软件体系结构性能评价研究*)

赵会群[1] 孙 晶[1] 王国仁[2] 高 远[2]

(北方工业大学计算机系 北京100041)[1] (东北大学计算机科学与工程系 沈阳110006)[2]

Study on Software Architecture Performance Evaluation

ZHAO Hui-Qun[1] SUN Jing[1] WANG Guo-Ren[2] GAO Yuan[2]

(Dept. of Computer Science, North China University of Technology, Beijing 100041)[1]

(Northeastern University, Dept. of Computer Science and Engineering, Shenyang 110006)[2]

Abstract The software architecture is a design of application system, performance evaluation of software architecture during the early stage of their development is really attractive. This paper proposes a new method for software architecture performance modeling. To achieve this, it adds new calculus into stochastic process algebra (SPA in short), the developed SPA called extended stochastic process algebra (ESPA in short). By ESPA, performance evaluation and software architecture modeling can combine perfectly. It defines a few performance terms for software architecture using reward structure derived from ESPA. To explain the performance terms it also designs an experiment.

Keywords Software architecture, Stochastic process algebra, Performance evaluation

1 引言

随着软件规模和复杂程度不断地扩大和增加，软件开发的成败已不再完全取决于数据结构和软件算法的选择，而是在很大程度上取决于软件系统体系结构(Software Architecture)的设计。软件体系结构已经成为一个新兴的计算机学科[1]。所谓的软件体系结构是问题解决方案的逻辑框架，它包括系统中各计算单元(组件)的功能分配、各单元间的高层交互(连接器)说明以及体系结构的约束[2]。

软件体系结构的热点研究问题之一是软件体系结构描述语言(software architecture description language, ADL)。所谓的ADLs是对软件体系结构的形式化描述，它只关心软件系统的高层结构，不关心软件的实现细节[2]。ADLs在系统的逻辑设计与程序设计之间架起一个“桥梁”，使系统设计者可以在系统设计阶段测试、分析系统的正确性、适应性。虽然ADL的研究历史不是很长，但ADLs的种类很多，它们采用不同的风格和手段建立软件体系结构视图[3]。随着研究的深入，对ADL要求也越来越高。在系统设计阶段对软件系统性能有效地分析将成为新的需求，而现有的ADLs却没有提供相应的研究结果[3]。

本文从软件体系结构建模角度，对软件系统性能分析方法进行讨论，提出一种基于SPA的软件体系结构建模机制和基于该建模机制的软件系统性能分析方法。文中简要介绍随机进程代数理论，并针对软件体系结构建模需求对SPA进行了适当的扩充；建立软件体系结构的随机进程代数模型，并给出各种软件系统结构性能定义和评价方法；最后通过实验检验性能评价方法的有效性。

2 基础知识

SPA是一种形式化的分布系统建模方法，它把系统看成独立实体组成的集合，实体又称代理(agents)，各代理的执行进程是随机原子活动(atomic actions)集合[4]。随机进程代数现已形成了若干分支。本文以PEPA为参考，PEPA定义的进程运算如下[4]：

(1)顺序组合(Sequential composition)‘·’：表示进程中的活动顺序执行。

(2)选择(Selection)‘+’：表示两个进程可选择其一执行。

(3)并发(Concurrence)‘‖’：表示两个进程中活动独立执行。

(4)协同或同步(Cooperation/Synchronization)‘Θ’：表示多个进程中的同类活动协同操作，H表示同类活动集。

(5)封装运算(Encapsulation)‘/’：表示进程对活动的隐埋。

运算(3)是运算(4)的特例，即当协同运算中的H为空时为并发运算。

由于PEPA是面向计算机(通讯)系统的(代数)建模语言，对软件系统的描述能力较弱，为此本文对PEPA进行扩展，增加描述软件系统行为的运算，并用具有时序特征的事件来定义进程运算，以此方便软件系统的仿真。

设$\forall e_1, e_2 \in E$是事件域中的两个事件，e_1和e_2的发生按时间和因果有以下基本关系：

事件e_1和e_2时间等于，记为：“$e_1 =_t e_2$”，表示e_1和e_2同时发生；

事件e_1和e_2时间小于等于，记为：“$e_1 <_t e_2$”，表示事件e_1在e_2前发生；

事件e_1和e_2原因等于，记为：“$e_1 =_c e_2$”，表示事件e_1和e_2互为原因；

事件e_1和e_2原因小于等于，记为：“$e_1 <_c e_2$”，表示事件e_2因e_1而产生。

*)国家八、六三高技术研究发展计划项目(项目号：863-511-946-003)资助。赵会群 博士，副教授，研究方向：软件体系结构。孙 晶 硕士，讲师，研究方向：软件体系结构。王国仁 教授，研究方向：分布对象技术。高 远 教授，主要从事软件容错及高可信计算机系统方面研究工作。

