САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Инструментальное средство автоматизации тестирования приложений EMC в среде IBM z/OS

Дипломная работа студента 545 группы

Котова Юрия Александровича

Научный руководитель	……	к. техн. н., доцент А.Ф. Губкин
Рецензент	……	старший преподаватель Н. Н. Вояковская
Допустить к защите, Зав. Кафедрой	……	д. ф.-м. н., профессор А. Н. Терехов

Санкт-Петербург
2012

SAINT PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Chair

Automation suite for testing EMC applications under IBM z/OS

Graduate paper by

Kotov Yury

Supervisor	……	assosiate professor A. F. Gubkin
Reviewer	……	associate teacher N. N. Voyakovskaya
“Approved by” Head of Department	……	professor A. N. Terekhov

Saint Petersburg
2012

Оглавление

1. Определения и сокращения.

ОС – операционная система.

z/OS – проприетарная серверная операционная система, разработанная компанией IBM для мейнфреймов собственного производства.

СХД – система хранения данных.

Symmetrix – производительная, надёжная и интеллектуальная корпоративная система хранения данных, продукт EMC.

TimeFinder – программный продукт EMC для управления локальной репликацией СХД Symmetrix с мейнфрейма.

SRDF – программный продукт EMC для управления удаленной репликацией СХД Symmetrix с мейнфрейма.

Mainframe Enablers – проект по созданию комплекса программных продуктов по управлению данными ОС z/OS, включающий в себя TimeFinder, SRDF и другие.

JCL – (Job Control Language) язык программирования, применяющийся в операционных системах мейнфреймов фирмы IBM.

TSO/E – (Time Sharing Option/Extension) подсистема разделения времени, реализует одновременную поддержку множества независимых параллельных пользовательских сеансов.

JES – (Job Entry Subsystem) подсистема управления заданиями.

2. Введение.

В настоящее время рынок ПО стремительно развивается, и необходимо постоянно поддерживать высокий уровень конкурентоспособности. Для обеспечения этого в первую очередь продукт должен быть качественным и развивающимся. Необходимо своевременно выпускать новые версии и патчи исправления ошибок, и при этом они должны быть полностью протестированы. Качество продукта можно определить как совокупную характеристику исследуемого ПО с учётом следующих составляющих: надёжность, сопровождаемость, практичность, эффективность, функциональность. Чтобы гарантировать каждую составляющую проводятся различные типы тестирования такие, как функциональное, нагрузочное, стресс-тестирование, тестирование безопасности и другие.

В данной работе рассматривается функциональное тестирование приложений EMC для управления локальной и удалённой репликацией. Репликация – механизм синхронизации содержимого нескольких копий объекта. В процессе репликации происходит копирование данных из одного источника на множество других и наоборот. Таким образом, при репликации изменения, сделанные в одной копии объекта, могут быть распространены в другие копии. Репликация называется локальной, если копирование происходит внутри одного хранилища, и удаленной, если операция проводится между несколькими различными хранилищами данных.

Данные приложения работают в операционной системе z/OS, предназначенной для продолжительной работы с большим количеством операций с высоким уровнем безопасности и устойчивости.

Функциональное тестирование – один из основных видов тестирования программного обеспечения, направленный на проверку реализуемости функциональных требований. При этом функциональное тестирование

обычно является обширным и очень трудоемким. Для сокращения времени тестирования общепринятым подходом является автоматизация [8]. Автоматизированное тестирование ПО – часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Автотесты – это программные модули, позволяющие проверить поведение приложения на соответствие требованиям или предоставляющие достаточно информации, чтобы осуществить подобную проверку (например, тесты на производительность могут лишь выдавать статистику, которую потом анализирует человек).

Основной целью данного диплома является создание инструментального средства автоматизации, которое позволит сократить время и повысить качество функционального тестирования приложений EMC.

3. Проект Mainframe Enablers.

Данный проект включает комплекс программных продуктов по управлению данными операционной системы IBM z/OS, серверной ОС для мейнфреймов. Symmetrix – это система хранения данных, которая может быть подключена к мейнфрейму [10]. Для сохранности данных Symmetrix использует технологии локальной и удалённой репликации. В состав Mainframe Enablers входят приложения по управления такими продуктами, как TimeFinder и SRDF.

Семейства программных продуктов EMC TimeFinder и SRDF являются наиболее мощными в отрасли пакетами программ для локальной и удаленной репликации. Эти средства позволяют создавать тома обеспечения непрерывности бизнеса (BCV) для операций с параллельной обработкой, включая резервное копирование, тестирование, разработку и локальное восстановление, а также создают удаленные реплицированные копии данных для защиты от аварий и простоев на основном узле. Семейства TimeFinder и SRDF фактически являются наиболее широко распространенным набором решений по локальной и удаленной репликации в отрасли. Копии этих продуктов установлены в десятках тысяч производственных сред по всему миру.

Программы комплекса Mainframe Enablers работают в ОС z/OS. Управляющие команды передаются продуктам по средствам JCL заданий [11]. Рассмотрим этот процесс с точки зрения системы.

Задание (job) – единица работы z/OS. Пользователь может запросить у системы выполнение какой-либо работы (конечно, связанной с запуском определенных приложений) с помощью специальным образом записанного и переданного системе текста. В тестировании используются пакетные задания (batch job). Они формируются пользователями на языке JCL и направляются на обработку по команде сеанса TSO SUBMIT. В задании описано, какие

программы, в какой последовательности и с какими данными будут исполнены, а также в какой форме и куда должны быть направлены результаты выполнения программ.

