UML 行为图驱动的 Java 程序运行时验证工具

邱晓康　陈铭松　王林章　李宣东　郑国梁

（南京大学计算机软件新技术国家重点实验室　南京大学计算机科学与技术系　南京 210093）

摘　要　UML 是一种标准的可视化建模工具，广泛应用于软件系统的描述、可视化、构建和建立文档。本文介绍了一种 UML 行为图驱动的 Java 程序运行时验证工具。该工具以一个随机的测试用例集作为输入，运行经过插装的被测 Java 程序，得到一组用于验证的程序运行轨迹。通过对程序运行轨迹和 UML 行为图中合法的事件序列的比较，该工具可以对程序的动态行为规约进行检查。本文描述了该工具的设计思想、算法和实现技术，并通过对实例研究对该工具的可用性和有效性进行了讨论。

关键词　运行时验证，UML 行为图，插装，随机测试，Java

UML Behavior Diagram Driven Tool for Runtime Verification of Java Programs

QIU Xiao-Kang　CHEN Ming-Song　WANG Lin-Zhang　LI Xuan-Dong　ZHENG Guo-Liang

（State Key Lab for Novel Software Technology，Nanjing University，Nanjing 210093）

（Department of Computer Science and Technology，Nanjing University，Nanjing 210093）

Abstract　The UML is a standard visual modeling language that is specified to specify，visualize，construct and document the artifacts of software systems. In this paper，we introduce a UML behavior diagram driven tool for runtime verification of Java programs. It takes a set of random test cases as input and run the instrumented Java programs to get runtime traces for verification. Then it check the dynamic behavior specifications of the programs by compare the program execution traces and the legal sequences of events. In this paper，we describe this tool in detail，including its design，algorithms and implementation，and present several cases to show its availability and effectiveness.

Keywords　Runtime verification，UML behavior diagrams，Instrumentation，Random testing，Java

1　引言

测试和验证是保证软件质量的两种主要方法。软件测试是为了发现程序中的错误而执行程序的过程，通过定义各种测试充分性，可以提高我们对被测软件质量的信心，但无法回答系统一定没有错误这样一类问题，因而无法从根本上确保系统的可靠性。而验证则通过基于数学的形式化方法，揭示系统的不一致性、歧义性和不完备性，可以从某一个角度回答系统一定没有错误这样一类问题，提高我们对系统可靠性的信心，往往被应用于一些关键领域的软件系统。运行时验证简单地说可以看作软件测试和形式化方法的结合[12]。它是一种保证程序可靠性的轻量化方法，其基本思想是收集程序运行中的相关信息并得到与程序相关的测试或操作中的某些性质。对于一个给定的程序执行轨迹，我们可以验证该轨迹是否满足程序规约。如不满足，则报告存在的缺陷[7]。

统一建模语言 UML(Unified Modeling Language)是一种标准化的可视化建模语言，它被用于对软件系统架构的可视化描述和构建，以及建立文档[24]。在最新的 UML2.0 版本中共有 13 种图，可以分为结构图和行为图两类。行为图包括活动图、用例图、状态机图、顺序图、交互概观图、通信图和时间图。UML 行为图用于展示一个系统中对象的动态行为，包括它们的方法、合作、活动和状态历史。一个系统的动态行为可以描述成系统随时间变化的一个序列。

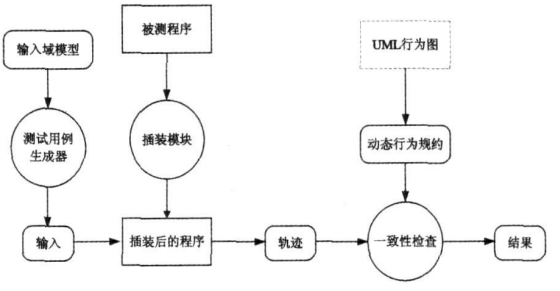


图 1　UML 行为图驱动的运行时验证工具框架

本文介绍了一个自动化的运行验证工具，该工具的设计框架如图 1 所示。它首先对被测程序进行插装，记录与 UML 动态行为规约相关的事件；然后根据被测程序的输入域模型，随机生成测试用例集合，并以此驱动插装后的待测程序运行，在运行过程中记录程序的运行轨迹。最后，工具对运行轨迹和满足 UML 行为规约的事件序列进行自动比较，检查被测程序行为与给定的行为规约之间的一致性。该工具可以接受 UML 状态机图、顺序图、交互概观图描述的行为规约，对 Java 程序进行验证。

