回归测试自动化工具研究*)

马雪英[1,2] 姚 砺[1] 叶澄清[1]

(浙江大学计算机科学系 杭州310027)[1] (浙江财经学院信息管理系 杭州310012)[2]

摘 要 回归测试是软件测试生命周期中非常重要但十分费时费力的阶段,我们通过研究事件捕获/回放技术,实现回归测试自动化;通过优化测试用例,降低测试费用。本文介绍了基于事件源识别的捕获/回放技术和基于事件跟踪的回放同步技术,以及测试用例优化算法。最后简单介绍了所开发的面向 Java 的事件捕获/回放工具-Panorama Player。测试自动化工具的使用,能简化测试过程,提高测试效率。

关键词 回归测试,事件捕获/回放,回放同步,测试自动化工具

Research on the Automated Testing Tool in the Regress Test

MA Xue-Ying[1,2] YAO Li[1] YE Cheng-Qing[1]

(Dept. of Computer Science and Engineering, Zhejiang University, Hangzhou 310027)[1]

(Dept. of Information Management, Zhejiang Institute of Finance & Economics, Hangzhou 310012)[2]

Abstract The regress test is an important but time-consuming stage in the software test lifecycle, we study the techniques of the event capture/playback in order to automate the regress test procedure, and lowdown the test costs by optimizing test case. First, we introduce the techniques of the event capture/playback based on event source recognition and playback synchronization based on event tracing, then we implement the algorithm of the test selection. Finally the Panorama Player, an event capture/playback tool of JAVA, is also introduced in this paper. By using the automated testing tool, We can simplify the test procedure and enhance the test efficiency.

Keywords Regress test, Event capture/playback, Playback synchronization, Automated testing tool

1 引言

回归测试是软件测试过程中的一个重要阶段。当代码修改、软件硬件平台变更或硬件配置改变后,都必须进行回归测试。作为软件生命周期的一个组成部分,回归测试在整个软件测试过程中占有很大的工作量比重。在渐进和快速迭代开发中,新版本的连续发布使回归测试进行得更加频繁,而在极端编程方法中,更是要求每天都进行若干次回归测试。而软件越是接近发行,已经进行过的测试数就越多,在此时每发现并改正一个错误就必须把以前做过的所有测试重做一遍,工作量就越大。因此,非常有必要通过选择正确的回归测试策略来改进回归测试的效率和有效性。可以通过两个途径来提高回归测试效率回归测试自动化和测试用例优化。

回归测试自动化就是测试工具能够自动选择测试用例进行回归测试。回归测试有两个很大的缺点:费时而烦琐。尤其在测试用户界面(GUI)时,回归测试变得很复杂,操作员也非常容易出错。但越来越多的应用程序需要和图形用户界面(GUI)一起工作。为了支持这些应用程序的测试,一个优秀的软件测试工具,必须提供捕获(Capture)用户操作(如击键,鼠标活动,等等)的能力和在代码被修改之后自动回放(Playback)用户操作的能力。

捕获/回放功能可以把用户在进行测试时的键盘和鼠标等输入操作记录下来,同时也把软件的响应记录下来,当对软件作了修改并重新运行这个测试时,就可以利用测试回放功能把这个测试以前所作的输入操作重新应用到本次测试中,并自动比较软件对本次测试和以前的测试的响应是否相同,如果不同,就表明对软件的修改产生了新的错误。

测试用例优化,就是在整组测试用例中选择用例进行回归测试,所选择的测试用例集覆盖度尽可能地大,而所花费的代价尽可能地少。这就是测试用例选择问题。图1表示了回归测试自动化工具的工作流程。

