

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

Пояснительная записка ?с., ? рис., ? табл., ? источников. Целью настоящего дипломного проекта является разработка программного обеспечения, предоставляющего возможность проведения тестирования взаимодействия Flex приложений с сервером через AMF протокол.

Полученные результаты: разработан программный модуль для тестового фреймворка Apache JMeter, предоставляющий возможность проведения функционального и нагрузочного тестирования взаимодействия Flex приложений с сервером через AMF протокол. Тестирование модуля показало его соответствие техническим требованиям.

Преимуществами разработанного программного обеспечения являются:

- а) возможность записи тестовых сценариев с помощью прокси сервера.
- б) кроссплатформенность;
- в) расширяемость;

Разработанный программный комплекс может быть использован программистами и инженерами по тестированию в различных организациях, принадлежащих сфере IT, в целях повышения качества разрабатываемого программного обеспечения.

Содержание

Введение	8
1 Анализ существующих решений	15
1.1 Обзор утилит для тестирования Flex приложений	15
1.1.1 HP QuickTest Professional	15
1.1.2 IBM Rational Functional Tester	16
1.1.3 NeoLoad	17
1.1.4 Apache JMeter	18
1.2 Выводы	19
2 Проектирование модуля	20
2.1 Требования к разрабатываемому модулю	20
2.1.1 Формулировка требований	20
2.1.2 Анализ требований	21
3 Программная реализация модуля	23
3.1 Выбор технических средств решения задачи	23
3.2 Реализация поддержки AMF протокола	25
3.2.1 Общие сведения о BlazeDS	25
3.2.2 Сериализация и десериализация сообщений	27
3.2.3 AMF клиент	31
3.3 Интеграция с JMeter	32
3.3.1 Общая архитектура JMeter	32
3.3.2 Реализация модуля отправки AMF сообщений	36
3.3.3 Реализация прокси-сервера	38
3.4 Тестирование	38
3.4.1 Модульное тестирование	39
3.4.2 Приёмочное тестирование	40
4 Технико-экономическое обоснование	42
4.1 Концепция экономического обоснования	42
4.2 Рынок и план маркетинга	43
4.2.1 Сегментирование рынка	43
4.2.2 Продвижение товара	43
4.3 Производство продукта	44
4.3.1 Статья «Материалы»	44

4.3.2	Статья «Спецоборудование»	45
4.3.3	Статья «Расходы на оплату труда»	45
4.3.4	Статья «Отчисления на социальные нужды»	46
4.3.5	Статья «Затраты по работам, выполняемым сто- ронними организациями»	46
4.3.6	Статья «Командировочные расходы»	47
4.3.7	Статья «Прочие прямые расходы»	47
4.3.8	Статья «Накладные расходы»	47
4.4	Организационный план проекта	48
4.5	Комплексная оценка эффективности НИР	48
4.5.1	Научно-технический эффект разработки	48
4.5.2	Экономический эффект	49
4.5.3	Расчет потребности в начальных инвестициях	49
4.6	Экономическая эффективность проекта	50
5	Охрана интеллектуальной собственности	52
5.1	Интеллектуальная собственность	52
5.2	Программа для ЭВМ	52
5.3	Авторское право на программу для ЭВМ	53
5.4	Правообладание	55
5.5	Нарушение прав на программу для ЭВМ и базу данных	55
5.6	Право на официальную регистрацию	55
5.7	Процедура официальной регистрации	56
5.8	Заявка на официальную регистрацию	57
5.9	Программный продукт и формы его продажи	58
5.10	Договор на использование программы для ЭВМ	58
	Заключение	59
	Список использованных источников	60
	А Программный код модуля	62
	Б План приёмочных испытаний приложения	83
	Б.1 Объект тестирования	83
	Б.2 Цели и приоритеты тестирования	83
	Б.3 Стратегии тестирования	83
	Б.4 Последовательность работ	83

Б.5	Критерии начала тестирования	83
Б.6	Критерии окончания тестирования	84
Б.7	Тестовое окружение	84
В	Сценарии тестирования приложения	85
В.1	Настройка тестового окружения	85
В.2	Функциональные тесты	85
В.2.1	Проверка записи тестовых запросов	85
В.2.2	Проверка отправки корректного AMF запроса че- рез AMF RPC Sampler	87
В.2.3	Проверка отправки некорректного AMF запроса че- рез AMF RPC Sampler	89
В.2.4	Проверка запуска содержимого Test Plan в несколь- ко потоков	91

Глоссарий

Фреймворк — в информационных системах структура программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Apache JMeter — инструмент для тестирования программного обеспечения, разрабатываемый Apache Jakarta Project.

Java — объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем, приобретённой компанией Oracle).

BlazeDS — серверная Java-технология для передачи данных.

Flex — технология для создания RIA, разработанная компанией Adobe.

Maven — средство для автоматизации сборки проектов.

Обозначения и сокращения

RIA — Rich Internet application, приложения, доступные через сеть Интернет, обладающие особенностями и функциональностью традиционных настольных приложений.

AMF — Action Message Format, бинарный формат обмена данными.

ПО — программное обеспечение.

GUI — Graphical user interface, графический интерфейс пользователя.

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[1].

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[2]. 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 ме-

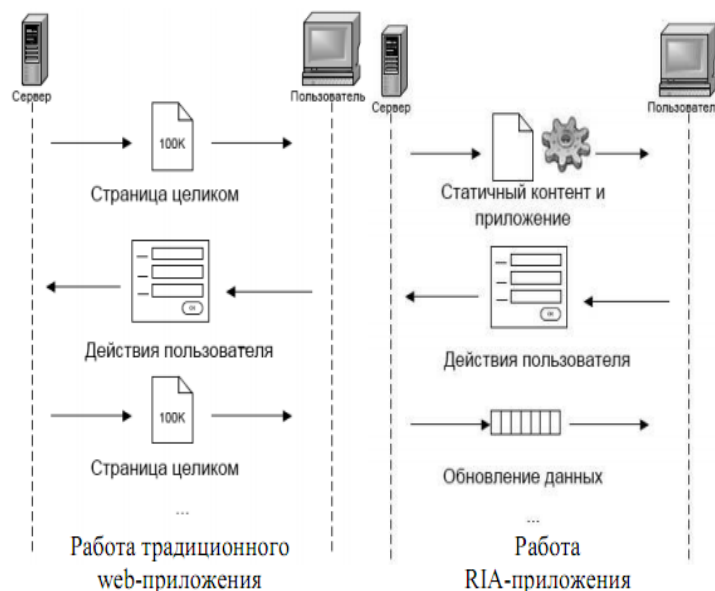


Рисунок 0.2 — Принципы работы традиционного web-приложения и RIA приложения

ре необходимости.[3] 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[4].

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 сокеты, 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[5]:

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[6].

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 является бинарным, 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. Сложность состоит в поддержке уникальности идентификатора сессии каждого клиента. Идентификатор клиента выделяется строго одному подключенному SWF-файлу и прошивается 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

На сегодняшний день существует ряд различных программных комплексов, предоставляющих возможность для проведения функционального, нагрузочного, регрессионного тестирования различного рода приложений. Проведём обзор наиболее распространённых и доступных утилит и укажем их функционал по обеспечению тестирования в области Flex технологий.