Задания, поступающие в систему, принимаются и обрабатываются специальным компонентом z/OS, который называется подсистемой управления заданиями JES (Job Entry Subsystem). JES принимает задания, регистрирует их, осуществляет анализ и формирует очереди заданий, а затем передает задания на выполнение базовой управляющей программе BCP. После завершения выполнения задания и получения результатов от BCP, JES формирует отчет по заданию (листинг), передает его пользователю или выводит на указанные устройства.

Подробнее об операционной системе z/OS можно почитать на официальном сайте компании IBM [12].

4. Тестирование.

Для тестирования продуктов используются пакетные задания, написанные на JCL. В общем виде тестовое задание выглядит как последовательность действий над дисковыми устройствами и включает команды продуктов Mainframe Enablers.

Постоянно поддерживается несколько версий программных продуктов и моделей Symmetrix, а также выходят новые версии. Для их тестирования приходится запускать одни и те же тестовые задания множество раз, изменяя либо подключаемые библиотеки и префиксы у команд, либо дисковые устройства, либо и то, и другое. Поскольку счёт тестов идёт на сотни, то делать изменения вручную – достаточно трудоёмкая задача. Возникает потребность в инструментальном средстве, которое позволит вносить эти изменения в одном месте и будет распространять их на все тесты.

Следующая проблема, о которой стоит сказать – это нехватка времени на полное тестирование в связи с тем, что тесты выполняются последовательно. Для тестирования отводится определённый набор ресурсов, задания могут использовать разные устройства из этого набора. Операционная система z/OS является мультипрограммной и позволяет запускать задания параллельно. Одновременное выполнение нескольких заданий сократит время тестирования, однако на практике тестировщику достаточно сложно отслеживать, какие устройства заняты, а какие свободны уже при двух параллельно запущенных заданиях, не говоря уже о том, что для запуска второго задания придётся вручную указывать устройства, что, в свою очередь, отнимает немало времени. В большинстве случаев при ручном тестировании в один момент времени выполняется один тест, и полное тестирование занимает много времени. От создаваемого средства потребуем автоматического отслеживания доступности ресурсов, и параллельного запуска тестов с использованием различных устройств.

Стоит добавить к требованиям поддержку зависимых тестов, когда дочерний тест должен быть запущен после завершения родительского или во время его выполнения. Ну и конечно, нужен отчет о проведённом тестировании: сколько тестов запущено, какие тесты и краткую информацию о неуспешных тестах.

5. Существующие технологии.

Был проведён анализ существующих технологий для решения обозначенных задач:

1. *Макрогенераторы.*

Макрогенератор – программа, выполняющая преобразование входного текста в выходной при помощи задаваемых ей правил замены последовательностей символов, называемых правилами макроподстановки. С помощью макросов можно из шаблонного теста генерировать тесты для различного окружения. Основным недостатком данного подхода является отсутствие целостной системы для запуска тестов. Так же нет поддержки зависимых тестов, параллелизма и сбора результатов.

2. Расширения эмуляторов терминала.

Дополнения к эмуляторам терминала такие, как TN3270 Plus [9], позволяют в автоматическом режиме воспроизводить действия пользователя. Они предназначены в первую очередь для тестирования интерфейсов, а не для запуска заданий. В случае с JCL заданиями потребуется большой объём подготовительных работ для каждого теста.

3. Hiperstation.

Крупный продукт от компании Compuware для автоматизированного тестирования мэйнфреймов. Hiperstation имеет множество дополнительной, не требующейся функциональности, не прост в настройке и использовании, не бесплатен. Детальную информацию об этом продукте можно посмотреть здесь [7].

3. JAT.

Утилита для настройки и тестирования приложений в ОС z/OS. Позволяет генерировать и запускать JCL задания. Данная утилита лучше остальных подходит для решения поставленных задач, но она не бесплатна. К минусам так же относится то, что она не позволяет запускать тесты на разных устройствах, нет параллельного выполнения, а вносить изменения в неё нет возможности. Подробная информация доступна здесь [6].

Поскольку целиком подходящего средства не нашлось, то было принято решение создавать инструментальное средство. Для этого определим задачу более формально.

6. Постановка задачи.

Задачами данной дипломной работы являются:

- Создание инструментального средства для запуска параметризованных тестовых заданий, включающее в себя:
  - поддержку зависимых тестов,
  - динамическое распределение ресурсов между тестами для их параллельного выполнения,
  - отчёт о проведённом тестировании;
- Внедрение данного инструмента в проект Mainframe Enablers.

7. Концепция.