在第 2 节中我们将介绍基于 UML 的动态行为规约。在第 3 节中描述工具的设计结构以及相关算法。第 4 节中介绍

邱晓康　硕士研究生，研究方向：软件工程、软件验证、形式化方法；**陈铭松**　硕士，研究方向：软件工程、软件验证、模型检验；**王林章**　博士，讲师，研究方向：软件工程、软件测试；**李宣东**　博士，教授，博士生导师，研究方向：软件工程、形式化方法、模型检验；**郑国梁**　教授，博士生导师，研究方向：软件工程、软件开发环境。

使用该工具进行的一些验证实例，并对工具的可用性和有效性进行了讨论。第5节是与相关研究工作的比较。最后总结了我们的工作以及对下一步工作的方向。

2 基于UML的动态行为规约

在本工具中，程序的动态行为规约用UML行为图来描述，主要包括UML状态机图、顺序图和交互概观图。

UML状态机图(State Machine Diagrams)用于对有限状态系统中的离散行为建模[20]。它使用状态机来描述各种建模元素的行为，例如一个类的实例。一个状态表示一个类在生命周期中的某个特定点，允许进行某一类特定的操作。调用对象的某种操作，通常是成员方法，可以触发状态的转换。一个状态机图包括一个初始状态，一个对象的生命周期从初始状态开始。如果将触发状态转换的方法调用看作事件，状态机图描述的一个对象行为就可以表示成一个事件序列。

UML顺序图(Sequence Diagrams)是最常用的一种UML交互图，主要描述一组生命线之间的消息交换。它通过描述消息交换的顺序以及相应的事件描述来描述一个交互场景[20]。一个顺序图有两个维度：垂直维度表示时间，水平维度表示不同的对象。每个对象用一个纵条表示，每个消息用一个水平的带标记的箭头表示[24]。在本文中，我们考虑只描述单一场景的简单顺序图，其中不包含分支和循环。在顺序图中，我们把消息的发送和接收都看作事件，因此可以把顺序图的语义表示为消息发送和接收时间的序列。根据每个对象的控制流所确定的可视化偏序关系，以及消息发送和接收事件之间的因果依赖关系，可以推导出一个顺序图中的事件序列[23,24]。

交互概观图(Interaction Overview Diagrams)是UML 2.0中一种新的交互图，主要描述以各种交互场景作为节点的控制流。交互概观图使用了活动图的表示方法，其中每个活动节点代表一个活动，生命线以及消息等较低层次的元素不会显式地出现在这一概观层次中[20]。

在我们过去的工作[17,18]中已经形式化地定义了用UML状态机图、顺序图和交互概观图描述行为规约，以及一个事件序列满足此类行为规约时需满足的条件。为了降低复杂性，我们在这些定义中对UML行为图作了以下4个限制来进行简化：

(1)不包含任何并发组合，即不存在分支结点和合并结点；

(2)不包含任何时间约束；

(3)在顺序图中描述的对象均属于单体类，即顺序图中涉及的类在系统中只出现一个实例；

(4)一个交互概观图中的所有顺序图具有共同的生命线，即描述了一个共同的对象集合中的消息交换。

在本文中，我们以状态机图为例，介绍基于UML的动态行为规约的形式化描述。

定义1 一个状态机图是一个五元组 $G=(S, E, T, s^0, c)$，其中

S 是一个有穷状态集合；

E 是一个有穷事件集合，每个事件对应于一个成员方法的调用；

$T\subseteq S\times E\times S$ 是一个有穷转换关系集合，T 中的任何转换可以用(s, e, s')的形式表示，其中 $s, s'\in S$，分别表示转换前后的状态，$e\in E$，表示触发转换的事件；

