软件体系结构演化模型

王映辉[1,2], 王立福[1]
(1. 北京大学软件工程国家工程研究中心, 北京 100871; 2. 陕西师范大学计算机学院, 陕西西安 710062)

摘 要: 软件演化包括静态演化和动态演化两个方面. 作为软件的蓝图的 SA(软件体系结构), 为人们宏观把握软件的整体结构和软件演化提供了一条有效的途径. 在描述 SA 的构件-连接件模型的基础上, 首先针对 SA 的静态演化, 建立了 SA 邻接矩阵和可达矩阵, 凭借矩阵变换与运算对 SA 静态演化中的波及效应进行了深入地分析和量化界定, 同时给出了构件在 SA 中贡献大小相对量的计算方法. 其次, 针对 SA 的动态演化, 给出了 SA 动态语义网络模型, 分析了 SA 动态语义网络中基于不动点的浸润过程收敛的判定, 提出了邻接矩阵原子过滤的概念, 进而指出, SA 动态演化过程可用一系列邻接矩阵原子过滤在时刻上的逻辑衔接来描述. 最后给出了两个层面上对 SA 演化波及效应的分析框架. 为 SA 演化的管理、控制、利用、评价和量化描述奠定了基础.

关键词: 软件体系结构; 软件演化; 可达矩阵; 波及效应; 浸润
中图分类号: TP311.52 文献标识码: A 文章编号: 0372-2112 (2005) 08-1381-06

Research about Model and Ripple Effect Analysis of Software Architecture Evolution

WANG Ying-hui[1,2], WANG Li-fu[1]
(1. *National Engineering Research Center for Software Engineering, Peking University, Beijing* 100871, *China*;
2. *School of Computer Science, Shaanxi Normal University, Shaanxi, Xi'an* 710062, *China*)

Abstract: Software architecture(SA) acts as blueprint and skeleton of software. It's an availability approach for people to grasp whole macroscopical architecture and evolution of software based on SA. Firstly, for aim of static evolution of SA, SA relation matrix and reachability matrix are created. Depending on matrix shift & calculation, ripple effect of SA evolution can be analyzed and its quantity can be ascertained. At the same time, an approach for calculating relative quantity of component effect is shown. Secondly, for aim of dynamic evolution of SA, SA dynamic semantic network model is described, convergence condition of soak process in this SA dynamic semantic network model is addressed. A concept of atomic filtration of adjacency matrix is put forward, so SA dynamic evolution is expressed by a series of continuous atomic filtrations of adjacency matrix. An analysis framework of ripple effect of SA evolution based on two of levels is proposed. All are credible foundation to manage, control, utilize, value SA evolution, and describe SA evolution with quantity.

Key words: software architecture; evolution; reachability matrix; ripple effect; soak

1 引言

构造性和演化性是软件的两个基本特性[1]. 软件进行渐变并达到所希望的形态就是软件演化, 软件演化由一系列复杂的变化活动组成. 对软件变化的控制是软件开发者历来追求的目标. 引起软件变化的原因是多方面的, 如基础设施的改变、功能需求的增加、高性能算法的发现、技术环境因素的变化等等. 所以, 对软件变化甚至演化进行理解和控制显得比较复杂和困难.

近几年来, 软件体系结构 SA(Software Architecture) 已成为软件研究的热点之一, 它作为软件的蓝图为人们宏观把握软件的复杂性和整体变化性提供了一条有效途径. SA 的形式化描述、基于构件和 SA 的软件组装与开发、SA 实践、面向模式的 SA 研究等等, 这一切已构成了 SA 的基本技术和理论体系. 另外, SA 是软件生命周期的早期产品, 着重解决软件系统结构和需求向实现平坦过渡的问题, 是软件生命周期中开发、集成、测试和维护更改的基础; 此外, 在生命周期早期基于 SA 进行软件检测和修改的相对低代价性[2]. 显而易见, 要刻画复杂的软件演化, 并对演化中的影响效应进行观察和控制, 自然应从 SA 演化研究开始.

收稿日期: 2004-04-20; 修回日期: 2005-05-12
基金项目: 国家(863) 高技术研究发展计划(No. 2001AA113171); 国家重点基础研究发展规划(973)(No. 2002CB312006); 陕西省自然基金(No. 2003F35); 国家博士后基金(No. 20040350251)

目前,对 SA 的演化研究方面,形式化描述方法[3,4]更加偏向于对软件构造性的描述,而将 SA 的演化性隐藏在各种“运算”之中,不能直观而定量地反映 SA 演化活动及其波及效应;UML 方法[5]对 SA 的描述基于不同的视图,虽然从不同的角度实现了对 SA 结构的可视性,但也没有涉及 SA 演化活动的定量刻画方法;其它有关软件演化的研究基本注重软件演化过程的定性分析[6~8]. 本文提出一种 SA 演化的定量刻画模型,并结合矩阵运算对 SA 演化过程及其波及效应进行有效分析.