Следующий рисунок показывает концепцию создаваемого приложения JTEST:

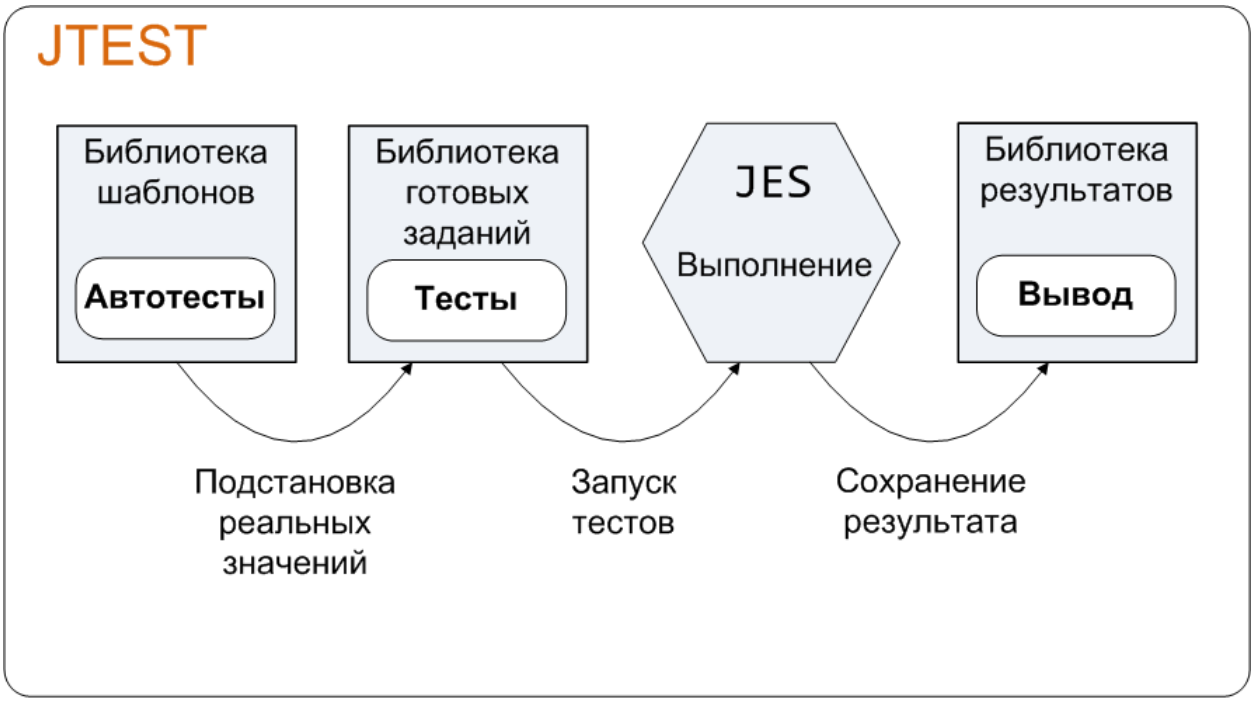

Рис. 1 Концепция инструментального средства JTEST.

Данный рисунок демонстрирует работу приложения JTEST с точки зрения пользователя (тестировщика). Шаблон теста, далее будем называть его автотест, представляет из себя JCL программу, содержащую переменные, при подстановке которых получается выполнимый тест. Автотесты хранятся в библиотеке шаблонов. Программа JTEST подставляет значения переменных, определённые тестировщиком. Полученные тесты будут сохранены в библиотеке готовых заданий. Затем они будут запущены, и по окончанию их листинг будет сохранён в библиотеке выводов.

С программной точки зрения инструмент JTEST будет рассмотрен далее.

8. Формат автоматизированного теста.

Автотест – это JCL программа, содержащая переменные. Переменные могут находиться в любом месте программы, включая комментарии.

При разработке формата переменных было принято во внимание то, что некоторые существующие тесты уже были локально параметризованы средствами JCL, поэтому для уменьшения затрат на перенос базовых тестов под автоматизацию было решено поддержать формат существующих JCL переменных. В общем виде этот формат можно описать так:

&<имя переменной>[.]

То есть, в начале идет знак амперсанда, затем имя переменной, а заканчивается символами: точка, запятая, круглые скобки или пробел. Только точка может входить в описание переменной, при подстановке значения переменной она будет заменена вместе с именем.

С другой стороны, была бы полезной возможность создавать переменные визуально различимые от JCL переменных. Для обеспечения этого был дополнительно введён формат переменных, начинающийся и заканчивающийся символом %. Итого, окончательный формат получился таким:

- Начальные символы переменной: & %
- Терминальные символы переменной: . %
- Символы означающие конец переменной: , ( ) пробел

Для большей гибкости было решено дать возможность определять и переопределять переменные внутри автотеста. Для их описания внутри автотеста используется JCL комментарий и специальный символ %. Получаем:

//*%<имя> = <значение>

9. Переменные и ресурсы.

Переменная определяется своим именем и имеет конкретное значение, которое будет подставлено в автотест. Глобальные переменные, значение которых распространяется на все тесты, задаются в блоке описания переменных и ресурсов (DD:SETVARS). Внутри автотеста можно задать локальную переменную, действие которой будет распространяться только на текущий тест.

Некоторые переменные допускают совместное использование, например, два теста могут параллельно выполняться и использовать одну и ту же библиотеку. Но есть переменные, не допускающие этого, например, физические устройства, будем называть их ресурсами.

Ресурсы – это переменные, не допускающие совместного использования. В один момент времени только одно задание может использовать конкретный ресурс. Исключение составляют зависимые задания, речь о которых пойдёт далее.

Зачастую, для тестирования имеется набор однотипных устройств, так что тесты могут использовать разные устройства из набора. JTEST позволяет задавать наборы ресурсов и выполнять тесты параллельно, динамически распределяя ресурсы между ними. Наборы ресурсов задаются с помощью директив, рассмотрим некоторые из них.