$s^0\in S$ 是起始状态；

c 是 G 描述的对象所属的类。

我们使用事件序列来表示状态机图的轨迹，即方法调用的时间顺序。一个事件序列可以用 $e_0\wedge e_1\wedge\cdots\wedge e_m$ 的形式表示，其中对任何 $i(0\leqslant i\leqslant m-1)$，事件 e_{i+1} 发生在 e_i 之后。

定义2 给定一个状态机图 $G=(S, E, T, s^0, c)$，一个事件序列 $e_0\wedge e_1\wedge\cdots\wedge e_m$ 是 G 的一个轨迹当且仅当存在一个状态序列 $s_0\wedge s_1\wedge\cdots\wedge S_{m+1}$，满足以下条件：

- $\forall i(0\leqslant i\leqslant m+1)$，$s_i\in S$；
- s_0 是状态 s^0；
- $\forall i(0\leqslant i\leqslant m)(s_i, e_i, s_{i+1})\in T$；

3 运行时验证工具设计

本工具的设计结构如图2所示。本节中将根据我们的验证工具中三个主要的模块，分别介绍使用的相关技术和算法。

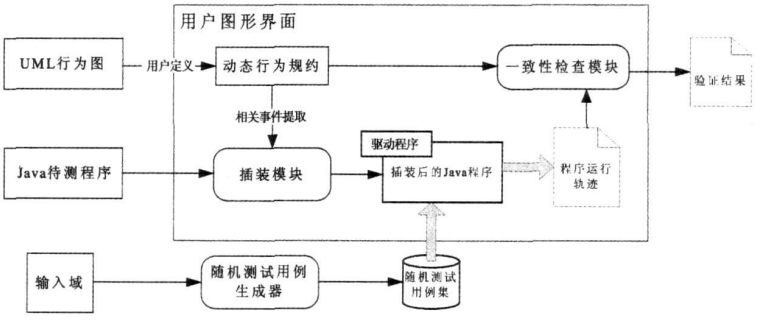


图2 验证工具的设计结构

3.1 随机测试用例生成器

随机测试和一般的系统性测试不同，在选择测试用例时没有明显的系统性，因此不同的测试之间不存在相关性[10]。尽管对于随机测试是否比划分测试更有效仍然存在争议[9]，它的两个优点使得我们在本文的验证工具框架中使用它：第一，它的成本低廉，而且易于实现，特别是当我们需要一个非常大的测试用例集时；第二，它可以使测试用例生成的过程自动化，从而可以降低工作量并减少错误。

对于不同的被测程序，需要构造不同的测试用例生成器。因此我们不能为所有的应用提供一种通用的解决方案，而只是从整体的意义上进行讨论。随机测试用例生成器接收被测程序输入域的模型作为输入，并且自动地生成输入。它还可以使用程序的运行特征(operational profile)，即对程序预期试用情况的描述。目前已经有一些用于单元测试的随机生成工具出现[21]。在纯粹的随机机制中也可以通过设定各个类和操作的权重，以及根据被测类创建的实例个数来进行参数

化。

在使用随机测试中的一个重要问题就是如何生成随机值。组成测试集的输入值并不总是有穷的。当然，由于存储和输入媒介的大小限制，任何计算机程序的不同输入必须被限定在有限的个数内。但实际上这种有限的可能性往往仍然很大，可以视作无穷。对于这种非常大的输入空间，随机值的意义并不大，因此我们没有找到一种通用的技术或工具来自动化地生成随机值。考虑到这种困难性，我们只为输入相对较为简单的程序生成随机测试用例。

3.2 插装模块

程序插装是动态测试中的一种常用技术[14]。它的主要思想是在程序中插入一些探针(probe)，即在源代码或目标代码中插入的一个或多个语句，用于记录程序动态的信息，同时保持被测程序的逻辑完整性。当被测程序执行时，探针也会被执行，并抛出运行时的特征数据。

在我们的测试工具中，通过对被测程序的插装来记录与消息发送和接收相关的事件。这种插装是通过程序自动完成的。在插装后的程序运行时，动态的事件序列就会记录下来，用于测试用例的选择和一致性的检查。

在 Java 程序中，一个方法的调用对应于顺序图或交互概观图中的一个消息发送事件，而该方法中第一个语句的执行对应于这个消息的接收事件或状态机图中的状态转换事件。因此，我们定义一个事件项如下：

