一种基于面向侧面技术的并发式软件建模方法*)

苏 旸 康 力 胡圣明 陈 平
(西安电子科技大学软件工程研究所 西安 710071)

摘 要 基于面向侧面(Aspect-Oriented)技术及统一建模语言状态图提出了并发式软件系统开发过程中横切特性的建模方法。本方法将并发软件系统的业务逻辑和横切行为分别封装到复合状态的不同正交区域中,并通过事件广播机制反映二者的交互关系。同时,以模块化的状态迁移系统(Modular Transition System)作为基本计算模型,对该建模方法进行形式化描述,给出了模型元素及建模过程的精确语义。实例研究表明,该方法在并发软件设计阶段实现了横切关注点的分离策略,并使得系统模型具有松耦合、适应性和可跟踪性的优点。

关键词 面向侧面,状态图,并发系统,横切特性,模块化迁移系统

Modeling Concurrent Software System Based on Aspect-oriented Techniques

SU Yang KANG Li HU Sheng-Ming CHEN Ping
(Software Engineering Institute, Xidian University, Xi'an 710071)

Abstract Based on aspect-oriented techniques and statechart diagrams of unified modeling language (UML), an approach on modeling crosscutting concerns in concurrent software system is presented. Modeled in UML statechart diagrams, the primary system functions and traversal features are enveloped in orthogonal regions of a composite state respectively. The mutual relations between orthogonal regions are represented by the orders of broadcasted events. With modular transition system, a basic computation model, formalism description of the aspect-oriented statechart model is discussed. Also, the precise semantics of model elements and modeling procedures are given. Illustrated by case study examples, the strategy of separation of crosscutting concerns is implemented in the design phrase of concurrent software system with this method. The advantages of this approach are loose couple, adaptability and traceability.

Keywords Aspect-oriented, Statechart diagrams, Concurrent software system, Modular transition system

在并发式软件系统中,主要业务功能通过多个进/线程的交替执行来实现,而并发特征(如多任务调度和同步)又需要由一组影响多个进/线程逻辑的横切行为来保障,这导致了系统的主要业务功能和横切特性在代码中的缠结,降低了系统的可维护性、可扩展性及易进化性。面向侧面(Aspect-Oriented)技术是近年来新出现的软件设计范型,它基于系统横切关注点分离的思想,并有助于提高系统在设计和实现层面上的重用水平[1,2]。由于 AOSD (Aspect-Oriented Software Development)研究尚在初始阶段[3,4],因此针对并发式软件系统的具体特征,缺乏相应的设计阶段建模方法及其形式化描述。

统一建模语言(UML)是软件系统需求分析和设计建模的一种成熟标准,许多 AOSD 研究者通过扩展 UML 来实现面向侧面的建模技术。本文采用标准的 UML 状态图对并发式软件系统进行面向侧面建模,并以模块化的迁移系统(Modular Transition System)[5]作为计算模型,对该方法进行形式化描述,给出了建模元素及建模过程的精确语义,为并发式软件系统的面向侧面模型验证研究提供了一个基本的框架。实例研究表明,该建模方法在并发式软件的设计阶段实现了横切关注点的分离,并具有松耦合、适应性和可跟踪性的优点。

1 并发系统的面向侧面状态图建模

并发系统的业务逻辑描述和时间有密切的关系,因此对它的建模方法应反映系统的动态行为,而 UML 状态图是一种通过状态、事件和动作对系统动态行为进行建模的标准方法,它适合表现并发系统的执行逻辑[6]。

"与状态"[7](AND-State)是 UML 状态图中的一种复合状态,它包含两个以上的正交区域(Orthogonal Regions),每个正交区域是一个相对独立的状态图。由于正交区域在逻辑上是并发的,它们表示了同一对象在同一时刻的不同描述,因此对象必定同时处在所有正交区域的相应状态上。正交区域之间的交互通过事件广播来完成,即当对象接收到一个事件时,这个事件被发送到该对象所有的正交区域中,受这个事件影响的区域中的状态将做出相应的状态迁移。用 UML 状态图和面向侧面技术建模并发系统的核心思想就是在一个"与状态"的不同正交区域中分别表示主要业务逻辑和其相应的并发约束操作,正交区域间的事件广播则反映了系统的主要业务逻辑在多个位置上被并发约束操作所影响,这种影响就是并发系统中主要的横切特性。建模的主要步骤包括(实例见论文第三部分):

