

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
“Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В.И.Ульянова (Ленина)”
(СПбГЭТУ)

Факультет КТИ

Специальность АСОИУ

Кафедра АСОИУ

К защите допустить:
Заведующий кафедрой

Советов Б. Я.

**Пояснительная записка
К ДИПЛОМНОМУ ПРОЕКТУ**

Тема: «Обеспечение тестирования взаимодействия Flex приложений с сервером через AMF»

Студент	_____	/ Тедикова О.В. /
Руководитель	_____	/ Выговский Л. С. /
Консультант по экономическому обоснованию	_____	/ Швецова О.А. /
Консультанты	_____	/ Берковская К.И. /

Санкт-Петербург
2012 г.

Реферат

В этой работе ...

Содержание

Введение	7
1 Анализ существующих решений	14
1.1 Формулировка критериев выбора	14
1.2 Обзор утилит для тестирования Flex приложений	15
1.2.1 HP QuickTest Professional	15
1.2.2 IBM Rational Functional Tester	17
1.2.3 NeoLoad	18
1.2.4 Apache JMeter	18
1.3 Итоги	19
2 Проектирование модуля	21
2.1 Требования к разрабатываемому модулю	21
2.1.1 Формулировка требований	21
2.1.2 Анализ требований	22
2.2 Архитектура модуля	22
3 Программная реализация модуля	23
3.1 Выбор технического средства решения задачи	23
3.2 Руководство пользователя	23
3.2.1 Установка JMeter	23
3.2.2 Установка модуля amf-translator	23
3.2.3 Запуск приложения	23
3.2.4 Настройка прокси-сервера	23
3.2.5 Запись тестового сценария	24
3.2.6 Создание тест-плана	25
3.2.7 Запуск тестов	27
3.3 Методика тестирования	28
4 Техничко-экономическое обоснование	29
4.1 Концепция экономического обоснования	29
4.2 Рынок и план маркетинга	30
4.2.1 Сегментирование рынка	30
4.2.2 Продвижение товара	30
4.3 Производство продукта	31
4.3.1 Статья «Материалы»	31

4.3.2	Статья «Спецоборудование»	32
4.3.3	Статья «Расходы на оплату труда»	32
4.3.4	Статья «Отчисления на социальные нужды»	33
4.3.5	Статья «Затраты по работам, выполняемым сто- ронними организациями»	33
4.3.6	Статья «Командировочные расходы»	34
4.3.7	Статья «Прочие прямые расходы»	34
4.3.8	Статья «Накладные расходы»	34
4.4	Организационный план проекта	35
4.5	Комплексная оценка эффективности НИР	38
4.5.1	Научно-технический эффект разработки	38
4.5.2	Экономический эффект	38
4.5.3	Расчет потребности в начальных инвестициях . . .	38
	Заключение	40
	А Программный код модуля	41

Глоссарий

Распределённый — Слово, которое нельзя употреблять. Но надо протестировать длинные строки в глоссарии.

Обозначения и сокращения

АИС — Автоматизированная информационная система.

Введение

Данный дипломный проект ориентирован на практическое применение навыков автоматизации тестирования: обзор и анализ существующих методов и средств нагрузочного тестирования, разработку программного обеспечения, в значительной степени снижающего сложность осуществления функционального и нагрузочного тестирования взаимодействия клиента и сервера по AMF-протоколу, разработку методики функционального и нагрузочного тестирования с использованием предложенного ПО.

В настоящее время активно развивается рынок Web-приложений. Все больше различных услуг предоставляется через Интернет, классические информационные системы в различных компаниях также становятся Web-приложениями, предоставляя для своих сотрудников и партнеров доступ к ресурсам компании не только из локальной, но и из глобальной сети. Концепции и технологии, используемые при разработке web-приложений, постоянно развиваются и совершенствуются. Оптимизируется использование ресурсов и времени, улучшаются возможности по отображению предоставляемой информации, динамичность и интерактивность web-приложений.

На сегодняшний день одним из наиболее перспективных подходов к обеспечению всего вышеперечисленного является концепция Rich Internet Application (в дальнейшем RIA) - это приложения, доступные через сеть Интернет, обладающие особенностями и функциональностью традиционных настольных приложений.

Приложения RIA, как правило:

- а) передают веб-клиенту необходимую часть пользовательского интерфейса, оставляя большую часть данных (ресурсы программы, данные и пр.) на сервере;
- б) запускаются в браузере и не требуют дополнительной установки программного обеспечения;
- в) запускаются локально в среде безопасности, называемой «песочница» (sandbox).

На Рис. 0.1 демонстрируется место Rich Internet Applications среди других технологий программных систем

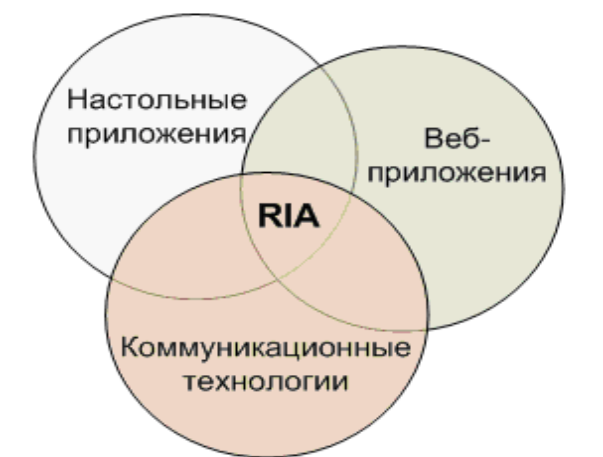


Рисунок 0.1 — RIA среди других технологий

Термин «RIA» впервые был упомянут компанией Macromedia в официальном сообщении в марте 2002 года. Подобные концепции существовали и несколькими годами ранее, например:

- а) DHTML — HTML страницы с динамическим содержимым;
- б) Java-applet — это программы написанные на языке Java, как правило, предназначенные для загрузки посредством браузера;

Для начала рассмотрим архитектурные особенности построения web-приложений базирующихся на RIA концепции. Для этого сравним принцип работы традиционного web-приложения и RIA-приложения (Рис. 0.2).

Работа традиционных web-приложений сконцентрирована вокруг клиент серверной архитектуры с тонким клиентом. Такой клиент переносит все задачи по обработке информации на сервер, а сам используется в основном для отображения статического контента. Основной недостаток этого подхода в том, что все взаимодействие с приложением должно обрабатываться сервером, что требует постоянной отправки данных на сервер, ожидания ответа сервера, и загрузки страницы обратно в браузер. В отличие от традиционного web-приложения в RIA значительная часть функционала выполняется на стороне клиента, поэтому появляется возможность отправлять и получать данные с сервера только по мере

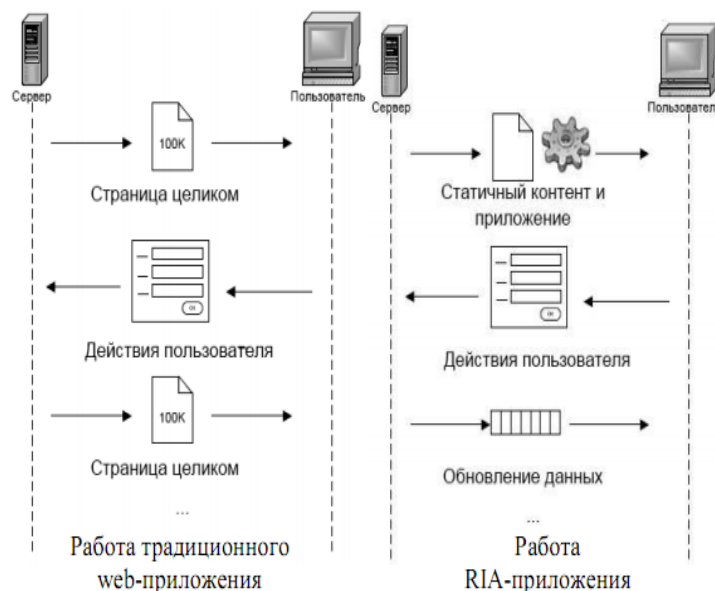


Рисунок 0.2 — Принципы работы традиционного web-приложения и RIA приложения

необходимости. Если подключение нестабильно, клиентская часть RIA-приложения обладает возможностями кэширования данных и работы без подключения к сети в режиме offline.