1.1 Обзор утилит для тестирования Flex приложений

1.1.1 HP QuickTest Professional

HP QuickTest Professional (QTP) — один из инструментов автоматизации функционального тестирования, является флагманским продуктом 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 приложение 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.

1.1.2 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[7]. 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.) недо-

статочно, 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.

1.1.3 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.

1.1.4 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 автоматической сборки проектов Maven и Ant, через которые осуществляется интеграция со многими серверами сборки, такими как Jenkins и Hudson.

JMeter является бесплатным кросс-платформенным Java-приложением с открытым исходным кодом и удобным API. Существует боль-

шое количество плагинов к JMeter, значительно расширяющих его базовую функциональность.

1.2 Выводы

Был проведён обзор ряда тестовых утилит, указаны их ключевые особенности, а также подробно рассмотрены возможности в области тестирования Flex приложений.

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 модуль должен предоставлять возможность записи действий пользователя, производимых с тестируемым приложением (проксирование запросов). Проксирование является важным функциональным требованием, так как на этапе подготовки тестирования позволит создавать тестовый сценарий, на основе действий пользователя с приложением.

б) Программный модуль должен предоставлять возможность отправки AMF сообщений через http соединение по указанному url

в) Программный модуль должен полностью поддерживать протокол AMF с целью кодирования, декодирования и замены сообщений. Все полученные или отправленные сообщения должны предоставляться пользователю в человекочитаемом формате.

г) Реализация программного модуля должна полностью поддерживать основные принципы нагрузочного тестирования имитирующего работу большого количества пользователей одновременно.

д) Разрабатываемое приложение должно интегрироваться с системами автоматической сборки проектов, такими как maven и ant

е) 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 авто-

мативированных 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.

ж) Кроссплатформенность. Созданная нами тестовая утилита должна работать под управлением различных операционных систем.

з) 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.

2.1.2 Анализ требований

Проанализируем сформулированные требования к разрабатываемому модулю.

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 требова-

ния, 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.

Для наглядности и удобства сведём их в таблицу.

Таблица 2.1 — Список требований к модулю

Описание	Приоритет	Трудоёмкость	Риск
Возможность записи действий пользователя, производимых с тестируемым приложением(проксирование запросов)	Высокий	Высокая	Средний
Возможность отправки AMF сообщений через http соединение по указанному url	Высокий	Высокая	Средний
Поддержка протокола AMF с целью кодирования, декодирования и замены сообщений	Высокий	Высокая	Средний
Поддержка основных принципов нагрузочного тестирования	Высокий	Высокая	Средний
Интеграция с системами автоматической сборки проектов	Высокий	Высокая	Средний
Возможность запуска тестов из командной строки	Высокий	Высокая	Средний
Кроссплатформенность	Высокий	Высокая	Средний
Условия распространения продукта	Высокий	Высокая	Средний

3 Программная реализация модуля

3.1 Выбор технических средств решения задачи

Как показали результаты обзора существующих тестовых фреймворков, на данный момент нет решения полностью удовлетворяющего поставленным требованиям. Однако многие рассмотренные программные комплексы уже предоставляют часть необходимого нам функционала, поэтому для решения поставленной на дипломный проект задачи нет необходимости создавать тестовую утилиту с нуля, разумнее будет выбрать одно из существующих решений, и доработать его. Чтобы выбрать из предложенного разнообразия подходящий продукт, составим таблицу, где укажем, в какой степени каждый тестовый фреймворк удовлетворяет поставленным требованиям.

Анализ показал, что оптимальным выбором, в наибольшей степени удовлетворяющим поставленным требованиям, является Apache JMeter. Именно на базе его функционала будет реализовано решение задачи дипломного проекта.

Разработка модуля будет осуществляться с использованием языка Java, что обусловлено одним из требований технического задания — кроссплатформенность приложения. Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор, что обеспечивает полную независимости байт-кода от операционной системы и оборудования и позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Также выбранный нами тестовый фреймворк Apache JMeter является стопроцентным Java приложением, поэтому для интеграции с ним целесообразно использовать именно этот язык.

На текущий момент основным средством обеспечения взаимодействия Flex клиентов с Java приложениями является технология BlazeDS — серверная Java-технология для передачи данных, поддерживающая AMF протокол. BlazeDS является бесплатным приложением с открытым

исходным кодом, разработано компанией Adobe. В силу распространённости BlazeDS, одно из основных функциональных требований к реализации модуля — обработка AMF сообщений — будет решаться с помощью средств именно этой технологии.

В качестве инструмента автоматизации сборки проектов был выбран Apache Maven — фреймворк для автоматизации сборки проектов, специфицированных на XML-языке POM (Project Object Model). Основными преимуществами Maven являются:

а) Независимость от OS. Сборка проекта происходит в любой операционной системе. Файл проекта один и тот же.

б) Управление зависимостями. Редко какие проекты пишутся без использования сторонних библиотек(зависимостей), которые зачастую тоже в свою очередь используют библиотеки разных версий. Мавен позволяет управлять такими сложными зависимостями, что позволяет разрешать конфликты версий и в случае необходимости легко переходить на новые версии библиотек.

в) Возможна сборка из командной строки. Такое часто необходимо для автоматической сборки проекта на сервере (Continuous Integration).

г) Хорошая интеграция с средами разработки. Основные среды разработки на java легко открывают проекты которые собираются с помощью maven. При этом зачастую проект настраивать не нужно — он сразу готов к дальнейшей разработке. Как следствие - если с проектом работают в разных средах разработки, то maven удобный способ хранения настроек. Настроечный файл среды разработки и для сборки один и тот же — меньше дублирования данных и соответственно ошибок.

д) Декларативное описание проекта. В файлах проекта pom.xml содержится его декларативное описание, а не отдельные команды.

Эффективность разработки программного обеспечения в любом современном проекте подразумевает возможность вести разработку параллельно с другими участниками проекта. Для оптимизации совместной работы над дипломным проектом было принято решение о размещении всех файлов проекта в репозитории системы контроля версий Git. Git —

это быстрая, масштабируемая, распределенная система управления версиями с большим набором команд, которые обеспечивают как операции верхнего уровня, так и полный доступ к внутренним механизмам.

В качестве среды разработки был выбрана IntelliJIdea[8].

3.2 Реализация поддержки AMF протокола

3.2.1 Общие сведения о BlazeDS

BlazeDS — серверная Java-технология для передачи данных. Предоставляет ряд сервисов, которые позволяют приложениям клиента взаимодействовать с сервером, а также осуществляет передачу данных между несколькими клиентами, подключенными к серверу BlazeDS, в режиме реального времени. BlazeDS приложение состоит из двух частей: клиентского приложения и серверного J2EE web-приложения. Архитектура клиента представлена на Рис. 3.1.