定义 3 一个事件项是一个四元组 $V=(m, t, s, r)$，其中 m 是一个被发送(接收)的消息，用方法名以及以毫秒为单位的协调世界时(coordinated universal time)来表示，作为消息 ID；$t\in\{S, R\}$ 是事件的类型(我们用 S 和 R 分别表示消息的发送和接收)；$s(r)$ 是 m 的发送(接收)对象，用 Java 虚拟机中给出的哈希码表示的对象 ID，以及所属类的名称 $\tau(V)$ 来表示。

我们的插装算法详见文[18]。使用一个流分词器(stream tokenizer)对源代码进行分析，分成一些标记。当读到一些特定的标记时，就可以确定方法调用和定义的位置。由于无法在运行时记录下一个完整的事件项，我们在消息发送和接收时分别记录下关于发送者和接受者的信息。在程序执行结束后，根据消息 ID 对这两部分信息进行匹配。消息 ID 会作为一个额外的形式参数，在插装时添加到被插装的方法中。有时一个语句中可能存在多个方法调用的嵌套，在这种情况下，嵌套层次较深的方法应该比嵌套层次较浅的方法先调用。因此我们定义了一个调用栈来记录方法调用的嵌套关系，在扫描程序时对嵌套的方法调用从外向里依次推入栈中，最后出栈的顺序即方法调用的顺序。

3.3 一致性检查模块

在得到程序的执行轨迹和一致性规约后，就可以进行一致性的验证。根据上述的插装算法，通过运行插装后的程序，可以得到与一个给定的 UML 行为规约相关的程序运行轨迹。下面将介绍如何通过对程序运行轨迹和基于 UML 行为图的动态行为规约的比较来进行验证。

运行插装后程序得到的运行轨迹可以表示成一个事件项(见定义 3)的序列：$\sigma=v_0\wedge v_1\wedge\cdots\wedge v_n$。为了对程序执行轨迹和交互概观图的轨迹进行匹配，定义一个程序运行的踪迹如下：

定义 4 给定一个基于 UML 图的行为规约 G，一个程序执行轨迹 $\sigma=v_0\wedge v_1\wedge\cdots\wedge v_n$ 满足 G，如果存在一个相应的 G 的轨迹 $e_0\wedge e_1\wedge\cdots\wedge e_n$，使得

- 对任何 $i(0\leqslant i\leqslant n)$，$\tau(v_i)=\zeta(L(e_i))$；
- 对任何 $i(0\leqslant i\leqslant n)$，如果 e_i 是一个消息发送(接收)事件，则 v_i 对应于相同的消息发送(接收)事件；
- 如果 G 中存在一个顺序图 s，且 (e_i, g, e_j) 在 s 的有穷消息集合 M 中$(0\leqslant i<j\leqslant n)$，则 v_i 和 v_j 具有相同的 m，即对应于同一消息的发送和接收；
- 对任何 v_i 和 $v_j(0\leqslant i<j\leqslant n)$，如果 $\tau(v_i)=\tau(v_j)$，则具有相同的发送者(接收者)。

根据以上定义，我们给出了一致性检查算法[18]，检查是否一个执行轨迹文件中的所有轨迹满足给定的 UML 动态行为规约 G。该算法遍历算法中的每个轨迹，从每个轨迹的起始事件开始。已遍历过的轨迹段被保存在一个列表变量 currentpath 中。一旦发现 currentpath 与 G 一致，则继续检查下一条轨迹；如果一个轨迹扫描结束时仍然未与 G 一致，就返回 false，G。