На основе вышесказанного, можно выделить ряд преимуществ RIA как перед настольными приложения, так и перед стандартными веб-приложениями:

- а) для работы с RIA не требуется установка приложения, как правило, все что необходимо, это установка плагина для браузера;
- б) пользователи могут использовать приложение на любом компьютере, имеющем соединение с Интернет, причем обычно не важно, какая операционная система на нем установлена;
- в) интерфейс RIA обладает большей интерактивностью, он не ограничен лишь использованием языка HTML, применяемого в стандартных веб-приложениях. Наиболее сложные приложения RIA предлагают внешний вид и функциональность, близкие к настольным приложениям;
- г) при работе веб-приложения компьютер пользователя гораздо меньше подвержен риску вирусному заражению, чем при запуске исполняемых бинарных файлов;

д) использование вычислительных ресурсов клиента и сервера лучше сбалансировано;

е) взаимодействие клиента с сервером осуществляется без ожидания каких-либо действий пользователя.

Для реализации RIA многими IT-компаниями предлагаются различные технологии. Наиболее известными из них являются AJAX, Flash, Flex (в дальнейшем Flex) и AIR фирмы Adobe; ActiveX, WPF и Silverlight корпорации Microsoft; Java FX и Java Applets компании Sun Microsystems.

В настоящее время одной из наиболее распространенных технологий разработки RIA является Adobe Flex, которая представляет собой среду с открытым кодом для создания и обслуживания web-приложений, совместимую со всеми распространенными браузерами, платформами персональных компьютеров и версиями операционных систем и имеющую максимальную поддержку со стороны разработчиков.

Flex представляет собой набор классов, расширяющих возможности Flash. Flex позволяет описывать интерфейс web-приложения на MXML – декларативном языке описания и настройки свойств визуальных элементов интерфейса, основанном на XML. Логика web-приложения пишется на ActionScript – полноценном объектно-ориентированном языке программирования, который является одним из диалектов ECMAScript. Результатом компиляции является файл SWF, предназначенный для выполнения в браузере (на платформе Flash Player, которая существует в виде плагина к браузеру) или как самостоятельное приложение (на платформе AIR).

Flex-framework включает возможности локализации, стилизации приложения, разработки модульного приложения, встроенные валидаторы и форматоры текстовых полей — все те инструменты, которые нужны разработчикам приложений, работающих online. Также Flex предоставляет полные мультимедийные возможности Flash Platform, такие как потоковое мультимедиа, возможность получить доступ к веб-камере и микрофону пользователя, бинарные сокет, обширные возможности сетевых коммуникаций (HTTP-запросы, веб-сервисы, встроенный формат

сериализации AMF), оперирование координатами трехмерного пространства).

Вся разработка во Flex ориентирована на применение готового набора расширяемых компонентов, внешний вид которых позволяет гибко настраивать CSS, что облегчает задачу разработчика.

Flex SDK является бесплатным инструментарием с июня 2007 года с открытым исходным кодом, распространяемым на условиях Mozilla Public License. Для работы с процедурами и классами этого фреймворка можно использовать как бесплатные (Eclipse WTP IDE, FlashDevelop IDE) так и платные (Flex Builder IDE, IntelliJ IDEA IDE, Aptana Studio IDE, PowerFlasher FDT IDE) среды разработки.

Flex приложения предоставляют возможность реализации клиент-серверного взаимодействия на основе бинарного формата обмена данными — AMF (Action Message Format). AMF используется для сериализации структурированных данных, таких как объекты Action Script или XML, и обмена сообщениями между Adobe Flash клиентом и удалённым сервисом. AMF более экономичен по трафику по сравнению с XML и позволяет передавать типизированные объекты.

Как известно огромную роль в жизненном цикле программного обеспечения играет фаза тестирования. Тестирование — один из важнейших этапов контроля качества в процессе разработки ПО. Автоматизированное тестирование является его составной частью. Оно использует программные средства для выполнения тестов и проверки их результатов, что помогает сократить время тестирования и упростить его процесс, а также может дать возможность выполнять определенные тестовые задачи намного быстрее и эффективнее чем это может быть сделано вручную. Автоматизация тестирования даёт следующие преимущества:

а) Автоматизация — выполнение сложно воспроизводимых вручную тестов, таких как нагрузочное и стресс-тестирование;

б) Меньшие затраты на поддержку — когда автоматические скрипты уже написаны, на их поддержку и анализ результатов требуется, как правило, меньшее время чем на проведение того же объема тестирования вручную;

- в) Повторяемость — все написанные тесты всегда будут выполняться однообразно, то есть исключен «человеческий фактор». Тестировщик не пропустит тест по неосторожности и ничего не напутает в результатах;
- г) Быстрое выполнение — автоматизированному скрипту не нужно сверяться с инструкциями и документациями, это сильно экономит время выполнения;
- д) Отчеты — автоматически рассылаемые и сохраняемые отчеты о результатах тестирования.

Если сложить все вышесказанное, то больший объем тестирования, может быть достигнут с меньшими усилиями, давая при этом возможность улучшать как качество, так и продуктивность.

Однако использование AMF вызывает ряд трудностей для реализации автоматизации функционального и нагрузочного тестирования взаимодействия сервера и Flex клиента, связанных с бинарной природой протокола. Так как amf сообщение представляет собой совокупность байтов, разработчикам и тестировщикам трудно считывать и изменять содержащуюся в нём информацию.

Отдельной проблемой является нагрузочное тестирование таких приложений — имитация работы с приложением большого количества пользователей за счёт запуска набора тестов в несколько потоков. Сложность состоит в поддержке уникальности идентификатора сессии каждого клиента. Идентификатор клиента выделяется строго одному подключенному SWF-файлу и прошит в передаваемых данных. Стандартный порядок тестирования — запись запросов пользователя с использованием прокси-сервера и запуск сохранённого сценария в несколько потоков — в данном случае будет неэффективным, так как все перехваченные запросы будут иметь одинаковый идентификатор клиента и тесты пройдут только в 1 потоке, а остальные будут возвращены с ошибкой о неправильной сессии.

Таким образом, можно сформулировать наличие противоречия между необходимостью проведения функционального и нагрузочного тестирования и сложностью его реализации на технологии Flex и поставить следующие задачи на дипломную работу:

- а) разработка программного обеспечения, в значительной степени снижающего сложность осуществления функционального и нагрузочного тестирования взаимодействия клиента и сервера по AMF-протоколу;
- б) разработка методики функционального и нагрузочного тестирования с использованием предложенного ПО.

1 Анализ существующих решений

В этом разделе выполнены следующие работы: анализ существующих решений в области автоматизации тестирования, сравнительная оценка вариантов возможных решений, выбор наиболее оптимального средства автоматизации.

1.1 Формулировка критериев выбора

На сегодняшний день существует ряд различных программных комплексов, предоставляющих возможность для проведения функционального, нагрузочного, регрессионного тестирования Flex приложений, поэтому для решения поставленной на дипломный проект задачи нет необходимости создавать тестовую утилиту с нуля, разумнее будет выбрать одно из существующих решений, и, в случае, если оно не полностью удовлетворяет нашим условиям, доработать его. Чтобы выбрать из предложенного разнообразия подходящий продукт, сформулируем ряд требований, которым должна удовлетворять выбранная нами тестовая утилита.

а) Возможность проведения нагрузочных тестов. Так как большинство web-приложений являются многопользовательскими, проведение нагрузочных тестов является обязательным условием качественного тестирования. Поскольку запуск тестов в несколько потоков является довольно сложной задачей, инструмент для автоматизации должен обеспечивать удобный и надёжный способ многопоточного прогона тестовых сценариев.

б) Поддержка протокола AMF. Утилита должна осуществлять тестирование взаимодействия Flex приложения с сервером, предоставляя пользователю возможность наблюдать трафик. Тестирование пользовательского интерфейса для нашей задачи нецелесообразно.

в) Простота создания тестовых сценариев. Утилита должна предоставлять пользователю удобных механизм записи и редактирования тестов. такими как maven и ant, и серверами интеграционного тестирования. Это условие

г) Возможность запуска тестов из командной строки. Данная возможность позволяет выполнять тесты в автоматическом режиме в системах непрерывной интеграции, что является необходимостью, особенно в тех случаях, когда над разными частями тестируемой системы разработчики трудятся независимо и необходимо выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. Часто производители тестовых фреймворков предоставляют свои продукты для проведения интеграционного тестирования, однако зачастую они узко заточены под конкретный круг задач, поэтому нас будет интересовать возможность запуска тестов на таких инструментах интеграции, как Jenkins и Hudson, которые на данный момент широко распространены и имеют множество плагинов, позволяющих значительно расширить их существующую функциональность.