· 识别并发系统的主要业务功能,将其封装到不同的"业务类"中,系统中多个并发的进/线程通过向相应"业务类"的实例发送消息来完成各自的功能。

· 根据进/线程间的并发约束条件,将影响"业务类"的横切行为分别封装到不同的 Aspects 组件中,Aspects 可以被看

*)项目基金:国家自然科学基金(SN:60473063),教育部博士学科科研基金(SN:40103230087)。苏 旸 博士研究生,研究方向:软件建模及面向侧面的软件工程。

作和“业务类”在同一层面上的一种特殊的类。

·根据对系统动态行为的分析，分别生成“业务类”及其相应的 Aspects 的状态图。

·将“业务类”和其相应的 Aspects 状态图作为正交区域，共同组合成一个“与状态”。整合过程就是定义出有时序关系的广播事件集合，这些事件在有正交关系的状态图之间传播，事件之间的时序关系反映了系统的主要业务功能中横切行为的位置(图 1)。

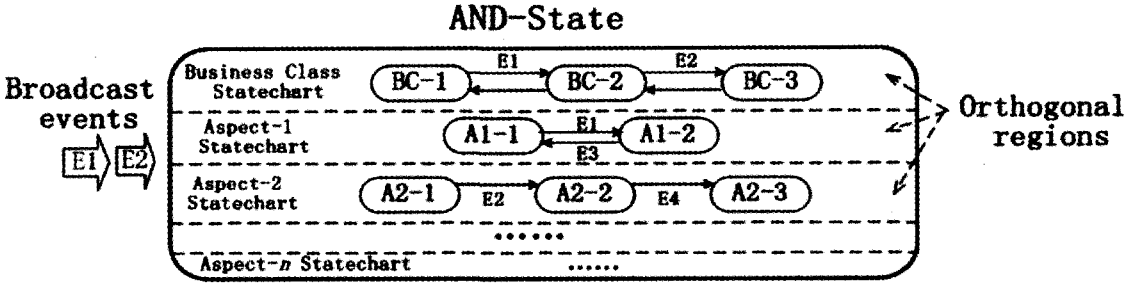

图 1　用“与状态”建模并发系统的横切特性

该建模方法具有以下优点：

·松耦合。并发系统在设计阶段能够被分成多个松耦合的自治行为模块(“业务类”及其相关的 Aspects 的状态图)，并可以通过设计一组广播事件在“业务类”的执行逻辑中加入相应的横切逻辑，这种关注点分离的策略减少了并发系统设计的复杂性。

·适应性。对于已经设计完成的状态图模型，如果需要增加新的系统横切属性，则能够在基本不影响现有正交区域的情况下，生成新的 Aspects 状态图并更改相应正交区域中的广播事件的类型和它们之间的序列关系。

·可跟踪性。由于系统的主要业务功能和横切行为被封装为类结构及相似的 Aspects，并且状态图建模产生了它们的行为规约，其相应的语义可以从设计阶段到实现阶段被一直保持下来，从而建立了设计模型和实现代码之间的相关一致性。

2　面向侧面状态图建模的形式化描述

UML 状态图是系统的一种高层设计模型，可以辅助代码实现，但形式化语义的缺乏导致了模型验证的困难。因此，本文基于一种形式化计算模型——模块化状态迁移系统对并发系统的面向侧面状态图建模方法进行形式化描述，给出建模元素和建模过程的精确语义，为面向侧面的模型验证以及自动化代码产生奠定基础。

2.1　模块化状态迁移系统

一个反应式系统(Reactive System)可以用状态迁移系统 $TS=\langle V,\Sigma,\Theta,T\rangle$ 描述：

·$V=\{\pi,y_0,y_1,\cdots,y_m\}$：系统的状态变量有穷集合，包括控制变量 π 和数据变量 $y_i(0\leqslant i\leqslant m)$，$\pi$ 指示了系统控制流程中当前的执行位置。