%RANGE

Задаёт диапазон значений для одной переменной. Менеджер ресурсов будет выдавать тесту, запросившему эту переменную, свободное значение, не занятое другим тестом.

Синтаксис:

<имя> = %RANGE(<начальное значение>-<конечное значение>) [опции]

Опции:

HEX – значения в шестнадцатеричном формате;

SOLID – диапазон рассматривается как единое целое. При запросе любого количества значений из диапазона будут резервироваться все значения из него.

Пример:

DEV = %RANGE(15A0-15AF) SOLID HEX

при этом переменные будут следующие:

&DEV2 – второе значение из диапазона = 15A1.

%DEV+1% – второе значение из диапазона = 15A1.

В процессе тестирования чаще всего приходится иметь дело с дисковыми устройствами. Каждое устройство имеет 3 параметра: номер диска в ОС, метку тома и Symmetrix номер. По отдельности они не могут быть ресурсами, поскольку относятся к одному устройству, которое не может быть совместно использовано. Соответственно, такой ресурс (устройство) должен иметь не просто переменную со значением, а значение для каждого параметра.

В ходе работы был разработан следующий формат. Ресурс-устройство задаётся суффиксом, включает в себя 3 переменные со значениями:

UCB<suffix> – номер диска в ОС;

VOL<suffix> – метка тома;

DEV<suffix> – Symmetrix номер диска.

Для задания таких ресурсов может быть использована следующая директива:

%DEVREQ

Директива %DEVREQ ищет последовательные устройства в конфигурационном файле. Файл содержит информацию обо всех устройствах внутри одного Symmetrix, их владельцах, параметрах и атрибутах (тип, размер, и т.д.).

Синтаксис:

<суффикс> = %DEVREQ(<СХД>,<владелец>,<кол-во>) [<атрибуты>] [опции]

Опции:

SOLID – устройства будут рассматриваться как единое целое. При запросе любого количества устройств будут резервироваться все устройства.

Пример:

STD = %DEVREQ(NTG,YKOTOV,16) 1 STD

В результате данного запроса будет создано 16 стандартных ресурсов-устройств на СХД Symmetrix NTG, владелец – YKOTOV, размер – 1. С каждым ресурсом будут ассоциированы 3 переменные: UCBSTD, VOLSTD и DEVSTD.

10. Управление запуском тестов.

Управление запуском тестов отвечает за то, какие тесты и в каком порядке будут выполняться. Пользователь может определять зависимости между тестами, выбирать набор тестов для тестирования и выделять начальные и терминальные тесты, которые будут запущены перед всеми остальными или после соответственно. Задаётся это с помощью специальных директив в DD карте DEPEND.

10.1 Зависимости между тестами.

Два теста называются зависимыми, если они связаны одним из двух следующих условий:

1. Зависимый тест должен быть запущен после окончания теста родителя;
2. Зависимый тест должен выполняться вместе с тестом родителем, то есть будет запущен сразу после запуска родительского.

Разрешается создавать цепочки зависимых тестов и множественные зависимости от одного теста, но без образования циклов. При этом зависимые тесты должны быть выполнены на одних и тех же ресурсах.

Синтаксис:

[зависимый тест] <- [родительский тест] : [опции]

Опции:

SUB – запустить задание сразу после запуска задания родителя.

При отсутствии опций зависимый тест будет запущен после окончания теста родителя.

10.2 Выбор тестов для тестирования.

При тестировании бывает полезно запустить на выполнение несколько конкретных тестов, а не все из библиотеки, как это происходит по умолчанию. Управлять этим можно с помощью директив %SKIP и %RUN.

Синтаксис:

{<тест> | *} : [%SKIP |%RUN]

%SKIP – не выполнять тест.

%RUN – выполнять тест, если он прежде был убран из выполнения.

* – применить действие ко всем тестам.

10.3 Начальные и терминальные тесты.

В процессе тестирование часто встречаются ситуации, когда один или несколько тестов должны быть запущены перед всеми основными тестами, для создания определённой конфигурации, такие тесты назовём начальными. Терминальными называются тесты, которые должны выполнится после окончания основных тестов.

Для определения порядка запуска начальных (терминальных) тестов разрешается задавать зависимости между ними, однако не допускается зависимость между начальным и не начальным тестами. То же самое можно сказать и про терминальные тесты. JTEST позволяет задать такие тесты с помощью директив %INIT и %TERM.

Синтаксис:

<тест> : [%INIT | %TERM]

%INIT – тест должен быть запущен перед всеми остальными тестами.

%TERM – тест должен быть запущен после всех остальных тестов.

11. Программа JTEST.

Запустить тесты на выполнение можно с помощью JCL программы, запускающей JTEST. Рассмотрим запуск на примере:

```
//JTESTSUB JOB (EMC),MFQA
//JTEST    EXEC PGM=JTEST,PARM='-KEEP'
//STEPLIB  DD DISP=SHR,DSN=YKOTOV.JTEST.LOADLIB
//SYSPRINT DD SYSOUT=*
//INITPDS  DD DISP=SHR,DSN=YKOTOV.JTEST.MYPROJ.MODEL
//CUSTPDS  DD DISP=SHR,DSN=YKOTOV.JTEST.MYPROJ.JCLLIB
//OUTPUT   DD DISP=SHR,DSN=YKOTOV.JTEST.MYPROJ.OUTPUT
//SETVARS  DD *
CCUU = %RANGE(6800-6809)
VER = 504
//DEPEND   DD *
* : %SKIP
JOBT0001 : %RUN
```

Программа использует следующие DD карты:

- `STEPLIB` – расположение исполняемого модуля JTEST;
- `SYSPRINT` – вывод сообщений программы JTEST;
- `INITPDS` – библиотека с автотестами для запуска;
- `CUSTPDS` – библиотека готовых к выполнению тестов;
- `OUTPUT` – библиотека выводов завершённых заданий;
- `SETVARS` – блок задания переменных и ресурсов;
- `DEPEND` – блок управления запуском тестов.