д) Кроссплатформенность. Выбранная нами тестовая утилита должна работать под управлением различных операционных систем.

е) Условия распространения продукта. В идеале программное обеспечение должно быть бесплатным и иметь открытый исходный код с возможностью создания собственных расширений, это является одним из основных критериев отбора, так как тестировщики и разработчики всегда должны иметь возможность доработки и усовершенствования существующего функционала тестового фреймворка, чтобы адаптировать его под специфику работы тестируемого ими приложения.

ж) Документация. Так как современные инструменты тестирования не менее сложны, чем среды разработки, большим плюсом будет наличие подробной пользовательской документации, документации для разработчиков, различных форумов и сайтов, посвящённых данному инструменту тестирования.

1.2 Обзор утилит для тестирования Flex приложений

1.2.1 HP QuickTest Professional

HP QuickTest Professional (QTP) — один из инструментов автоматизации функционального тестирования, является флагманским продук-

том компании HP в своей линейке. Для разработки автоматизированных тестов QTP использует язык VBScript (Visual Basic Scripting Edition) — скриптовый язык программирования, интерпретируемый компонентом Windows Script Host. Он широко используется при создании скриптов в операционных системах семейства Microsoft Windows. QTP поддерживает ряд технологий, среди которых есть и Macromedia Flex. Поддержка Flex осуществляется за счёт установки плагина, предоставляемого компанией Adobe (Flex QTP add-in).

Чтобы приступить к тестированию приложения необходимо скомпилировать, создав для него HTML-оболочку, и развернуть его либо локально, либо на web-сервере. Создание тестов в QTP осуществляется следующим образом: приложение открывается в браузере и все действия, совершаемые пользователем с пользовательским интерфейсом приложения записываются в виде строчек Visual Basic скрипта. QTP поддерживает запись большинства наиболее часто используемых во Flex приложениях событий (операций), связанных с пользовательским интерфейсом. Однако часть из них, например некоторые атомарные операции, игнорируются во время записи тестов. Пользователь имеет возможность добавить их в текст скрипта вручную. Для проверки правильности выполнения теста пользователь задаёт ожидаемые значения для выполняемых операций — checkpoints. Во время автоматического прогона тестов запускать браузер уже не нужно, достаточно лишь указать HTML страницу, используемую для тестов. Для выявления причины возникновения ошибок в тестах или каких-либо других неполадок можно обратиться к логам Flash Player или настроить и включить логирование в QTP. Возможность прогона тестов в несколько потоков в QTP отсутствует.

Для проведения тестов с HP QuickTest Professional необходимо использовать браузер Internet Explorer версии 6 или выше. HP QuickTest Professional работает с операционными системами семейства Windows и является платным программным обеспечением.

1.2.2 IBM Rational Functional Tester

IBM Rational Functional Tester является средством автоматизированного регрессивного тестирования, предназначенным для тестирования Java, .NET, Web-приложений, включая Flex-приложения, и терминальных приложений на платформах Windows и Linux. Functional Tester поддерживает два языка сценариев: Java и Visual Basic.NET. Для тестирования Java-приложений в программу Functional Tester включена открытая среда разработки Eclipse. Установка дополнительных компонентов не требуется. Если требуется использовать язык сценариев Visual Basic.NET, перед установкой IBM Rational Functional Tester необходимо установить Visual Studio.NET.

Перед началом тестирования приложение загружается в браузер, затем все действия, совершаемые пользователем записываются в виде Java или Visual Basic.NET скрипта (в зависимости от настроек). Пользователь может редактировать скрипты, а также задавать ожидаемые результаты выполнения той или иной операции для проверки работы приложения. Как и QTP, IBM Rational Functional Tester во время записи тестов добавляет компоненты приложения в свой набор объектов. Далее тесты можно запускать в автоматическом режиме. Также в IBM Rational Functional Tester имеется возможность многократного запуска тестов с различным набором данных. Если стандартного набора объектов (таких как кнопки, текстовые поля и т.д.) недостаточно, существует возможность самостоятельно добавить во фреймворк необходимые объекты. Это возможно благодаря Rational Functional Tester proxy software development kit (SDK), который имеет удобное API и довольно подробную сопроводительную документацию. Также IBM Rational Functional Tester упрощает механизм регрессионного тестирования. Functional Tester использует усовершенствованную технологию ScriptAssure для того, чтобы «изучить» контрольные характеристики пользовательского интерфейса, что позволяет идентифицировать те же самые средства управления в новой версии, несмотря на внесенные изменения. Эти характеристики сохраняются в объектной карте, совместный доступ к которой могут получить различные скрипты и участники

проекта. Благодаря этой карте изменения, внесенные в характеристики распознавания объекта, будут отражены во всех скриптах тестирования, что существенно упрощает обслуживание.

Rational Functional Tester поддерживает операционные системы Windows 2000, XP, Vista, 7, Linux. Является платным программным обеспечением.

1.2.3 NeoLoad

NeoLoad — профессиональный инструмент нагрузочного тестирования веб-приложений, в том числе и Flex, работающий на многих платформах, в том числе Windows, Solaris, Linux. Осуществляя моделирование большого числа пользователей, которые обращаются к приложению, NeoLoad делает проверку надежности и производительности приложения при разных нагрузках.

Тестирование Flex приложений осуществляется засчёт записи AMF трафика во время взаимодействия пользователя с приложением. Все перехваченные запросы отображаются в xml формате, что позволяет редактировать содержащиеся в них данные. Далее записанные тесты могут быть запущены в несколько потоков.

NeoLoad является платным программным обеспечением с закрытым исходным кодом.

1.2.4 Apache JMeter

JMeter — инструмент для проведения нагрузочного тестирования, изначально разрабатывался как средство тестирования web-приложений, в настоящее время он способен проводить нагрузочные тесты для JDBC-соединений, FTP, LDAP, SOAP, JMS, POP3, IMAP, HTTP и TCP. В JMeter реализованы возможность создания большого количества запросов с помощью нескольких компьютеров при управлении этим процессом с одного из них, логирование результатов тестов и разнообразная визуализация результатов в виде диаграмм, таблиц и т. п.

Ключевое понятие в JMeter — план тестирования. План тестирования приложения представляет собой описание последовательности шагов, которые будет исполнять JMeter. План тестирования может содержать:

- а) Группы потоков (Thread Groups) — элемент, позволяющий конфигурировать многопоточный запуск тестов
- б) Контроллеры — позволяют создавать тесты со сложной логической структурой.
- в) Слушатели — отображают результаты выполнения тестов
- г) Соответствия — позволяют сравнивать полученные результаты с ожиданиями пользователя
- д) Сэмплеры — основные элементы тест-плана, в которых формируется тело запроса, тестовый шаг.

На данный момент в JMeter нет отдельных средств для тестирования Flex приложений, однако есть возможность записи http запросов, содержащих в себе тело AMF сообщения, с помощью прокси-сервера. Далее перехваченные запросы могут быть перенесены в план тестирования и запущены в режиме нагрузочного тестирования. Созданные в JMeter тесты могут быть запущены с помощью систем сборки проектов Maven и Ant. Также JMeter свободно интегрируется со многими серверами сборки, такими как Jenkins и Hudson.

JMeter является бесплатным кросс-платформенным Java-приложением с открытым исходным кодом и удобным API. Существует большое количество плагинов к JMeter, значительно расширяющих его базовую функциональность.

1.3 Итоги

В предыдущем разделе был рассмотрен ряд инструментов для тестирования Flex приложений. Для наглядности и простоты анализа все исследованные продукты были собраны в таблицу и охарактеризованы по следующему набору критериев:

- а) Возможность проведения нагрузочных тестов;

- б) Поддержка протокола AMF.
- в) Простота создания тестовых сценариев.
- г) Возможность запуска тестов из командной строки.
- д) Кроссплатформенность;
- е) Условия распространения продукта;
- ж) Документация.

Для каждого критерия (в таблице указаны их номера), была дана оценка по пятибальной шкале, характеризующая степень того, в какой мере тот или иной продукт удовлетворяет данному критерию.

Таблица 1.1 — Таблица сравнительного анализа тестовых утилит

Инструмент тестирования	1	2	3	4	5	6	7
HP QuickTest Professional	2	0	3	4	0	1	4
IBM Rational Functional Tester	2	0	4	3	5	1	4
NeoLoad	5	5	4	4	5	1	3
Apache JMeter	5	3	4	5	5	5	4

Из таблицы видно, что по обозначенным критериям наиболее подходящим для решения поставленной задачи является инструмент тестирования Apache JMeter, решающим фактором стало условие распространения продукта с открытым исходным кодом, а также его популярность. Для того, чтобы JMeter мог обеспечивать полную поддержку тестирования Flex приложений, необходимо будет расширить его функционал, добавив отображение AMF сообщений в понятной пользователю форме и возможность запуска тестов в несколько потоков без дублирования сессий с сервером.