·$\Sigma=\{s_0,s_1,\cdots,s_t\}$：状态的非空集合，$s_i(0\leqslant i\leqslant t)$ 对应 V 的一次赋值。第一个为初始状态，即 $s_0\models\Theta_S$，并且对于每一个 $i\geqslant 0$，状态 s_{i+1} 是状态 s_i 的 $\tau_{i+1}(\tau_{i+1}\in T)$ 迁移后继。

·$\Theta=\{\pi_0=l_0)\wedge(Y_0=\{y_1^0,y_2^0,\cdots,y_m^0\})$：声明了系统初始状态的一阶公式，$l_0$ 代表了系统开始执行时的入口位置，Y_0 是系统中数据变量的初始赋值集合。

·$T=\{\tau_0,\tau_1,\cdots,\tau_u\}$：有限的状态迁移集合，$\tau_i(0\leqslant i\leqslant u)$ 是一个状态迁移函数：$\Sigma\mapsto 2^{\Sigma}$，用 $\tau(s)$ 来表示状态 s 经 $\tau\in T$ 迁移后的所有后继状态集合，如果 $\tau(s)\neq\Phi$ 则称一个迁移 τ 是可行的，否则对于状态 s 迁移 τ 是不可行的。τ 能用一阶公式 $\rho_\tau:(\pi=l)\wedge(\pi'=l')\bigwedge_{t=0}^{m}(y_t=y'_t)$ 描述，ρ_τ 称为迁移关系，其中 l 和 l' 分别代表状态迁移前后系统在控制流程中的执行位置，分别表示为 pre-location(ρ_τ)和 post-location(ρ_τ)，y'_t 表示迁移后系统数据变量的值。

模块化的状态迁移系统就是用多个功能相对独立，并且在逻辑上有并发关系的子迁移系统模块构成一个完整的系统。其中每一个子迁移系统模块 $m:\langle\langle I\rangle,\langle B\rangle\rangle$ 包括模块接口 $I:\langle V_{In},V_{Out},V_{Global}\rangle$ 和模块体 $B:\langle V_M,\Theta_M,T_M\rangle$。

V_{In}：模块的输入变量集合，它可以被其它模块修改，但不能被模块本身修改。V_{Out}：模块的输出变量集合，它的值只能由模块自身的操作修改，但能被其它模块读取。V_{Global}：系统的全局变量，它的值能被系统中所有的模块修改和读取。V_M,Θ_M,T_M 分别代表了模块的局部变量集、模块的初始条件和模块中的状态迁移集合。

2.2　建模元素的形式化定义

并发系统 $P(p_1,p_2,\cdots,p_n)$ 可以描述为一个模块化迁移系统 $M:\langle m_1,m_2,\cdots,m_n\rangle$，其中的迁移系统子模块 $m_i:\langle\langle V_{i,In},V_{i,Out},V_{i,Global}\rangle,\langle V_{i,M},\Theta_{i,M},T_{i,M}\rangle\rangle(0\leqslant i\leqslant n)$ 表示 P 中的一个进/线程 $p_i(0\leqslant i\leqslant n)$。$C=\{c_0,c_1,\cdots,c_m\}$ 表示系统中“业务类”的集合，进/线程 p_i 通过向 $Z=\{z_0,z_1,\cdots,z_t\}(Z\subseteq C)$ 中“业务类”$z_j(0\leqslant j\leqslant t)$ 实例发送消息来实现其相应的功能，即 z_j 中方法的调用执行反映在迁移系统子模块 m_i 的迁移集合 $T_{i,M}$ 中，因此 z_j 状态图中的迁移集合可以用 $\Gamma_j\subset T_{i,M}$ 来表示。$A_j=\{a_{j,0},a_{j,1},\cdots,a_{j,s}\}(0\leqslant j\leqslant t)$ 表示横切“业务类”z_j 的 Aspects 集合，$a_{j,q}(0\leqslant q\leqslant s)$ 和 z_j 状态图的整合过程可以表示为子迁移系统模块的语义变更操作：$a_{j,q}$ 向子迁移系统模块 m_i 中引入全局或局部的变量，并且调整现有的迁移，增加新的迁移。对于 m_i，它的一个横切 Aspect 的可以描述为 $a:\langle V_a,V'_a,\Theta_a,T_a,\mu,\sigma\rangle$，其中，

