ISSN1000-9825 Journal of Software 软件学报 2000,11(8):1053~1059

# 一个面向分布式程序的测试系统框架*

顾庆 陈道蓄 韩杰 谢立 孙钟秀

(南京大学计算机软件新技术国家重点实验室 南京 210093)

E-mail: guq@dislab.nju.edu.cn

**摘要** 提出了一个面向分布式程序的测试系统框架TFDS(test system framework for distributed software system),并介绍了它在异构网络中的一个实现原型PSET*(distributed program structure and event trace, revised version).框架的主要功能是对分布式程序进行单元测试和集成测试.包括面向规约设计和源码分析的静态部分和面向程序执行和事件序列分析的动态部分.在构件的基础上,PSET*的功能可以较容易地得以改进和增强.

**关键词** 分布式系统,测试系统框架,不确定性,事件序列,分布式程序设计语言.

**中图法分类号** TP311

分布式程序运行在多台机器构成的网络环境中[1],需要处理分布单元(进程)间的通信和同步以及不同平台结构的差异.传统的面向串行程序的测试可以由各种简单的工具来完成,测试工具本身不需要有复杂的内部结构[2].但对于分布式系统,测试工作不仅仅局限于单个进程或模块,它还要考虑运行时刻各分布进程间协作的正确性和效率.例如,对于Client/Server结构的程序,测试工具就必须考虑:

(1) Client和Server程序本身的正确性,可以利用传统的测试工具来检错;

(2) 网络通信通道的无差错性和性能;

(3) Client和Server交互的正确性和性能.

后两个问题都需要针对分布式程序的特有的测试工具.这些测试工具要面对异构的分布平台,还需要监控分布进程的运行以处理不确定性问题.所有这些决定了针对分布式程序系统的测试工具应是一个具有一定内部结构的测试系统.鉴于分布式系统发展很快,这个测试系统应易于更新和改进.

为了将测试工具实现为一个系统,需要一个测试系统框架来作为实现的参考.本文提出了一个行之有效的面向分布式程序的测试系统框架,它由面向程序规约和源码的静态部分和面向程序执行的动态部分构成,主要完成对分布式程序的单元测试和集成测试.根据该框架可以实现易于重构的分布式程序测试系统,以完成不同的测试功能和针对不同类型的分布式语言.

## 1 分布式程序测试

### 1.1 基本运行模式

目前,分布式程序系统普遍采取层次结构的运行模式,如CORBA,Java Applet,可以表示成如图1所示的结构.其中,操作系统为分布式程序的本机运行提供底层支持,这一层可能是异构的,如机器$A$运行的是Windows NT,而机器$B$上的则是UNIX.通信机制由各种通信协议构成,为使分布式程序成为一个相互协作的有机

* 本文研究得到国家"九五"重点科技攻关项目基金(No.98-780-01-07-03)资助.作者顾庆,1972年生,博士,讲师,主要研究领域为分布式系统与测试.陈道蓄,1947年生,教授,博士生导师,主要研究领域为分布式计算与并行处理.韩杰,1976年生,硕士生,主要研究领域为分布式计算,软件质量保障.谢立,1942年生,教授,博士生导师,主要研究领域为分布式计算,并行处理.孙钟秀,1936年生,教授,博士生导师,中国科学院院士,主要研究领域为分布式计算,并行处理.

本文通讯联系人:顾庆,南京 210093,南京大学计算机软件新技术国家重点实验室

本文2000-01-17收到原稿,2000-06-13收到修改稿

整体,一般由这一层来屏蔽底层的不同[1],如双方都采用 TCP/IP 协议或 PVM 等.再上一层是语言编译器及支撑系统,支撑系统协助程序在不同的平台上运行——如 Java 虚拟机,可同时完成调度和资源分配等功能.

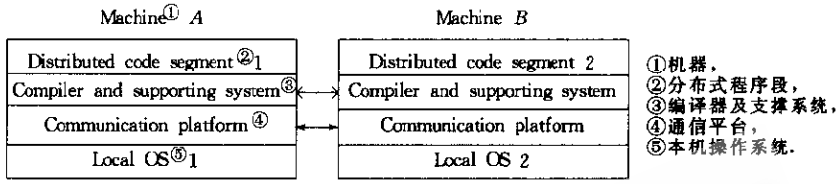


Fig. 1　A running mode of distributed programs
图1　分布式程序运行模式

### 1.2　分布式程序测试的主要问题

分布式程序同传统串行程序(imperative)的根本区别在于程序运行的不确定性.即:在运行的某一时刻,程序处于状态 $s$ 时,其下一步的动作及由该动作所导致的状态不可预知.以生产者-消费者问题为例,当某一生产者生产了一个产品并将其放入仓库后,接下来的可能动作或者是生产者继续生产产品,或者是消费者从仓库中取出产品;不同的动作会导致不同的程序状态(包括仓库的当前库存).该不确定性所导致的最大问题是程序执行过程的不可预测和不可重复[3].