2 Проектирование модуля

2.1 Требования к разрабатываемому модулю

2.1.1 Формулировка требований

Сформулируем ряд функциональных требований к разрабатываемому модулю. Требования должны быть составлены таким образом, чтобы результат их реализации удовлетворял одной из поставленных на дипломный проект задачи — разработанное ПО должно в значительной степени снижать сложность осуществления функционального и нагрузочного тестирования взаимодействия клиента и сервера по AMF-протоколу.

а) Разрабатываемый программный комплекс должен предоставлять возможность записи действий пользователя, производимых с тестируемым приложением (проксирование запросов). Проксирование является важной опцией на этапе подготовки тестирования, поскольку позволяет создавать тестовый сценарий, полностью имитирующий работу пользователя с приложением.

б) Все перехваченные на этапе проксирования запросы, а также ответы сервера при приёме тестов, должны предоставляться пользователю в человекочитаемой форме. Поскольку используемый во Flex приложениях AMF протокол является бинарным, преобразование тела AMF запроса в понятную пользователю форму является обязательным условием для корректного анализа передаваемых в нём данных.

в) Программный модуль должен предоставлять пользователю возможность создания, редактирования и удаления шагов тест плана.

г) Реализация программного модуля должна полностью поддерживать нагрузочное тестирование взаимодействия Flex приложений с сервером — имитацию работы с приложением большого количества пользователей одновременно, а именно обеспечение уникальности идентификатора сессии клиент-серверного взаимодействия для каждого потока и его сохранность в рамках одного потока.

2.1.2 Анализ требований

Проанализируем сформулированные требования к разрабатываемому модулю. Для наглядности и удобства сведём их в таблицу. Для классификации требований были выбраны следующие критерии:

а) Приоритет — критерий оценки полезности реализации требования для конечного пользователя, важности для достижения поставленных перед проектом целей.

б) Трудоёмкость — критерий отображает предварительную оценку сложности реализации требования, количество привлекаемых для этого ресурсов.

в) Риск — интегральный критерий, введением которого предпринимается попытка оценить во-первых возможность невыполнения требования, во-вторых возможную ошибку в оценке по двум предыдущим критериям (в первую очередь трудоёмкости).

Для каждого из критериев введена шкала из трёх уровней: низкий, средний, высокий. Размерность шкалы выбрана минимальной исходя из соображений простоты и наглядности. Так как размер проекта и количество предъявляемых к нему требований незначительны, то таких шкал вполне достаточно для исчерпывающей классификации требований а также принятия проектных и организационных решений.

2.2 Архитектура модуля

3 Программная реализация модуля

3.1 Выбор технического средства решения задачи

3.2 Руководство пользователя

3.2.1 Установка JMeter

Для начала необходимо скачать с сайта производителя zip архив, содержащий все необходимы для установки файлы, а затем распаковать его на диске. Место установки JMeter далее будем называть JMeter_HOME.

3.2.2 Установка модуля amf-translator

Чтобы подключить к JMeter модуль amf-translator, достаточно добавить jar-файл приложения в каталог JMeter_HOME/lib/ext.

3.2.3 Запуск приложения

Приложение запускается с помощью файла jmeter.bat или ApacheJMeter.jar, находящихся в каталоге JMeter_HOME/bin.

3.2.4 Настройка прокси-сервера

После запуска в окне приложения с левой стороны нам доступно дерево элементов. Чтобы создать прокси сервер с поддержкой протокола AMF, необходимо правой кнопкой мыши кликнуть по элементу WorkBench, а затем добавить элемент AMF Proxy Server (WorkBench > Add > Non-Test Elements > AMF Proxy Server).

В поле AmfProxy Port необходимо указать номер порта, который будет слушать наш прокси сервер. Если указать, например, 9090, то прокси-сервер будет запущен на localhost:9090. Затем точно такие же настройки прокси-сервера устанавливаются в браузере, с помощью которого будет производиться тестирование. Также стоит убедиться, что указанный Вами порт уже не занят другим приложением.

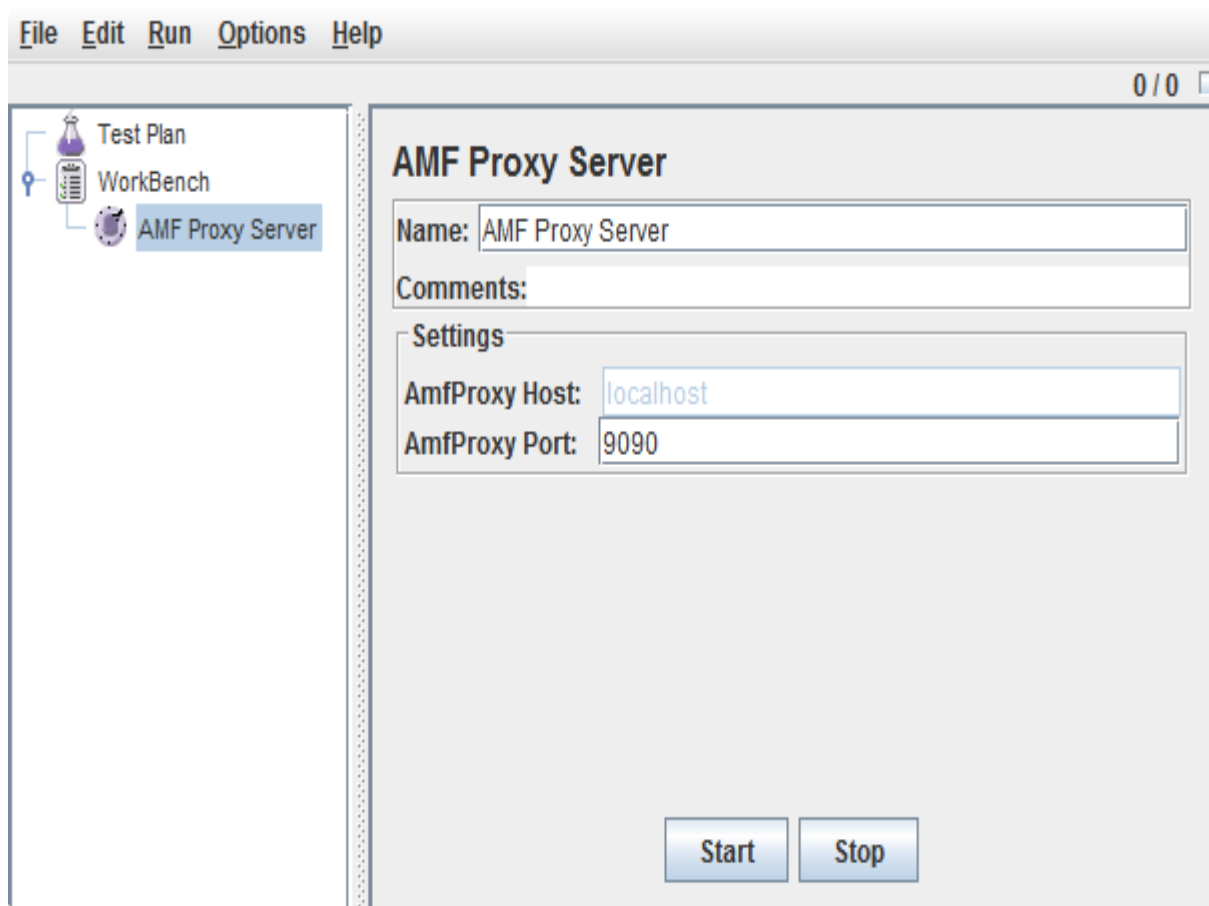


Рисунок 3.1 — Настройка прокси-сервера

3.2.5 Запись тестового сценария

После того, как в AMF Proxy Server установлены все необходимые параметры, нажимается кнопка Start, запускающая прокси-сервер.

Затем тестируемое приложение открывается в браузере, для которого также применены соответствующие настройки прокси-сервера, и пользователь может выполнять с Flex приложением необходимые операции, которые будут записываться AMF Proxy Server в виде элементов AMF RPC Sampler и в дальнейшем могут быть перенесены в тест-план. Чтобы завершить запись тестовых запросов, необходимо нажать кнопку Stop. После завершения записи тестов, все перехваченные запросы отображаются в дереве элементов JMeter в качестве дочерних элементов AMF Proxy Server.