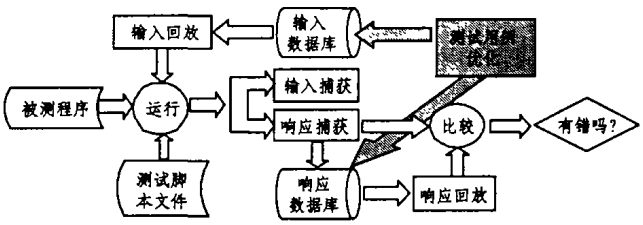

图1 捕获/回放的自动化流程

2 捕获和回放

从图1可以看出,设计回归测试自动化工具,主要是解决测试用例的输入/响应捕获,输入/响应回放。其关键技术包括如何设计类装载器、基于事件源识别的捕获技术、基于事件源识别的回放技术以及基于事件跟踪的回放同步。

2.1 设计类装载器

*)本课题得到国家自然科学基金(项目编号60073027)和浙江省教育厅科研基金(项目编号119034031)资助。马雪英 博士研究生,主要研究方向为软件测试,数据库设计,网络安全。姚 砺 博士,主要研究方向为软件测试,计算机网络。叶澄清 教授,博士生导师,主要研究方向为软件测试、高性能及智能机系统、多机并行处理系统、多媒体计算机技术。

事件捕获/回放器是通过监视被测应用程序(Java 类)的系统事件队列来记录/回放被测程序的输入事件的[10],设计时将其放在与被测应用程序运行时的同一进程内,且必须能够在运行时动态地装载应用程序。

Java 作为一种极具动态性的语言,通过反射(Reflection)机制为我们提供了动态类载入机制。Java 中提供此功能的包是 java.lang.reflect。其中 java.lang.Class 类封装了动态装载类库的功能。以下是一段示例程序,简单地说明了利用 Class 类设计类装载器的过程。

```
public class MyClassLoader
{//该段程序从命令行接受需装载的类名,然后装载该类,并执行该类的 main 方法。
    public static void main(String [] args)throws Exception
    {
        Class LoadCls = Class.forName(args [0]); //从命令行参数中获取需要装载的类名
        Method mainMethod = getMain(LoadCls); //获取装载类的 main'方法
        mainMethod.invoke(null,null); //执行装载类的 main 方法
    }
    private static Method getMain(Class cls)throws Exception
    {
        Method [] methods = cls.getMethods(); //获得类的所有方法
        for (int i=0; i<methods.length; i++)
        {
            if (methods [i].getName().equals("main"))//如果当前方法是 main 方法
                return methods [i];
        }
        return null;
    }
}
```

2.2 基于事件源识别的事件捕获

Java 事件是由事件源产生的。事件源可以是 GUI 组件、Java Bean 或任何有生成事件能力的对象。

当操作环境响应用户的一个动作(如鼠标点击)从而生成一个事件时,Java 虚拟机中同操作环境通信的 AWT 部分会收到一个通知,AWT 便将它转化为一个 AWT 事件并添加进系统事件队列。此时,事件处于 AWT 的事件分派线程(dispatchEvent())的控制下。分派线程依次从系统事件队列中取出每个事件,送到生成该事件的组件的分发过程[11]。可见在用户输入事件(KeyEvent 和 MouseEvent)的生存周期内,一直都是存储在系统事件队列中的,而且是由 dispatchEvent()方法统一分派的。所以我们只要能重载 dispatchEvent()方法就可以获取系统的所有事件,包括用户输入事件。

Java 提供了 EventQueue 类来访问、操纵系统事件队列,该类中封装了对系统事件队列的各种操作。事件捕获的过程是:首先,从 EventQueue 类中派生一个新类;然后,用 EventQueue 类中的 push()方法,把当前的 EventQueue 类替换为派生类,此时所有的系统事件都会转发到我们的派生 EventQueue 类[12];接着,在派生类中重载 dispatchEvent()方法,截获所有的系统事件,包括用户输入事件;最后通过 ToolKit 类中定义的 getSystemEventQueue()方法来获得对 EventQueue 的引用。