·V_a：Aspect 全局变量的有限集，可以被系统中所有子迁移模块访问。

·V'_a：Aspect 局部变量的有限集，只能被本模块访问。

·Θ_a：$\Theta_a=(\sigma_0=h_0)\wedge(U^0=\{u_0^0,u_1^0,\cdots,u_n^0\})$ 是一阶公式，它声明了 Aspect 的初始状态，h_0 代表了 Aspect 结构中横切操作的入口位置，U^0 是 Aspect 中变量的初始赋值集合。

·T_a：Aspect 的状态迁移集合，它描述了包含 Aspect 状态图的正交区域中状态迁移集合。

·μ：迁移删除指令集合，$\mu=\{(\varphi_0,\omega_0)(\varphi_1,\omega_1),\cdots,(\varphi_v,\omega_v)\}$ 是一个二元组集合，其中的每一条指令对应 m_i 中需要被

修改的的迁移关系 ρ，φ_i 表示系统控制流程的位置，ω_i 表示系统运行到 φ_i 时的数据变量集合。根据 μ 的一系列指令，m_i 的迁移集合中的迁移公式 ρ 如果包含和 $\varphi_i \wedge \omega_i (0 \leqslant i \leqslant v)$ 相等的项，则将 ρ 中该项删除。μ 描述了面向侧面状态图整合过程中，由于一个 Aspect 正交区域的引入，“业务类”状态图中原有的若干个迁移及相应的数据变化应该被取消。

• σ：迁移增加指令集合，$\sigma = \{(\psi_0, \varepsilon_0, \chi_0), (\psi_1, \varepsilon_1, \chi_1), \cdots, (\psi_u, \varepsilon_u, \chi_u)\}$ 是一个三元组集合，其中的每一条指令对应 m_i 中需要被修改的迁移关系 p 和 Aspect 中需要被修改的迁移关系 ρ_a。ψ_i 表示 m_i 中需要添加新迁移的控制流程位置，ε_i 表示 Aspect 结构中操作流程的当前位置，χ_i 表示 Aspect 的操作执行到 ε_i 时 Aspect 的数据变量集合。根据 σ 的一系列指令，Aspect 的迁移集合中的迁移公式 ρ_a 如果包含和 $\varepsilon_i \wedge \chi_i$ $(0 \leqslant i \leqslant u)$ 相等的项，则将 ρ_a 中该项添加（析取关系）到 ψ_i 所指示的 m_i 中相应的迁移公式 ρ 中。σ 描述了面向侧面状态图整合过程中，由于一个 Aspect 正交区域的引入，Aspect 状态图中若干个迁移及相应的变量变化情况应该被“业务类”状态图中某些广播事件所触发。

2.3 面向侧面状态图整合过程的操作语义

以模块化的迁移系统为基本的形式化计算模型，针对并发系统建模的面向侧面状态图整合过程可以描述为：一个子迁移系统模块 m_i：$\langle\langle V_{i,In}, V_{i,Out}, V_{i,Global}\rangle, \langle V_{i,M}, \Theta_{i,M}, T_{i,M}\rangle\rangle$ 和一个 Aspect 模块 a：$\langle V_a, V'_a, \Theta_a, T_a, \mu, \sigma\rangle$ 组合后语义发生变更，形成新的子迁移系统模块

$m_i(a)$：$\langle\langle V^*_{i,In}, V^*_{i,Out}, V^*_{i,Global}\rangle, \langle V^*_{i,M}, \Theta^*_{i,M}, T^*_{i,M}\rangle\rangle$：