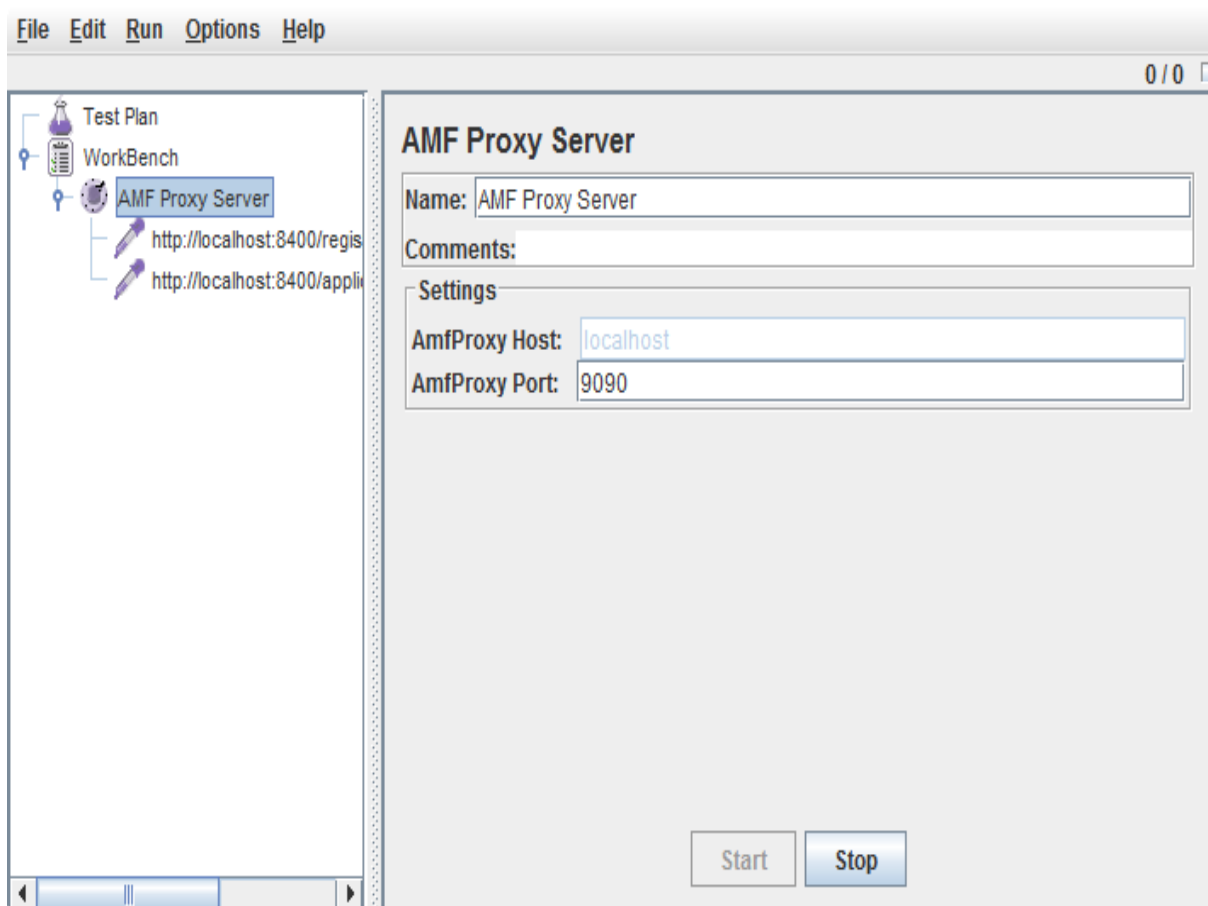


Рисунок 3.2 — Запуск прокси-сервера

3.2.6 Создание тест-плана

Создание тест-плана в JMeter осуществляется следующим образом. В первую очередь добавляется группа потоков - Thread Group (Test Plan > Threads (Users) > Thread Group).

Этот элемент является ключевым в тест-плане JMeter, именно его функционал отвечает за реализацию нагрузочного тестирования — многопоточного запуска последовательности тестовых шагов. В данном элементе задается следующее:

- а) Действия, которые будут производиться в случае, если в тест выполняется с ошибкой (Action to be taken after a Sampler error);
- б) Число потоков, в которое будут запускаться шаги тест-плана (Number of Threads);
- в) Интервал, в течение которого будет запущено указанное в предыдущем параметре число потоков (Ramp-Up Period);

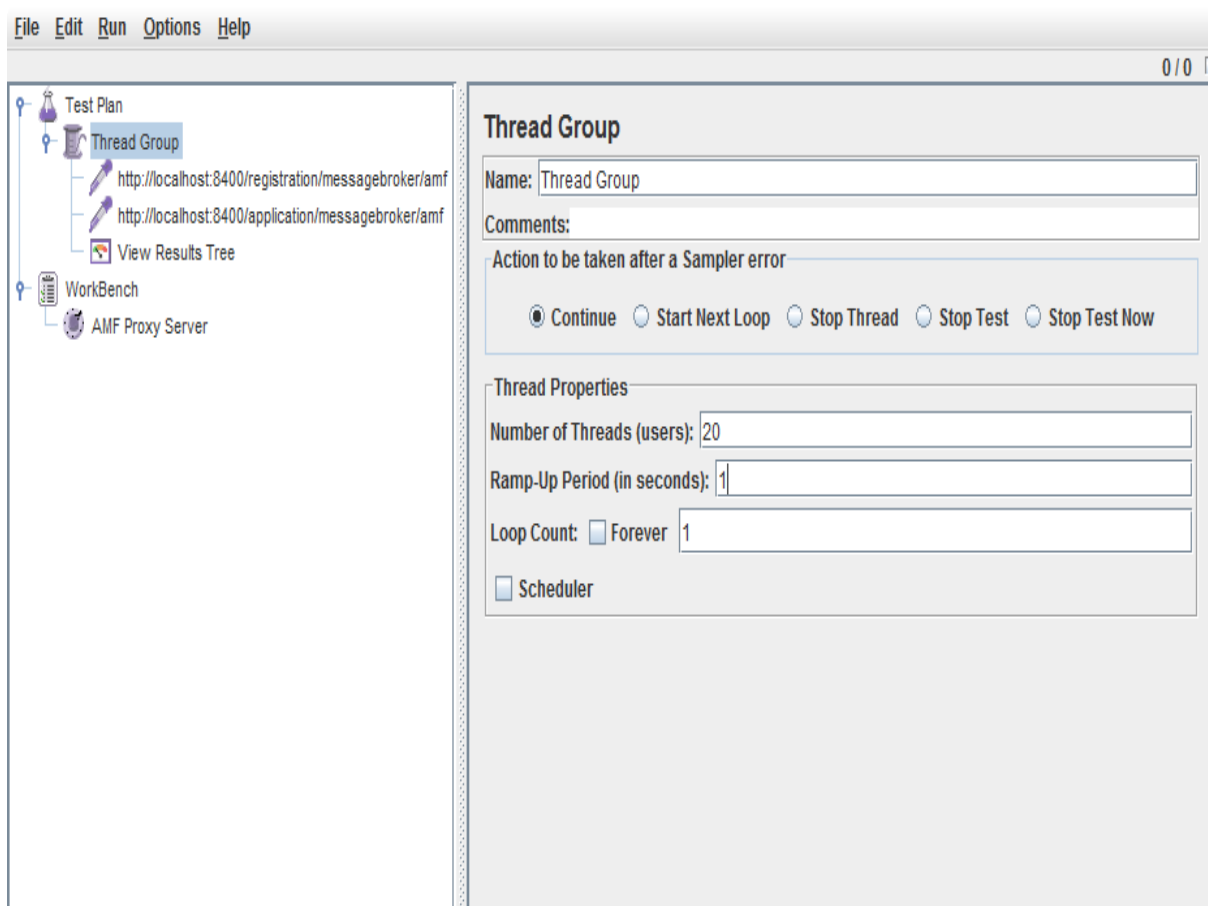


Рисунок 3.3 — Создание тест-плана

- г) Число повторений набора тестов (Loop Count);
- д) Расписание запуска тестов (Scheduler).

Затем в Thread Group в качестве дочерних элементов добавляются шаги тестов, которые будут запускаться с указанными характеристиками. В нашем случае мы переносим элементы AMF RPC Sampler, записанные с помощью прокси.