2 软件体系结构模型

目前对 SA 的定义形式多样,本文采用许多文献[2,6,9~11]中比较公认的定义,即 SA 是组成系统的构件以及构件与构件之间交互作用关系(连接件)的高层抽象.

定义 **1** 构件 **Com** 是系统中承担一定功能的数据或计算单元. $Com=\langle \text{Ports}, Imp\_Bs\rangle$,其中 Ports 是构件接口集合,$Imp\_Bs$ 是构件的实现. $\text{Ports}=\{\text{Port}_1, \text{Port}_2, \cdots, \text{Port}_n\}$,而 $\text{Port}_i=\langle \text{ID}, Publ_i, Ext_i, Prvt_i, Beha_i, Msg_i, Cons_i, NFun_i, Ply_i\rangle$,其中,ID 为构件标识,$Publ_i$ 是 Port_i 向外提供的功能的集合,Ext_i 是外部通过 Port_i 向构件提供的功能的集合,$Prvt_i$ 是 Port_i 和 $Beha_i$ 是分别是 Port_i 的私有属性集合和行为方法集合,Msg_i 是 Port_i 产生的消息的集合,与事件有关,$Cons_i$ 是 Port_i 的行为约束,$NFun_i$ 是 Port_i 的非功能说明,Ply_i 是与连接件交互点自集合.

定义 **2** 连接件 **Con** 是系统中承担构件间交互语义的连接运算单元. $Con=\langle \text{ID}, Beha, Msg, Nfun, Cons, Role\rangle$. 其中,ID 为连接件的标识,$Beha$ 是连接件行为语义的描述,Msg 是构件与各 Role 交互事件产生的消息的集合,$Nfun$ 为连接件的非功能描述,$Cons$ 是连接件语义约束的集合,Role 是连接件与构件交互点的集合.

定义 **3** **SA** 是由构件通过连接件及其之间的语义约束形成的拓扑网络 $N_{SA}=\langle Coms, Cons, Const\rangle$(称 SA 网络或 SA 网络模型),其中 $Coms$ 是构件的集合,$Cons$ 是连接件的集合,$Const$ 是构件与连接件之间的语义动态、静态约束.

定义 **4** **SA** 简化模型 $SA_S=\langle Coms, \text{Connectors}\rangle$;其中,$Coms$ 同定义 3;$\text{Connectors}=\{\text{Connector}_{lr}\}$,$\text{Connector}_{lr}=\langle Cons, G\text{-}CConst\rangle$,$Cons$ 同定义 3,而 $G\text{-}CConst$ 表示构件之间的语义约束.

要说明的是,定义 3 中的连接件和定义 4 中的连接件之间是一种“展开”和“合并”的关系,分别适应于本文中对 SA 动态演化和静态演化的研究. 下面不再作特别说明.

定义 **5** 由于系统需求、技术、环境和分布等因素的变化而最终导致的 SA 按照一定的目标形态的变动,称之为 SA 演化.

就单一软件来说,SA 演化被划分为静态演化和动态演化两个方面. 对 SA 在非运行时刻的修改和变更称为 SA 的静态演化(如软件版本的升级等),而软件在运行时刻的 SA 变换称为 SA 的动态演化. 本文从静态[16]和动态演化两个方面建立其刻画模型,并对 SA 的演化进行统一描述.

3 SA 静态演化模型及其波及效应

为了描述的方便,根据定义 4 先给出一个实例.

假设一个系统的 SA 由 5 个构件和 6 个连接件构成,其交互关系模型如图 1 所示. 其中构件 Component1 通过连接件 Connector1、Connector3、Connector5 Connector6 分别与构件 Component2、Component3、Component4 和 Component5 发生交互关系;其它构件之间的关系依然.

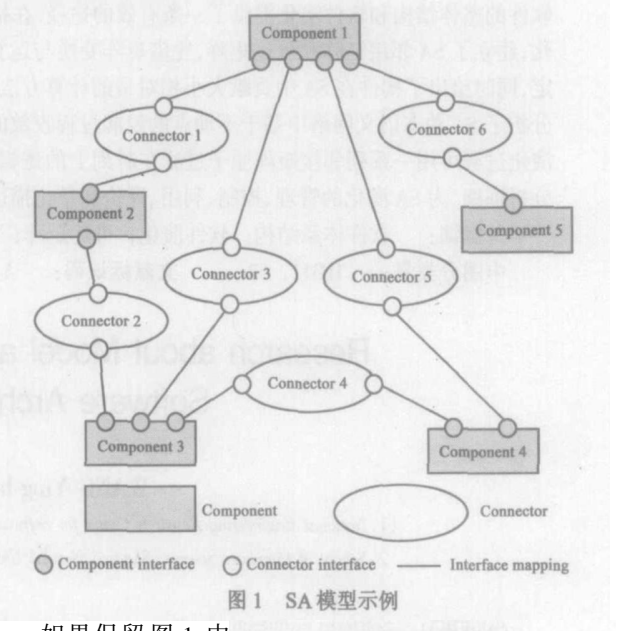

图 1 SA 模型示例

如果保留图 1 中连接件的基本语义即方向语义,则得到图 2 所示的有向图. 其中,构件 Component2 和构件 Component3 之间存在双向连接语义,说明图 1 中的连接件 Connector2 至少承载着双向的交互方向语义关系.