由于上述原因,通常的测试方法被称为不确定性测试——为增加可靠性,同一个输入(测试用例)需要执行多次.另外,还要增加确定性测试,包括干预程序的执行、修改有关的竞争条件等,以使程序按照某一事先规定的路径执行,从而增加测试的充分性.

由于不确定性的存在,测试分布式程序不能仅考虑输入和对应的输出.可以定义同步事件[3],并采用事件序列来反映程序的运行过程以及分布进程间的交互.

## 2　一个面向分布式程序的测试系统框架 TFDS

测试一个程序 $P$ 应该包括两个方面.首先,从规约角度入手,分析程序的结构,确定测试方案,考虑代码的装备;然后,再选择测试用例运行和跟踪执行程序,比较中间结果,分析程序的质量.

按照这种模式,为在异构网络环境中对分布式程序进行有效的测试,我们设计了一个面向分布式程序的测试系统框架 TFDS(test system framework for distributed software system).

TFDS 按照测试流程来组织测试系统.如图 2 所示,系统分为两个部分:针对程序规约和源代码分析的静态部分与针对程序执行和事件序列监测的动态部分.它们建立在同构的通信平台上,从而在分布测试时各段测试代码和模块能够有效地相互协作.

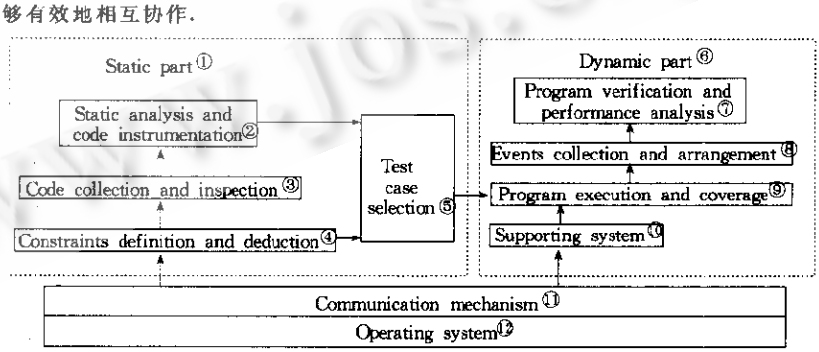


Fig. 2　The test framework TFDS for distributed program system
图2　分布式程序系统的测试框架TFDS

### 2.1 TFDS的静态部分

TFDS的静态部分由4个构件组成:规约定义与演算、代码收集和检查、结构分析和代码装备以及测试用例选择.下面分别加以介绍.

#### 2.1.1 规约定义与演算

规约定义与演算定义程序的验证准则.分布式程序的正确性包含两个方面:程序执行过程的正确性和结果状态的正确性.设执行过程用事件序列$s$表示,结果状态表示为$R$,两者组成一个二元组$(s,R)$,对分布式程序$P$,验证准则可表示为:对任意$X\in Domain(P)$,定义集合$Valid(P,X)$,它由所有对应于输入$X$有效的二元组$(s_i,R_i)$组成.对于复杂的分布式程序,定义$Valid$集往往不可行.可以针对事件序列中的同步事件定义事件约束,事件约束规定了在给定的输入$X$下各同步事件间在时间上或因果关系上的依赖以及这种依赖的或然性.可以采用CSPE(constraints on succeeding and preceding events)及其扩充[3]来描述事件约束.

对于分布式程序,事件的定义可以分成3个层次:

(1) 规约级.按照程序的规约可以定义同步事件,一般与程序中的功能函数相对应;

(2) 语言(程序)级.根据语言的同步设施及对应的语句定义同步事件;

(3) 实现级.进一步考虑底层的支撑系统,如通信机制和操作系统等来定义同步事件.

采用规约级事件可以对应于传统的黑箱测试;采用语言级事件可以对应于传统的白箱测试;采用实现级事件可以方便程序的调试,便于进行确定性测试.对于不同层次的事件,其验证准则的定义应有所不同.