AMF RPC Sampler представляет собой форму для вызова процедур java-объектов на стороне сервера. Чтобы вызвать метод удалённого объекта, необходимо заполнить следующие поля:

- а) Endpoint Url - URL, по которому отправляется запрос
- б) AMF Call - имя удалённого объекта и процедуры, которая должна быть вызвана; Например, если мы хотим вызвать у объекта registrationDestination метод registerUser, в этом поле следует написать registrationDestination.registerUser;

в) Request Parameters - параметры метода, вызываемого у объекта (если они существуют).

На Рис. 3.4 представлен интерфейс AMF RPC Sampler.

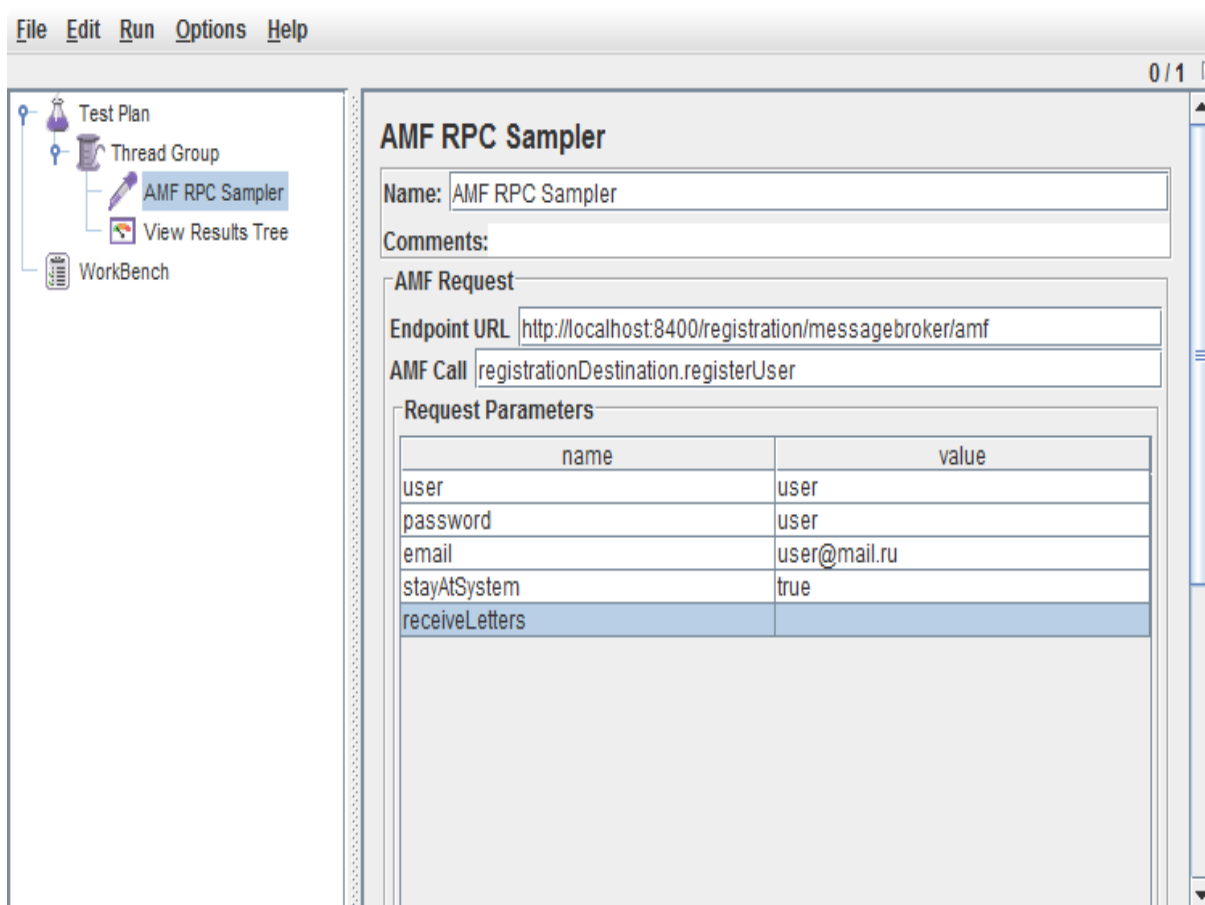


Рисунок 3.4 — Интерфейс элемента AMF RPC Sampler

Помимо сэмплера в тест-план также следует добавить визуалайзер результатов, чтобы иметь возможность отслеживать ход тестового сценария (Thread Group > Add > Listener). JMeter предлагает большой выбор таких элементов, для примера будем использовать View Results Tree. После того как план сформирован, он может быть сохранён. (File > Save Test Plan As...)

3.2.7 Запуск тестов

Чтобы запустить содержимое элемента Test Plan, необходимо выбрать в основном меню Run > Start. После завершения прогона тестов результаты их выполнения можно наблюдать в View Results Tree.

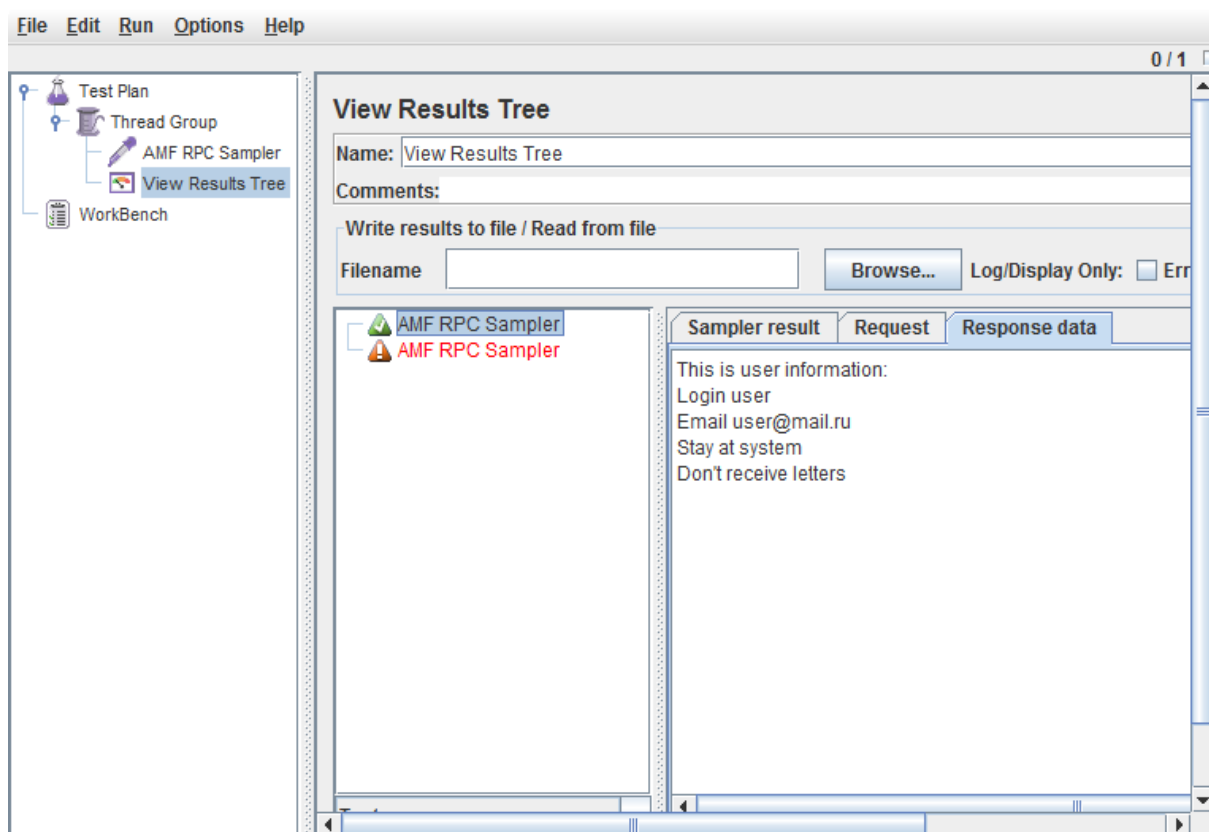


Рисунок 3.5 — Отображение результатов тестов в визуализаторе View Results Tree

В правой части визуализатора отображается дерево элементов тест плана и статус их выполнения - если элемент подсвечен зелёным цветом, то шаг выполнен успешно. В случае с AMF RPC Sampler это значит, что соединение с сервером было установлено и вызов метода удалённого объекта прошёл без ошибок. Иначе тест не считается пройденным и элемент подсвечивается красным цветом. Также на вкладках слева предоставляется возможность просмотра отправленного запроса и полученного от сервера ответа.

