АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ЭЛЕМЕНТОВ ВЫСОКОПРОИЗВОДИТЕЛЬНОГО ПРОГРАММНОГО КОМПЛЕКСА

Б.Я. Штейнберг, Е.В. Алымова, А.П. Баглий, Р.И. Морылев, З.Я. Нис , В.В. Петренко, Р.Б. Штейнберг

Работа поддержана грантом ***ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2012г» ОКР 2008-04-2.4-15-02-003 (2008-2009) «Разработка высокопроизводительного программного комплекса для квантово-механических расчетов и моделирования наноразмерных атомно-молекулярных систем и комплексов».***

Введение.

Данная работа посвящена методике тестирования корректности элементов высокопроизводительных программных комплексов. Для автоматизации тестирования предложены специальные инструментальные программные средства.

К настоящему моменту известен [2] ряд средств автоматизации тестирования: HP (QuickTest Professional, WinRunner), IBM (Robot, Functional Tester), Borland (SilkTest) и AutomatedQA (TestComplete), представляющие собой средства автоматизации регрессионного тестирования, тестирования графических интерфейсов и Web-приложений.

Работа [3] посвящена вопросам разработки требований к крупным системам и комплексам программ, а также тестированию и реализации требований к крупным комплексам программ. Значительное внимание уделено верификации, трассированию и обеспечению баланса требований к крупным комплексам программ в условиях ограниченных ресурсов. Рассмотрены стратегии, планирование и затраты ресурсов на тестирование комплексов, а также инспекции и критические просмотры реализации требований к архитектуре.

В работе [1] излагается метод автоматической генерации набора тестов для графического интерфейса пользователя, моделируемого детерминированным конечным автоматом с помощью UML диаграмм действий. Метод заключается в построении обхода графа состояний системы с применением экономичного алгоритма обхода и компиляции построенного обхода в тестовый набор.

В Windows Vista реализована библиотека UI Automation, которую можно использовать для тестирования приложений Win32, приложений .NET Windows Forms, а также приложений WPF на компьютерах, использующих операционные системы с поддержкой .NET Framework 3.0. В работе [4] предлагается подход к самостоятельной реализации автоматического тестирования графического интерфейса средствами UI Automation.

Высокопроизводительный программный комплекс состоит из набора пакетов вычислительных модулей и интеллектуального интерфейса. К особенностям высокопроизводительного комплекса можно отнести параллельность вычислительных модулей и использование языка C++ для разработки некоторых модулей. Эти особенности оказывают влияние на тестирование.

Разработана инструментальная система тестирования, которая содержит генератор случайных входных данных, подсистему исполнения программ на этих данных, подсистему сравнения результатов и вывод результатов тестирования. Эта система позволяет тестировать вычислительные модули по принципу «черного ящика» [5]. Для тестирования по принципу «белого ящика» [5] используются дополнительные средства, позволяющие отслеживать прохождение тестов по различным участкам программы и, этим самым контролировать покрытие всех передач управления в программе (критерий полноты тестирования). Инструментальная система может быть использована в следующих ситуациях:

- для исследуемой программы имеется тест, представляющий пару <набор входных данных, эталонный набор выходных данных>;
- имеется эталонная программа (например, для параллельной программы имеется аналогичная ей по функциональности последовательная);
- исполнение одной и той же программы на разных вычислительных системах (например, параллельная программа на кластере из 4 узлов и на кластере из 10 узлов для тестирования масштабируемости и переносимости).

Разработана инструментальная программа для тестирования человеко-машинного интерфейса. Эта программа перебирает различные варианты использования (use-case) интерфейса программного комплекса.

Рассмотрены методики тестирования вычислительных модулей программного комплекса, связанные с некоторыми особенностями их разработки.

Автоматизация тестирования по принципу «черного ящика».