2.3 基于事件源识别的事件回放

2.3.1 *回放定位* 传统的事件回放器对鼠标事件的回放是利用绝对屏幕位置方式,即在捕获鼠标事件时记录下此时鼠标点击处在屏幕上的绝对位置坐标,此后回放鼠标事件时便以记录下的屏幕绝对位置坐标作为鼠标回放时的位置。这样设计的好处是回放器设计简单,但前提是每次回放时对应的窗口/组件都必须处于相同的位置,而且窗口/组件对应于鼠标点击位置的部分必须可见,不能被隐藏或其他窗口遮蔽。

但实际上,由于应用程序窗口的运行受到窗口管理器的控制,并不能保证同一窗口/GUI 组件在多次运行中都出现于相同的位置。而且,即使同一窗口/GUI 组件在多次运行中每次出现于相同的位置,如果受其它应用程序或执行过程中某个条件失败的干扰,都会导致后续的回放失去目标。例如,其它应用程序突然弹出一个窗口,或者执行过程中打印机缺纸、申请的网页不存在等,都会导致目标回放的窗口/组件失位。

有鉴于此,我们设计的事件回放器通过事件源(窗口/GUI 组件)的标题来识别窗口,辅以相对屏幕位置管理。其设计原理如下:

每当捕获一个鼠标/键盘输入事件时,我们记录下响应该鼠标/键盘事件的窗口/组件的名称,以及鼠标相对于该窗口/组件的坐标位置以及其它一些基本信息。组件名称的命名规则是:容器1…….容器 *n*.组件类型.组件名称。

当回放用户输入事件时,通过直接操纵系统事件队列实现输入事件的回放。具体步骤是:

首先,通过记录下的窗口/组件的名称获得对应的窗口/组件的引用;然后,结合记录的其它鼠标/键盘事件信息,重构鼠标/键盘输入事件;最后,将重构的鼠标/键盘事件直接放入系统事件队列中,由分派线程执行后续的事件分派工作。

这样,我们就解决了由于窗口/组件的位置变动带来的回放定位问题。

2.3.2 *窗口组件的引用获得* 第二个需要解决的关键问题是如何能根据窗口/组件的名称获得其引用。这还是通过系统事件队列来实现的。因为 Java 程序在新建/删除一个容器时都会向系统事件队列发出一个 Containerevent 事件,其中包含了对该容器的引用,所以事件回放器在载入被测程序后便监视系统事件队列,截获所有的 ContainerEvent 事件。如果是新建容器事件,便从中获得新建 Container 的引用。因为所有的 Container 都实现了函数 getComponents(),该函数返回容器包含的所有其它组件或容器的引用,我们通过一个简单的递归便可以得到一个容器包含的所有组件或容器,我们将窗口/组件的引用和其名称保存在 Vector 向量中,这样在回放过程中,只需知道窗口/组件的名称,便可从 Vector 向量中找到对应窗口/组件的引用。

2.4 基于事件跟踪的回放同步

一个设计良好的捕获/回放测试工具除了能捕获并回放用户输入事件外,更重要的是还必须能同步回放过程,因为在回放过程中充满了各种会导致回放失败的变数。

同步回放过程的目的主要在于:

① 决定何时回放下一个用户输入事件;② 如果回放失败,必须能尽早察觉;③ 如果可能,应当在错误恢复后继续回放。

而现行的绝大多数捕获/回放工具,对回放事件的控制是依据捕获时输入事件之间的时间间隔。例如,如果连续两个输入事件在捕获时的时间间隔为5秒,那么在回放时,在前一个输入事件回放后等待5秒,然后回放第二个输入事件。

但是由于不同的机器运算速度(跨硬件平台测试)、运行环境(跨操作系统平台测试)对回放过程会造成影响,使得两次回放之间的时间间隔难以控制。即使在理想状况下我们可以不计这一因素,但是,还有一种情况是无法避免的。例如,我们测试网络程序,需要从 Web 服务器下载 Applet 运行。此

时，即使是在同一台机器上进行回放，由于网络速度的影响而无法控制其延时。更有甚者，回放时由于服务器忙或关闭而联接失败。这样，本来后续动作是点击下载的 Applet 的某个 GUI 组件，现在不论等待多久，回放的动作肯定错误。