3.3 Методика тестирования

4 Технико-экономическое обоснование

4.1 Концепция экономического обоснования

На сегодняшний день одним из наиболее перспективных направлений в разработке web-приложений является концепция Rich Internet Application (в дальнейшем RIA) — это приложения, доступные через сеть Интернет, обладающие особенностями и функциональностью традиционных настольных приложений. Одной из наиболее распространенных технологий разработки RIA является Adobe Flex. Flex приложения предоставляют возможность реализации клиент-серверного взаимодействия на основе бинарного формата обмена данными — AMF (Action Message Format). AMF более экономичен по трафику по сравнению с XML и позволяет передавать типизированные объекты. Как известно огромную роль в жизненном цикле программного обеспечения играет фаза тестирования. Автоматизированное тестирование является его составной частью. Оно использует программные средства для выполнения тестов и проверки их результатов, что помогает сократить время тестирования и упростить его процесс, а также может дать возможность выполнять определенные тестовые задачи намного быстрее и эффективнее чем это может быть сделано вручную. Однако использование AMF вызывает ряд трудностей для реализации автоматизации функционального и нагрузочного тестирования взаимодействия сервера и Flex клиента, связанных с бинарной природой протокола. Так как amf сообщение представляет собой совокупность байтов, разработчикам и тестирующим трудно считывать и изменять содержащуюся в нём информацию. Отдельной проблемой является нагрузочное тестирование таких приложений — имитация работы с приложением большого количества пользователей за счёт запуска набора тестов в несколько потоков. Выходом из сложившейся ситуации является разработка программного обеспечения, в значительной степени снижающего сложность осуществления функционального и нагрузочного тестирования взаимодействия клиента и сервера по AMF-протоколу, а также разработка методики функционального и нагрузочного тестирования использованием предложенного ПО.

Целью технико-экономического обоснования является определение экономической целесообразности реализации проекта. Этапами ТЭО являются:

- а) трудоемкость и календарный план выполнения НИР;
- б) смета затрат на проведение НИР;
- в) комплексная оценка эффективности НИР.

4.2 Рынок и план маркетинга

4.2.1 Сегментирование рынка

Сегментирование рынка состоит в выделении сегментов анализируемого рынка и оценки спроса на продукцию в каждом сегменте рынка.

Итак, выделим основные сегменты:

- а) Компании, осуществляющие разработку программного обеспечения.
- б) Компании, специализирующиеся на внешнем тестировании программного обеспечения.
- в) Программисты, осуществляющие разработку ПО в частном порядке (для личных или коммерческих целей).

Проведем анализ требований различных групп потенциальных покупателей к продукции:

- а) технический уровень, качество и надежность в эксплуатации, уровень послепродажного обслуживания;
- б) технический уровень, качество и надежность в эксплуатации, уровень послепродажного обслуживания;
- в) цена продукции, качество и надежность в эксплуатации.

4.2.2 Продвижение товара

Для эффективного продвижения товара, а так же для поддержания спроса необходима реклама, предоставление скидок, демо-версии для привлечения новых клиентов. Поскольку продукт является специализированным, следует давать рекламу в журналы, газеты, размещать

на сайтах и форумах той же тематики. Скидки, как правило, стимулируют уже существующих клиентов, поскольку важно, привлекая новых, не потерять уже существующих пользователей.

4.3 Производство продукта

Проводится расчет затрат, связанных с проведением НИР. Основные статьи калькуляции приведены ниже:

- а) Материалы
- б) Спецоборудование
- в) Расходы на оплату труда
- г) Отчисления на социальные нужды
- д) Затраты по работам, выполняемым сторонними организациями
- е) Командировочные расходы
- ж) Прочие прямые расходы
- з) Накладные расходы

4.3.1 Статья «Материалы»

На статью относятся затраты на сырье, основные и вспомогательные материалы, покупные полуфабрикаты и комплектующие изделия, необходимые для выполнения конкретной НИР с учетом транспортно-заготовительных расходов. Калькуляция расходов по статье «Материалы» приведена в табл. 4.1.

Таблица 4.1 — Материалы

Наименование материалов	Количество, шт	Цена, р.	Сумма, р.
Бумага формата А4	1	120	120
Заправка картриджа для принтера	1	300	300
Записываемый диск CD-RW	1	50	50
Итого:			470
Транспортные расходы, 15%:			70
Всего:			540

4.3.2 Статья «Спецоборудование»

На статью «Спецоборудование» относятся затраты на приобретение (или изготовление) специальных приборов, стендов, другого специального оборудования, необходимого для выполнения конкретной НИР. Для данной НИР дополнительного спецоборудования не требуется.

4.3.3 Статья «Расходы на оплату труда»

На статью относится заработная плата научных сотрудников, инженеров и прочего инженерно-технического персонала, непосредственно занятых выполнением конкретной НИР. Эти расходы складываются из основной и дополнительной заработной платы. Средняя зарплата за один рабочий день определяется, исходя из месячного оклада и среднего количества рабочих дней за месяц, принимаемого за 22 дня.

Основная зарплата рассчитывается по формуле:

$$C_{зо} = \frac{T \cdot C_{зо.мес}}{22}, \quad (4.1)$$

где T – трудоемкость выполнения работ по НИР, $C_{зо.мес}$ – месячный оклад.

Дополнительная зарплата рассчитывается по формуле:

$$C_{зд} = \frac{C_{зо} \cdot H_d}{100}, \quad (4.2)$$

где $C_{зо}$ – основная зарплата, H_d – норматив дополнительной зарплаты.

Производим расчет основной и дополнительной зарплат на основании следующих данных:

а) трудоемкость выполнения работ исполнителем $T_{исп} = 106$ чел.-дн.;

б) трудоемкость выполнения работ СНС $T_{рук} = 7$ чел.-дн.;

в) месячный оклад инженера $C_{зо.мес.исп} = 15\,000$ р.;

г) месячный оклад руководителя $C_{зо.мес.рук} = 20\,000$ р.;

д) норматив дополнительной зарплаты $H_d = 12\%$.

$$C_{\text{зо.исп}} = \frac{106 \cdot 15000}{22} = 72272 \text{р.}, \quad (4.3)$$

$$C_{\text{зо.рук}} = \frac{7 \cdot 20000}{22} = 6363 \text{р.}, \quad (4.4)$$

$$C_{\text{зо}} = 72272 + 6363 = 78635 \text{р.}, \quad (4.5)$$

$$C_{\text{зд.исп}} = \frac{72272 \cdot 12}{100} = 8672 \text{р.}, \quad (4.6)$$

$$C_{\text{зд.рук}} = \frac{6363 \cdot 12}{100} = 763 \text{р.}, \quad (4.7)$$

$$C_{\text{зд}} = 8672 + 763 = 9435 \text{р.} \quad (4.8)$$

Общие расходы на оплату труда составляют 88 070 р.

4.3.4 Статья «Отчисления на социальные нужды»

На статью относят затраты, связанные с выплатой единого социального налога. Данная статья рассчитывается пропорционально зарплате в размере 26,2%:

- а) федеральный бюджет – 20%;
- б) фонд социального страхования – 3,2%;
- в) фонд обязательного медицинского страхования – 2,8%;
- г) страхование от несчастных случаев – 0,2%.

$$C_{\text{зд}} = \frac{P_{\text{от}} \cdot H_{\text{сн}}}{100}, \quad (4.9)$$

где $H_{\text{сн}}$ – суммарный норматив отчислений на социальные нужды – 26,2

$$C_{\text{сн}} = \frac{88070 \cdot 26,2}{100} = 23074 \text{р.}, \quad (4.10)$$

4.3.5 Статья «Затраты по работам, выполняемым сторонними организациями»

Расходов на работы, выполняемые сторонними организациями, не существует.

4.3.6 Статья «Командировочные расходы»

На статью относятся расходы на все виды служебных командировок работников, выполняющих задания по конкретной НИР. В данной НИР расходов, связанных со служебными командировками, нет.

