面向对象系统自适应多级嵌套抗衰技术及模型

王　湛，赵颜利，刘凤玉，张　宏
（南京理工大学计算机系，南京 210094）

[摘要] 对面向对象计算系统执行自适应的细粒度软件抗衰策略，可以进一步增强软件抗衰技术的适用性及灵活性，同时能更大程度地降低抗衰成本，提高软件的可靠性。针对面向对象软件系统中各级组件的性质和特点，制定了将抗衰粒度细化到活动级的重启策略；同时根据各级组件间控制、调用及数据访问的关系，分析了组件间的耦合度，给出了计算组件重启相关性和相关度的方法，判定了组件可达集，确定了各级组件重启群，最终制定出自适应的多级嵌套的软件抗衰策略，并在此基础上构建了策略实施模型，为实现智能化面向对象软件系统的细粒度软件抗衰提供了支持。

[关键词] 软件抗衰；重启相关性；可达集；抗衰粒度，随机高级 Petri 网
[中图分类号] TP302.7 [文献标识码] A [文章编号] 1009－1742(2008)07－0062－07

1 前言

软件老化是指在软件的长期不间断运行过程中，由于系统内存占用和泄漏、未释放的文件锁、数据更新不及时、存储空间碎片以及舍入误差的累积等原因，导致软件性能衰退的过程[1~3]，它最终将导致软件的失效。而软件抗衰（software rejuvenation）是软件性能衰退到一定程度时，终止程序的继续运行，重启系统清理其内部状态（如进行垃圾收集、刷新操作系统内核表、重新初始化内部、数据结构等），从而释放操作系统资源，使软件性能得到恢复[2,3]。为了将软件抗衰技术智能化地应用于目前广泛使用的面向对象软件系统，将抗衰执行到更细粒度，可从系统的底层组件入手，分析组件间的重启相关性，得出组件可达集，并在此基础上得到各级组件重启群，从而实现面向对象软件系统智能化的细粒度软件抗衰。

George Candea 提出的微重启（microreboot）[4,5]是针对传统的结构化软件系统，对模块逐级重启执行软件抗衰，对现在广泛应用的面向对象系统适性较差，且其抗衰粒度相对较粗。要执行智能化面向对象软件系统的细粒度的软件抗衰策略，可以借鉴文献[4,5]的思想，并针对面向对象技术的特点，从系统的底层组件开始，依次确定其可达集，进而得到各级组件重启群，并在此基础上建立易于理解和分析的系统抗衰重启模型，从而为实现面向对象软件系统智能化细粒度软件抗衰提供强力支持。

2 面向对象软件系统

面向对象（object oriented，OO）技术充分体现了分解、抽象、模块化、信息隐蔽等思想，可以有效地提高软件生产率、缩短软件开发时间、提高软件质量，是控制软件复杂性的有效途径[6]。与传统的结构化软件开发方法相比，面向对象软件开发方法具有更大的优势和更为广泛的应用。

面向对象软件软件系统中组件包括用例、类、对象、活动、接口等。同时系统还提供了用例图、顺序图、类图、活动图等不同类型的图来描述各级组件的交互关系和工作流程。系统中，活动（activity）表示的是某流程中任务的执行，它可以表示某算法过程中语句的执行[6]。活动是实现系统功能的最基本的结构组件和行为组件，是实现系统功能的最基本元素。由此将抗衰粒度细化到活动级，是实现细粒度的软件抗衰策略的首要前提。

[收稿日期] 2006－09－11；修回日期 2007－01－25
[基金项目] 国家自然科学基金资助项目(60273035)；国防科工委基础应用资助项目(K1704060511)
[作者简介] 王　湛(1982－)，女，江苏连云港市人，南京理工大学博士研究生，主要研究方向为软件性能保持

3 组件重启相关性与重启相关度

一个组件重启时，可能导致其他组件的故障或错误，称为组件间的重启相关性[7]，表示这种相关性程度的数值称为组件间的重启相关度。组件间重启相关性取决于它们的耦合程度。一个组件重启时，与其耦合程度越低的组件出错的可能性也越低。