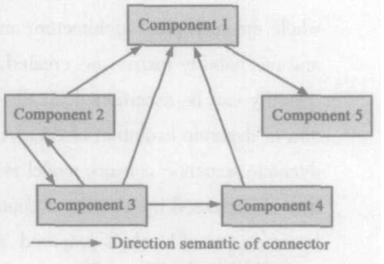

图 2 图 1 的简化模型

3.1 SA 邻接矩阵与变换

定义 **6** 图 3 是由图 2 构造出的 SA 结构关系邻接矩阵图.

图 3 中每一个黑色的圆圈关联着两个具有直接交互方向

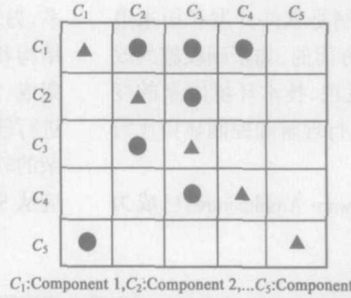

图 3 图 2 的 SA 结构关系邻接矩阵图

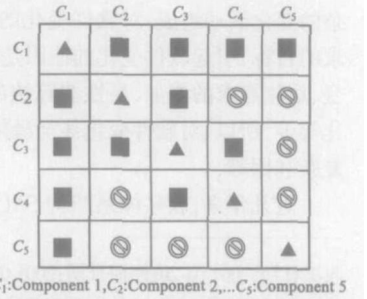

图 4 图 1 的 SA 语义关系连接矩阵图

关系构件, 其方向性由行和列分别表示, 行为被关联方向的末尾构件, 列为被关联方向的起始构件. 另外, 图 3 中的三角形表示构件的自交互关系, 这种关系隐藏在构件自身内部. 本文并不关心构件自身内部的交互关系.

定义 **7**　图 4 是由图 1 构造出的 SA 语义关系连接矩阵图.

图 4 中三角形的含义与图 3 中的相同, 黑色方块表示行列中对应两个构件之间存在着直接的交互语义关系, 带横线的圆圈表示行列中对应两个构件之间不存在或存在间接的交互语义关系.

图 4 中的黑色方块和带横线的圆圈是对不同环境和应用中语义形式的抽象, 在具体化时, 矩阵图在表面上的这种对称性可能会被打破, 变为非对称形式(非对称矩阵图). 例如, 当连接件的语义表示两个构件间最大无环可达路径上构件的个数(包括两端)时, 列 C_3 和行 C_5 交叉处的圆圈变为数值 4(参见图 2), 即 $C_3 \rightarrow C_2 \rightarrow C_1 \rightarrow C_5$, 而列 C_5 与行 C_3 交叉处的圆圈变为数值 0, 即不可达; 等等.

由此可见, SA 结构关系邻接矩阵图是 SA 语义关系连接矩阵图的一种简单形式. 也就是说, 在 SA 结构关系邻接矩阵中, 我们将语义信息简化成了构件间是否存在直接交互以及交互的方向语义, 有利于 SA 结构演化的研究.

定义 **8**　给 SA 结构关系邻接矩阵图和 SA 语义关系连接矩阵图中的十字交叉处填充上具有一定语义的数值后, 形成的对应矩阵分别称为 **SA** 结构关系邻接矩阵和 **SA** 语义关系连接矩阵. SA 结构关系邻接矩阵简称 SA 邻接矩阵或邻接矩阵.

例如, 在图 2 中, 当自构件 C_i 到构件 C_j($i, j = 1, 2, \cdots, 5$)存在一条直接的连接时, 图 3 中的第 C_i 行和 C_j 列交叉处填充为 1, 否则为 0, 其它连接依次类推. 这样, 就形成了对应的 SA 结构关系邻接矩阵. 又如, 在图 1 中, 当自构件 C_i 到构件 C_j($i, j = 1, 2, \cdots, 5$)存在一条可达路径时, 将图 4 中的第 C_i 行和 C_j 列交叉处填充为 1, 否则为 0, 其它连接依次类推. 这样, 就形成了对应的 SA 语义关系连接矩阵.

由图论的相关理论[13]可知, SA 结构关系邻接矩阵通过一定的运算(如闭包运算)后可以转换为 SA 语义关系连接矩阵.

3.2　SA 静态演化中的波及效应和构件贡献

定义 **9**　组成 *SA* 中构件的变化所引起的 *SA* 中其它构件的一系列的连带变化过程称为波及效应.

在 SA 的静态演化中, 表面上看是对构件的增加、替换、删除, 但这种变化蕴涵着一系列的连带和波及效应, 更多的表现为变化的构件或连接件与其相关联的构件或连接件的重新组合和归整.