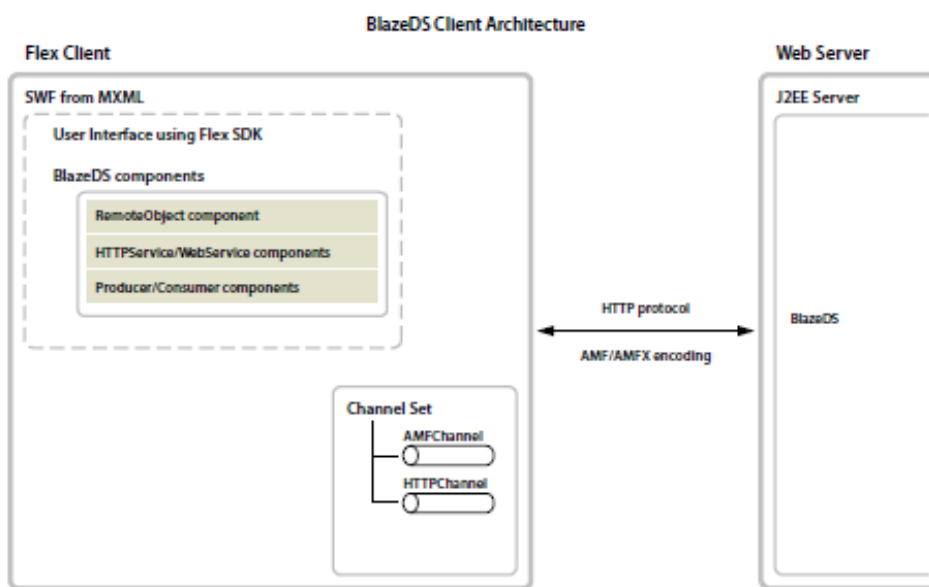


Рисунок 3.1 — Архитектура клиента BlazeDS

Клиентское приложение BlazeDS обычно представляет собой Adobe Flex или AIR приложение. В его состав входят:

- а) Пользовательский интерфейс приложения. Создаётся с помощью Flex SDK;
- б) Один или несколько компонентов BlazeDS:

- 1) RemoteObject — компонент, предоставляющий клиентскому приложению доступ к методам Java-объектов на стороне сервера;
- 2) HTTPService — компонент, позволяющий клиентскому приложению с помощью http запросов взаимодействовать с JSP, сервлетами, ASP страницами через сервер BlazeDS;
- 3) WebService — компонент, предназначенный для взаимодействия с веб-сервисами;
- 4) Producer — компонент-отправитель сообщений, предназначен для взаимодействия с сервером сообщений;
- 5) Consumer — компонент-получатель сообщений, предназначен для взаимодействия с сервером сообщений.

в) Набор каналов. На стороне клиента определяются каналы, которые инкапсулируют соединение между Flex клиентом и сервером BlazeDS. Для клиентского приложения задаётся набор каналов, упорядоченных по предпочтению. Flex компонент пытается подключиться по первому каналу, указанному в списке, и в случае неудачи выбирает следующий канал и т.д, до тех пор пока соединение не будет установлено, либо список каналов не кончится. Flex клиенты могут использовать различные типы каналов, такие как AMFChannel и HTTPChannel. AMFChannel использует бинарный AMF протокол, а HTTPChannel - небинарный формат AMFX (AMF, преобразованный в XML). Выбор канала зависит от ряда факторов, например от типа создаваемого приложения, формата передачи данных, требуемого размера сообщений.

Архитектура сервера BlazeDS представлена на Рис. 3.2.

BlazeDS сервер базируется на технологии J2EE. Взаимодействие клиента и сервера BlazeDS происходит следующим образом : Flex клиент посылает запрос по определённому каналу, далее запрос направляется в соответствующий каналу компонент endpoint, который является точкой обработки получаемых сервером сообщений различного типа. Затем сообщение декодируется и проходит через ряд Java-объектов - MessageBroker , Service object, Destination object и Adapter object. Adapter object либо обрабатывает запрос локально, либо связывается с какой-либо backend

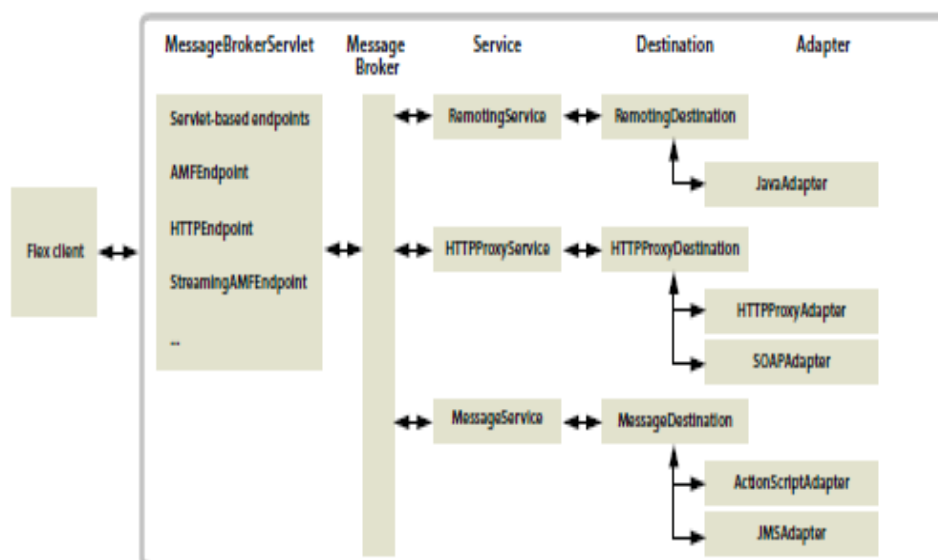


Рисунок 3.2 — Архитектура сервера BlazeDS

системой или удалённым сервером. После запроса происходит обратный процесс.

Таким образом для обеспечения поддержки AMF протокола в разрабатываемом программном обеспечении, необходимо будет реализовать модуль, осуществляющий сериализацию сообщений перед их отправкой от клиента к серверу и наоборот, десериализацию полученных клиентом сообщений сервера.

3.2.2 Сериализация и десериализация сообщений

Рассмотрим подробнее спецификацию AMF.

AMF — бинарный формат, используемый для сериализации структурированных данных, таких как объекты Action Script или XML, и обмена сообщениями между Adobe Flash клиентом и удалённым сервисом[9]. Action Message Format более экономичен по трафику по сравнению с XML и позволяет передавать типизированные объекты. Первая версия протокола, AMF0, была применена в Flash Player 6 в 2001 и оставалась неизменной в реализациях ActionScript 2.0 в Flash Player 7 и Flash Player 8. В Flash Player 9 был внедрён Action Script 3.0 с усовершенствованной виртуальной машиной ActionScript Virtual Machine (AVM+). В новой версии протокола, AMF3, оптимизирован формат сообщений и введена под-

держка новых типов. Далее будет рассматриваться именно обновлённая версия протокола.