所以，传统的基于时迟的回放技术已经无法适应网络环境下的软件测试。我们采用基于事件跟踪的回放同步技术，从以下四个方面来同步回放过程：

(1)根据记录下的连续两个输入事件间的时间间隔，但这只是回放的必要条件，还必须满足下列条件；

(2)因为连续两个输入事件间可能还有其它系统事件，例如，点击菜单后弹出一个窗体，此时就会产生 FocusEvent、WindowsEvent 等系统事件，因此在捕获输入事件时，需记录下连续两个输入事件间发生的其它系统事件，以便在回放一个输入事件时，保证在该输入事件前发生的其它系统事件都已按捕获时的顺序出现过。这样做还可以发现某些回放错误，例如，某个窗体/容器没有出现。

(3)在回放某一个输入事件时，必须保证前一个输入事件已经被事件源处理完毕。

因为事件分派线程对于系统事件队列中的事件是一个一个顺序处理的，前一个事件处理完了才分派系统事件队列中的下一个事件。因而可以采用监视系统事件队列为空的办法来判断前一个事件是否处理结束。但是这样一方面需占用大量 cpu 时间，另一方面，系统事件队列为空并不表示前一事件已处理结束，因为可能还正在处理中。

改进的方法是：事件回放器每放一个输入事件到系统事件队列，就随之放入一个自定义的事件，该事件源是事件回放器。这样，当事件分派线程处理完输入事件后，就会将自定义事件分派给事件回放器，从而事件回放器就能确认前一个输入事件已处理结束，如果此时又满足前两个条件，就可以继续回放下一个输入事件[Harper02]。

(4)在回放输入事件时，必须保证该事件的事件源存在。

因为我们设计的事件回放器是基于对事件源的识别来回放输入事件的，所以如果在回放过程中发现某个输入事件的事件源不存在，说明发生错误。此时：

①等待事件源的出现，然后继续回放；

②通过测试人员的介入来解决问题，或者继续回放，或者终止回放。

3 测试用例优化

3.1 遗传算法

回归测试是对改正后的程序重复以前作过的各种测试，以保证没有出现新的错误，同时再测定覆盖率，以保证能达到既定的覆盖率。通常情况下，我们原先设计的一组测试用例所获得的覆盖率很有可能只需其中的几个测试用例就可以获得了，那些冗余的测试用例在每一次重新测定覆盖率时都浪费了大量的时间和资源。测试用例选择目的就是要剔除冗余的测试用例，以提高测试效率。

测试用例选择问题是 NP-hard 问题。它可以形式化地描述成以下测试代价的最小化问题，即给定一个覆盖度的下限 K，找出满足该覆盖度的一个测试用例集合且代价最小[1]。

$$\min C(T)$$

$$\text{subject to } \mathrm{COV}(T) \geqslant K \qquad (1)$$

其中，T 是 TS 的真子集，TS 是一组已知的测试用例集。

给定覆盖度的下限 K 后，测试部门往往采用简单的贪心算法或者改进的贪心算法，在给定的用例集 TS 中选择出最优用例集 T。虽然贪心算法能以较快的速度得到优化解，但是难以得到最优解，所以总是设计启发式算法来求出问题的一个次优解。在实际应用中，设计启发式算法的指导原则应该是使这两部分时间之和最小，而不应该是单纯的求优化时间短或者是优化效果好。所以在选择用例集优化算法时，主要考虑使用尽可能短的优化时间得到理想的优化效果。

在这里，我们采用遗传算法进行测试用例选择。遗传算法(Genetic Algorithm，简称 GA)是由进化论和遗传学机理而产生的一种优化方法，在本质上是一种不依赖具体问题的直接搜索方法。