#### 2.1.2 代码收集和检查

一个分布式程序由分布在多台机器上的程序段所构成,为进行有效的代码分析,应该将这些程序段汇集在一台机器上作整体分析以了解程序的分布结构.

为保证程序编写的正确性,在编程过程中应随时对程序段作词法和语法检查.编译器也可以完成这部分工作,但编译器只适于检查已初步完成的程序.当程序只是部分完成时,应有一个便利的测试模块对其进行部分检测.另外,当程序是由串行语言所编写时,内嵌的分布语法的正确性也需要由特定的测试构件来保证.不同的测试构件可以针对不同类型的分布式语言.

#### 2.1.3 结构分析和代码装备

首先对程序的分布和并发结构进行分析和判断,探查进程间循环调用错误和调用目标错误,确定程序复杂度和算法潜在的并行度等.随后,根据分析结果构筑程序状态图,如可达状态图和并发状态图,借以选择测试用例.

为了在分布式程序运行时能够产生和收集各种事件,需要对程序作一定的变换,例如,在程序的关键位置插入测试代码.对于不同层次的事件有不同的代码装备方式,如对实现级的事件需要底层支撑系统的配合才能产生,甚至可能要修改支撑系统.

#### 2.1.4 测试用例选择

对于分布式程序,测试用例可以有一般输入和事件序列两种形式.前者与串行程序等同,而后者可以用于确定性测试.选择测试用例可以有两种途径:从程序实现出发和从程序规约出发.从程序实现出发可以根据结构分析时得到的程序状态图来确定所需覆盖的程序路径.该路径应该涉及同步设施或并发结构.确定路径后可据此划分程序的输入域,进而选择测试用例.从程序规约出发要考虑规约的不同表示,例如,对于FSM(finite state machine)表示可以根据变迁遍历或状态遍历选择输入/输出序列.而基于CSPE约束表示则可以根据对约束的违反和覆盖准则等来选择测试用例.

在上述讨论的基础上,可以进一步根据规约和实现按照压力测试的原则选择测试用例,如按照超常的规模和频率申请系统资源,寻找输入数据的敏感组合等,以测试系统在异常情形下的健壮性.

### 2.2 TFDS的动态部分

选择好测试用例后需要通过动态执行分布式程序测试其正确性和性能.动态部分包括3个构件:程序执行和覆盖、运行事件收集和整理以及程序验证和性能分析.

2.2.1 程序执行和覆盖

测试分布式程序需要考虑程序运行的中间状态,而程序运行本身是不确定的,因此需要有不同于串行程序的测试方案.主要包括:

(1) 不确定性测试.不干涉程序的运行,但同一个测试用例一般要反复运行多次;

(2) 确定性测试.通过插入代码或增加环境约束人为干涉程序的运行,使其按照某一指定的路径执行,以达到充分性的要求;

(3) 可达性测试.这是上述两者的结合,首先通过不确定性测试获得若干事件序列,然后再按照确定性测试方案逐次试探,并产生新的事件序列.

2.2.2 运行事件收集和整理

在程序执行过程中由测试代码或支撑系统生成的各种事件需要经过收集和整理以形成完整的事件序列.可以有两种方式:

(1) 集中记录和收集.可以得到唯一的事件序列,但等于附加了一个集中控制器,限制程序的运行,影响程序的性能,并且程序的规模不能过大.

(2) 分布记录和收集.由产生相应事件的分布单元来记录.例如,由各进程来记录与自己相关的事件;或通过同步数据对象来记录,如生产者-消费者问题中的仓库对象.由于记录的是多个不完整的事件子序列,需要采用某种方法,如时间戳等,将其归并为一个或几个完整的事件序列.

2.2.3 程序验证和性能分析

一般根据验证准则判定被测程序的正确性.验证准则对任一输入 $X$ 规定了 $Valid(P,X)$,而收集到的信息代表一种可行性,即 $Feasible(P,X)$.判定程序正确性的标准是这两个集合的比较.在实践中可以定义事件约束来代表 $Valid(P,X)$,通过检测事件序列是否符合事件约束来测定程序的正确性.

在性能方面可以确定程序的关键事件,如 barrier 事件,然后根据相邻两个此类事件间的时间差来分析和比较程序的性能,包括同一规约不同程序实现间的比较和同一程序实现在不同平台上运行情况的比较等.在规模测试的前提下,还需要检查系统是否在超常的输入和压力下发生了异常等等.