AMF протокол поддерживает следующие типы данных:

а) `undefined` — любой неспецифицированный протоколом формат данных;

б) `null`;

в) `boolean`;

г) `integer` — в ActionScript 3.0 `integer` представляет собой 29-битное число переменной длины без знака или 28-битное со знаком. Если выделенных битов не достаточно для представления числа, то AVM+ сериализует его как тип `double`;

д) `double` — 8-байтовое число с плавающей точкой в формате IEEE 754 с порядком байтов от старшего к младшему (`network byte order`)

е) `string` — строка в кодировке UTF-8, литерал или ссылка;

ж) `xml-doc` — данные в xml формате в кодировке UTF-8. При сериализации преобразуются в строку;

з) `date` — сериализуется как число миллисекунд, прошедших с 00:00:00 первого января 1970 в часовом поясе UTC, данные о локальном часовом поясе не посылаются;

и) `array` — массивы данных;

к) `object` — `amf3` обрабатывает Action Script объекты, а также разработанные пользователем классы, которые можно разделить на следующие группы:

- 1) анонимные — Action Script объекты или экземпляры пользовательских классов, не имеющие зарегистрированных в протоколе псевдонимов. В процессе десериализации анонимные объекты будут интерпретироваться как `Object`;
- 2) типизированные — экземпляры классов с зарегистрированными в протоколе псевдонимами;
- 3) динамические — экземпляры классов, общедоступные переменные которых могут динамически изменяться во время выполнения кода;

- 4) сериализуемые — экземпляры классов, реализующих интерфейс `flash.utils.IExternalizable`, которые полностью контролируют сериализацию своих членов.
- л) `xml` — в Action Script 3.0 введена поддержка XML синтаксиса E4X;
- м) `byte-array` — в Action Script 3.0 введён новый формат хранения массива байтов, `ByteArray`. AMF3 сериализует данные этого типа с использованием 29-битного значения, передаваемого в качестве длины массива байтов.

AMF сообщение содержит информацию об отдельной транзакции. Оно определяет вызываемую удалённую операцию, операцию, выполняемую клиентом в случае успеха или ошибки запроса, и используемые при этом данные. Структуры сообщений, представляющих запросы клиента и ответы сервера, одинаковы.

Первым полем AMF сообщения является `target URI`, которое описывает, какая операция, функция или метод должны быть вызваны. Спецификация AMF не определяет точного формата этого поля — его формат зависит от конкретной реализации используемого сервера.

Вторым полем AMF сообщения является `response URI`, определяющее имя операции, которая будет вызвана в зависимости от ответа на запрос клиента.

Третье поле AMF сообщения — длина тела сообщения в байтах.

Четвёртым заключительным полем AMF сообщения является тело сообщения. Оно содержит фактические данные, связанные с выполняемой операцией. Если сообщение является запросом клиента, то оно должно содержать параметры, передаваемые удалённому методу. Если сообщение является ответом сервера, то оно должно содержать либо ответ на запрос, либо сообщение об ошибке.

Отправка AMF сообщений осуществляется пакетами, каждый из которых имеет следующую структуру:

- а) версия AMF протокола;
- б) количество заголовков пакета;
- в) заголовки пакета;

- г) число сообщений пакета;
- д) сообщения пакета;

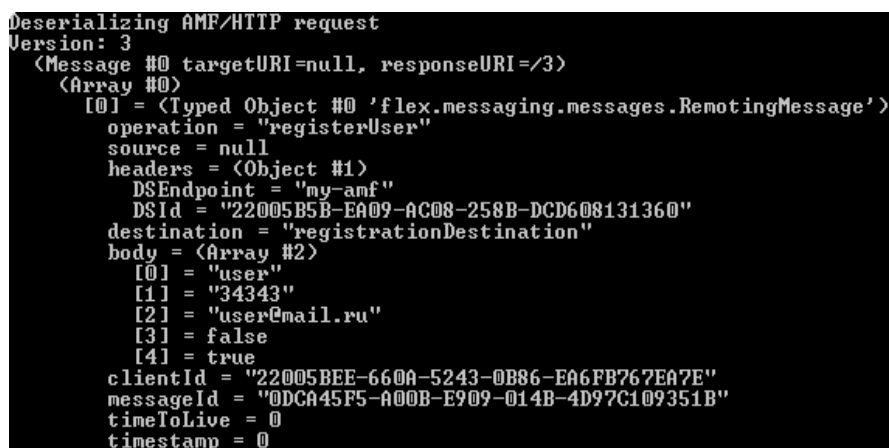
В библиотеке BlazeDS существует класс, соответствующий структуре AMF пакетов — `ActionMessage`. Чтобы получить экземпляр `ActionMessage` из входного потока данных, используется класс `AmfMessageDeserializer`. Следующий код демонстрирует этот процесс:

Листинг 3.1 — Получение `ActionMessage` из входного потока данных

```
1 AmfMessageDeserializer deserializer = new AmfMessageDeserializer();
2 ActionMessage message = new ActionMessage();
3 ActionContext context = new ActionContext();
4 AmfTrace amfTrace = new AmfTrace();
5 SerializationContext serializationContext =
    SerializationContext.getSerializationContext();
6 deserializer.initialize(serializationContext, inputStream, amfTrace);
7 deserializer.readMessage(message, context);
```

Метод `initialize(SerializationContext context, InputStream in, AmfTrace trace)` устанавливает контекст для чтения данных из указанного входного потока. Метод `readMessage(ActionMessage m, ActionContext context)` считывает десериализованные данные в переданный ему в качестве параметра экземпляр `ActionMessage`.

Пример десериализованного сообщения представлен на Рис. 3.3.



```
Deserializing AMF/HTTP request
Version: 3
{Message #0 targetURI=null, responseURI=/3}
{Array #0}
[0] = {Typed Object #0 'flex.messaging.messages.RemotingMessage'}
  operation = "registerUser"
  source = null
  headers = {Object #1}
    DSEndpoint = "my-amf"
    DSId = "22005B5B-EA09-AC08-258B-DCD608131360"
  destination = "registrationDestination"
  body = {Array #2}
    [0] = "user"
    [1] = "34343"
    [2] = "user@mail.ru"
    [3] = false
    [4] = true
  clientId = "22005BEE-660A-5243-0B86-EA6FB767EA7E"
  messageId = "0DCA45F5-A00B-E909-014B-4D97C109351B"
  timeToLive = 0
  timestamp = 0
```

Рисунок 3.3 — Десериализованное сообщение

3.2.3 AMF клиент

В BlazeDS существует механизм, Java AMF Client, позволяющий совершать удалённые вызовы методов и обрабатывать ответы сервера. Преимущество использования Java AMF Client заключается в том, что сериализация и десериализация AMF сообщений, отправляемых клиентом и сервером, а также установка http соединения, полностью обеспечивается данной технологией.

Классы, реализующие функционал Java AMF Client, находятся в пакете `flex.messaging.io.amf.client`. Основным классом является `AMFConnection`. Пример его работы показан ниже.

Листинг 3.2 — Получение `ActionMessage` из входного потока данных

```
1 String url = "http://localhost:8400/team/messagebroker/amf";
2 AMFConnection amfConnection = new AMFConnection();
3 try {
4     amfConnection.connect(url);
5 } catch (ClientStatusException e) {
6     logger.error("Couldn't connect to " + url, e);
7 }
8 try {
9     Object result = amfConnection.call("remoting_AMF.echo", "echo me1");
10 } catch (ClientStatusException cse) {
11     handleException(cse);
12 } catch (ServerStatusException sse) {
13     handleException(sse);
14 }
```

`AMFConnection` устанавливает соединение с удалённым объектом по указанному URL с помощью метода `connect()`. В случае успешной установки соединения метод `call()` отправляет AMF запрос пользователю, в качестве параметров метод принимает имя вызываемого на стороне сервера метода и его параметры, представленные в виде массива объектов.