在遗传算法中，有一个包含个体 x_i 的群体 $P = \langle x_1, \cdots, x_n \rangle$，个体 x_i 代表问题的一个解，群体就是问题的一些解的集合。评价函数 $F(x_i)$ 被用来对这些候选解进行评价，目标是优化该评价函数(搜索该函数的最大值或最小值)以解决给定的问题。这些候选解通常用位串(bit string)的形式表示，借用生物学的术语称之为染色体(chromosome)。在算法的每次迭代中，评价函数按照优化标准对每个个体 x_i 进行度量，计算其适应度 f_i，适应度最高的个体被选择允许再生，以产生新的一代。下面是算法的形式化描述：

```
choose an intial population
determine the fitness of each individual
perform selection
repeat
    perform crossover
    perform mutation
    determine the fitness of each individual
    perform selection
until some stopping criterion applies
```

遗传算法中的再生过程主要包括三个遗传算子：(1)选择；(2)交叉；(3)变异。在选择过程中，适应度高的个体被直接复制到下一代群体中。适应度越高的串，产生后代的概率就越高。在交叉过程中，两个串的部分位(称为基因)进行交换从而产生一个新串作为下一代的个体。变异用来随机地改变染色体的部分基因。交叉和变异的使用都有一定的概率，分别称为交叉概率和变异概率。

对于问题(1)，我们按如下设计遗传算法来求解问题的最优或次优解：

染色体编码：对于此问题，编码方案非常直接，即 x 相量，x 是 n 位相量的二进制表示，n 是被测程序段的个数，每一位代表了它所对应的用例选中与否，1表示选中，0表示未选中[8]。

选择算子：采用通常的选择算子，即适应度最高的个体被直接复制到下一代群体中。

交叉算子：采用简单的单点交叉。

变异算子：变异算子可以遵循这样的原则：染色体中的每一位有 $1/L$ 的变异概率(L 为染色体中位的数量)[4]。

适应度函数：适应度函数 $f(x)$ 取决于 x 对应的测试集的费用以及覆盖率。由于规划问题的要求，x 必须首先满足最低覆盖率的要求，同时要求测试费用最低，因此构造下列特性的适应度函数 $f(x)$，使其：

在 x 达到最低覆盖率的前提下，费用越低，$f(x)$ 值越大；

在 x 不达到最低覆盖率的情况下，给 $f(x)$ 加上一个惩罚，以降低它的适应度函数。本文采用的适应度函数为：

```
if Req(x)>=MiniReq then
    fitness(x)=1/Cost(x)
else
    fitness(x)=0
```

其中，$Req(x)$是x满足覆盖率的程度，即x对应的w^Tz。$Cost$是x的花费，即c^Tx。本文实现的遗传算法的几个重要参数如下：

交叉概率：0.8

变异概率：$1/n$，n为染色体编码长度

种群规模：50左右

终止准则：进化最大150代

3.2 实验结果

实验结果表明：得到优化解的时间：贪心法<遗传法，两种算法运行时间都小于0.1s，且随规模增长呈平稳缓慢增长趋势，没有指数爆炸问题（见图1）；优化效果：穷举法>遗传法>贪心法，而且遗传算法非常接近最优解，优化效果明显好于贪心算法（见图2）。图1中纵坐标 Tim 表示得到优化解的时间，图2中的 Time 表示优化后的测试用例集的总运行时间，横坐标均为原始测试用例集的大小。