Рассматривается инструментальное программное средство, предназначенное для автоматизации тестирования методом черного ящика вычислительных модулей, входящих в состав высокопроизводительного

программного комплекса. Под тестированием «черного ящика» подразумевается отсутствие доступа к исходному коду тестируемой программы, а значит и отсутствие представления об её внутреннем устройстве.

Работа системы тестирования заключается в последовательном запуске тестируемой программы с различными входными данными и на различных конфигурациях вычислительной системы. По итогам каждого запуска автоматически производится оценка успешности теста. По результатам выполнения всех тестов формируется файл отчета.

Входные файлы с данными для программ генерируются автоматически по их описанию. Для генерации данных используется генератор псевдослучайных чисел из стандартной библиотеки C.

В рамках данной системы тестирования принимаются следующие ограничения на тестируемую программу:

- Программа не является интерактивной, т.е. не содержит графического интерфейса пользователя и не считывает данные из консоли.
- Входные данные считываются из заранее определенных файлов.
- Выходные данные сохраняются в заранее определенные файлы.
- Файлы с входными и выходными данными имеют фиксированный формат.
- В случае невыполнения данных ограничений корректная работа системы тестирования не гарантируется.

Система тестирования выполнена в виде консольной программы и имеет следующие параметры командной строки:

- Путь к тестируемой программе.
- Файл конфигурации с описанием списка тестовых случаев.
- Имя списка тестовых случаев, который будет использован для выполнения тестирования.
- Таймаут завершения тестируемой программы в секундах.

В файле конфигурации тестирующей системы описывается следующая информация:

- Конфигурации запуска тестируемых программ.
- Форматы входных и выходных данных тестируемых программ.
- Тестовые случаи (test cases).

Под конфигурацией запуска понимается конфигурация вычислительной системы, на которой будет запускаться тестируемая программа.

Возможны следующие способы запуска программ:

1. Запуск программы на 1 процессоре.
2. Запуск программы на нескольких процессорах (с использованием OpenMP). Предоставляется возможность указания количества используемых процессоров.
3. Запуск программы на нескольких узлах кластера (с использованием MPI+OpenMP). Предоставляется возможность указания количества используемых узлов кластера.

Входные и выходные данные тестируемых программ задаются в специальном формате IODataDescription. Последовательности токенов описаний представляют собой описание строки файла с данными, а последовательность описания строк описывает весь файл целиком.

Это XML-описание можно создать как в обычном текстовом редакторе, так и в специально разработанном для данной системы редакторе IODataDesriptionEditor. Этот редактор позволяет рассматривать данные в файле как последовательность токенов, представленную в виде таблицы, в каждой ячейке которой можно указать описание того или иного данного.

Описание IODataDescription используется в библиотеке IODataTuner. С помощью этой библиотеки система тестирования получает следующие возможности:

- Генерация файлов со случайными входными данными по описанию.
- Проверка корректности (соответствия описанию) входных/выходных файлов с данными.
- Проверка эквивалентности двух файлов с выходными данными с точки зрения описания. Если, например, один файл содержит вещественное число 3.1415, а второй - 3.1414, и при этом в описании задана точность этого данного 0.001, то такие данные считаются равными, а файлы их содержащие – эквивалентными.

  Тест задается комбинацией:

  - конфигурации запуска;
  - набор входных данных;
  - условия успешности.

Тест считается непройденным, если хотя бы одно из его условий успешности не выполняется. Условия успешности делятся на общие и специальные. Общие условия проверяются вне зависимости от того, указаны ли они в описании теста или нет.

К общим условиям относятся:

- Программа должна вернуть нулевой код возврата.
- После завершения работы тестируемой программы выходные файлы должны быть записаны на диск.

На данный момент предусмотрены два вида специальных условий успешности теста:

1. Корректность выходных данных.
2. Эквивалентность выходных данных при запуске на разных конфигурациях.

В зависимости от специального условия меняется алгоритм выполнения теста.

В первом случае тест состоит из одного запуска тестируемой программы и проверки выходных данных на соответствие описанному формату.