• m_i 的输入及输出变量保持不变：$V_{i,In} = V^*_{i,In}$ 并且 $V_{i,Out} = V^*_{i,Out}$。

• m_i 的全局变量集合和 Aspect 的全局变量集合合并：$V^*_{i,Global} = V_{i,Global} \cup V_a$，$V^*_{i,Global}$，可以被新的迁移系统中所有的子迁移系统模块访问。

• m_i 的局部变量集合和 Aspect 的局部变量集合合并：$V^*_{i,M} = V_{i,M} \cup V'_a$，$V^*_{i,M}$ 只能被子迁移系统模块 $m_i(a)$ 访问。

• m_i 的初始化条件和 Aspect 的初始化条件合并：$\Theta^*_{i,M} = \Theta_{i,M} \wedge \Theta_a$。

• m_i 中迁移集合的变更过程由 Aspect 模块 a 中的指令集合 μ 和 σ 所指示。

• Aspect 的一系列迁移删除指令 μ 和迁移增加指令 σ 对 m_i 迁移集合中相应的迁移关系进行重定向操作（删除原有的部分迁移关系，并在相应位置加入新的迁移关系），生成新的迁移集合 $T^*_{i,M}$。迁移变更的操作过程如下：如果 $\tau \in T_{i,M}$ 是 m_i 迁移集合中的一个迁移，用迁移关系 ρ_τ 表示；$\beta_i = (\varphi_i, \omega_i)$ 和 $\gamma_j(\psi_j, \varepsilon_j, \chi_j)$ 分别是迁移删除和增加指令 $(\varphi_i \leftrightarrow \psi_j)$，则变更后的迁移关系（表示迁移 $new\_\tau \in T^*_{i,M}$）为：$\rho_{new\_\tau} = (\rho_\tau \vee \varphi_i \vee \omega_i) \wedge \psi_j \wedge \varepsilon_j \wedge \chi_j$。

3 实例研究

在一个并发软件系统中，有多个进程对一个有界内存缓冲区进行数据读写操作，具体的功能需求包括：①缓冲区的容量有限；②进程只能通过 read() 和 write() 方法读写缓冲区数据；③缓冲区允许同一时刻存在多个读操作；④缓冲区只允许同一时刻有一个写操作；⑤当缓冲区正进行一个写操作时，系统阻塞所有其它操作请求；⑥缓冲区空时系统阻塞所有的读操作请求；⑦缓冲区满时系统阻塞所有的写操作请求。

通过对需求的分析，⑥和⑦是控制有边界缓冲区的同步需求，③至⑤是多个进程间的调度需求。“对缓冲区进行读写”是系统的主要业务功能，因此可以将它封装到“业务类” Main_Buffer 中，多个进程通过向 Main_Buffer 的实例发送消息来实现各自的功能，进程间同步和调度需求是主要业务功能的横切逻辑。使用 UML 状态图对系统进行设计建模时，“业务类”Main_Buffer、同步侧面 Synchronization_Aspect 和调度侧面 Scheduling_Aspect 三者的状态图分别作为一个“与状态”的正交区域，正交区域中迁移上的广播事件反映了 Aspects 结构引入主要业务逻辑后，对业务控制流程的重定向操作，从而实现系统的横切逻辑（图 2）。例如，当进程通过 read() 和 write() 方法读写缓冲区时，事件 read 和 write 广播发送给“与状态”的三个正交区域，由于 Synchronization_Aspect 区域中相关迁移的触发事件与之匹配，则产生相应迁移，进程同步的约束得以实现，并同时产生 READ 和 WRITE 广播事件；这进一步触发了 Scheduling_Aspect 区域中相应的迁移变化，在实现进程调度功能的同时，产生 read_event 和 write_event 事件；最后，Main_Buffer 中相应状态触发，缓冲区实际读写操作完成。