4

。

UML 状态机图、

。

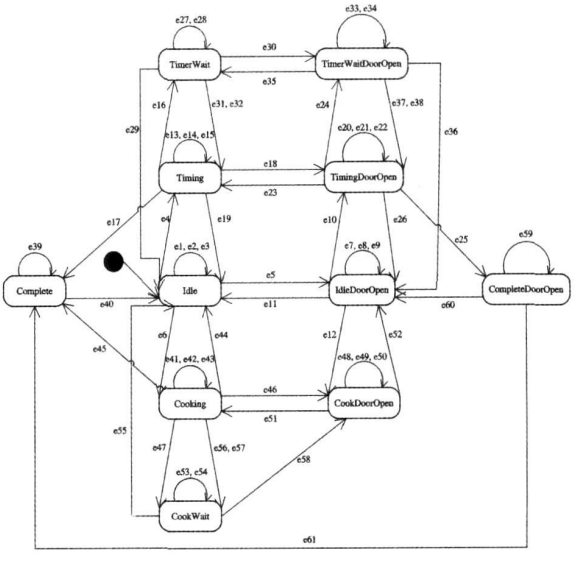
4.1

。Java 程序实现，17 个类、113 个方法。MicrowaveOven 类的状态机图作为设计规约，3 所示。11 个状态和 61 个状态转换事件。：

Cooking 状态发生了 openDoor 事件进入 CookDoorOpen 状态，pause 事件，，CookDoorOpen 状态没有与 pause 事件对应的状态转换。，

。，

。



3 MicrowaveOven 类的状态机图规约

另一个实例是基于一个用 Java 实现的养老保险系统 ORIS，17 个类和 241 个方法。ORIS 是一个工业界的实际项目并已投入实际使用。ORIS 的设计规约并不是用

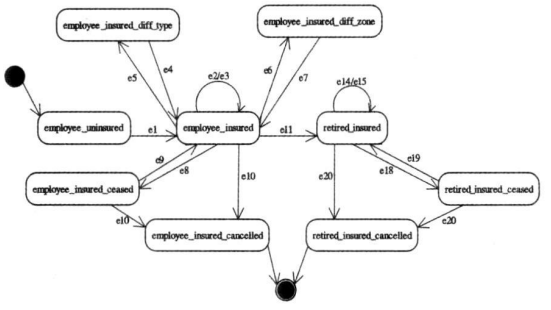
UML 描述的,

。 Person 类为例,

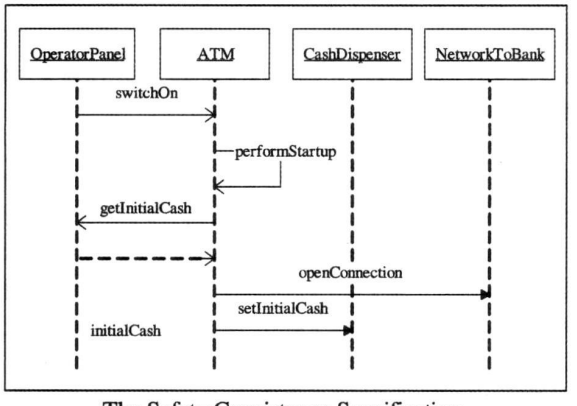
9 个状态和 17 个状态转换事件的状态机图, 4 所示。 , 。

, 。

。



4 Person 类的状态机图规约



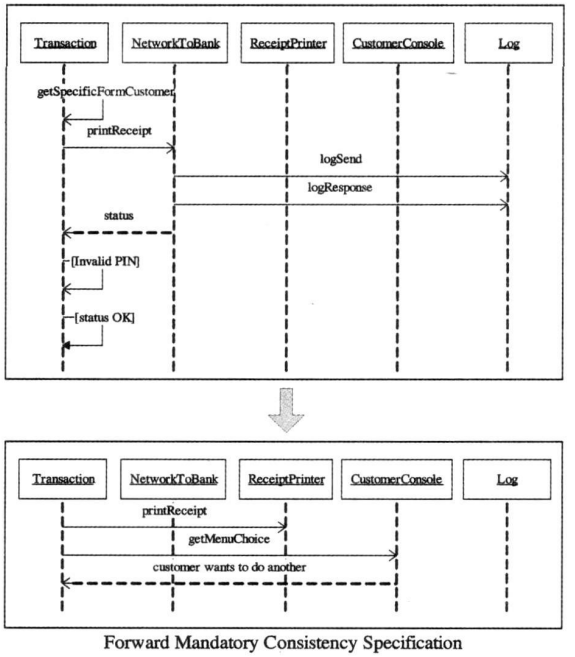


图 5 ATM 系统的顺序图规约

4.2

, ATM 模拟系统[2]。 、 ,

[2] 、 。

,

, 5 所示。 , [18]