定义 **10**　设 SA 的邻接矩阵:

$M_R = (C_{ij})$, 其中 C_{ij} 表示构件 C_i 与构件 C_j 之间的连接件; $i, j = 1, 2, \cdots, n$, 并且

$$C_{ij} = \begin{cases} 1, \text{当 } C_i \text{ 与 } C_j \text{ 之间存在直接交互关系时;} \\ 0, \text{当 } C_i \text{ 与 } C_j \text{ 之间不存在直接交互关系时;} \end{cases}$$

另外, 设集合 $X = \{C_1, C_2, \cdots, C_i, \cdots, C_n\}$, 则邻接矩阵 M_R 对应的关系: $R \subseteq X^2$,

于是, 关系 R 的传递闭包 $R^+ = R \cup R^2 \cup \cdots \cup R^n$.

则 R^+ 对应的矩阵 $M_{R+} = M_R \vee R_R^2 \vee \cdots \vee M_R^n = \bigvee_{k=1}^{n} M_R^k$, 其中 $M_{R_i}^i = M_{R_{i-1}}^{i-1} \vee M_R$, $i = 2, 3, \cdots, n$;

此时, 称 M_{R^+} 为 SA 的可达矩阵, 简称可达矩阵.

例如, 由图 3 表示的 SA 对应的邻接矩阵

$$\boldsymbol{M}_R = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$; 则 SA 的可达矩阵

$$\boldsymbol{M}_{R^+} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

由可达矩阵各列可以直接得出(参照图 2): 构件 C_1(第 1 列)到构件 C_5(第 5 行)可达; 构件 C_2(第 2 列)和 C_3(第 3 列)到其它构件(各行)分别都可达; 构件 C_4(第 4 列)到 C_1(第 1 行)和 C_5(第 5 行)分别可达; 构件 C_5(第 5 列)到其它构件都不可达.

由此可见, 通过可达矩阵非常容易界定某一构件变化所影响的其它构件. 进而, 当某一组构件发生变化时, 可以圈定被影响或波及到的其它构件的范围.

另外, 通过可达矩阵, 可以对各个构件对 SA 影响的大小(称为贡献或贡献大小)进行确定和排序. 具体办法是, 求取可达矩阵各列的元素之和, 其大小代表相应列头构件所影响的其它构件的个数. 当然, 这个数值越大, 该构件对 SA 的影响(贡献)越大. 例如, 图 2 的可达矩阵 $\boldsymbol{M}_{R^+}$ 的各列头构件 C_1、C_2、C_3、C_4 和 C_5 对应的各列元素之和分别为 1、5、5、2 和 0, 所以各构件对 SA 贡献的大小排列为 $C_2 \geqslant C_3 > C_4 > C_1 > C_5$.

SA 静态演化中的基本活动包括删除构件、增加构件、修改构件、构件合并与构件分解五种, 各种活动引起的影响具体参见文献[16].

最后要说明的一点是, 在对 SA 的静态演化分析中, 虽然仅仅保留了方向语义, 但可以看出, 这在宏观层面研究 SA 的静态演化的波及效应分析是非常必要的.

4　SA 动态演化模型及其波及效应

SA 的动态演化比静态演化更为复杂. 为了能把握其宏观动态性, 仅仅依靠邻接矩阵和可达矩阵来刻画是不够的. 下面结合基于动态语义网的润湿理论[14], 对 SA 动态演化做进一步地描述.

4.1　SA 动态语义网

在 SA 的动态演化研究中, 要求 SA 模型不仅具有刻画静态结构特性的能力, 还应具有描述构件状态变化、构件间通过连接件的相互作用和局部网络的集群行为等动态特性的

能力. 因此, 需要将构件间的相互作用与约束细化为构件与连接件间的相互作用与约束. 由定义 3 可得出如下的推论.

推论 **1** SA 网络模型 N_{SA} 能用有向图刻画.

在SA 的动态演化中, 可以设想, 信息流(数据流或控制流) 是通过 NSA 中的连接件在构件之间流动的, 当这种流动到达某一构件时, 继续驱动目标构件, 进而辐射出去或终止. 这种反复出现的过程称为 NSA 浸润.

定义 **11** 对被考察的系统 S 形式化后的动态语义网是一个有向图 $G_S=\langle V(G_S), E(G_S), F_S\rangle$, 其中 $V(G_S)$ 是策动源; $E(G_S)$ 是浸润传播途径集; F_S 是 $V(G_S)$ 到 $E(G_S)$ 的有序集合上的函数, 表示某种可预定义的形式化语义联系.

定理 **1** NSA 与 G_S 同构.

证明略.

定义 **12** 在 SA 的动态演化中, 将 N_{SA} 称为 SA 动态语义网, 记为 G_{SA}.

可见, SA 动态语义网也是一个有向图 $G_{SA}=\langle V(G_{SA}), E(G_{SA}), F_{SA}\rangle$, 其中 $V(G_{SA})$ 是策动源和被激构件集; $E(G_{SA})$ 是浸润传播途径集; F_{SA} 是 $V(G_{SA})$ 到 $E(G_{SA})$ 的有序集合上的函数, 表示某种可预定义的形式化语义联系.