为了帮助程序员检查和判断分布式程序的执行情况,很多测试工具[4~7]都以图形的方式向程序员展示程序的动态执行过程,包括 replay 方式和 animation 方式.

上述 TFDS 中的各模块在实现时只需遵从标准接口就可以按需要撤换某一部分的组成构件,从而提供更强的功能或支持不同的分布式语言和平台.

## 3 基于 TFDS 的测试系统原型 PSET*

根据 TFDS,我们重新设计并改进了原有的一个面向 NC++语言的测试系统 PSET(distributed program structure and event trace).

### 3.1 NC++语言

NC++是我们自行研制和设计的一个基于工作站网(NOW)的分布式程序设计语言.它具有和 Ada 语言类似的并发语言设施,如会合机制,但同时也增加了新的分布式语言结构,如进程组机制等.

NC++程序的运行模式如图 3 所示.其中"Master"是用于语言支撑的系统进程,负责各进程的创建、终止、调度和 I/O 重定向等,同时也作为会合机制的中介."Group Server"负责管理进程组,它通过会合机制代表组内进程向外界提供服务.

### 3.2 PSET* 中的事件定义

PSET* 主要以执行事件为基础验证分布式程序的执行情况.其事件定义在实现层,为一个六元组:$\langle T,S,D,E,I,O\rangle$.各项的涵义分别是:

(1) 类型(type).即事件的类型,例如,"EVT_USR_RPCSEND"代表进程间调用的一个发起事件.

(2) 源进程(source).事件的发起进程,如发起进程间调用的进程.由进程的 pid 表示,可以转换为进程名

(即对象名).

(3) 目标进程(destination).事件针对的目标进程,不同类型的事件涵义不同,如对"EVT_USR_RPCSEND"事件,它代表提供入口的被调用进程;而对于"EVT_USR_GJOIN"(进程申请加入进程组)事件,它代表管理目标进程组的"GroupServer".

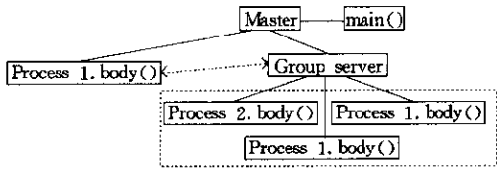


Fig. 3 The running mode of a NC++ program
图3 NC++程序的运行模型

(4) 调用入口号(entry).它只对与会合机制相关的事件有意义,代表被调用进程或进程组定义的一个入口,一般对应于一个对象方法.

(5) 时间戳(timestamp).这是指事件的发生时间,用于对事件的排序以及性能分析.

(6) 其他(other).同事件有关的其他信息,如产生事件的相关语句.

一个完整的事件表示可能是(EVT_USR_RPCSEND,262159,524304,1,756266,"B.putChar(C)").

### 3.3 PSET*的运行结构

这里列出动态部分的实现结构,如图4所示.

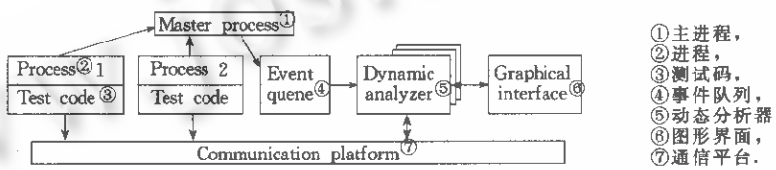


Fig. 4 The architecture of the dynamic part of PSET
图4 PSET动态部分的实现模式

测试系统按"Client/Server"模式被分为两部分:动态分析器和图形界面.动态分析器完成动态测试的主要功能,它对应于TFDS动态部分的两个模块:运行事件的收集和整理以及程序验证和性能分析.图形界面处于启动测试系统的主控台上,提供测试系统用户界面,接收用户指令和反馈测试结果,包括事件表的演示.

图4中所示的是由"Master"进程集中收集执行事件的模式."Master"进程负责监控程序的执行.各分布进程产生的执行事件被"Master"进程集中放置在事件队列中,分析器通过事件序列来分析和判断程序的执行情况,并将结果通过用户界面提交给测试用户.多个动态分析器可以服务于同一个图形界面,即一次可同时测试多个NC++程序.

当分散收集执行事件时,动态分析器中将嵌入一个模块,它向上同原有的构件接口,提供排序好的事件序列,向下接收从多个队列转来的事件集,并按时间戳等将其排序.分析器的其他部分无需改动.