由事件的$=_c$、$<_c$和$=_t$、$<_t$可以定义事件时间和原因的小于等于关系$\leqslant_c$和$\leqslant_t$。

规则1 $e_1<_t e_2\Rightarrow\neg(e_2<_c e_1)$；$e_1<_c e_2\Rightarrow\neg(e_2<_t e_1)$（$\Rightarrow$为永真蕴涵，下同。）。

在上述事件关系基础上可以定义事件的“调用”、“激发”、“同步”、“独立”、“顺序”等关系运算。为方便描述问题，定义一些常用的事件：Start 表示开始、Finish 表示结束、Succ 表示成功。

定义2.1 设$\forall e_1,e_2\in E$(E 是事件域)，如果$(e_1<_c e_2)\wedge(\text{finish}(e_2)<_t\text{finish}(e_1))$成立，则称事件 e_1调用 e_2，记为：$e_1\triangleright e_2$；如果$e_1<_c e_2$成立，则称事件 e_1激发 e_2，记为：$e_1\rightarrow e_2$；如果$e_1<_c e_2$并且$e_1=_t e_2$成立，则称事件 e_2与 e_1同步，记为：$e_1\Theta e_2$；如果$\neg(e_1\rightarrow e_2)\wedge\neg(e_2\rightarrow e_1)$，则称事件 e_1与 e_2独立，记为：$e_1\|e_2$；如果$\text{finish}(e_1)<_t\text{start}(e_2)$成立，则称事件 e_1和 e_2顺序发生，记为：$e_1\cdot e_2$。

显然，上述事件关系满足传递性。

定义2.2 一个进程是由一组原子活动组成的集合，每个原子活动由该活动所经历的事件构成。用 βp 表示进程 P 所对应的活动集，用 αA 表示活动 A 所对应的事件集。

定义2.3 设 P 表示进程域，E 表示对应的事件域，p_1、$p_2\in P$ 为进程，进程运算如下：

(1)p_1激发 p_2，记为：$p_1\rightarrow p_2$；$\beta(p_1\rightarrow p_2)=\{(a_1,a_2)\mid a_1\in\beta p_1\wedge a_2\in\beta p_2\wedge\forall e_2\in a_2(\exists e_1(e_1\in a_1))\Rightarrow e_1\rightarrow e_2\}$；

(2)p_1调用 p_2，记为：$p_1\triangleright p_2$；$\beta(p_1\triangleright p_2)=\{(a_1,a_2)\mid a_1\in\beta p_1\wedge a_2\in\beta p_2\wedge\forall e_2\in a_2(\exists e_1(e_1\in a_1))\Rightarrow e_1\triangleright e_2\}$；

(3)p_1并发 p_2，记为：$p_1\|p_2$；$\beta(p_1\|p_2)=\{(a_1,a_2)\mid a_1\in\beta p_1\wedge a_2\in\beta p_2\wedge\forall e_1,e_2(e_1\in a_1\wedge e_2\in a_2)\Rightarrow e_1\|e_2)\}$；

(4)p_1或 p_2，记为：p_1+p_2；$\beta(p_1+p_2)=\{(a_1,a_2)\mid a_1\in\beta p_1\wedge a_2\in\beta p_2\wedge\forall e_1,e_2(e_1\in a_1\wedge e_2\in a_2)\Rightarrow e_1\cup e_2)\}$；

(5)p_1同步 p_2，记为：$p_1\Theta p_2$；$\beta(p_1\Theta p_2)=\{(a_1,a_2)\mid a_1\in\beta p_1\wedge a_2\in\beta p_2\wedge\exists e_1,e_2(e_1\in a_1\wedge e_2\in a_2)\Rightarrow e_1\Theta e_2)\}$；

(6)p 重复执行，记为：$(T,\lambda)\cdot p$；$\beta((T,\lambda)\cdot p)=\{(a_1,a_2)\mid T\subset p\wedge a_1,a_2\in T\wedge\forall e_1,e_2(e_1\in a_1\wedge e_2\in a_2)\Rightarrow e_1\cdot e_2\}$；其中：T 中元素个数为重复次数，λ 为 T 中活动随机转换率。