4.3.7 Статья «Прочие прямые расходы»

На статью относятся расходы на получение специальной научно-технической информации, платежи за использование средств связи и коммуникации, а также другие расходы, необходимые для проведения НИР. На всех этапах работы требуется выход в Интернет. Стоимость работы в Интернете составляет 350 р. в месяц. Затраты на Интернет на весь период составляют:

$$C_{\text{и}} = \frac{350}{22} \cdot 106 = 1686 \text{ р.}, \quad (4.11)$$

Также требуется использование телефона. Примем эти расходы ориентировочно равными $C_{\text{т}} = 300$ р. Следовательно, суммарно прочие прямые затраты составляют:

$$C_{\text{ппз}} = C_{\text{и}} + C_{\text{т}} = 1686 + 300 = 1986 \text{ р.}, \quad (4.12)$$

4.3.8 Статья «Накладные расходы»

В статью включаются расходы на управление и хозяйственное обслуживание НИР.

$$C_{\text{нр}} = \frac{P_{\text{от}} \cdot H_{\text{нр}}}{100}, \quad (4.13)$$

где $H_{\text{нр}}$ – норма накладных расходов, равная 20%.

$$C_{\text{нр}} = 88070 \cdot \frac{20}{100} = 17614 \text{ р.} \quad (4.14)$$

На основании полученных данных в табл. 4.2 приведена калькуляция себестоимости разработки.

Таблица 4.2 — Смета затрат на проведение НИР

№ п/п	Наименование статьи	Сумма, р.
1	Материалы	540
2	Спецоборудование	—
3	Расходы на оплату труда	88070
4	Отчисления на социальные нужды	23074
5	Затраты по работам, выполняемым сторонними организациями	—
6	Командировочные расходы	—
7	Прочие прямые расходы	1986
8	Накладные расходы	17614
Себестоимость НИР:		131 284

4.4 Организационный план проекта

Таблица 4.3 — Трудоемкость и календарный план выполнения НИР

№	Этапы и работы	Трудоемкость, чел.-дн.		Численность, чел.	Длительность, дн.							
		Исполнитель	Руководитель НИР			4	9	14	19	24	29	33
1	•	•	•	•	•	•	•	•	•	•	•	•
2	•	•	•	•	•	•	•	•	•	•	•	•
3	•	•	•	•	•	•	•	•	•	•	•	•
4	•	•	•	•	•	•	•	•	•	•	•	•
5	•	•	•	•	•	•	•	•	•	•	•	•
6	•	•	•	•	•	•	•	•	•	•	•	•
7	•	•	•	•	•	•	•	•	•	•	•	•
8	•	•	•	•	•	•	•	•	•	•	•	•
9	•	•	•	•	•	•	•	•	•	•	•	•
10	•	•	•	•	•	•	•	•	•	•	•	•
11	•	•	•	•	•	•	•	•	•	•	•	•
12	•	•	•	•	•	•	•	•	•	•	•	•
13	•	•	•	•	•	•	•	•	•	•	•	•
14	•	•	•	•	•	•	•	•	•	•	•	•
15	•	•	•	•	•	•	•	•	•	•	•	•
16	•	•	•	•	•	•	•	•	•	•	•	•

Продолжение на след. стр.

Трудоемкость выполнения работы исполнителем составляет 106 чел.-дней, а руководителем 7 чел.-дней. Общая продолжительность выполнения данной НИР 113 дней (чуть больше 15 недель).

4.5 Комплексная оценка эффективности НИР

4.5.1 Научно-технический эффект разработки

Концепции и технологии, используемые при разработке web-приложений, постоянно развиваются и совершенствуются - оптимизируется использование ресурсов и времени, улучшаются возможности по отображению предоставляемой информации, динамичность и их интерактивность.

Flex является одним из самых распространённых и перспективных инструментов разработки web-приложений. Разрабатываемый в дипломном проекте программный модуль поможет обеспечить контроль качества Flex приложений, а также повысит эффективность тестирования за счёт автоматизации тестовых сценариев.

4.5.2 Экономический эффект

Выгоды, которые может получить потребитель от использования разрабатываемой продукции:

- а) Повышение стабильности разрабатываемого приложения за счёт обеспечения качественного тестирования.
- б) Повышение скорости тестирования за счёт автоматизации труда инженеров по качеству.
- в) Понижение затрат на тестирование, т.к. внедрение разрабатываемого модуля позволяет отказаться от использования платных средств автоматизированного тестирования.

4.5.3 Расчет потребности в начальных инвестициях

Потребность в начальном капитале определяется средствами, израсходованных на НИР – 131 284 р., а также дополнительными средствами, необходимых для приобретения ПЭВМ и принтера. При стоимости

ПЭВМ 20 000 р., сроке ее службы – 5 лет, времени копирования программного обеспечения 0,5 ч. потребность в ПЭВМ составит:

Заключение

В результате проделанной работы стало ясно, что ничего не ясно...

Приложение А Программный код модуля

Листинг А.1 — Код реализация декодера сообщений

```
1 package edu.leti.amf;
2
3 import flex.messaging.io.ClassAliasRegistry;
4 import flex.messaging.io.SerializationContext;
5 import flex.messaging.io.amf.ActionContext;
6 import flex.messaging.io.amf.ActionMessage;
7 import flex.messaging.io.amf.AmfMessageDeserializer;
8 import flex.messaging.io.amf.AmfTrace;
9 import flex.messaging.messages.AcknowledgeMessageExt;
10 import flex.messaging.messages.AsyncMessageExt;
11 import flex.messaging.messages.CommandMessageExt;
12 import org.slf4j.Logger;
13 import org.slf4j.LoggerFactory;
14
15 import java.io.IOException;
16 import java.io.InputStream;
17
18 /**
19  * Класс для сериализации и десериализации amf сообщений
20  *
21  * @author Tedikova O.
22  * @version 1.0
23  */
24 public class MessageDecoder {
25
26     private static final Logger logger =
27         LoggerFactory.getLogger(MessageDecoder.class);
28
29     public MessageDecoder() {
30         ClassAliasRegistry registry = ClassAliasRegistry.getRegistry();
31         registry.registerAlias(AsyncMessageExt.CLASS_ALIAS,
32             AsyncMessageExt.class.getName());
33         registry.registerAlias(AcknowledgeMessageExt.CLASS_ALIAS,
34             AcknowledgeMessageExt.class.getName());
35         registry.registerAlias(CommandMessageExt.CLASS_ALIAS,
36             CommandMessageExt.class.getName());
37     }
38
39     /**
40      * Метод считывает amf сообщение из входного потока и преобразует его в
41      * экземпляр
42      *
43      * @return Action Message
44      */
45 }
```

```

39      * @param inputStream входной поток , содержащий amf сообщение
40      * @return сообщение, преобразованное в ActionMessage
41      * @throws IOException в случае ошибки чтения/записи/
42      * @throws ClassNotFoundException в случае ошибки работы с классами
43      */
44      public ActionMessage getActionMessage(InputStream inputStream) throws
IOException, ClassNotFoundException {
45          AmfMessageDeserializer deserializer = new AmfMessageDeserializer();
46          ActionMessage message = new ActionMessage();
47          ActionContext context = new ActionContext();
48          AmfTrace amfTrace = new AmfTrace();
49          SerializationContext serializationContext =
SerializationContext.getSerializationContext();
50          deserializer.initialize(serializationContext, inputStream,
amfTrace);
51          deserializer.readMessage(message, context);
52          logger.debug("Received message\n" + amfTrace);
53          return message;
54      }
55  }

```

Листинг А.2 — Интерфейс обработки http-запросов

```

1  package edu.leti.jmeter.proxy;
2
3  import org.apache.jmeter.protocol.http.sampler.HTTPSamplerBase;
4  import org.apache.jmeter.samplers.SampleResult;
5  import org.apache.jmeter.testelement.TestElement;
6
7  /**
8   * Интерфейс, отвечающий за обработку http запросов.
9   *
10  * @author Tedikova O.
11  * @version 1.0
12  */
13  public interface SamplerDeliverer {
14      /**
15       * Метод обрабатывает полученный http запрос.
16       *
17       * @param sampler http sampler, созданный на основе полученного запроса.
18       * @param subConfigs настройки тестового элемента.
19       * @param result информация об обработке запроса.
20       */
21      public void deliverSampler(HTTPSamplerBase sampler, TestElement[]
subConfigs, SampleResult result);
22  }

```

Листинг А.3 — Интерфейс прокси сервера

```
1 package edu.leti.jmeter.proxy;
2
3 /**
4  * Прокси представляет собой поток вычитывающий поток данных из сокета.
5  *
6  * @author Tedikova O.
7  * @version 1.0
8  */
9 public interface ProxyInterface {
10
11     /**
12      * Вызывается приложением для начала записи трафика.
13      */
14     void start();
15
16     /**
17      * Вызывается приложением для завершения записи трафика.
18      */
19     void stop();
20
21     /**
22      * Возвращает порт подключения прокси сервера.
23      */
24     int getLocalPort();
25 }
```