В данном примере возможно появление двух видов исключительных ситуаций:

а) `ServerStatusException` — в случае появления сообщения об ошибке от сервера.

б) `ClientStatusException` — в случае ошибки установки соединения с сервером или при непредвиденном разрыве соединения.

3.3 Интеграция с JMeter

3.3.1 Общая архитектура JMeter

Структура проекта JMeter состоит из следующих директорий:

- а) bin — содержит в себе .bat и .sh файлы для запуска JMeter, файл ApacheJMeter.jar и файлы настроек;
- б) docs — директория, содержащая документацию по проекту;
- в) extras — дополнительные файлы для утилиты ant;
- г) lib — jar файлы библиотек, используемых в JMeter;
- д) lib/ext — jar файлы ядра и отдельных компонентов JMeter;
- е) src — исходные коды JMeter;
- ж) test — юнит-тесты;
- з) xdocs — xml файлы для документации (JMeter генерирует документацию из xml).

Общая логическая структура модулей проекта представлена на Рис. 3.4:

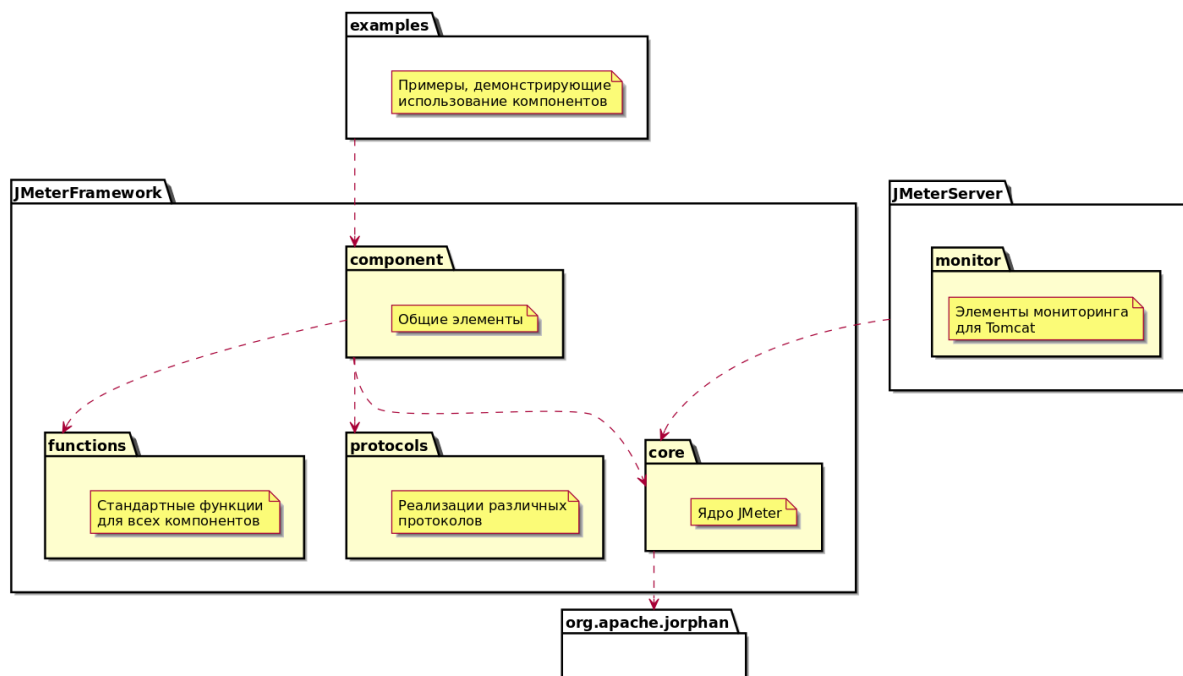


Рисунок 3.4 — Диаграмма модулей

где,

- а) `component` — директория, содержащая общие для различных протоколов элементы, такие как визуализеры, соответствия и т.д;
- б) `core` — ядро JMeter, содержит базовые интерфейсы и абстрактные классы;
- в) `examples` — примеры, демонстрирующие использование компонентов фреймворка;
- г) `functions` — стандартные функции, используемые всеми компонентами;
- д) `jorphan` — утилитные классы;
- е) `monitor` — элементы мониторинга сервера Tomcat 5;
- ж) `protocol` — содержит реализации компонентов Jmeter для различных протоколов.

В архитектуре JMeter ядро, содержащее в себе интерфейсы и абстрактные классы, а также базовый функционал, отделены от конкретных реализаций компонентов для различных протоколов. Это сделано для того, чтобы разработчики могли добавлять поддержку новых протоколов без сборки всего приложения. Таким образом, для того, чтобы добавить в JMeter элементы тестирования Flex приложений, нужно будет переопределить несколько базовых компонентов JMeter, собрать jar файл модуля, и поместить его в директорию `lib/ext` — новый функционал будет автоматически подхвачен JMeter.

Прежде чем приступить к созданию различных компонентов, опишем ряд общих правил их реализации, которые необходимы для того, чтобы элемент правильно работал в среде JMeter. В основном это относится к графическому интерфейсу пользователя (GUI).

На Рис. 3.5 изображена диаграмма объектов, отображающая MVC модель.

В JMeter код GUI элемента отделён от функционального кода элемента, поэтому реализуя новый компонент следует создавать отдельные классы для рабочего функционала и графического представления. GUI элемент, в зависимости от его предназначения, должен расширять один из представленных в таблице абстрактных классов.