4.2 SA 动态语义网中的浸润过程

动态语义网的浸润过程[14]可以抽象为有向图结点间的数据或控制信息的驱动, 即是一个由源结点策动的传播.

定义 **13** 浸润域是浸润发生过程的载体, 它是一个有向图 $G=\langle V(G), E(G), F\rangle$, 其中 $V(G), E(G), F$ 含义类似于定义 11.

推论 **2** 在 G_{SA} 上同样可以定义与定义 13 完全相同含义的浸润域.

以下概念的描述是在的浸润域 G_{SA} 上进行的, 不再做特别说明.

定义 **14** 策动源 $Act\_Nodes^{(t)}=\{V_i \mid V_i\in V(G)\wedge S_i=1\wedge 0\leqslant i<|V(G)|\}$, 即 t 时刻 $V(G)$ 中状态为 1 的所有结点构成的集合. S_i 表示 V_i 的状态, 定义为

$$S_i=\begin{cases}1, & F_i(x_1, x_2, \cdots, x_n)>\theta(x_1, x_2, \cdots, x_n)\\ 0, & \text{其他}\end{cases}$$

其中: $F_i(x_1, x_2, \cdots, x_n)\in R$ 是 V_i 的激励变换函数; $\theta_i(x_1, x_2, \cdots, x_n)\in$ 是 V_i 的门限函数.

可见 $S_i\in\{0, 1\}$, 其中 0 表示抑制, 1 表示活跃; $x_i(1\leqslant i\leqslant n)$ 表示结点 V_i 的输入.

另外, 如果定义 $Y_i=F_i(x_1, x_2, \cdots, x_n)/S_i$ 作为结点 V_i 的输出; 可见, 当 S_i 抑制时 Y_i 是无意义的.

定义 **15** 图 G 的邻接矩阵 $\boldsymbol{X}=(x_{ij}), 0\leqslant i, j<|V(G)|$, 其中 $x_{ij}=\begin{cases}1, & S_i=1\wedge\langle i, j\rangle\in F_G(E(G))\\ 0, & \text{其他}\end{cases}$; $F_G(E(G))$ 表示图 G 对应的边集 $E(G)$ 中对应的所有边的两端结点之间的某种可预定义的语义联系的集合.

为了有效地描述 SA 的动态演化, 定义 15 对定义 8 做了进一步的细化.

t 时刻图 G 的全部结点状态的集合表示为 $S^{(t)}=\{S_i \mid V_i\in V(G), 0\leqslant i\leqslant|V(G)|\}$.

定义 **16** 对整个矩阵 $\boldsymbol{X}$ 的操作, 等价于对矩阵 $\boldsymbol{X}$ 中每个元素的操作. 同样 θ_i 对整个矩阵 $\boldsymbol{X}$ 的操作, 等价于对矩阵 $\boldsymbol{X}$ 中每个元素的操作.

定义 **17** 浸润算子 Soak 是在时刻 t 起作用的、限定 F_i 和 θ_i 对 $X^{(t)}$ 进行一个离散时间步长的、逻辑上并行运算的操作.

定义 **18** 一个浸润步是指浸润算子 Soak 作用在三元组 $\langle Act\_Nodes^{(t)}, X^{(t)}\rangle$ 上的结果: $\langle \bigcup_{V_i\in Act\_Nodes^{(t)}}\{V_j \mid \langle i, j\rangle\in F_G(V(E))\wedge F_i(X^{(t)}, S^{(t)})\rangle\theta(X^{(t)}, S^{(t)})\rangle^{(t+1)}, X^{(t+1)}, t+1\rangle$, 记为 $Soak\_Step^{(t+1)}$.

性质 **1**

$Soak\_Step^{(t+1)}=Soak(Soak\_Step^{(t)}), t=0, 1, \cdots, N.$

定义 **19** 浸润过程是有限个时刻相继的浸润步的逻辑衔接.

定义 **20** 在浸润过程中, 当 t 到达某一时刻 T 时, 如果出现了 $X^{(T+1)}=X^{(T)}$, $S^{(T+1)}=S^{(T)}$, 则称此时刻的点为不动点.

定义 **21** 存在不动点的浸润过程称为收敛的, 否则就称为不收敛的.

定理 **2**[14] 一个浸润过程收敛的充要条件是浸润中的每个结点 V_i 的 F_i 当 $t=T$ 足够大时收敛于 θ_i.

证明 参阅文献[14].

定义 **22** 在 G_{SA} 的浸润过程中, 当 t 足够大到 T 时, 如果所有结点的状态和对应的邻接矩阵的元素全不为 0, 则称 SA 是最优的.

可见, 在最优的 SA 中, 各构成元素之间"紧密"合作, 表现为内聚性较高.

4.3 SA 动态演化分析

SA 的动态演化分析比静态演化分析复杂, 建立 SA 动态演化过程模型是动态分析的关键.

定义 **23** SA 特定条件域 U_i 是指与 SA 相关的软件系统在一次运行中所对应的环境和约束条件集.