### 3.4 实例研究

为了判断PSET*的测试能力,我们用NC++语言实现了一些小型的分布式程序,并采用PSET*对它们进行了测试.其一是生产者-消费者问题,其中生产者、消费者和仓库都表示为进程对象,以进程间调用(会合机制)的形式完成相互间的协作.

在正常情况下,单个进程间的一次会合按顺序涉及如下事件:

(1) EVT_USR_RPCSEND:调用进程发起RPC调用;

(2) EVT_SYS_TRANSRPC:系统进程"Master"传递这一调用;

(3) EVT_USR_RCVRPC:该调用申请进入被调用进程的入口等待队列;

(4) EVT_USR_DORPC:被调用进程开始执行对应的方法;

(5) EVT_USR_RPCRESP:被调用进程返回调用结果(同步调用);

(6) EVT_USR_RPCGETR:调用进程收到调用结果(同步调用).

这些事件呈匹配关系.其中,异步调用(调用进程发出调用信息后立即返回)只需前4个事件;后两个事件专为同步调用而设,调用结果返回时不需要"Master"进程作中介,这是因为被调进程已经知道调用进程的确切位置.

验证程序的正确性首先需要事件的收集,在事件收集之前需要代码装备,NC++预编译系统可以完成此项功能.对于生产者-消费者问题,可以写出如下的命令行:

nc-t cont.nc

其中 cont.nc 为程序名,它在所生成的目标码中自动插入了测试代码.

另一个例子是一个打印机组管理程序"print.nc"[8],其间涉及进程与进程组之间的调用问题,即一个进程调用一个进程组的入口.此时,被调用对象是一个进程组,由组内进程执行对应的方法.对于这种情况,PSET* 增加了以下几个事件以代替原有的事件:

(1) EVT_USR_GGETCALL:"Group Server"接收到调用申请,并把它置入入口调用等待队列.由于"Group Server"由用户定义,因此与之相关的仍是用户事件("USR"表示).它代替上述的"EVT_USR_RCVRPC".

(2) EVT_USR_GSNDCALL:"Group Server"将调用申请发往指定的组内成员.

(3) EVT_USR_MGETCALL:某组内成员获得该调用申请并执行对应方法.它代替上述的"EVT_USR_DORPC".

对于同步调用,执行结果仍由组内成员直接返回给发起调用的进程.

除正常情况的事件外,PSET* 还定义了一些对应于异常情况的事件,如

(1) EVT_SYS_NODST:调用目标找不到,可能对应进程已经终止.

(2) EVT_SYS_NOENTRY:目标进程没有给定的调用入口.由于编译时可以发现入口不匹配的错误,出现这一事件即表示系统发生故障.

(3) EVT_SYS_QOVERFLOW:目标进程的入口调用等待队列溢出.

上述事件是针对进程间调用的,其他类型的事件还包括:与进程创建和终止相关的事件,如 EVT_SYS_PROCREATE 和 EVT_USR_PROBEGIN;与进程 I/O 相关的事件,如 EVT_USR_OUTPUT 和 EVT_SYS_OUTREDIRECT,以及与多目通信相关的事件等.

在 PSET* 中,根据收集的事件序列判断程序的正确性可以从如下几方面入手:(1) 查看是否出现了异常事件;(2) 观察事件间的匹配情况;(3) 将多个紧密关联的事件组合在一起,构成规约级的事件,如上述几个事件可以归并成一个调用事件,这样就可以使用针对规约的验证准则来判定程序的正确性,如采用 CSPE 约束描述等.

对上述的打印机组管理问题,由于实现时组管理者将打印任务派发给组内进程后未能及时将该任务从打印队列清除,致使一个空闲的打印机进程在主动申请时又重复得到了这一任务.该错误在事件匹配时即被 PSET* 检测出来.

在进行性能分析时,可以根据实现级的事件计算各进程的通信时间、调用等待时间和执行时间等,再配合事件表的演示[8]可以很方便地发现程序的瓶颈,以选择更好的实现算法.

### 3.5 与相关测试工具的比较

通过 TFDS,PSET* 在原有的 PSET[8] 的基础上得到了进一步的改进.首先,PSET 没有采用构件方式实现,缺乏必要的系统结构,测试功能难以扩充.而 PSET* 的功能却有明显的增强,如增加了在线监控和时钟调节,测试准确性得到了提高.其次,PSET 同语言的联系过于紧密,难以分清各自的界限,如解析器同编译器间的关系就不好处理,容易出现错误.而 PSET* 的实现更加容易,原有的一些设计问题,如怎样同预编译器合作,得到了良好的解决,使得测试系统结构更加清楚,降低了测试系统本身的错误率.