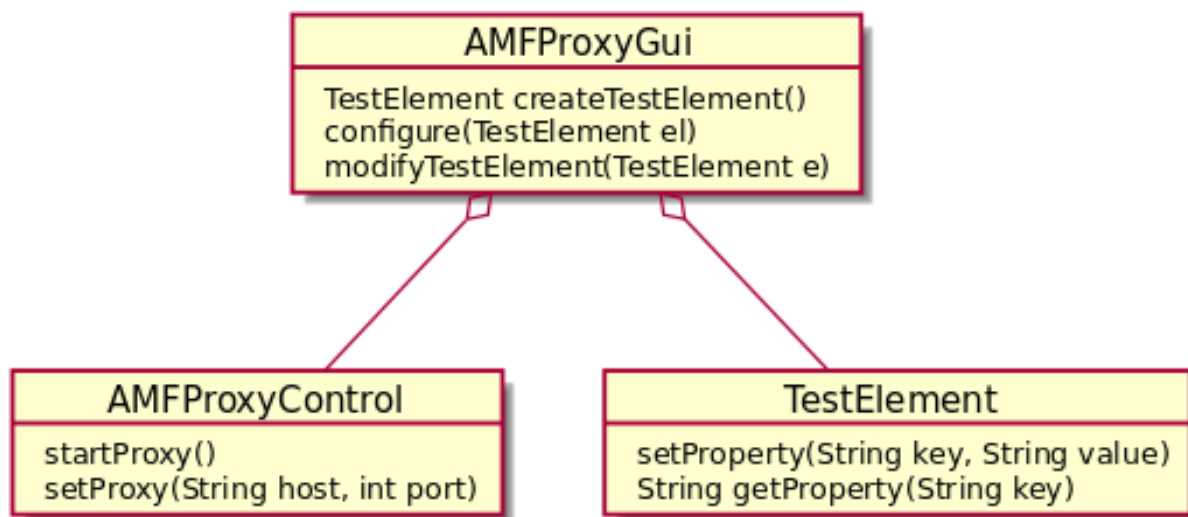


Рисунок 3.5 — Диаграмма объектов

Таблица 3.1 — Абстрактные классы GUI JMeter

Имя класса	Предназначение
AbstractSamplerGui	Класс GUI для элемента Sampler, предназначенного для отправки запросов серверу
AbstractAssertionGui	Класс GUI для элемента Assertion, осуществляющего проверку результатов выполнения запроса
AbstractConfigGui	Класс GUI для Configuration Element, применяемого для настройки параметров Sampler
AbstractControllerGui	Класс GUI для элементов Controller, осуществляющих управление вызовом содержимого Test Plan
AbstractPostProcessorGui	Класс GUI для Post-Processor Element, обозначающего действия, которые должны быть выполнены после отправки запроса

Продолжение на след. стр.

Продолжение таблицы 3.1

AbstractPreProcessorGui	Класс GUI для Pre-Processor Element, обозначающего действия, которые должны быть выполнены перед отправкой запроса
AbstractVisualizer	Класс GUI для элемента Listener, предоставляющего возможность наблюдать результаты выполнения запросов
AbstractTimerGui	Класс GUI для элемента Timer, предназначенного для установки временных задержек между запуском элементов Test Plan

Следующим шагом является реализация метода `getResourceLabel()`. Этот метод должен возвращать имя ресурса, представляющего компонент.

Чтобы GUI создаваемого вами тестового элемента соответствовало стилю JMeter, следует добавить в него стандартную рамку JMeter, которая создаётся следующим образом — `setBorder(makeBorder())`. Также необходимо создать панель с именем элемента методом `makeTitlePanel()`. Обычно её располагают в верхней центральной части панели элемента.

Важным пунктом является реализация метода `public void configure(TestElement el)`, который отвечает за отображение параметров тестового элемента в GUI. Первой строчкой в методе должен быть вызов `super.configure(e)`, что обеспечит выполнение некоторых стандартных действий, как например отображение имени элемента. Обратное действие — передача данных из GUI в тестовый элемент — обеспечивается методом `public void modifyTestElement(TestElement e)`, который также нужно реализовать. В имплементацию этого метода тоже включается строчка `super.configureTestElement(e)`.

Другой необходимый в создании GUI метод — `public TestElement createTestElement()`. Он должен создать новый экземпляр тестового элемента и

передавать его в метод `modifyTestElement(TestElement)` в качестве параметра.

Все вышеобозначенные правила разграничения GUI и функционала элемента облегчают разработчикам реализацию графического интерфейса. Хотя программист и не освобождается от процедуры создания и размещения компонентов графического интерфейса, всё взаимодействие GUI и соответствующего ему тестового элемента обеспечивается JMeter.

3.3.2 Реализация модуля отправки AMF сообщений

Согласно одному из требований технического задания, разрабатываемое программное обеспечение должно предоставлять возможность отправки AMF сообщения серверу. Данное требование осуществляется реализацией такого компонента JMeter, как сэмплер(Sampler). Сэмплеры в JMeter предназначены для отправки запросов серверу и ожидания ответов от них.

Чтобы добавить свой собственный сэмплер в JMeter, необходимо создать класс, наследующий абстрактный класс `AbstractSampler` и реализующий метод `public SampleResult sample(Entry e)`. Именно этот метод будет вызываться у каждого сэмплера, входящего в тест план во время его прогона. Метод `sample` должен осуществлять подключение к серверу, ожидание ответа от него и возвращать экземпляр класса `SampleResult`, содержащий в себе информацию о выполнении запроса.

В рамках решения этой задачи был создан класс `AmfRPCSampler`. В методе `sample` данного класса производится вызов метода удалённого объекта на стороне сервера с помощью AMF клиента (алгоритм работы AMF клиента описан в разделе 3.2.3). Метод возвращает экземпляр класса `AmfRPCSamplerResult`, в который по ходу выполнения метода `sample` записывается информация о передаваемых в запросе данных, ответ сервера, а также статус выполнения запроса (в случае, если отсет сервера получен и не содержит сообщений обошибках, запрос считается успешно выполненным).

Также метод `sample` класса `AmfRPCSampler` отвечает за одно из важных технических требований к модулю — возможность проведения

полноценных нагрузочных тестов. В JMeter имитация работы с приложением большого числа пользователей одновременно осуществляется за счёт запуска тест-плана в несколько потоков и реализуемый сэмплер должен обеспечивать уникальность идентификатора сессии AMF клиента с сервером для каждого потока, выполняющего тест-план, а также гарантировать его сохранность в рамках одного потока. Данная задача решается за счёт введения в классе AmfRPCSampler статической переменной `ThreadLocal<AMFConnection> AMF_CONNECTION_THREAD_LOCAL`. Работа с данной переменной происходит следующим образом. В методе `sample` проверяется текущее значение `AMF_CONNECTION_THREAD_LOCAL` вызовом метода `AMF_CONNECTION_THREAD_LOCAL.get()`, который возвращает экземпляр `AMFConnection` для текущего потока. Если соединение `AMFConnection` для данного потока уже было создано, то дальнейшая работа осуществляется с полученным экземпляром `AMFConnection`. Если вызов `AMF_CONNECTION_THREAD_LOCAL.get()` возвращает `null`, то создаётся новый экземпляр `AMFConnection`, производится попытка подключения к серверу и значение `AMFConnection` записывается в `AMF_CONNECTION_THREAD_LOCAL`. Таким образом все элементы `AmfRPCSampler`, запускаемые одним потоком, будут работать с одним и тем же экземпляром `AMFConnection`, а остальные потоки будут использовать другие соединения.

Разработанный `AmfRPCSampler` также имеет графический интерфейс пользователя, реализованный в классе `AmfRPCSamplerGui`. `AmfRPCSamplerGui` создан согласно правилам реализации GUI в JMeter и расширяет класс `AbstractSamplerGui`. Интерфейс содержит следующие компоненты для ввода данных, необходимых для вызова метода объекта на стороне сервера:

- а) `endpointUrlField` — поле для URL, по которому отправляется запрос;
- б) `amfCallField` — поле для имени процедуры, которая должна быть вызвана;
- в) `parametersPanel` — таблица параметров вызываемого метода.

3.3.3 Реализация прокси-сервера

Создание прокси-сервера необходимо для возможности записи трафика между приложением и сервером и дальнейшего использования записанных запросов в тест-плане. В рамках модуля реализован прокси сервер, который запускается по адресу localhost:port (порт указывается пользователем), обрабатывает полученные http запросы и отправляет их адресату. В случае, если тело полученного http запроса содержит AMF сообщение, происходит его десериализация с использованием AmfMessageDeserializer и создаётся экземпляр AmfRPCSampler, в который записывается информация из AMF сообщения. Все созданные во время работы прокси-сервера AMF запросы могут быть перенесены в тест план. На Рис. 3.6 изображена диаграмма последовательности, отображающая процесс записи AMF запросов.