在 SA 特定条件域 U_i 下, SA 对应的软件系统在运行中数据和控制信息的流动总是在若干个确定的构件和连接件之间传播, 这若干个确定的构件和连接件组成了新的 SA, 称为 SA 特定条件域 U_i 下的 SA, 记为 SA_i.

定义 **24** SA 条件域 U 是指软件系统所有可能的 SA 特定条件域 U_i 的并集, 即 $U=\bigcup_{i=1}^{N}$, N 为 U 的非空子集的个数.

定义 **25** $\forall U_i\in U, \exists SA_i\subseteq SA(i=1, 2, \cdots, N, N$ 为 SA 条件域 U 的非空子集的个数), 称为不动点软件体系结构 SA_i, 简称不动点 SA_i. 其中 $SA_i\subseteq SA$, $N_{SA}=\langle Coms, Cons, Const\rangle$, $N_{SA_i}=\langle Coms_i, Cons_i, Const_i\rangle$, 且 $Coms_i\subseteq Coms$, $Cons_i\subseteq Cons$, $Const_i\subseteq Const$.

定义 25 说明, 当 SA 特定条件域确定之后, 对应的 SA_i 也是确定的. 从 G_{SA} 浸润的全过程来看, $SA_i\subseteq SA$ 相当于 SA 在此特定条件域下的一个稳态.

定义 **26** 在一个 SA 中, 自一个不动点 SA_i 到另一个不动

点 $SA_j(i, j = 1, 2, \cdots, N, N$ 为 SA 条件域 U 的非空子集的个数$)$之间的变换称为 SA 的不动点转移.

可见, 软件运行环境条件组合的变化决定了 SA 的不动点转移, 即 SA 条件域 U_i 的变化决定了 SA 的不动点转移. 另外, SA 的动态演化过程可用在 G_{SA} 的浸润过程来刻画, 而浸润步是 SA 动态演化的基本活动.

通过不动点的转移, SA 由一种形态变为另外一种形态. 实际上是: 软件在特定的环境条件下(包括运行的硬件环境, 输入和使用人员等), 软件只有部分成分(构件和连接件)在起作用. 也就是说, 数据或信息只在部分成分中流动, 且决定了在这种特定环境下的局部的 SA, 即不动点 $SA_i(i = 1, 2, \cdots, N, N$ 为 SA 条件域 U 的非空子集的个数$)$.

定义 27　定义 21　SA_k 对应的邻接矩阵为 $\boldsymbol{X}_k = (x_{ij}), 0 \leqslant i, j < |V(G_{SA})|$, 其中

$$x_{ij} = \begin{cases} Y_i,\ S_i = 1 \wedge \langle i, j\rangle \in F_{G_{SA}}(E(G_{SA})) \wedge Con_i \in SA_k \wedge Con_j \in SA_k \\ 0, \text{其他} \end{cases};$$

$F_{G_{SA}}(E(G_{SA}))$ 与定义 9 中相同, $Con_i \in SA_k$ 和 $Con_j \in SA_k$ 分别表示构件 Con_i 和 Con_j 都属于 SA_k 中的构件. 而 $k = 1, 2, \cdots, N, N$ 为 SA 条件域 U 的非空子集数.

设不动点 SA_i 对应的邻接矩阵为 $\boldsymbol{X}_i$, 则 $\boldsymbol{X} = \bigvee_{i=1}^{N} \boldsymbol{X}_i$, $\vee$ 表示矩阵元素的"逻辑或"运算. 例如, 当 $N = 3$ 时, 设

$$\boldsymbol{X}_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{X}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, X_3 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \text{则}$$

$$\boldsymbol{X} = \begin{bmatrix} 1\vee 0\vee 1 & 1\vee 0\vee 1 & 1\vee 1\vee 0 \\ 1\vee 1\vee 1 & 0\vee 0\vee 0 & 0\vee 0\vee 1 \\ 0\vee 1\vee 1 & 0\vee 0\vee 1 & 1\vee 1\vee 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

定义 28　SA 条件域 U 的每一个非空子集 $U_j(j = 1, 2, \cdots, N)$ 对应的一个 X_j 称为 $\boldsymbol{X}$ 的一个过滤, 简称 $\boldsymbol{X}$ 过滤, 且 X_j 和 X 的阶数相同, $\boldsymbol{X} = \bigvee_{i=1}^{N} \boldsymbol{X}_i$, $\vee$ 表示矩阵元素的"逻辑或"运算, N 为 SA 条件域 U 的非空子集的个数.

按照定义 22 中"逻辑或"运算规则, 将 $\boldsymbol{X}$ 进行"拆分", 可得 M 个 $\boldsymbol{X}$ 过滤.

性质 2　互异的 $\boldsymbol{X}$ 过滤的个数 M 最大为 $2^{X\text{中元素个数}}$ 个. 当 SA 条件域 U 非空子集的个数 N 大于 $2^{X\text{中元素个数}}$ 时, 对应的 $\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_N$ 中有相同的.