3.1 组件耦合性

在面向对象软件系统中，尽管每个结构组件是一个独立的实体，但其间经常发生交互作用，这种组件间连接的紧密程度称为耦合性[7]。组件间的连接越紧密，联系越多，耦合程度越高。根据组件间交换数据的情况，将面向对象软件系统中组件间的耦合性分为4类：非直接耦合、数据耦合、控制耦合、公共耦合。

如果组件间的联系只是通过上层组件的控制和调用来实现，称为非直接耦合。若上层组件调用下层组件完成指定功能，且调用关系是通过参数传递来实现，称为控制耦合。如果组件间分别通过一个公共环境进行数据交换，称为公共耦合。这个公共环境可以是全局变量，全局数据结构，通信缓冲区和数据(库)文件等。如果组件间通过接口的参数表交换信息数据，称为数据耦合。上述四种耦合的耦合度依次增加，其中公共耦合的适性最差。

3.2 组件重启相关性的判定

当系统需要重启一组件时，有些组件需要同时重启，则将该组件及其所有重启组件视为一个重启群[7]。在对特定组件执行重启操作前，必须先判定其与其他组件间的重启相关性和重启相关度，否则会出现数据丢失，数据不一致，甚至软件失效等运行错误。

对应组件间的几种耦合性，将重启相关性分为3类：相互独立，功能相关，状态相关。

定义1 若组件 A 调用组件 B，而组件 B 未调用组件 A，则 A 与 B 功能相关，记为 $A\rightarrow B$。

性质1 功能相关具有传递性，若 $A\rightarrow B$，$B\rightarrow C$，则 $A\rightarrow C$。

定理1 若组件 A 与 B 功能相关，则 A 重启，B 同时重启，但 B 重启时，A 不一定重启。

假定 $A\rightarrow B$，则当组件 A 重启时，B 必须同时重启，否则被调用的组件 B 不能与调用组件 A 的状态保持一致；而当组件 B 重启时，A 则不一定重启，因为 B 并不向其上层组件 A 传递控制和调用参数，不会影响其状态。

定义2 若组件 A 与 B 有数据或状态共享，则称 A 与 B 状态相关，记 $A\wedge B$。

性质2 状态相关具有交换性和传递性。若 $A\wedge B$，则 $B\wedge A$；若 $A\wedge B$，$B\wedge C$，则 $A\wedge C$。

定理2 若组件 A 与 B 状态相关，则其中一个组件重启，另一组件同时重启，即 A 与 B 总是同时重启。

假定 $A\wedge B$，则组件 A，B 总是同时重启，因为 $A(B)$ 间接使用了 $B(A)$ 的数据，如果不同时重启，会导致数据不一致或数据冲突。

定义3 若组件 A 与 B 既非功能相关也非状态相关，则 A 与 B 相互独立，记为 $A\varnothing B$。

性质3 相互独立具有交换性。若 $A\varnothing B$，则 $B\varnothing A$。

在数据耦合和非直接耦合的情况下，相应的组件相互独立。相对独立不存在传递性。

3.3 组件重启相关度的计算

当两组件非直接相连时，判定它们的重启相关性需要考虑其经过的所有组件间的重启相关性。

定义4 一个组件 A 重启与另一组件 B 重启的相关程度称为重启相关度 $D[A\cdot B]$，取值为{0，1，−1，2，3}，即 $D[A\varnothing B]=0$，$D[A\rightarrow B]=1$，$D[A\wedge B]=2$。另外定义 $D[A\cdot A]$ 为组件 A 自身重启相关度，且 $D[A\cdot A]=3$。若 $A\rightarrow B$，则记 B 与 A 的重启相关度为 $D[B\cdot A]=-1$。

根据重启相关性可以计算重启相关度，同样，由重启相关度可以确定重启相关性。即 $D[A\cdot B]=0$，1，2⇔(A 相对 B 独立)，(A 与 B 功能相关)，(A 与 B 状态相关)。