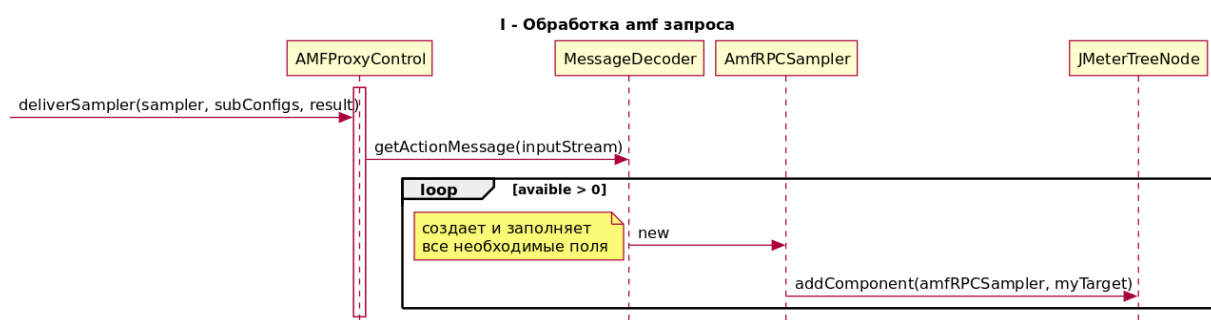


Рисунок 3.6 — Обработка запроса

GUI прокси-сервера содержит поле для ввода порта, на котором сервер будет запущен, а также кнопки для запуска и остановки сервера.

3.4 Тестирование

Тестирование программного обеспечения — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом[10]. В более широком смысле, тестирование — это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов, выполнению тестирования и анализу полученных результатов. Тестирование является важным и неотъемлемым этапом жизненного цикла программного обеспечения,

оно позволяет выявить ошибки уже на ранних стадиях разработки, что сокращает время разработки и повышает надежность программного продукта. При разработке модуля использовались следующие виды тестирования:

- а) Компонентное (модульное) тестирование(Unit Testing);
- б) Приёмочное тестирование.

3.4.1 Модульное тестирование

Компонентное (модульное) тестирование(Unit Testing) проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.)[11]. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. Этот тип тестирования обычно выполняется программистами[12].

Во процессе реализации модуля было осуществлено покрытие классов, заключающих в себе основной функционал, юнит-тестами (классы, реализующие GUI, тестами не покрывались). Тестирование проводилось с помощью JUnit — библиотеки для модульного тестирования программного обеспечения на языке Java. Использование Maven в разработке приложения гарантирует, что все созданные юнит-тесты будут автоматически запускаться перед сборкой модуля, и в случае, если хотя бы один тест не выполнится, сборка приложения не будет запущена.

Как и любая технология тестирования, модульное тестирование не позволяет отловить все ошибки программы. В самом деле, это следует из практической невозможности трассировки всех возможных путей выполнения программы, за исключением простейших случаев. Кроме того, происходит тестирование каждого из модулей по отдельности. Это

означает, что ошибки интеграции, системного уровня, функций, исполняемых в нескольких модулях не будут определены. Кроме того, данная технология бесполезна для проведения тестов на производительность. Таким образом, модульное тестирование более эффективно при использовании в сочетании с другими методиками тестирования.

3.4.2 Приёмочное тестирование

Приемочное тестирование или Приемо-сдаточное испытание (Acceptance Testing) представляет собой формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

- а) определения удовлетворяет ли система приемочным критериям;
- б) вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

Приемочное тестирование выполняется на основании набора типичных тестовых случаев и сценариев, разработанных на основании требований к данному приложению. Решение о проведении приемочного тестирования принимается, когда:

- а) продукт достиг необходимого уровня качества;
- б) заказчик ознакомлен с Планом Приемочных Работ (Product Acceptance Plan);

Фаза приемочного тестирования длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения.

План Приемочных Испытаний - это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Также важным этапом в подготовке приёмочного тестирования является проектирование тестовых случаев (Test Cases) — артефактов,

описывающих совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. Чаще всего test case состоит из трёх частей:

а) предусловия — список действий, которые приводят систему к состоянию пригодному для проведения основной проверки, либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состоянии;

б) шаги теста — список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям;

в) Expected Results — описание состояния системы, в котором она должна находиться после выполнения шагов теста.

Набор тестов должен быть спроектирован таким образом, чтобы обеспечивать оптимальное тестовое покрытие тестируемого приложения.

План приёмочных испытаний, а также набор тест кейзов, используемых при тестировании разработанного модуля находятся в приложении к к диплому.

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:

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

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

Дополнительная зарплата рассчитывается по формуле:

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

где $C_{\text{зо}}$ – основная зарплата, $H_{\text{д}}$ – норматив дополнительной заработной платы.

Производим расчет основной и дополнительной зарплат на основании следующих данных:

- а) трудоемкость выполнения работ исполнителем $T_{\text{исп}} = 106$ чел.-дн.;
- б) трудоемкость выполнения работ СНС $T_{\text{рук}} = 7$ чел.-дн.;
- в) месячный оклад инженера $C_{\text{зо.мес.исп}} = 15\,000$ р.;
- г) месячный оклад руководителя $C_{\text{зо.мес.рук}} = 20\,000$ р.;
- д) норматив дополнительной заработной платы $H_{\text{д}} = 12\%$.

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

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

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

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

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

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

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:

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

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

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 распространенных 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:

$$K_{\text{ПЭВМ}} = \frac{P_{\text{ПЭВМ}} \cdot T_{\text{маш}}}{T_{\text{сл}} \cdot k}, \quad (4.15)$$

где $P_{\text{ПЭВМ}}$ - цена ПЭВМ, р; $T_{\text{сл}}$ – срок службы ПЭВМ, лет; k - число рабочих часов в году, $T_{\text{маш}}$ – время использования ПЭВМ при разработке программного обеспечения.

В данном случае потребность в начальном капитале на ПЭВМ для реализации проекта составит:

$$K_{\text{ПЭВМ}} = \frac{20000 \cdot 824}{1984 \cdot 5}, \quad (4.16)$$

Аналогично рассчитывается потребность в начальном капитале для принтера:

$$K_{\text{ПЭВМ}} = \frac{6000 \cdot 5}{1984 \cdot 5}, \quad (4.17)$$

Затраты на разработку программного обеспечения предполагается покрыть за счет собственных средств.

Результаты расчета потребности в начальном капитале приведены в табл. 4.3.

Таблица 4.3 — Потребность в начальном капитале

Статьи	Сумма, р.
проектирование системы	131 284
ПЭВМ	1 661
принтер	3
ИТОГО:	132 948

4.6 Экономическая эффективность проекта

Показатель рентабельности инвестиций (ROI – Return On Investments) определяется как отношение среднегодовой прибыли к суммарным инвестиционным затратам в проект.