。 。 ,

, , , ,

, 。

4.3

FIPA-OS v2.2.0 的[19]。 FIPA 协议族的开源系统。

FIPA 迭代契约网交互协议[8](IP, Iteration Protocol) FIPA 契约网协议的一个扩充, ,

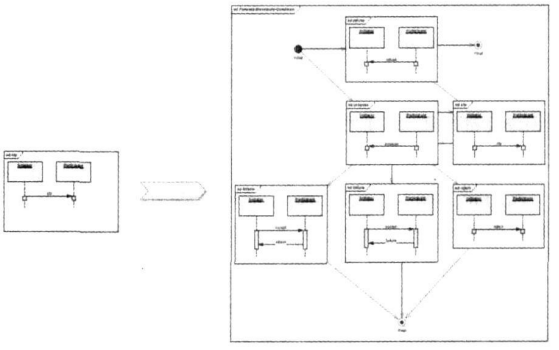
。

。

, , 6 所示。

, FIPA-OS v2.2.0 的架构, agent 之间的交互来完成某些任务。 5000 个任务作为程序输入, 5000 次执行中的运行轨迹。

5000 条轨迹中, 259 条违反了图 6 中描述的向前强制一致性规约。 259 条非法轨迹都在顺序图‘propose’ 。 , FIPA-OS v2.2.0 中的缺陷引起的。 , 6 中顺序图‘propose’ ‘cfp’ , ‘cfp’ ‘cfp’ 。 FIPA-OS v2.2.0 的运行时成功的验证发现了一个错误。



6 FIPA 迭代契约网交互协议的交互概观图规约

5

UML 状态机图和顺序图的运行时验证的研究工作[18,17]。 , UML 交互图表示的规约对 Java 程序作运行时验证的文献还很少。

Java 程序和其它程序上[12,1,15,3,25,6]。 ,

,

,

, ,

。

, UML 图使用更加广泛, UML 行为图描述的规约可以从软件开发过程的产品中直接产生。 ,

,

, 。 ,

UML 行为图作为规约语言的验证工具比采用基于时序逻辑的规约语言更容易被接受。 ,

UML 交互模型是否完备。 [16] ,
(Live Sequence Charts, LSCs)[5]
, 。
Java 程序的工作[13,11,22] [4] ,
, 。
UML 行为图驱动的对 Java 程序运行时验证。 ,
Java 程序,
。 UML行为图中合法的事件序列的比较, 。
, ,
。
。
, 。 ,
。
,
UML 规约的验证,
。 ,
, ,
。