与其他测试工具[4~7]相比,按照 TFDS 实现的分布式程序测试系统具有如下两个特点:

(1) 有系统地实现面向分布式程序的测试工具,在功能方面比较全面.构件实现模式保证测试系统易于更新和重构,从而测试不同类型的分布式程序.

(2) 其他测试工具可以纳入到 TFDS 的范畴.如"N_MAP"可以按照静态部分的模式实现;"orcshot"和"GDDT"则可归结于动态测试;"QBVT"也是一种动态测试工具,它通过监控进程的在线演示来测试运行中的分布式程序.通过构件替换和重组,测试系统可以实现上述工具的功能.

## 4 结 论

由于并发的存在,分布式程序在运行时具有不确定性.在测试一个分布式程序时需要同时考虑程序执行过程的正确性以及运行结果(结束状态)的正确性.因此,在传统测试方法的基础上,对分布式程序需要增加确定性测试以保证测试的充分性和可靠性.在进行确定性测试的过程中,测试工具必须监控分布进程的运行,必须面对底层的支撑平台.这些决定了分布式程序测试工具必须具有一定的系统结构.

本文介绍了一个面向分布式程序的测试系统框架TFDS,它包含了测试分布式程序的全过程,包括规约的定义与演算、代码的收集和检查、结构分析和代码装备、程序的执行和覆盖、运行事件的收集和整理以及程序验证和性能分析.面向NC++程序的测试系统PSET*是它的一个原型实现,它可以有效地帮助程序员检测分布式程序结构上和执行中的一些错误和性能缺陷.下一步的工作是,如何按照TFDS使PSET*的功能得到进一步的改善和扩充.

## 参考文献


1 Tanenbaum A S. Distributed Operating Systems. Upper Saddle River, NJ: Prentice Hall, Inc., 1996

2 Jorgensen P C. Software Testing——A Craftsman's Approach. Boca Raton, Florida: CRC Press, 1995

3 Carver R H, Tai Kou-chung. Use of sequencing constraints for specification-based testing of concurrent programs. IEEE Transactions on Software Engineering, 1998,24(6):471~490

4 Fersche A, Johnson J. Performance prototyping of parallel applications in N_MAP. In: Benjamin Lian ed. Proceedings of the IEEE 2nd International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-96). Singapore: Inst. of Electrical and Electronics Engineers Inc., 1996. 84~92

5 Hofman R, Langendoen K, Bal H. Visualizing high_level communication and synchronization. In: Benjamin Lian ed. Proceedings of the IEEE 2nd International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-96). Singapore: Inst. of Electrical and Electronics Engineers Inc., 1996. 37~43

6 Koppler R, Grabner S, Volkert J. Graphical support for data distribution in SPMD parallelization environments. In: Benjamin Lian ed. Proceedings of the IEEE 2nd International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-96). Singapore: Inst. of Electrical and Electronics Engineers Inc., 1996. 21~28

7 Hart D, Kraemer E, Roman Gruia-Catalin. Interactive visual exploration of distributed computations. In: Los Alamitds ed. Proceedings of the 11th International Parallel Processing Symposium (IPPS'97). Los Alamitors, CA: IEEE Computer Society Press, 1997. 121~128

8 Gu Qing, Chen Dao-xu, Xie Li. A validation system for object oriented distributed programming language named NC++. Journal of Software, 1997,8(supplement):352~356
(顾庆,陈道蓄,谢立.PSET:一个面向分布式语言NC++的测试原型.软件学报,1997,8(增刊):352~356)


## A Test System Framework for Distributed Programming


GU Qing CHEN Dao-xu HAN Jie XIE Li SUN Zhong-xiu

(*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)



**Abstract** This paper puts forward a test system framework TFDS (test system framework for distributed software system) for the testing of distributed programs, and as an illustration, introduces its prototype implementation PSET* (distributed program structure and event trace, revised version) on heterogeneous network platforms. The main functions of TFDS are for unit testing and integration testing. And the framework is divided into two parts: one for specification design and source code analysis, which works statically; and the other for program execution and event sequence manipulation, which works dynamically. Along with TFDS, and by a component-based architecture, the functionality of the PSET* can be easily improved and reinforced.



**Key words** Distributed system, test system framework, non-determinism, event sequence, distributed programming language.