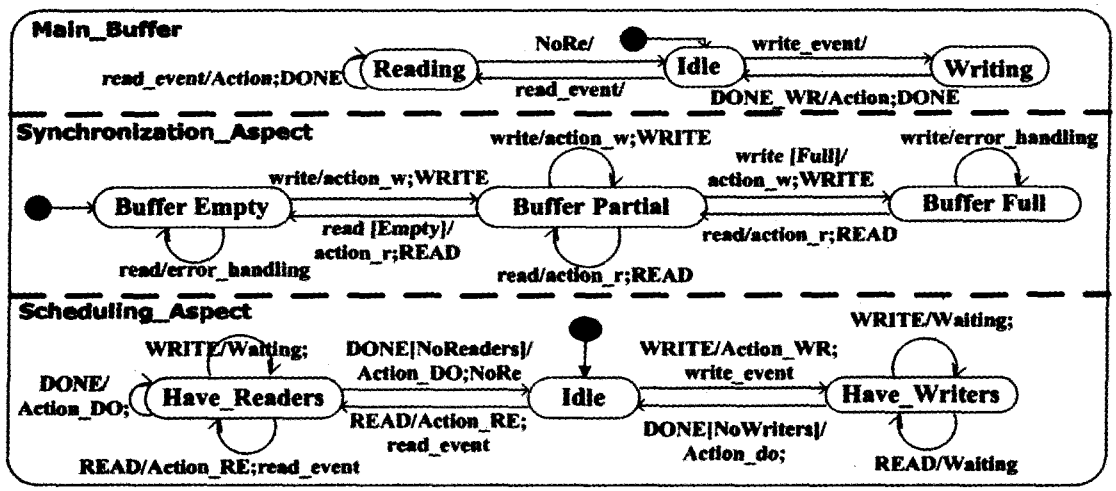

图 2 包含进程同步和调度横切特性的状态图模型

结论 面向侧面的并发系统建模体现了早期关注点分离的软件开发原则，简化了并发软件系统的开发、测试及维护工作。由于模块化状态迁移系统能够为该建模方法提供精确的

（下转第 262 页）

结论：I 是定义 2.1、2.2 和 2.3 的界面模型，$P=(X, Situation, Action, Layout, Element, Role, s_0, s_n)$的规划过程可以验证界面模型 I 的正确性。

证明：不失一般性，假设 s_0 是任意的某个初始界面，s_n 是目标界面，并且在界面关系模型 R 中存在 ts 转换序列，使得 $s_0 \xrightarrow{ts} s_n$。

对于规划问题 $P=(X, Situation, Action, Layout, Element, Role, s_0, s_n)$，

1）如果不存在 s_0 到 s_n 的规划，根据 $Poss_V$ 和 do_V 的定义，以及动作和界面转换之间的对应关系，有 $Valid\_S(s_0)=$ False 或者 s_0 到 s_n 的规划终止于零情景 Φ。于是可以验证界面模型 I 是无效的。

2）如果存在 $a=\langle a_0,\cdots,a_{n-1}\rangle$，使得 a 是 s_0 到达 s_n 的动作序列。即

$$s_0 \xrightarrow{a} s_n \equiv (\exists a_0,\cdots,a_{n-1})\ s_n = do_V(\langle a_0,\cdots,a_{n-1}\rangle, s_0) \wedge s_0 \leqslant s_n$$

根据 $Poss_V$ 和 do_V 的定义，$Valid_S(s_0)=$True，$Valid_S(s_n)=$True，并且 $\forall a \in \langle a_0,\cdots,a_{n-1}\rangle$，$Valid_A(a)=$True。根据公理 3.1，对于 s_0 到 s_n 规划过程中的每一个 s，也有 $Valid_S(s)=$True。

考虑 s_0 的任意性，可以验证界面模型 I 是正确的。

综合 1）和 2），得到 P 的规划可以验证界面模型 I 的正确性。

主要工作总结 我们通过引入问题相关的检验机制，基于情景演算的规划，为用户界面设计的验证提供了方法。我们的主要工作是：首先，对界面建模，建模的依据是业务需求，从而使得验证过程能够体现实际的业务流程；其次，提前在设计阶段检验界面是否正确，而不采用为应用系统生成测试用例的方式来验证界面，避免把错误带到开发阶段；第三，利用情景演算处理复杂状态变化的优势以及自动推理和计算能力，将验证转化为在规划过程中自动完成，解决了需要逐一对各种界面状态进行正确性判断的问题。但是，我们可以看到，前面讨论的检验机制在情景正确性和动作正确性判断过程中，对同一情景的正确性判断处理了两次，因此，更进一步的研究是简化验证过程，提高规划的效率，从而提高界面设计验证的效率。