1 Bartetzko D, Fischer C, Moller M, et al. Jass-Java with Assertions. Electronic Notes in Theoretical Computer Science, 2001, 55(2)
2 Bjork R C. The Simulation of an Automated Teller Machine. http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/Links.html
3 Brorkens M, Moller M. Dynamic event generation for runtime checking using the jdi. Electronic Notes in Theoretical Computer Science, 2002, 70(4): 21~35
4 Clarke E M, Grumberg O, Peled D A. Model Checking. MIT Press, 1999
5 Damm W, Harel D. LSCs: Breathing Life into Message Sequence Charts. Formal Methods in System Design, 2001, 19(1): 45~80
6 Drusinsky D. Semantics and Runtime Monitoring of TLCharts: Statechart Automata with Temporal Logic Conditioned Transitions. Electronic Notes in Theoretical Computer Science, 2001
7 Finkbeiner B, Sankaranarayanan S, Sipma H. Collecting Statistics over Runtime Executions. Electronic Notes in Theoretical Computer Science, 2002, 70(4): 1~19
8 Foundation for Intelligent Physical Agents. FIPA Iterated Contract Net Iteration Protocol Specification. http://www.fipa.org/specs/fipa00030/, 2002
9 Gutjahr W J. Partition Testing vs Random Testing: The Influence of Uncertainty. IEEE Trans on Software Engineering, 1999, 25(5): 661~674
10 Hamlet R. Random Testing. Encyclopedia of Software Engineering, 1994 970~978
11 Havelund K, Pressburger T. Model checking JAVA Programs Using JAVA PathFinder. Int J Software Tools for Technology Transfer, 2000(2): 336~381
12 Havelund K, Rosu G. Monitoring Java Programs with Java PathExplorer. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 1~18
13 Holzmann G J, Smith M H. Software Model Checking: Extracting Vrificatioin Models from Source Code. In: Proc. 12th Int'l Conference on Formal Description Techniques (FORTE/PSTV'99), 1999
14 Huang J C. Program Instrumentation and Software Testing. Computer, 1978, 11(4): 25~32
15 Kim M, Kannan S, Lee I, et al. Java-MaC: A Runtime Assurance Tool for Java Programs. Electronic Notes in Theoretical Computer Science, 2001, 55(2)
16 Lettrai M, Klose J. Scenario-based monitoring and testing of real-time UML models. In: Proc. 4th Int'l Conference on Unified Modeling Language (UML'01) Lecture Notes in Computer Science 2185, 2001 45~80
17 Li X, Qiu X, Wang L, et al. UML State Machine Diagram Driven Runtime Verification of Java Programs. Manuscript, 2006
18 Li X, Wang L, Qiu X, et al. Runtime Verification of Java Programs for Scenario-Based Specifications. In: Proc. 11th Int'l Conference on Reliable Software Technologies (AE'06), 2006
19 Nortel Networks Corporation. FIPA-OS Distribution Notes. http://fipa-os.sourceforge.net, 2002
20 Object Management Group. UML Superstructure Specification, v2.0. http://www.omg.org/docs/formal/05-07-04.pdf, 2005
21 Oriat C. Jartege: a Tool for Random Generation of Unit Tests for Java Classes. In: Proc. 2nd Int'l Workshop on Software Quality (SOQUA'05) Lecture Notes in Computer Science 3712, 2005 242~256
22 Park D Y, Stern U, Skakebak J U, et al. Java Model Checking. In: Proc. 1st Int'l Workshop on Automated Program Analysis, Testing and Verification, 2000
23 Peled D A. Software Reliability Methods. Springer, 2001
24 Rumbaugh J, Jacobson I, Booch G. Unified Modeling Language Reference Manual, 2nd ed. Addison-Wesley, 2004
25 Stolz V, Huch F. Runtime Verification of Concurrent Haskell Programs. Electronic Notes in Theoretical Computer Science, 2004

(上接第 264 页)

5 Luckham D C, Veral J. An event-based architecture definition language. IEEE Transaction on Software Engineering, 1995, 2(9): 717~734
6 Taylor R, Medvidovic N, Anderson K, E, et al. A component and message-based architectural style for GUI software [J]. IEEE Transactions on Software Engineering, June 1996. 390~406
7 Allen R, Garlan D. A formal basis for architectural connection. ACM Transactions on Software Engineering and Methodology, 1997, 6(3): 213~249
8 Magee J, Kramer J. Dynamic structure in software architectures. In: Kaiser G. E. ed. Proceedings of the ACM SIGSOFT'96: the 4th Symposium, Foundations of Software Engineering (FSE4), New York: ACM Press, 1996. 3~14
9 王晓光, , .ABC/ADL: XML 的软件体系结构描述语言. , 2004, 141(1): 1521~1531
10 冯铁, , , . , 2000, 11(8): 1078~ 1086
11 马俊涛, , . A-ADL: . , 2000, 11 (10): 1382~1389
12 骆华俊, , . XYZADL , 2000, 11(8): 1024~1029
13 张家晨, , , . , 2000, 11(8): 1024~1029
14 蔡谊, , . [J]. , 2004, 27(5): 620~624
15 Grsecurity. http://www.grsecurity.net/
16 Mei Hong, Chang Jichuan, Yang Fuqing. Composing software components at architectural level. The Int Conf on SoftwareTheoryandPractice, Beijing, 2000
17 梅宏, , , . ABC: 、 . , 2003, 14(4): 721~732
18 Kiczales G, Lamping J, Mendhekar A, et al. Aspect-Oriented programming. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241, Springer-Verlag, 1997. 220~242. http://citeseer.nj.nec.com/63210.html