约定1 当2个组件之间存在多种重启相关性时，取重启相关度最高的相关性。当2个组件之间存在多种耦合时，取最高的耦合。耦合程度越高，组件间连接关系越紧密，重启相关度也越高。这样可以保证重启的完善性和有效性。

定义5 若组件 A 与组件 B 功能相关，认为从 A 可达 B，但从 B 不可达 A；若 A 与 B 状态相关，认为 A 与 B 相互可达；若 A 与 B 相互独立，认为 A 与 B 相互不可达。反之亦然。

定义6 若组件 A 与组件 K 非直接连接，但 A 经若干组件可达 K，则从 A 到 K 间有一条可达路径 $R[A\sim K]$，经历的组件用“−”连接。

约定2 若从组件 A 到组件 K 存在可达路径 R

$[A\sim K]=A—K_1—K_2—\cdots—K_n—K$，则 A 与 K 在此路径上的重启相关度为 $D[A\cdot K]=\min\{D[A\cdot K_n],\ D[K_n\cdot K]\}$。

推论 1　若组件 A 与组件 K 间存在可达路径 $R[A\sim K]$，但不存在可达路径 $R[K\sim A]$，则 A 与 K 功能相关。

推论 2　若组件 A 与组件 K 间既不存在可达路径 $R[A\sim K]$，也不存在可达路径 $R[K\sim A]$，则 A 与 K 相互独立。

推论 3　若组件 A 与 K 间既存在可达路径 $R[A\sim K]$，又存在可达路径 $R[K\sim A]$，则 A 与 K 状态相关。

对推论 1 的证明用反证法。若组件 A 与组件 K 非功能相关，则可能是其他两种情况即相互独立、状态相关。若 A 与 K 相互独立，按定义 5 A 与 K 相互不可达，即不存在可达路径 $R[A\sim K]$，与假设矛盾。若 A 与 K 状态相关，由定义 5 组件 A 与 K 相互可达，再由定义 6 两组件间必存在一条可达路径 $R[K\sim A]$，与假设矛盾。因此 A 与 K 必为功能相关。用类似方法可证推论 2、推论 3。

按以下步骤确定组件间的重启相关性。

Step 1　确认初始组件和终结组件。因为可达路径是单向的。

Step 2　查找从初始组件到终结组件的所有可达路径。

Step 3　按每一条可达路径求初始组件和终结组件间的重启相关度。

Step 4　确定初始组件和终结组件的最终重启相关度及重启相关性。

4　判定组件可达集及组件重启群

定义 **7**　若从组件 A 到组件 K 存在可达路径 $R[A\sim K]$，则 K 为 A 的可达组件。A 的所有可达组件构成 A 的可达组件集，简称可达集。用 $S[A]$ 表示。

定理 3　组件 A 与其任意可达组件 K 的重启相关度 $D[A\cdot K]>0$。

若 K 为 A 的可达组件，则存在可达路径 $R[A\sim K]$，由推论 1、推论 3 得 A 与 K 间至少为功能相关，据定义 4，$D[A\cdot K]>0$。

为了执行细粒度的软件抗衰，可从系统的活动图入手。首先将活动重启群、对象重启群、类重启群、用例重启群置空，当监测到系统的某一活动出现故障需执行重启策略时，执行以下 4 步操作。

Step 1　确定问题活动可达集，得到活动重启群，初步判定类重启群、对象重启群、用例重启群。

分析系统活动图执行以下具体操作。

1)确定该活动的耦合活动及其重启相关性，得出该问题活动的可达集，定为活动重启群，为执行活动级重启提供支持。

2)查找活动重启群中所含活动的所属泳道的实现类，将其放入类重启群中。

3)查找活动图中的对象流，查出活动重启群所含活动的所在流程的关联对象，将其放入对象重启群中。

4)查找活动重启群中活动所属用例，放入用例重启群中。

Step 2　根据 Step 1 提供的对象重启群和系统交互图，确定重启群所含对象的可达集，放入对象重启群，由此得到最终的对象重启群。进而查出重启群中所有对象的所属类，将其放入类重启群中，得到新的类重启群。并根据交互图确定问题用例，放入用例重启群。