性质 2 也意味着, 通过 $\boldsymbol{X}$"拆分"获取的每一个 $\boldsymbol{X}$ 过滤并不都是有意义的. 也就是说, 这种"拆分"下的每个 X 过滤对应的 SA 不总是实际存在的.

定义 29　设 G_{SA} 对应的邻接矩阵 $\boldsymbol{X} = (X_{11})$, $0 \leqslant i, j < N$, $N = |V(G_{SA})|$, 称 $X_{11}, X_{12}, \cdots, X_{1n}, X_{21}, X_{22}, \cdots, X_{2n}, X_{n1}, X_{n2}, \cdots, X_{nn}$ 为 $\boldsymbol{X}$ 的一个序列, 其中 $X_{ij} \in \{0, 1\}$.

定义 30　对 $\boldsymbol{X}$ 的一个序列按 0 和 1 的组合值从小到大进行编码, 形成一个编码序列, 称为 $\boldsymbol{X}$ 的编码序列. 在 $\boldsymbol{X}$ 的编码序列中, 除第一个元素之外的每个元素都有前驱, 除最后一个元素之外的每个元素都有后继. 对应的 0 和 1 的组合值称为 $\boldsymbol{X}$ 编码序列元素的序号.

例如, 对于 $\boldsymbol{X}$ 的一个序列 $X_{11}, X_{12}, X_{21}, X_{22}$, 即 $\boldsymbol{X}$ 为 2* 2 的矩阵, 按 0 和 1 的组合从小到大编码序列为: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 则元素 1001 的前驱为 1000, 而后继为 1010, 序号为十进制的 9; 其它依次类推.

定义 31　如果将 X 编码序列中元素的序号当作 X 的下标, 并用此元素中的 0 或 1 代替矩阵 $\boldsymbol{X}$ 中相应元素的值时, 形成的一个矩阵系列 $\boldsymbol{X}_0, \boldsymbol{X}_1, \cdots, \boldsymbol{X}_{2^{N*N}-1}$($N$ 为 $\boldsymbol{X}$ 矩阵的阶数), 称为 $\boldsymbol{X}$ 的过滤系列. 每个 $\boldsymbol{X}_i$ 称为一个 $\boldsymbol{X}$ 原子过滤, 简称原子过滤 $\boldsymbol{X}_i$.

例如, 上例中 $\boldsymbol{X}$ 的编码序列中的元素 1001, 对应的 $\boldsymbol{X}_9 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, 元素 1110 对应的 $\boldsymbol{X}_{14} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$.

可见, $\boldsymbol{X}$ 过滤系列中的元素是矩阵, 除了第一个和最后一个元素之外, 每个元素也有前驱和后继.

推论 3　SA 动态演化过程可用一系列在时刻上相继的 $\boldsymbol{X}$ 原子过滤的逻辑衔接来刻画.

推论 3 给出了 SA 动态演化过程的量化矩阵描述模型. 由此可见, 通过矩阵的变换运算[13], 可以对 SA 动态演化过程进行较好地量化分析和描述. 另外, 由于每一个特定的 $\boldsymbol{X}$ 原子过滤对应一个特定的 SA. 所以, 对此特定的 SA 静态演化波及效应分析可利用第 2 节中的相关方法.

从不动点转移的角度来看一个软件, SA 静态演化实质上是 SA 动态演化的一个子过程. 在 SA 动态演化研究中, 为了宏观地把握演化的动态性, 假定被研究系统的 SA 静态结构是确定的, 且构成了稳定的网状通道, 在系统运行时, 信息经过这个网状通道传播, 即信息的流动范围随着运行状态的变化而扩张或收缩, 这种扩张或收缩反映了系统运行在 SA 特定条件域 U_i 中时 SA 的形态, 这种特定的形态及其变化的波及效应可用 $\boldsymbol{X}$ 原子过滤及其序列来描述.

5　SA 动态与静态演化模型之间的关系

由于本文涉及的内容比较丰富, 既包括 SA 演化动态演化模型及其波及效应分析, 又包括 SA 动态演化模型及其波及效应分析. 因此, 为了在整体上更为清晰地呈现本文的思想, 在此专门对演化模型的总体关系给予描述.

首先, 本文基于 SA 的构件-连接件动态语义简化模型, 建立了 SA 邻接矩阵和可达矩阵, 凭借矩阵变换和运算直接对 SA 演化特别是 SA 的静态演化中的波及效应进行了刻画分析和量化界定, 并对静态演化中的构件的删除、增加和修改、以及构件的合并和分解等变化活动(演化的表现形态)所引起的各种波及效应进行了区分和阐述; 提出了构件贡献的概念, 并给出了其大小相对量的计算方法.

其次, 基于 SA 的构件-连接件动态语义模型, 建立了 SA 动态语义网 G_{SA}, 凭借动态语义网中的浸润理论和不动点特征, 分析了 SA 动态语义网的浸润过程及其收敛性, 并给出了 SA 不动点转移的概念和 $\boldsymbol{X}$ 过滤以及 X 原子过滤的概念. 最后指出, SA 动态演化过程可用一个不动点转移序列对应的一

个矩阵(X 原子过滤) 序列来刻画和描述.