Во втором случае производится несколько запусков тестовой программы с одними и теми же входными данными, но на различных конфигурациях. После каждого запуска выходные данные проверяются на корректность и сохраняются. В конце производится проверка эквивалентности выходных данных, полученных после запусков тестируемых программ на различных конфигурациях.

По результату выполнения всех требуемых тестов генерируется файл отчета.

Автоматизация тестирования по принципу «белого ящика».

Описанная выше программная система, предназначенная для автоматизации тестирования программных модулей методом «черного ящика», может быть расширена до инструментальной системы тестирования по принципу «белого ящика». Расширение основано на внутреннем представлении программ [7], использующемся в Открытой распараллеливающей системе (ОРС) [6], и на работе с управляющим графом программы. Это расширение разрабатывается для программ, написанных на языках Си и ФОРТРАН, поддерживаемых ОРС.

Управляющий граф программы [8, с.258] – ориентированный граф с одним источником и одним стоком, вершины которого – операторы, а дуги – передачи управления.

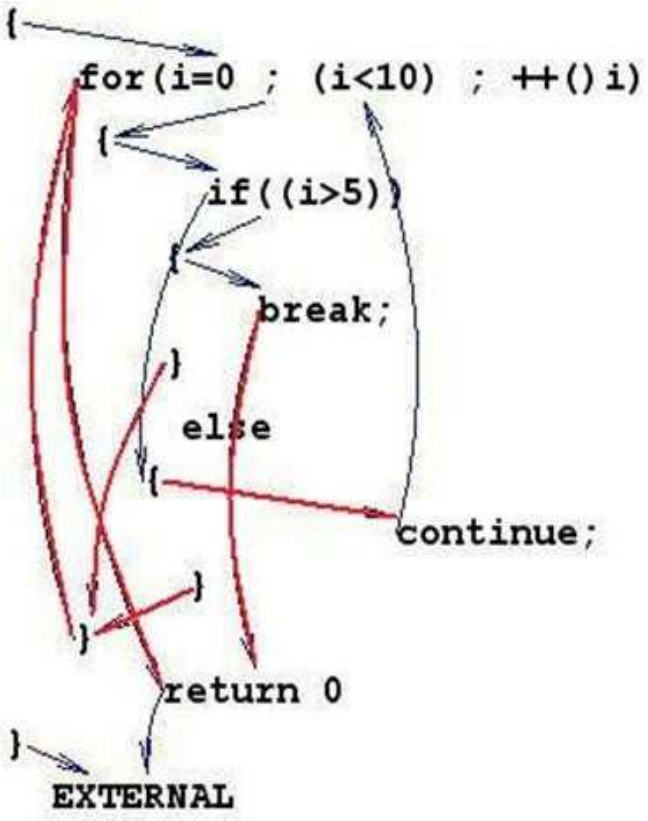

Рис. 1. Для управляющего графа фрагмента программы выделено множество контрольных дуг.

В данной работе в качестве критерия полноты тестирования используется покрытие передач управления – один из наиболее естественных и используемых критериев. Для использования этого критерия

требуется регистрировать прохождение программы по дугам управляющего графа. Для автоматизации такой регистрации в местах программы, соответствующих дугам управляющего графа, можно расставить специальные операторы – регистраторы. Добавление регистраторов в текст программы увеличивает время исполнения. Регистраторы достаточно расставлять не на всех дугах, а на некотором подмножестве. Возникает задача определения такого подмножества, которое содержало бы наименьшее количество дуг. Алгоритм нахождения такого множества описан в работе [9]. В рамках данного проекта этот алгоритм программно реализован.

Реализована подсистема, которая по программе, представленной во внутреннем представлении, и имени функции-регистратора строит управляющий граф, находит минимальное множество дуг, для каждой дуги позволяет вставить в исходную программу регистратор для нее. Регистратор должен быть реализован разработчиком тестируемой программы в виде функции, получающей единственный целочисленный аргумент (номер дуги) и сохраняющей при ее вызове информацию о том, что по дуге с данным номером произошла передача управления. Далее полученную программу можно вывести в виде исходного кода на одном из поддерживаемых языков и использовать для тестирования.