分析交互图执行以下具体操作。

1)确定对象重启群所有对象的耦合对象及其重启相关性，确定其可达集，放入对象重启群，得到最终对象重启群，为执行对象级重启提供支持；同时查出对象重启群中对象的所属类，放入类重启群。

通常对象间重启相关性由其相互间的消息类型决定，通常系统有调用消息、异步消息、返回消息、阻止消息和超时消息 5 种消息类型。假设 A，B 两个对象，且它们之间通过消息实现交互作用，若 a. A，B 间的消息类型属调用消息，则 A，B 属于内容耦合，$D[A\wedge B]=2$。b. A，B 间的消息类型属异步消息，且 A 为发送者，则 A，B 属于控制耦合，且 $D[A\rightarrow B]=1$。c. A，B 间的消息类型属返回消息，则 A，B 间属于控制耦合，且 $D[A\rightarrow B]=1$。d. A，B 间的消息类型属阻止消息或超时消息，则 A，B 间属于数据耦合，$D[A\rightarrow B]=0$。

确定消息类型后，读此消息。若此消息是线程间同步消息，则查找出必须在此消息发送前发送的消息组，并查找出消息的发送和接收对象。此时它们和问题对象间属控制耦合。

2)确定该交互图所属用例，放入用例重启群。

Step 3　确定类重启群中所有类的可达集，放入类重启群，得到最终类重启群，为执行类级重启提供

支持。同时查出类重启群中所有类的所属用例，放入用例重启群。

分析类图执行以下具体操作。

1)查找问题类的关联类。假设 A，B，C 三个类，A，B 间存在关联关系，而 C 为描述此关系的关联类，则 C 与 A，C 与 B 之间属控制耦合，且 $D[C\rightarrow A]=1$，$D[C\rightarrow B]=1$；而 A，B 属数据耦合。

2) 查找问题类的聚集类和组合类。假设 A，B 两个类，若 A，B 之间属聚集关系，且 A 为整体，根据聚集的定义则 A，B 且 $D[A\rightarrow B]=-1$；若 A，B 之间属组合关系，根据组合的定义则 A，B 属数据耦合。

3) 查找问题类的泛化类。根据泛化的定义，具有泛化关系的类之间属数据耦合。

4) 查找问题类的依赖类。假设 A，B 两个类，若 A，B 之间属依赖关系，且 A 为被依赖类，则 A，B 间属控制耦合，且 $D[A\rightarrow B]=1$ 。

5) 确认此问题类是否为控制类。此时若该问题类属于控制类，则确定该控制类所控制的用例，得出问题用例。

Step 4　根据 Step 3 提供的用例重启群和系统的用例图，确定用例重启群中所有用例的可达集，放入用例重启群中，得到最终的问题重启群，为执行用例级重启提供支持。

通常系统用例间包含泛化、包含和扩展 3 种关系。根据定义可知，具有以上 3 种关系的用例间属控制耦合。

现举例说明。图 1 是银行 ATM 系统用例图；图 2 是现金提取用例对象顺序图，图 2 中对象 a：银行读卡操作，对象 b：客户回执操作，对象 c：现金发送操作，对象 d：现金存放操作，对象 e：凭单打印操作，对象 f：执卡交易操作，对象 g：现金业务处理操作，对象 h：银行系统界面操作，参与者 A：银行客户，参与者 B：银行系统；图 3 是 ATM 取款用例活动图。利用所提出的方法可计算得到其对应的各级组件重启相关度，分别如表 1、表 2、表 3 所