(7)p_1隐藏 p_2，记为：p_1/p_2；$\beta(p_1/p_2)=\{(a_1,a_2)\mid a_1\in\beta p_1\wedge a_2\in\beta p_2\wedge p_2\subset p_1\wedge\forall e_1,e_2(e_1\in a_1\wedge e_2\in a_2)\Rightarrow e_1\cup e_2=e_1)\}$；

(8)p_1串行 p_2，记为：p_1*p_2；$\beta(p_1*p_2)=\{(a_1,a_2)\mid a_1\in\beta p_1\wedge a_2\in\beta p_2\wedge\forall e_1,e_2(e_1\in a_1\wedge e_2\in a_2)\Rightarrow(e_1\cdot e_2)\}$。

可以证明运算“$\rightarrow$”、“$\triangleright$”、“$\|$”、“Θ”、“$*$”和“$+$”分别满足结合律；运算“$\|$”、“Θ”和“$+$”分别满足交换率；运算“$\rightarrow$”、“$\triangleright$”、“$\|$”、“$*$”和“Θ”对运算“$+$”满足分配律。

定义2.4 设$\forall p_1,p_2,\cdots,p_n\in P$ 为进程，定义进程表达式如下：

(1)$p_i(i\in N)$为进程表达式；

(2)对 p_i 有限次地进行定义2.3中定义的运算后仍是进程表达式，进程表达式也称扩展随机进程代数模型，简称 ESPA 模型。ESPA 模型对应的事件集称为 ESPA 事件模型或仿真模型。

定义2.5 设 P 是进程模型中所有进程的集合，O 是进程模型中对进程所施加运算的集合，由 P 和 O 组成的二元组 E=⟨P,O⟩称为 ESPA 系统。

定理2.1 设 E=⟨P,O⟩是 ESPA 系统，则 S 是一个代数系统。

证明：由进程运算的封闭性可得定理2.1的正确性。 □

上述运算的优先级按由高到底为：·、/、*（同级）、→、▷（同级）、‖、Θ(同级)、+。

3 软件体系结构与进程代数模型

在讨论软件体系结构建模之前，首先根据文[3]提出的软件体系结构特征框架给出组件、连接器和软件体系结构的定义。

3.1 软件体系结构建模

根据本文讨论需要，下面只从功能和构成角度给出组件和连接器的描述定义，更严格的定义参见文[3]。

定义3.1 组件是一个数据单元或一个计算单元，它由组件接口和组件实现模块组成。

定义3.2 连接器是软件体系结构的一个组成部分，是组件交互协议的实现，连接器也由连接器接口和连接器实现两部分构成。

如果把连接器理解成为组件的“运算”，定义软件体系结构表达式如下。

定义3.3 设$\forall c_1,c_2,\cdots,c_n\in C$ 为组件，定义软件体系结构表达式如下：(1)$c_i(i\in N)$为软件体系结构表达式；(2)对 c_i 有限次地进行连接运算后仍是软件体系结构表达式，简称结构表达式。

定义3.4 设 C 是结构表达式中所有组件的集合，O 是结构表达式中对组件所施加“运算”的集合，由 P 和 O 组成的二元组 S=⟨C,O⟩称为软件体系结构，简称结构。

定理3.1 设 S=⟨P,O⟩是结构，则 S 是一个代数系统。

证明：由连接器的定义可得软件体系结构对运算封闭，所以定理3.1得证。 □

如果把软件体系结构与 EPSA 系统进行比较，把组件的执行看成一个进程，组件的方法(Publ,Exte,Priv)对应“进程”的原子活动，进程运算看成组件的运算；把连接器理解成组件运算(·、/、*、→、▷、‖、Θ、+)的实现，则不难看出两个概念体系有着一种对应关系，为此有下面定理。

定理3.2 设 S=⟨P,O⟩是结构，则总可以建立一个 ESPA 系统 E=⟨P,O⟩使得 S 与 E 同构。

证明：根据代数系统同构定义可以得证，略。 □

由定理3.2可知 ESPA 模型是软件体系结构执行过程的一种抽象，所以我们可以用 ESPA 对软件体系结构建模和分析。下面给出一个三层 C/S 结构应用系统的建模实例。

案例1 一个学生注册组件 Enroll 调用登记组件的方法 Register.reserveSeat()预定座位和调用缴费组件的方法 Bill.addToBill()填写缴费记录。