Инструментированную таким образом программу следует прогнать на тестовых наборах данных и проанализировать достигнутое покрытие, сведения о котором сохраняются при выполнении функции-регистратора. При неполном покрытии управляющего множества дуг графа тестируемой программы тестами следует расширить набор тестов, используя знания о уже достигнутом покрытии.

Тестирование человеко-компьютерного интерфейса.

Задачей тестирования графического интерфейса является обнаружение ошибок следующего характера:

1. необработанные исключения при взаимодействии с интерфейсом;
2. потеря или искажение данных, передаваемых через элементы интерфейса.

Тестовое воздействие формируется с помощью параметризации тестового сценария. Тестовый сценарий – это последовательность действий с управляющими элементами (нажатие кнопок, заполнение текстовых полей и т.п.). Параметрами тестового сценария являются возможные значения текстовых полей. В тестовые сценарии включаются контрольные точки, содержащие ожидаемый результат выполнения сценария. После прогона теста реальный результат сравнивается с ожидаемым, после чего делается заключение о наличии или отсутствии ошибки.

Набор тестовых воздействий формируется следующим образом: для каждого тестового сценария генерируется набор допустимых (позитивных) и недопустимых (негативных) значений. Таким образом, количество тестовых воздействий равно количеству тестовых сценариев, умноженных на два.

Полный перебор верных и неверных параметров для каждого элемента не происходит, потому что главной задачей тестирования является контроль передачи значений. Набор негативных тестов проверяет, все ли элементы интерфейсов снабжены контролем корректности вводимых данных.

Тестируемый интеллектуальный графический интерфейс условно делится на четыре части:

1) Элементы управления, отвечающие за авторизацию в системе.
2) Элементы управления, отвечающие за работу с проектами и задачами.
3) Стандартные диалоговые окна.
4) Мастер Экспертной системы.

Все части графического интерфейса кроме мастера Экспертной системы изучаются вручную, поскольку содержат небольшое число возможных комбинаций действий.

Мастер Экспертной системы изучается автоматически, так как представляет собой конечные последовательности страниц, содержащие различные элементы управления. Переход от страницы к странице осуществляется по кнопкам с заданными именами (например, назад или далее). Признаком последней страницы является кнопка с также заранее заданным именем (например, «Готово»). Каждый элемент на странице имеет набор состояний, перемножив их, получаем набор состояний страницы.

Тестовый сценарий строится по следующему шаблону:

- Перевести страницу в новое состояние (воздействуя на элементы методом полного перебора).
- Нажать кнопку «Далее», если она обнаружена, иначе нажать «Готово» и закончить процесс.
- Повторять пункты 1-2.

Описанная структура Экспертной системы дает однозначный критерий завершения генерации тестового сценария и перехода к генерации следующего сценария в наборе.

Набор тестовых сценариев генерируется с одновременным построением дерева состояния всей Экспертной системы по специально разработанному алгоритму. По построению существует только один путь от корня до каждого листа дерева состояний. Критерием покрытия для набора тестовых сценариев является посещение всех листов дерева интерфейса.

Для тестирования графического интерфейса разработан инструмент Macro Recorder, решающий следующие задачи:

– Изучение окна интерфейса;
– Запись действий с окном интерфейса, которые производит пользователь;

– Воспроизведение ранее записанных действий с проверкой корректности результата;
– Слежение за появлением необработанных исключений в процессе воспроизведения сценариев.

Тестирование специфических свойств комплекса.

Особенности параллельного программирования приводят к классу ошибок, неприсущих последовательному программированию.

Во-первых, это ошибки, связанные с некорректным использованием параллельных команд. Эта некорректность может быть связана с неверно организованной синхронизацией процессов или с некорректными обращениями к памяти при наличии информационных зависимостей в параллельно выполняемых процессах.