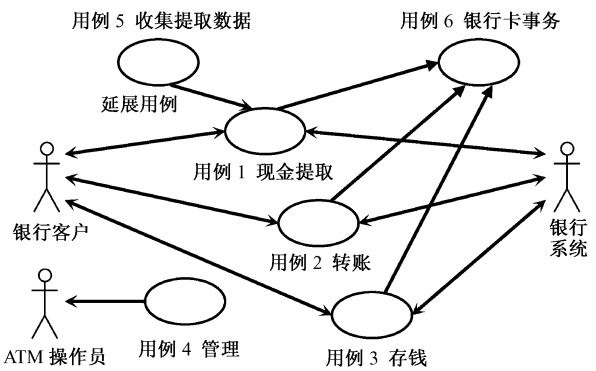

图 1　银行 ATM 系统用例图

Fig.1　The instance figure of the system in the bank

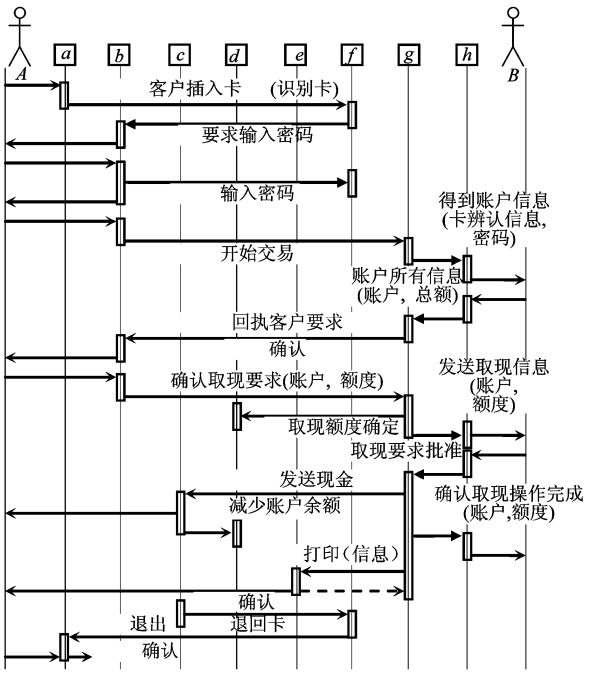

图 2　现金提取用例对象顺序图

Fig.2　The sequence figure of the objects in cash-withdrawing instance

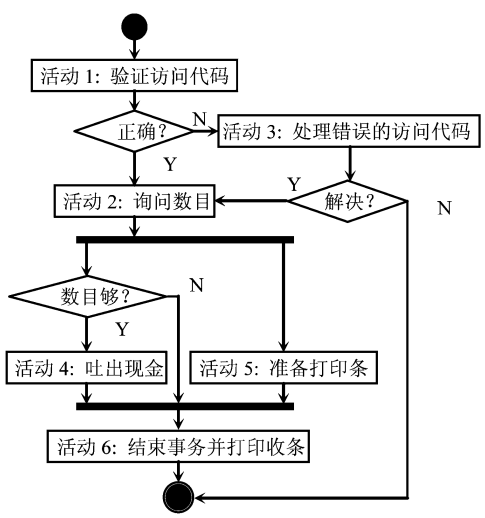

图 3　ATM 取款用例的活动图

Fig.3　The activity figure of the cash-withdrawing instance in ATM system

示。当诊断出活动 3 为问题活动时，则由表 3 知，$S[3]=\{2,3,4,5,6\}$；由于活动 3 属于对象 f，由表 1 知 $S[f]=\{b,c,d,e,f,g,h\}$；而对象 f 属于类 f，而类 f 包含在用例 1、用例 2、用例 3 中，由表 2 知，$S[1]=\{1,5\}$，$S[2]=\{2\}$，$S[3]=\{3\}$。由此活动重启群为{2，3，4，5，6}，对象重启群为{b，c，d，e，f，g，h}；用例重启群为{1，2，3，4，5}。

表 1　图 1 中用例间的重启相关度

Table 1　The restart dependence degree between the instances in Fig.1

重启相关度 初始用例 \ 终结用例	1	2	3	4	5	6
1	3	0	0	0	1	−1
2	0	3	0	0	0	−1
3	0	0	3	0	0	−1
4	0	0	0	3	0	0
5	−1	0	0	0	3	−1
6	1	1	1	0	1	3