参 考 文 献

1 Schlungbaum E. Model-based User Interface Software Tools: Current state of declarative models. GIT-GVU-96-30, Graphics, Visualization & Usability Center, Georgia Institute of Technology, Atlanta, November 1996

2 White L, Almezen H. Generating test cases for GUI responsibilities using complete interaction sequences. In: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE 2000), October 2000. 110～121

3 顾玉良，王立福. 界面类对象建模技术研究. 计算机工程，1999，25(7)：21～23

4 杜栓柱，谭建荣，陆国栋. 基于界面构件关联图的软件功能测试技术. 计算机研究与发展，2002，39(2)：148～152

5 张涌，钱乐秋，王渊峰. 基于扩展有限状态机测试中测试输入数据自动选取的研究. 计算机学报，2003，26(10)：1295～1303

6 Memon A M, Pollack M E, Soffa M L. Using a Goal-driven Approach to Generate Test Cases for GUIs. In: Proceedings of the 21st International Conference on Software Engineering, ACM Press, May 1999. 257～266

7 Memon A M, Pollack M E, Soffa M L. Plan Generation for GUI Testing. The Fifth International Conference on Artificial Intelligence Planning and Scheduling, AAAI press, April 2000. 226～235

8 McCarthy J. Situations, actions and causal laws: [Technical report]. Stanford University, 1963. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass, 1968. 410～417

9 梁伟晟，李磊. 基于表单的业务系统界面逻辑模型获取的研究. 计算机工程，已录用

10 Levesque H, Pirri F, Reiter R. Foundations for the Situation Calculus. Linköping Electronic Articles in Computer and Information Science, Linköping, Sweden, 1998, 3(18)

11 李磊. 会议报告：机器学习在需求工程中的应用. 见：第八届中国机器学习学术会议，2002

12 Reiter R. Proving properties of states in the situation calculus. Artificial Intelligence, 1993, 64: 337～351

（上接第 254 页）

语义，并且 UML 状态图直接反映了系统组件（类和 Aspects）的动态行为，所以并发系统的行为语义在设计和实现阶段保持了相关的一致性。今后的工作应该是结合具体 AOP（Aspect-Oriented Programming）环境如 AspectJ，实现并发软件系统的面向侧面代码自动化生成。

参 考 文 献

1 Kiczales G, Lamping J, Mendhekar A, Maeda C, et al. Aspect oriented programming [A]. In: Proceedings of the 11th Europeen Conf. Object-Oriented Programming [C]. Berlin: Springer-Verlag, 1997. 220～242

2 Andrews J H. Process-algebraic foundations of aspect oriented programming [A]. In: Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns [C]. Berlin, Heidelberg: Springer-Verlag, Sept. 2001. 187～209

3 Loughran N, Rashid A. Mining Aspects [A]. In: Proceedings of Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop [C]. Enschede, Holland: IEEE Computer Society Press, 2002. 15～23

4 Moreira A, Ara′ujo J, Brito I. Crosscutting Quality Attributes for Requirements Engineering [A]. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering [C]. New York: ACM Press, 2002. 167～174

5 Pnueli A. A temporal logic of concurrent programs [A]. 18th IEEE Symposium on Foundation of Computer Science [C]. Providence, Rhde Island: IEEE Computer Society Press, 1977. 67～77

6 Richner T, Ducasse S. Recovering high level views of object-oriented applications from static and dynamic information [A]. In: Proceedings of International Conference on Software Maintenance [C]. Oxford, UK: IEEE CS Press, 1999. 13～22

7 Booch, Rumbaugh, Jacobson. The Unified Modeling Language User Guide [M]. New York: Addison Wesley, 1999. 25～37

8 Manna Z, Pnueli A. The Temporal Logic for Reactive and Concurrent Systems: Specification [M]. New York: Spring-Verlag, 1991. 103～176