К параллельным программам предъявляется требование масштабируемости, то есть требование увеличения быстродействия при переносимости на новую подобную вычислительную систему с большим количеством вычислительных узлов. Тестирование масштабируемости состоит в сравнении выполнения параллельной программы на вычислительных системах с различными параллельными конфигурациями. Отличаться могут и количество ядер в процессорах, и количество процессоров в кластере, и коммуникационные системы и др.

Параллельные процессы выполняются недетерминировано, то есть неизвестно, какой процесс завершится раньше, а какой – позже. Может так случиться, что при одном наборе входных данных один процесс завершается раньше, а при другом, редко появляющемся наборе входных данных, – другой. В этом случае некорректно написанная синхронизация процессов может редко проявляться. Такую ошибку можно обнаружить, если искусственно увеличивать длительность некоторых процессов, оставляя неизменными длительности других.

Один из тестов для параллельных программ должен состоять в изменении длины используемых массивов и количества итераций распараллеливаемых циклов, сканирующих эти массивы. Например, при распараллеливании цикла с 1000 итераций на 4 узлах и на 8 узлах каждый узел получит для исполнения одинаковое количество итераций. Но при распараллеливании на 16 узлов картина изменится: 1000 не делится на 16 нацело. Управляет ли программа такой ситуацией автоматически или требует ручного вмешательства в исходный текст?

Использование языка C++ для разработки некоторых модулей может приводить к утечкам памяти. Эти ошибки могут не проявляться на быстро обрабатываемых входных данных. Но при долго обрабатываемых входных данных утечки памяти могут перейти критическую границу, после которой программа начнет работать неверно. Поэтому при тестировании написанных на C++ вычислительных модулей должен быть хотя бы один долго обрабатываемый тестовый набор. Если рассматривается тестирование «белого ящика» и осуществляется тестовое покрытие элементов программы (операторов, модулей…), то такие долго обрабатываемые тесты должны проходить через каждый элемент программы, способный вызывать утечки памяти, причем покрывать такой элемент надо много раз.

Тестирование утечек памяти можно производить полуавтоматическим способом. Удобно использование следующих шаблонов для поиска в исходных текстах программ мест выделения памяти:

Оператор new.
Оператор new[].
Функция malloc().
Далее обнаруживаются места освобождения памяти по следующим шаблонам:
Оператор delete.
Оператор delete[].
Функция free().

ЛИТЕРАТУРА:

1. Калинов А.Я., Косачёв А.С., Посыпкин М.А., Соколов А.А., Автоматическая генерация тестов для графического пользовательского интерфейса по UML диаграммам действий. Труды института Системного Программирования РАН, 2005г.
2. Можаев П. Средства автоматизированного тестирования // Открытые системы. 2009. №3
3. Липаев В.В. Тестирование крупных комплексов программ на соответствие требованиям. Учебник. М.: ИПЦ "Глобус", 2008г.
4. Dr. James McCaffrey. Test Run: The Microsoft UI Automation Library // MSDN Magazine: сетевой журн. 2008. URL: http://msdn.microsoft.com/en-us/magazine/cc163288.aspx (дата обращения: 31.05.2009)
5. Майерс Г. «Искусство тестирования программ» - М.: Финансы и статистика, 1982г.
6. Открытая распараллеливающая система. www.ops.rsu.ru
7. Петренко В. В. «Внутреннее представление REPRISE распараллеливающей системы» PACO'2008, Труды четвертой международной конференции «Параллельные вычисления и задачи управления», Москва: 27-29 октября 2008г.
8. Евстигнеев В. А., Касьянов В. Н. «Графы в программировании: обработка, визуализация и применение» - «БХВ-Петербург», 2003г.

9. Штейнберг Б. Я., Напрасникова М.В. «Минимальное множество контрольных дуг при тестировании программных модулей»//«Известия высших учебных заведений. Северо-Кавказский регион. Естественные науки» №4, 2003г., с.15-18.