Программа JTEST может быть запущена со следующими параметрами:

- `OCUST` – не посылать задания на выполнение, произвести только подстановку значений в автотесты;
- `KEEP` – сохранять вывод программы в компоненте JES, по умолчанию выводы будут удалены после сохранения в DD OUTPUT;

- `TSO` – использовать подсистему TSO/E для соединения с компонентом JES, по умолчанию связь будет осуществляться по средствам FTP. Для использования TSO необходимо указать дополнительные DD карты, пример приведён в Приложении 1;
- `MAXACT <кол-во>` – указывает максимальное возможное количество параллельно запущенных тестов.

По окончанию работы программы в DD карте SYSPRINT будет выведен отчет о проведенном тестировании. Отчет содержит информацию о том, какие тесты были запущены, результат прохождения каждого теста (успех/неудача) и для неуспешно завершившихся тестов указан шаг, на котором произошла ошибка.

Приложение 1 содержит рабочий пример JCL программы по запуску двенадцати тестовых заданий. В приложении 2 представлен соответствующий отчёт о тестировании.

12. Архитектура JTEST.

В этой главе рассматриваются основные особенности программы JTEST.

12.1 Очереди тестов.

Вначале из шаблонной библиотеки выбираются тесты, которые должны быть выполнены. Затем они разделяются на 3 очереди: начальную, основную и терминальную, которые делят запуск тестов по времени на 3 фазы соответственно. Во время каждой фазы должны быть выполнены все тесты, относящиеся к ней, только после этого программа приступает к следующей фазе.

С точки зрения архитектуры между очередями нет разницы, они отличаются только по времени, поэтому будем рассматривать работу программы с одной очередью, далее она будет называться входная очередь тестов.

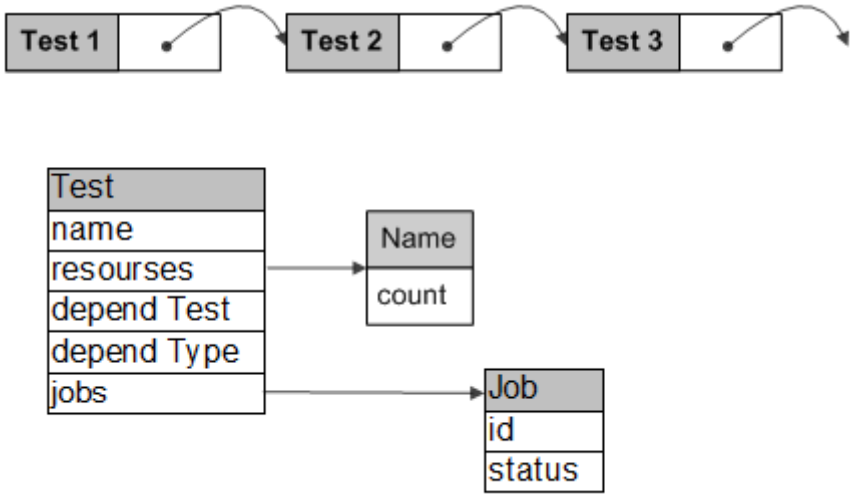

Рис 2. Входная очередь тестов.

На рисунке представлена структура сущности теста. Она содержит имя теста, список требуемых ресурсов, зависимость, если она есть, и её тип, а так же список запущенных заданий. Далее представлен процесс обработки

входной очереди тестов:

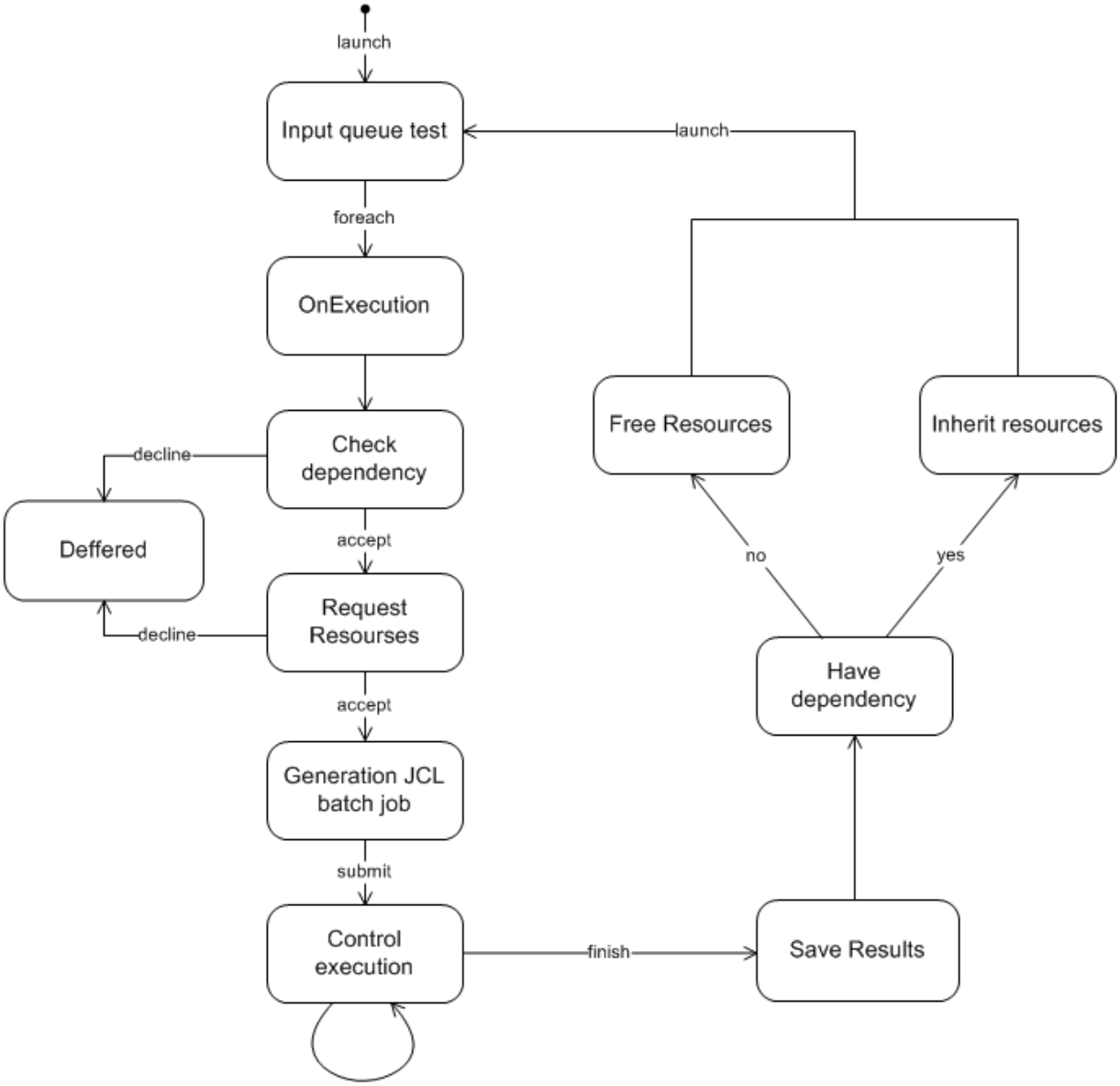

Рис 3. Обработка входной очереди тестов.

В начале работы программы или по определённому сигналу начинается обработка очереди. Каждый тест проверяется на возможность запуска, сначала по зависимости – был ли завершён тест родитель, то есть завершены все его задания, а затем по ресурсам – наличие и доступность. В случае возможности запуска происходит резервирование ресурсов для данного теста, и управление передаётся модулю генерации JCL, который подставляет реальные значения в автотест, в результате получается готовый к

выполнению тест. Он сохраняется в отдельную библиотеку и запускается на выполнение. При старте JCL программы происходит запуск одного или нескольких заданий (jobs), которые заносятся в структуру сущности теста. Модуль контроля выполнения тестов периодически проверяет состояние запущенных заданий и по завершению всех заданий теста отмечает тест как пройденный. Вывод пройденного теста сохраняется в библиотеке выводов. Далее для зависимого теста происходит передача ресурсов дочернему тесту, а для теста, который не имеет зависимостей, происходит освобождение ресурсов. После чего посылается сигнал на обработку входной очереди тестов.

12.2 Распределение ресурсов.

Программа JTEST оперирует несколькими различными наборами ресурсов, при этом каждый набор может быть рассмотрен независимо. Поэтому будем рассматривать задачу распределения одного набора ресурсов между тестами, для остальных будет использован аналогичный принцип распределения.

Тест должен получить все необходимые ресурсы до запуска, и ресурсы должны быть последовательными. Таким образом, задача распределения ресурсов аналогична задаче управления памяти в ОС с использованием только основной памяти, то есть без перемещения процессов между оперативной памятью и диском [1, 2].

Существуют два основных подхода к распределению ресурсов:

- Распределение фиксированными разделами;
- Распределение разделами переменной величины.

Примером первого подхода может служить задача распределения памяти в ОС MFT [3]. Для второго подхода можно привести пример распределения памяти в ОС MVT [4].

На практике тесты используют разное количество устройств, вплоть до

использования всех доступных. Получается, что использовать подход с разделами фиксированного размера возможно только, резервируя все ресурсы из набора для одного теста. Средство JTEST позволяет использовать ресурсы в таком режиме, исключая совместное использование. Для этого необходимо указать ключевое слово SOLID при задании набора ресурсов. По умолчанию используется другой подход с разделами переменной величины, позволяющий выполнять тесты параллельно.

Основной принцип работы по распределению ресурсов заключается в следующем. Сначала все ресурсы свободны, каждому вновь поступающему тесту выделяются необходимые ему ресурсы. Если ресурсов не достаточно, то тест не принимается на выполнение и стоит в очереди. После завершения теста ресурсы освобождаются, и их могут использовать другие тесты.