总体上来看,该文的思路建立在两个层面之上,第一个是将SA 作为一个整体,来观察它运行时的整体变化,这种变化其实就是不同时刻的SA 稳态之间的迁移和波动;另外一个层面是针对每一个稳态,再来观察 SA 在这个稳态下的静态演化行为和波及效应.也就是说,对 SA 演化的波及效应分析,首先是通过不动点转移所构成的不动点转移链上的波动分析;其次是在特定的不动点支撑下的 SA_i 静态演化过程的波及效应分析.不动点转移链上的波及过程实质上可由一系列 X 原子过滤来量化描述,而特定不动点支撑下的 SA 静态演化可用邻接矩阵和可达矩阵来量化描述.这样,将 SA 动态演化模型和 SA 静态演化模型在不同层面上得到了统一.

6 结束语

软件演化技术是软件需求和维护中的关键技术[15~17],而SA 作为软件的支撑骨架,为软件开发提供了统一的规约环境.所以,对软件的维护和变更管理首先表现在对 SA 的维护和变更管理之上.近几年来,对软件变化的研究呈现不断的上升趋势,但专门针对 SA 演化的研究,特别是模型支撑下的定量研究相对比较贫乏.本文的研究为 SA 演化的管理、控制、利用和评价等提供了可靠的依据,并为基于矩阵变换的 SA 演化量化测量奠定了基础,特别是在定量模型方面的研究具有一定的创新性.

进一步的研究工作包括:SA 演化中的波及效应能量衰减计算和界定,基于 X 过滤和 X 原子过滤序列的 SA 演化过程的评估,计算机辅助 SA 演化工具的研制和应用等.

参考文献:

[1] Liu Y, Zhang SK, Wang LF, Yang FQ. Component-Based software frameworks and role extension form[J]. Journal of Software, 2003, 14(8): 1364- 1370(in Chinese).

[2] Bass L, Clements PC, Kazman R. Software Architecture in Practice [M]. Aonton: Addisorr Wesley, 1998.

[3] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description languages[J]. IEEE Trans on Software Engineering, 2000, 26(1): 70- 93.

[4] Luo HJ, Tang ZS, Zheng JD. Visual architecture description language XYZ/ADL[J]. Journal of Software, 2000, 11(8): 1024- 1029(in Chinese).

[5] Rational Rose Corporation. UML notation guide. 2003[R]. http://www.rational.com/uml

[6] Sun CA, Jin MZ, Liu C. Overviews on software architecture research [J]. Journal of Software, 2002, 13(7): 1228- 1237(in Chinese).

[7] Bohner SA. Impact analysis in the software change process: A year 2000 perspective[A]. Proc. of the Int'l Conf. on Software Maintenance (ICSM' 96)[C]. Washington: IEEE, 1996. 42- 51.

[8] Ryder BG, Tip F. Change impact analysis for object-oriented programs [A]. Proc. of 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering[C]. New York: ACM Press, 2001. 46- 53.

[9] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development[J]. Journal of Software, 2003: 14(4): 721- 732 (in Chinese).

[10] Garlan D, Shaw M. An introduction to software architecture. In: Ambriola V, Tortora G, eds. Advances in Software Engineering and Knowledge Engineering, Vol II[M]. Hackensack: World Scientific Publishing, Co., 1993.

[11] Gao Yuan, Wang Guoren, Zhao Huiqun. An Abstract model of software architecture[J]. Chinese Journal of Computers, 2002, 25(7): 730- 736(in Chinese).

[12] Zhang SK, Wang LF, Yang FQ. Software architecture style based tier message bus[J]. Science in China (Series E), 2002, 32(3): 393- 400 (in Chinese).

[13] Li PL, Li LS, Li Y, Wang CL. Discrete Mathematics[M]. Beijing: Higher Education Press, 2001 (in Chinese).

[14] Wei Hui, Zhu Ping, He Xinggui. The object-oriented design for dynamic semantic network and the soaking on it[J]. System Engineering and Electronic Technology. 1998, 20(3): 56- 69.

[15] Baxter ID, Pidgeon CW. Software change through design maintenance [A]. Proc. of the Int'l Conf. of Software Maintenance[C]. Washington: IEEE, 1997. 250- 259.

[16] Wang Yinghui, Zhang Shikun, Liu Yu, et al. Ripple effect analysis of software architecture evolution based reachability matrix[J]. Journal of Software, 2004, 15(8): 1107- 1115(in Chinese).

[17] Kung D, Gao J, Hsia P, et al. Change impact identification in object oriented software maintenance[A]. Proc. of the Conference on software maintenance[C], Washington: IEEE, 1994. 202- 211.

作者简介:

王映辉 男,1967 年出生,博士后,教授,分别于 1989、1999 和 2002 获得学士、硕士和博士学位,目前研究方向为软件体系结构和软件演化技术. E-mail: wyh- 925@ 163. com.

王立福 男,1945 年出生,博士,教授,博士生导师,研究方向为软件工程、软件测试和软件体系结构.