表 2　图 2 中对象间的重启相关度

Table 2　The restart dependence degree between the objects of Fig.2

重启相关度 初始对象 \ 终结对象	a	b	c	d	e	f	g	h
a	3	1	1	1	1	1	1	1
b	−1	3	1	1	1	2	1	1
c	−1	−1	3	1	−1	−1	−1	−1
d	−1	−1	−1	3	−1	−1	−1	−1
e	−1	−1	1	1	3	−1	2	2
f	−1	2	1	1	1	3	1	1
g	−1	−1	1	1	2	−1	3	2
h	−1	−1	1	1	2	−1	2	3

表 3　图 3 中活动间的重启相关度

Table 3　The restart dependence degree between the activities of Fig.3

重启相关度 初始活动 \ 终结活动	1	2	3	4	5	6
1	3	1	1	1	1	1
2	−1	3	−1	1	1	1
3	−1	1	3	1	1	1
4	−1	−1	−1	3	0	0
5	−1	−1	−1	0	3	0
6	−1	−1	−1	0	0	3

当具体实施软件抗衰策略时，首先确定系统的软件性能参数，并对其进行监测，当系统出现异常时，首先通过分析各性能参数确定出问题活动；由此确定出系统各级组件重启群，同时根据各级组件的运行情况以及系统监控数据，预算出各级组件的抗衰成本，灵活确定适合系统的抗衰粒度，制定出自适应的系统多级嵌套的抗衰重启策略。由此更有效地节约成本，同时增强了抗衰策略的灵活性和实用性。

5　建立抗衰策略的实施过程模型

要实现以上的抗衰策略，必须在此基础上建立简约明确的重启抗衰策略实施模型，使其易于理解和分析，适用性和可读性都较以往的抗衰模型有所提高。分析 SPN（随机高级 Petri 网）以及 DFA（有限自动机）两种形式化描述方法的特点，综合两种方法优点制定了以下构建理论：用 SPN 描述每次抗衰的具体实施过程，用 DFA 控制各次抗衰的返回位置，明确描个抗衰子过程之间的关联与转移；这样既避免了模型状态空间爆炸问题，又避免了采用 DFA 模型不能描述策略的实施细节的缺点，使重启模型简单而明确，容易理解和分析。

由此构建出系统多级嵌套的抗衰重启策略的实施过程模型如图 4 所示。

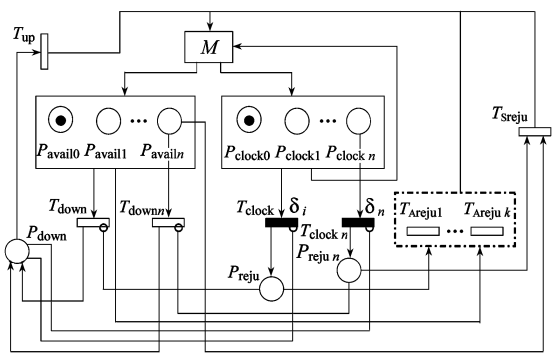

图 4　系统多级嵌套的抗衰重启策略的实施过程模型

Fig.4　Automaton model of nested rejuvenation policy of system

图 4 中，T 表示系统的变迁，P 表示系统当前位置，将表示活动级、对象级、类级及用例级重启的变迁为一组，如图 4 中点划线框所示，框内变迁的总数等于组件重启的级数之和。自动机 M 控制不只是每次重启的初始位置，还要控制该次重启应该执行哪一个变迁，它由自动机的状态转移规则确定，因为

在一个确定的状态下只能执行一种组件级重启。

用 M 的有限状态集 Q 的状态 $q_{i,j}$ 控制图 4 中位置组{ $P_{\text{avail}0}$, $P_{\text{avail}1}$, …, $P_{\text{avail}n}$ }和{ $P_{\text{clock}0}$, $P_{\text{clock}1}$, …, $P_{\text{clock}n}$ }中的标记，下标 $n=\sum_{k=0}^{m}N[k]+m+1$，m 为组件级重启次数。定义状态值为

$$
\begin{aligned}
&q_{00} = 1000\cdots000,\\
&\qquad \sum_{k=0}^{m}N[k]+m+1\\
&q_{01} = 0100\cdots000,\\
&\vdots\\
&q_{i,j} = \underbrace{00\cdots0}_{\sum_{k=0}^{i-1}N[k]+i+j}\ 10\cdots00,\\
&\qquad 0<i\leqslant m,\ 0\leqslant j<N[i],\\
&\vdots\\
&q_{Mn[m]} = 0000\cdots001
\end{aligned}
\tag{1}
$$