Менеджер ресурсов выполняет следующие задачи:

- ведение списка свободных и занятых ресурсов;
- при поступлении запроса на ресурсы от теста – анализ запроса, поиск и выбор свободного раздела нужного размера;
- блокировка выбранных ресурсов для теста;
- после завершения теста освобождение занятых ресурсов или передача их дочернему тесту.

Выбор раздела для вновь поступившего теста может осуществляться по разным правилам таким, например, как "первый попавшийся раздел достаточного размера", или "раздел, имеющий наименьший достаточный размер", или "раздел, имеющий наибольший достаточный размер". Программа JTEST выбирает первый попавшийся раздел достаточного размера, поскольку этот метод самый понятный для тестировщика. Ему будет проще всего понять, на каких устройствах будет запущен следующий тест и анализировать ситуацию в случае неуспешного теста.

13. Внедрение.

Вначале было автоматизировано несколько десятков тестов продукта TimeFinder, а после проведена демонстрационная презентация продукта JTEST для коллег. Далее было показано, как автотесты могут быть запущены на разных версиях ПО с использованием разных устройств. Для этого потребовалось изменить несколько строчек в блоке задания переменных, хотя ранее необходимо было вносить изменения в каждый тест. Так же было продемонстрировано параллельное выполнение тестов, использующих различные устройства.

После этого была написана пользовательская документация, которая вместе с программой доступна на внутреннем портале компании. Коллегам было предложено попробовать самостоятельно запустить программу, настроив её под себя. По их отзывам были внесены корректировки в интерфейс и добавлены некоторые директивы, например %DEVREQ, которая описывается в разделе 9.

Однако для полноценного использования средства в тестировании необходимо автоматизировать тесты, что занимает немало времени. Никто не готов был потратить часть своего времени, так как не было уверенности в положительном результате. Поэтому для демонстрации реальной пользы я автоматизировал в общей сложности более двухсот тестов продукта TimeFinder, что составляет более 75% одной его функциональности. При замере, сколько времени требуется тестировщику для ручного прохождения данных тестов и при использовании автоматизации, было получено, что в первом случае времени потребовалось в полтора раза больше. Первое использование автотестов оказалось достаточно успешным, поэтому было решено внедрить автоматизированное тестирование в рабочий процесс.

На данный момент инструментом пользуются 7 сотрудников из двух разных отделов. Для продукта TimeFinder автоматизировано более 400 тестов

(из 1500 общего числа тестов). Недавно началось использование JTEST в работе над продуктом SRDF, и сейчас автоматизировано около 100 тестов, что пока составляет малый процент (около 5%), но ведётся работа по дальнейшей автоматизации.

Созданное инструментальное средство достаточно универсально и позволяет производить тестирование практически любых приложений, вызываемых из JCL. Так JTEST уже используется для тестирования внутренних приложений EMC.

14. Результаты.

В результате данной дипломной работы:

- Создано инструментальное средство для запуска параметризованных тестовых заданий, включающее в себя:
  - поддержку зависимых тестов;
  - динамическое распределение ресурсов, позволяющее параллельно запускать тестовые задания. Время тестирования при параллельном выполнении тестов примерно в 2 раза меньше, чем при последовательном;
  - генерацию отчёта о проведённом тестировании, который содержит краткую информацию о запущенных тестах и более подробную о неуспешно завершившихся тестах;
- Данный инструмент был внедрен в проект Mainframe Enablers и активно используется при тестировании.

15. Список литературы.

[1] *Олифер Н. А., Олифер В. Г.* Сетевые операционные системы – СПб.: Питер, 2002. – 544 с.

[2] *Таненбаум Э., Вудхалл А.* Операционные системы. Разработка и реализация – СПб.: Питер, 2006. – 576 с.

[3] IBM System/3S0 Operating System: MFT Guide, GC27-6939-10 (March, 1972).

[4] IBM System/3S0 Operating System: MVT Guide, GC28-6720-4 (March, 1972).

[5] EMC mainframe technology overview, (http://uk.emc.com/collateral/hardware/technical-documentation/h6109-mainframe-technology.pdf)

[6] JCL Automation Tool – JAT, (http://www.dcmsi.com/jata2.html)

[7] Mainframe Testing and Auditing with Hiperstation, (http://www.compuware.com/mainframe-solutions/hiperstation-automated-mainframe-testing.html)

[8] Test automation, (http://en.wikipedia.org/wiki/Test_automation)

[9] TN3270 Plus, (http://sdisw.com/)

[10] Symmetrix Family, (http://www.emc.com/storage/symmetrix-vmax/symmetrix-vmax.htm)

[11] z/OS MVS JCL Reference, (http://publib.boulder.ibm.com/infocenter/zos/v1r13/index.jsp?topic=%2Fcom.ibm.zos.r13.ieab600%2Fiea2b6b0.htm)

[12] z/OS operating system, (http://www-03.ibm.com/systems/z/os/zos/)

Приложение.

