上，引入Orso等人在文献[13]和[14]中给出的元数据方法，生成了43个测试用例，这里将该方法称为T-Metadata方法。然后对构件RegisterStuGrade的源代码进行分析，分别计算三种方法生成的测试用例对构件RegisterStuGrade源代码的覆盖情况，结论如表1所示。

表1 三种方法生成的测试用例覆盖情况表

测试覆盖准则(以%表示)	Spec方法	T-Metadata方法	Matrix方法
语句覆盖率	91.1%	93.5%	96.9%
分支覆盖率	91.3%	92.1%	95.7%
因果覆盖率	91.7%	93.8%	94.1%
边界值覆盖率	92.1%	94.2%	94.3%
c-使用	90.8%	92.7%	94.5%
p-使用	91.4%	92.8%	95.1%

注：表中所取的数据均为平均值

结束语 本文在对构件之间的依赖关系进行概述和分析的基础上，给出了构件直接依赖图、构件间接依赖图和构件依赖图的定义，并依据这些定义，给出了构件依赖关系矩阵C_mDM的概念，用以描述构件C_m与构件系统中其它构件之间的依赖关系，接着进一步定义了复杂依赖关系矩阵C_mDDM，用以对上述C_mDM进行详细的描述。基于此，分析了依赖关系矩阵在应用实际中可以解决回归测试，构件变更处理等方面的问题；并进一步将基于依赖关系矩阵的方法应用于学校内部开发的构件RegisterStuGrade中，并从学校管理信息系统中选取了另外三个与之有关的构件，对其依赖关系进行分析，建立相应的构件依赖关系矩阵和复杂依赖关系矩阵，进而生成测试用例，并与Orso方法、仅依据规范说明的方法所生成的测试用例对于构件RegisterStuGrade源代码的覆盖情况进行了对比，从而验证了本方法的有效性。

下一步的工作是形成构件依赖关系自动确认方法，并用其它大型的应用实例进行说明以推动该方法的不断完善。

参考文献

[1] Brown A, Wallnau K. The current State of Component-based Software Engineering. IEEE Software, September 1998:37-47

[2] Stafford J A, Wolf A L. Architecture-level Dependence Analysis for Software System. International Journal of Software Engineering and Knowledge Engineering, 2001, 11(4):431-453

[3] Vieira M, Richardson D J. Classifying and Dealing with Dependencies in Large Component-Based Systems // 15th International Conference on Software & Systems Engineering & their Applications(ICSSEA 2002). Paris, December 2002:231-239

[4] Vieira M, Richardson D J. Analyzing Dependencies in Large Component-based Systems // Proceedings of the 17th IEEE International Conference on Automated Software Engineering. Edinburgh, UK, September, 2002:221-229

[5] Vieira M, Richardson D J. The Role of Dependencies in Component-Based System Testing and Evolution // Proceedings of th IWPSE-02, 24th Internarional Conference on Software Engineering(ICSE 02, Orlando, USA). May 2002:62-68

[6] Murhpy G C, Notkin D N. Lightweight Lexical Siurce Model Extractin. TOSEM, 1996, 5(3):262-292

[7] Ferrante J, Ottenstein K J, Warren J D. The Program Dependence Graph and its Use in Optimization. ACM Transaction on Programming Languages and Sytems, 1987, 9(3):319-349

[8] Sangal N, Jordan E. Using Dependency Models to Manage Complex Software Architecture // ACM SIGPLAN International Conference on Object-Oriented Programming. Systems, Language, and Applications 2005, San Diego, California, USA, 2005:167-176

[9] Zhao J. Using Dependency Analysis to Support Software Architecture Understanding. In: New Technologies on Computer Software. International Academic Publishers, September 1997:135-142

[10] Cheesman J, Daniels J. UML Component: A Simple Process for Specifying Component-Based Software. In: Addison Wesley ISBN 0-201-7-851-5, 2001:1-65

[11] Bixin L. Managing Dependencies in Component-based Systems Based on Matrix Model // Proceedings of. Net Object Days. Sept. 2003:67-78

[12] Horwitz S, Reps T, Binkley D. Interprocadural Slicing Using Dependency Graphs. In: ACM Transaction on Programming Languages and Systems. January, 1990, 22(1):26-60

[13] Orso A, Harrold M J, et al. Using Component Metacontent to Support the Regression Testing of Component-Based Software // Proceedings of IEEE International Conference on Software Maintenance (ICSM'01). Florence, Italy, November, 2001:716-725

[14] Harrold, Orso A, etc. Using Component Metadata to Support the Regression Testing of Component-based Software. Technical Report GIT-CC-01-38. College of Computing, Georgia Institute of Technology, 2001:1-10