对应初始状态 q_{00}，标记位于 $P_{\text{avail}0}$ 和 $P_{\text{clock}0}$；进入状态 $q_{i,j}=00\cdots010\cdots00$（第 $\sum_{k=0}^{i-1}N[k]+i+j+1$ 位为 1，$i>0$），标记位于 $P_{\text{avail}l}$ 和 $P_{\text{clock}l}$，其中 $l=\sum_{k=0}^{i-1}N[k]+i+j+1\ (i>0)$，系统执行了 l 次组件级重启；当系统到达状态 $q_{mN[m]}$ 时，标记位于 $P_{\text{avail}n}$ 和 $P_{\text{clock}n}$，准备执行系统级重启。

由此构建多级嵌套的抗衰重启策略的系统 DFA 模型如图 5 所示。

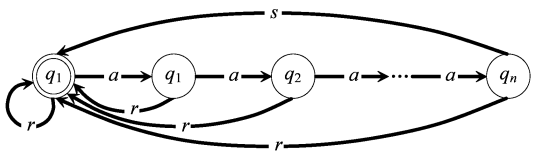

图 5　多级嵌套的抗衰重启策略的系统 DFA 模型

Fig.5　The DFA model of nested rejuvenation policy

图 5 中描述的状态转移规则可以简化为

$$
\begin{aligned}
&\delta(q_i,\ a) = q_{i+1} \quad 0\leqslant i<n,\\
&\delta(q_n,\ s) = q_0,\\
&\delta(q_i,\ r) = q_0 \quad 0\leqslant i\leqslant n
\end{aligned}
\tag{2}
$$

从 P_{clock} 位置组到自动机 M 的连接线，标示 M 的一个输入，由此获知系统的当前状态 q_{ij}；从变迁 T_{Areju} 组、T_{Sreju} 组、T_{up} 组到自动机 M，三线合一作为 M 的另一个输入，由此获知系统执行的重启类型（a，s 或 r）；M 的转移函数 θ 读取这两个输入，按式(1)所示规则，得到系统的下一状态，同时也得到系统在该状态时应执行的操作，将其作为 M 的输出，并按式(2)控制恢复子过程中标记的初始位置，以及系统从该位置出发所应执行的重启组件。如图 5 所示，其中 q_i 为系统状态，r 为系统意外失效后重启引导系统操作，s 为整个系统重启操作，a 为组件级重启。假定对于图 1、图 2、图 3 中，根据各级组件的运行情况以及系统监控数据，预算各级组件的抗衰成本后，确定重启的各级组件分别为：活动 3，对象 f，用例 1，且需执行 m 次组件级重启；根据图 5 系统首先确定 $q_{00}=10000\cdots000$，执行组件应用级重启 a，重启操作见图 4；然后由图 5 确定出系统返回位置状态 $q_{01}=01000\cdots000$，由此继续执行组件应用级重启 a，当执行了 $l(l=\sum_{k=0}^{i-1}N[k]+i+j+1,\ i>0)$ 次组件级重启操作 a 后，由图 5 确定出系统的返回位置为

$$
q_{i,j} = \underbrace{00\cdots0}_{\sum_{k=0}^{i-1}N[k]+i+j}\ 10\cdots00,
$$

按式(1)依次类推系统执行同样的重启操作，最终当由图 5 得到系统的当前位置在 $q_{mN[m]}$ 后，则按图 4 执行系统级重启操作 s。由此完成了整个系统的重启抗衰策略。