Приложение 1. Программа запуска.

```
//T5876    JOB  (EMC),'JTEST 5876',CLASS=A,MSGCLASS=X
//*
//JPROC    PROC
//JTEST    EXEC PGM=IKJEFT01,PARM='JTEST -TSO'
//STEPLIB  DD DISP=SHR,DSN=SYS3.TEST.AUTO.LOADLIB
//SYSEXEC  DD DISP=SHR,DSN=SYS3.TEST.AUTO.CLIST
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN  DD DUMMY
//SYSUT1   DD DCB=(RECFM=FB,LRECL=160),UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//INITPDS  DD DISP=SHR,DSN=APPLQA.TIMEFIND.JTEST.MODEL.JCLLIB
//CUSTPDS  DD DISP=SHR,DSN=YKOTOV1.TIMEFIND.JTEST.CUST.T5876
//OUTPUT   DD DISP=SHR,DSN=YKOTOV1.TIMEFIND.JTEST.OUTPUT.T5876
//DEVREQ   DD DISP=SHR,DSN=APPLQA.DEVICE.RESERV
//         PEND
//*
//#NIB3390 EXEC JPROC
//SETVARS  DD *
 AUTOREXX        = SYS3.TEST.REXX
 AUTOLIB         = SYS3.TEST.AUTO.LOADLIB
*----------------------
 MFELIB          = J74LIB
 SCFSUF          = V74
 CPFX            = V4
*----------------------
 LCLGK           = 5DD0
 RMTGK1          = 10D9    -  V12X NTC
*
 LCLRAG1         =  60     -  TFNTC2NIB
 RMTRAG1         =  60
*----------------------
STD = %DEVREQ(NIB,YKOTOV1,9) 1 NONE 2MIRR
BCV = %DEVREQ(NIB,YKOTOV1,9) 1 BCV  2MIRR
/*
//DEPEND   DD *
        *  : %SKIP      - SKIP ALL
*
 T3243001  : %RUN
```

```
 T3243002  : %RUN
 T3243006  : %RUN
 T3243007  : %RUN
 T3243008  : %RUN
 T3243021  : %RUN
 T3243025  : %RUN
 T3243030  : %RUN
 T3243033  : %RUN
 T3243040  : %RUN
 T3243050  : %RUN
 T3244053  : %RUN
/*
```

Представленная JCL программа – это рабочий пример задания по запуску двенадцати тестов для продукта TimeFinder.

Приложение 2. Отчёт о тестировании.

```
******************************** TOP OF DATA ********************************
 JTEST v2.4.011 started...
---------------------- PROGRAM PARAMETERS ----------------------------------
 Set TSO as interface to JES.
---------------------- DD SECTION ------------------------------------------
 DD:SETVARS has been opened.
 DD:CUSTPDS has been opened. 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876'
 DD:OUTPUT has been opened Output PDS(E) 'YKOTOV1.TIMEFIND.JTEST.OUTPUT.T5876'
 DD:DEPEND has been opened.
---------------------- EXECUTION -------------------------------------------
[Tue May 29 11:56:30 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243001)'
[Tue May 29 11:56:30 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243002)'
[Tue May 29 11:56:30 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243006)'
[Tue May 29 11:56:30 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243008)'
[Tue May 29 11:56:30 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243025)'
[Tue May 29 11:58:45 2012] Job T3243002 finished with RC=0
[Tue May 29 11:58:45 2012] Job T3243025 finished with RC=0
[Tue May 29 11:58:45 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3244053)'
[Tue May 29 11:59:06 2012] Job T3243008 finished with RC=0
[Tue May 29 11:59:06 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243030)'
[Tue May 29 12:00:08 2012] Job T3243006 finished with RC=0
[Tue May 29 12:00:08 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243007)'
[Tue May 29 12:01:21 2012] Job T3244053 finished with RC=8
[Tue May 29 12:01:52 2012] Job T3243030 finished with RC=0
[Tue May 29 12:01:52 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243040)'
[Tue May 29 12:02:03 2012] Job T3243001 finished with RC=0
[Tue May 29 12:02:03 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243033)'
[Tue May 29 12:02:24 2012] Job T3243007 finished with RC=0
[Tue May 29 12:02:24 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243021)'
[Tue May 29 12:05:06 2012] Job T3243040 finished with RC=0
[Tue May 29 12:05:06 2012] SUBMIT 'YKOTOV1.TIMEFIND.JTEST.CUST.T5876(T3243050)'
[Tue May 29 12:05:17 2012] Job T3243021 finished with RC=0
[Tue May 29 12:05:27 2012] Job T3243033 finished with RC=0
[Tue May 29 12:07:28 2012] Job T3243050 finished with RC=0
*****************************************************
... JTEST ended.
---------------------- FAILED JOBS DETAILS ---------------------------------
T3244053: RC=8
 12.00.54 J0003279  T3244053 TFREEST  EMCTF     0008     23      0      0
---------------------- OVERALL RESULTS -------------------------------------
 Total jobs ran    - 12
 Jobs with good RC - 11
 Jobs with bad RC  - 1
 Skipped members   - 212
 Total members     - 224
-----------------------------------------------------------------------------
******************************* BOTTOM OF DATA ********************************
```

Выше представлен отчёт о работе программы, приведённой в Приложении 1. Отчёт содержит следующие разделы:

- PROGRAM PARAMETERS

  Параметры, с которыми было запущено приложение JTEST.
- DD SECTION

  Используемые DD карты и связанные с ними наборы данных.
- EXECUTION

  Время запуска и завершения каждого тестового задания. Код, с которым завершился тест.
- FAILED JOBS DETAILS

  Шаг, на котором произошла ошибка, для неуспешно завершившихся тестов.
- OVERALL RESULTS

  Статистика проведённого тестирования: кол-во выполненных, успешных и неуспешных тестов.