ESPA 模型如下：

$S=(\text{Enroll}\rightarrow(\text{Register}\Theta R_1))+(\text{Enroll}\rightarrow(\text{Bill}\Theta R_2))$；

$\text{Register}=(\text{use}_1,r_1)\cdot(\text{task}_1,r_2)\cdot\text{Register}$；

$\text{Bill}=(\text{use}_2,r_3)\cdot(\text{task}_2,r_4)\cdot\text{Bill}$；

$R_1=(\text{use}_1,r_5)\cdot(\text{update}_1,r_6)\cdot R_1$

$R_2=(\text{use}_2,r_7)\cdot(\text{update}_2,r_8)\cdot R_2$。

上述 ESPA 模型中的同步活动集 H 为$\{\text{use}_1,\text{use}_2\}$，这里省略了 get 和 release 活动。同步活动 use_1和 use_2的转换率分别为 $\min\{r_1,r_5\}$和 $\min\{r_2,r_7\}$。

3.2 软件系统性能评价

文[4]证明 PEPA 进程模型与连续时间马尔可夫过程(Markov process)模型相对应。然而，因为 ESPA 模型中引入“调用”运算，所以 ESPA 模型无法保证以代理进程为状态的

随机过程具有 Markov 过程特性。但可以证明 ESPA 模型具有 semi-Markov 特性[5]，所以仍可以借鉴文[4]中的"回报结构"(Reward structure)定义软件体系结构的性能模型。

定义3.5 假设 ESPA 模型的代理进程与一个状态 Si 对应，$\Pi(S_i)$为系统在状态 S_i 的分布，λ_i 为状态 S_i 转换权值(率)，定义 ESPA 模型的性能方程为：$P=\sum\lambda_i\Pi(S_i)$。

其中：$\Pi(S_i)$是一般分布，并有 $\sum\Pi(s_i)=1$；$\Pi Q=0$，Q 为转移矩阵。

根据ESPA 模型的性能方程可以定义与 ESPA 模型同构的软件体系结构的各种性能评价指标。

定义3.6 设 A 为进程活动集，对$\forall a_{ij}\in A$，令 λ_{ij}为活动 a_{ij}的权值，当 $a_{ij}\in H$(H 为对资源使用的同步活动集)，λ_{ij}取得权值，否则权值为0。则把 $U=\sum\lambda_{ij}\Pi(s_i)$称为 ESPA 系统对资源的利用率。

其中：Si 为组件对应的状态。

定义3.7 设 A 为进程活动集，对$\forall a_{ij}\in A$，令 λ_{ij}为活动 a_{ij}的权值，当 $a_{ij}\in F$(F 为单位时间内完成任务的活动集)，λ_{ij}取得权值，否则权值为0。则把 $U=\sum\lambda_{ij}\Pi(s_i)$称为 ESPA 系统的吞吐量。

定义3.8 设 A 为进程活动集，对$\forall a_{ij}\in A$，令 λ_{ij}，μ_{mk}分别为两个不同类活动的权值，当 $a_{ij}\in B$(B 为获得系统资源使用权限的活动集)，λ_{ij}取得权值，否则权值为0；当 $a_{ij}\in C$(C 为系统释放资源使用权限的活动集)，μ_{mk}取得权值，否则权值为0；则称 $\sum\lambda_{ij}\Pi(s_i)$为 ESPA 系统的延迟；$W=\sum\lambda_{ij}\Pi(s_i)-\sum\mu_{mk}\Pi(s_m)$为 ESPA 的负载。

在上述定义中，如果把进程看作软件体系结构中的组件，则根据定理3.2可得与软件体系结构相应的各种性能定义。

基于 ESPA 模型的软件体系结构性能评价步骤如下：

(1)建立系统的一个 ESPA 模型。

(2)求解具有 Markov 或 Semi-Markov 特性的状态转换矩阵 Q，进而求解稳定状态概率分布。

(3)计算系统性能指标。

4 应用实例

案例2 [续案例1]评价学生注册系统对资源的利用律和业务的吞吐量。

案例2的 ESPA 形式化描述为：$Enrollment=(use,r_1)\cdot(task,r_2)\cdot Enrollment$；$Database=(use,r_3)\cdot(update,r_4)\cdot Enrollment$；$Systme=Enrollment\Theta_{use}Database$