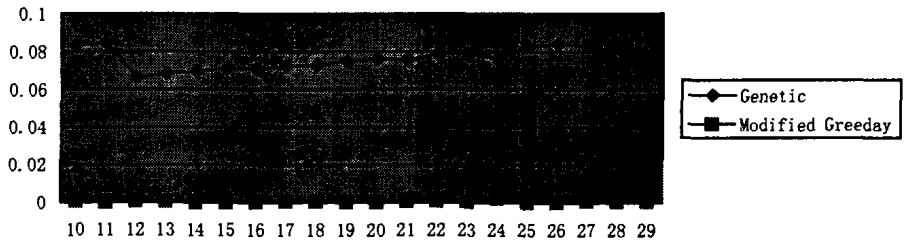

图1

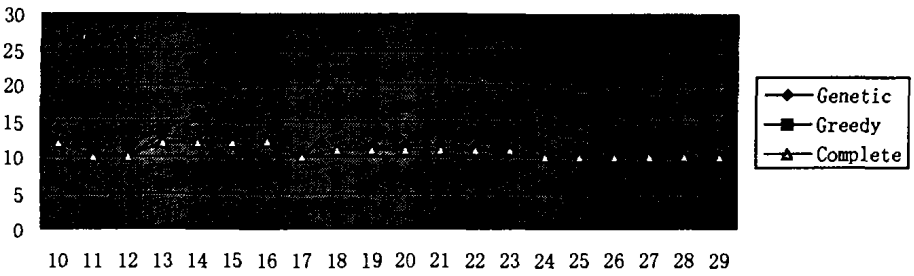

图2

4 Panorama Player

Panorama Player 是用来捕获/回放 Java 应用程序的输入事件的工具，它包括了事件记录器、事件回放器以及用例优化器。

事件记录器用类装载器（clasloader）来装入被测的 Java 应用程序，然后，通过监视应用程序的系统事件队列，记录下所有的用户输入事件（包括 KeyEvents 和 MouseEvents）和其它系统低级事件（包括 ComponentEvent、FocusEvent、WindowsEvent 和 ContainerEvent 等），并按事件产生的先后顺序将事件的相关信息存储到事件记录文本文件中。

用例优化器在事件回放之前，利用优化算法对事件记录数据库中的用例进行选择，剔除冗余的用例，优化回归测试用例集。

事件回放器用类装载器来装入被测的 Java 应用程序。然后，事件生成器从事件记录文件中按事件被记录下的顺序依次读出所有的事件。对于输入事件，先根据事件记录文件中保存的相关信息还原为原先的鼠标或键盘事件。最后，将生成的输入事件发送给对应组件。

结论 本文对回归测试自动化工具中的关键技术进行了研究，包括基于事件源识别的捕获/回放技术、基于事件跟踪的回放同步技术以及基于遗传算法的测试用例优化技术，目的是自动化回归测试过程，简化测试人员的测试复杂度，降低测试费用。面向 JAVA 的事件捕获/回放工具 Panaroma Player 的使用，必然会大幅度提高测试效率。

参 考 文 献

1 Csondes T, Kotnyek B. Greedy Algorithm for the Test Selection Problem in Protocol Conformance Testing. Journal of Circuit, System and Computers, World Scientific Publishing Company, 2002, 11(3): 273～281

2 International Software Automation Inc. Panorama-2 C/C++ User's Manual. International Software Automation Inc, 1993

3 Vilela P R S, Maldonado J C, Jino M. Program Graph Visualization. Software Practice and Experience, 1997, 27 (11)

4 Bäck T. Optimal mutation rates in genetic search. In: Proc. of the 5th Intl. Conf. on Genetic Algorithms (ICGA(93), Morgan Kaufmann, 1993. 2～9

5 Mitchell M. An Introduction to Genetic Algorithms. MIT Press, 1996

6 Beizer B. Software Testing Techniques [M]. New York: Van Nostrand Reinhold Company, 1990

7 Csödes T, kotnyek B. Automated test case selection based on subpurposes. Testing of Communication System Methods and Applications. In: GY, Csopaki, S. Dibuz, and K. Tarney, eds. Kluwer Acadmic publishers, 1999. 251～265

8 姚砺. 软件测试自动化工具：[博士学位论文]. 浙江大学，2002

9 孙霆. 软件测试工具的研究和实现：[硕士学位论文]. 浙江大学，1999

10 Horstman C S, Cornell G. 最新 Java2核心技术. 北京：机械工业出版社，2002

11 Palmer G. JAVA 事件处理指南. 北京：清华大学出版社，2002

12 Adair D. How to make an event log. Java Developer Connection. 2002