6　结语

执行智能化的面向对象软件系统的细粒度软件抗衰策略，进一步扩大软件抗衰的应用范围，提高其可用性和灵活性，更大程度地节省抗衰成本，提高软件的可靠性，必须具备 4 个前提条件：一是确定软件抗衰策略的抗衰粒度；二是软件系统各级组件的运行情况可监控；三是确定合理的组件可达集和各级组件重启群；四是构建易于理解的抗衰重启实施过程模型，为全面实行此策略提供支持。一般情况下，面向对象的软件系统都满足第二个条件。

对于第一个条件，考虑到系统中活动的独立性和基础性，最终确定将系统的抗衰粒度最细化到活动级；此时可将系统提供的活动图作为分析切入点，依次得出各级组件重启群。同时根据组件的运行情况以及系统监控数据，预算出各级组件的抗衰成本，灵活确定适合系统的抗衰粒度，由此增强策略的自适应。

第三个条件则是执行此抗衰策略的关键。根据面向对象软件系统中各级组件的控制调用及数据访问关系，分析了组件间的耦合程度，给出了判定组件重启相关性和重启相关度的方法，并在此基础上得

到了组件的可达集，最终得到各级组件重启群，从而为实现了细粒度的软件抗衰奠定基础。

要满足第四个条件必须分析各描述方法的优缺点。分析了 SPN 以及 DFA 两种形式化描述方法的特点；综合了两种方法，用 SPN 描述每次抗衰的具体实施过程，用 DFA 控制各次抗衰的返回位置，明确描述个抗衰子过程之间的关联与转移。这样既避免了模型状态空间爆炸问题，又避免了采用 DFA 模型不能描述策略的实施细节的缺点。由此，由此得出了简约明确的系统多级嵌套的重启抗衰策略实施模型，并以其易于理解和分析的优点为全面执行有效的面向对象软件系统的软件抗衰策略奠定了坚实的基础，由此进一步增强了抗衰技术的智能化，更大程度地节约了抗衰成本，提高了软件可靠性。

参考文献

[1] Huang Y, Kintala C, Kolettis N. Software rejuvenation: analysis, module and applications [J]. Proc of FTCS－25 [C]. Pasadena, CA, 1995

[2] Garg S, Moorsel A V, Vaidyanathan K. A methodology for detection and estimation of software aging [A]. Proceedings of the 9th International Symposium on Software Reliability Engineering [C]. Paderborn, Germany, 1998

[3] Castelli V, Harper R E, Heidelberger P. Proactive management of software aging [J]. IBM JRD, 2001, 45(2): 311－332

[4] Candea G, Kawamoto S, Fujiki Y. Microreboot-a technique for cheap recovery [A]. 6th Symposium on Operating Systems Design and Implementation [C]. San Francisco, CA, 2004

[5] Candea G, Cutler J, Fox A. Improving availability with recursive microreboots: a soft-state system case study [J]. Performance Evaluation Journal, 2004, 56(1—3): 213－248

[6] 王少峰. 面向对象技术教程[M]. 北京：清华大学出版社，2004

[7] 王　湛，游　静，赵颜利，等. 基于访问关系的进程重启相关性判定[J]. 计算机科学，2006，33(9)：274－277

Research and modeling analysis on adaptive nested fine-grained software rejuvenation of object-oriented software system

Wang Zhan, Zhao Yanli, Liu Fengyu, Zhang Hong

(*Department of Computer, Nanjing University of Science and Technology, Nanjing 210094, China*)

[**Abstract**] In order to enlarge the field of application, reduce the cost of software rejuvenation and improve software availability and reliability, the adaptive software rejuvenation with fine rejuvenation granularity of object-oriented software system would be put forward. Based on the characteristics of the component and coupling relation between the components of the system, this paper determines the finest rejuvenation granularity, analyzes the degree of the coupling relation, sets down a method to calculate the restart dependence and its degree of the components, so that determines the restart reachable set of each component and gets the restart gather of the components at every granularity, and finally sets down and models the adaptive nested software rejuvenation policy. So executing intelligent software rejuvenation at fine rejuvenation granularity in the object-oriented software system can be supported.

[**Key words**] software rejuvenation; restart dependence; restart reachable set; rejuvenation granularity; SHLPN