这里省略了对 Register 和 Bill 的调用，并假定状态转移满足 Markov 特性。上述 ESPA 描述对应的状态转换图如图1所示。

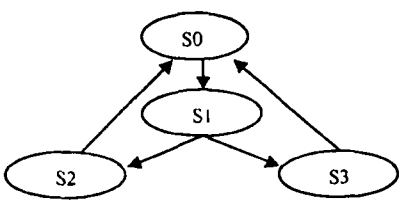

图1 学生注册过程模型状态转换图

其中：$S_0=P_{enroll}\Theta_{use}R_{database}$，$S_1=P'_{enroll}\Theta_{use}R'_{database}$，$S_2=P_{enroll}\Theta_{use}R'_{database}$，$S_3=P'_{enroll}\Theta_{use}R_{database}$。而 P'_{enroll}和 $R'_{database}$分别是 P_{enroll}和 $R_{database}$的一步执行结果。

转移矩阵 Q 为：$\begin{pmatrix}-r_{13} & r_{13} & 0 & 0\\ 0 & -(r_2+r_4) & r_2 & r_4\\ r_4 & 0 & -r_4 & 0\\ r_2 & 0 & 0 & -r_2\end{pmatrix}$ 其中：$r_{13}=\min(r_1,r_3)$；

解方程组 $\Pi Q=0$得：

$$\Pi(s_0)=r_2r_4(r_2+r_4)/((r_2+r_4)r_2r_4+r_{13}r_2r_4+r_{13}r_2^2+r_{13}r_4^2)$$

$$\Pi(s_1)=r_2r_4r_{13}/((r_2+r_4)r_2r_4+r_{13}r_2r_4+r_{13}r_2^2+r_{13}r_4^2)\quad(1)$$

$$\Pi(s_2)=r_{13}r_2^2/((r_2+r_4)r_2r_4+r_{13}r_2r_4+r_{13}r_2^2+r_{13}r_4^2)$$

$$\Pi(s_3)=r_{13}r_4^2/((r_2+r_4)r_2r_4+r_{13}r_2r_4+r_{13}r_2^2+r_{13}r_4^2)$$

执行 ESPA 事件模型(仿真)得：$r_1=2$，$r_2=2$，$r_3=6$，$r_4=8$，$r_{13}=\min(2,6)=2$。代入(1)中有：

$\Pi(s_0)=20/41$，$\Pi(s_1)=4/41$，$\Pi(s_2)=1/41$，$\Pi(s_3)=16/41$

计算系统资源利用率：系统资源利用率可认为是当 USE 或 UPDATE 发生时的系统加权概率和($R=\sum\rho_i\Pi(s_i)$)，对图1$\rho_0=1$、$\rho_1=1$、$\rho_2=1$、$\rho_3=0$。所以系统资源利用率为：

$U=\rho_0\Pi(s_0)+\rho_1\Pi(s_1)+\rho_2\Pi(s_2)+\rho_3\Pi(s_3)=25/41=60.98\%$。

结论 随机 Petri 网等建模理论中也有相关研究[7]，但这些研究都需要建立同构的 Markov 过程模型，并严格限制被评价系统的 Markov 过程特性。与上述研究不同，本文对 PEPA 进行扩展，提出更加适合软件体系结构建模的 ESPA 模型，该模型把软件系统设计与性能评价相结合，通过求解性能方程就可以在系统设计阶段计算系统的各种性能指标，从而指导系统设计。软件体系结构性能评价是首次提出，今后我们将在 ADLs 以及性能评价工具方面继续开展研究。

参 考 文 献

1 Shaw M, Garlan D. Software Architecture Perspectives On An Emerging Discipline. Prentice Hall, 1998

2 Allen R, Garlan D. Formalizing Architectural Connection. In: Proc. of the 16th Intl. Conf. on SW Engineering, Sirrebti Italy, May 1994. 71～80

3 Medvidovic N, Taylor R N. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE, 2000, 26(1).

4 Hillston J, Ribaudo M. Stochastic process algebra: a new approach to performance modeling. Modeling and Simulating of Advanced Computer Systems. Gordon Breach, 1998. 235～255

5 赵会群，高远，等. 基于组件的软件可靠性模型. 小型微型计算机系统，2002(6)

6 Stewart W J, Atif K, Plateau B. The Numerical Solution of Stochastic Automata Networks. European Journal of Operations Research, 1995, 86(3): 503～525

7 Ciardo G, Muppala J, Trivedi K S. SPNP: Stochastic Petri Net Package. In Petri Nets and Performance Models, pages 142(150, Kyoto, Japan, December 1989. IEEE. (p38)