Период возврата (срок окупаемости) инвестиций определяет промежуток времени от момента первоначального вложения капитала в проект, до момента, когда нарастающий итог суммарного чистого дохода становится равным нулю. Для определения периода возврата можно воспользоваться данными прогноза движения денежных средств и установить инвестиционный интервал, после которого показатель, определяемый как нарастающий итог ЧДП, становится положительной величиной. Этот интервал и определяет срок окупаемости инвестиций. Очевидно, что чем меньше период возврата инвестиций, тем проект является более экономически привлекательным.

Чистая текущая стоимость проекта NPV (Net Present Value) рассчитывается как разность дисконтированных денежных потоков поступлений и платежей, производимых в процессе реализации проекта за весь инвестиционный период.

Положительное значение NPV свидетельствует о целесообразности принятия решения о финансировании и реализации проекта, а при

сравнении альтернативных вариантов вложений экономически выгодным считается вариант с наибольшей величиной чистого дисконтированного потока.

Проект считается экономически выгодным, если внутренняя рентабельность превышает минимальный уровень рентабельности, установленный для данного проекта. Кроме того, этот показатель определяет максимально допустимую ставку ссудного процента (стоимости заемного капитала), при которой кредитование проекта осуществляется безубыточно, т. е. без использования для выплат за кредит части прибыли, полученной на собственный инвестированный капитал.

Разработанный модуль не является конечным коммерческим продуктом, но в то же время, он обеспечивает поддержку неотъемлемой части жизненного цикла программного обеспечения — тестирования, вследствие чего рассчитать экономическую эффективность не представляется возможным.

Таким образом, в процессе анализа экономической целесообразности данного проекта были получены следующие характеристики:

- а) трудоемкость НИР: исполнителя – 106 чел.-дн, руководителя НИР – 7 чел.-дн.
- б) себестоимость НИР составляет 131 284 руб.

По итогам расчетов можно уверенно сказать, что создание собственного программного средства стало разумным шагом в направлении облегчения тестирования Flex приложений и увеличении производительности труда инженеров по качеству.

5 Охрана интеллектуальной собственности

5.1 Интеллектуальная собственность

Согласно определению интеллектуальной собственности, принятому в российском законодательстве, а также на основании определения Стокгольмской конференции от 14 июля 1967 г., компьютерные программы относятся к объектам интеллектуальной собственности. Компьютерным программам предоставляется охрана нормами авторского права как литературным произведениям в соответствии с Бернской конвенцией. В Российской Федерации вопросы предоставления правовой охраны программам регулируются Гражданским кодексом РФ, Часть 4 (ГК РФ Ч.4).

5.2 Программа для ЭВМ

Под программой для ЭВМ понимается «... представленная в объективной форме совокупность данных и команд, предназначенных для функционирования ЭВМ и других компьютерных устройств в целях получения определенного результата». Кроме того, в понятие программы для ЭВМ входят «...подготовительные материалы, полученные в ходе разработки программы для ЭВМ, и порождаемые ею аудиовизуальные отображения» [2, ст. 1261]. С точки зрения программистов и пользователей программа для ЭВМ представляет собой детализацию алгоритма решения какой-либо задачи и выражена в форме определенной последовательности предписаний, обеспечивающих выполнение компьютером преобразования исходных данных в искомый результат.

Можно выделить следующие объективные формы представления программы для ЭВМ:

а) исходная программа (или исходный текст) — последовательность предписаний на алгоритмическом языке высокого уровня, предназначенных для автоматизированного перевода этих предписаний в последовательность команд в объектном коде;

б) рабочая программа (или объектный код) — последовательность машинных команд, т. е. команд, представленных на языке, понятном ЭВМ;

в) программа, временно введенная в память ЭВМ, — совокупность физических состояний элементов памяти запоминающего устройства ЭВМ (ОЗУ), сохраняющихся до прекращения подачи электропитания к ЭВМ;

г) программа, постоянно хранимая в памяти ЭВМ, — представленная на языке машины команда (или серия команд), выполненная в виде физических особенностей участка интегральной схемы, сохраняющихся независимо от подачи электропитания.

Исходная и рабочая программы представляются в электронном виде. Правовая охрана программ для ЭВМ распространяется только в отношении формы их выражения и «... не распространяется на идеи, концепции, принципы, методы, процессы, системы, способы, решения технических, организационных или иных задач, открытия, факты, языки программирования» [2, ст.1259, п. 5].

5.3 Авторское право на программу для ЭВМ

Предпосылкой охраноспособности программы для ЭВМ и базы данных является их творческий характер, т. е. они должны быть продуктом личного творчества автора. Творческий характер деятельности автора предполагается до тех пор, пока не доказано обратное [2, ст. 1257].

Момент возникновения авторского права является важнейшим юридическим фактом, который устанавливается в силу создания программы для ЭВМ. «Для возникновения, осуществления и защиты авторских прав не требуется регистрация произведения или соблюдение каких-либо иных формальностей» [2, ст.1259, п.4].

Закон устанавливает, что обнародование программы не является обязательным условием для возникновения прав на нее: «Авторские права распространяется как на обнародованные, так и на необнародо-

ванные произведения, выраженные в какой-либо объективной форме ...» [2, ст. 1259, п. 3].

Таким образом, только сам факт создания программы, зафиксированной в объективной форме, является основанием возникновения авторского права на нее.

Каждая составляющая понятия использования программы для ЭВМ имеет конкретное содержание, которое также определено законом:

а) воспроизведение — «... изготовление одного или более экземпляров произведения или его части в любой материальной форме, ... в том числе запись в память ЭВМ» [2, ст. 1270, п. 2, п.п.1];

б) распространение — предоставление доступа к произведению «... путем продажи или иного отчуждения его оригинала или экземпляров» [2, ст. 1270, п. 2, п.п.2];

в) публичный показ — «... любая демонстрация оригинала или экземпляров произведения непосредственно ... либо с помощью технических средств в месте, открытом для свободного посещения, или в месте, где присутствует значительное число лиц ...» [2, ст. 1270, п. 2, п.п.3].

В целях оповещения о своих правах правообладатель «... вправе использовать знак охраны авторского права, который помещается на каждом экземпляре произведения и состоит из следующих элементов: латинской буквы С в окружности; имени или наименования правообладателя; года первого опубликования произведения» [2, ст. 1271].

Исключительные права на программу переходят по наследству в установленном законом порядке, и их можно реализовать в течение срока действия авторского права.

Передача прав на материальный носитель не влечет за собой передачи каких-либо прав на программу для ЭВМ [2, ст.1227]. Иными словами, передача носителя информации (например, диска) с зафиксированной на нем программой третьему лицу не означает передачи каких-либо прав на эту программу.

5.4 Правообладание

Если человек (или группа людей) самостоятельно, по личной инициативе создал программу для ЭВМ, то он является одновременно и автором, и правообладателем созданного произведения, что позволяет ему по собственному усмотрению использовать эту программу (или базу данных) в личных целях, продавать, раздавать бесплатно, разрешать тиражировать и распространять или иным образом распоряжаться своими исключительными правами.

Кроме личных (неимущественных) прав автору “служебной” программы для ЭВМ (базы данных) принадлежит право на вознаграждение при условии использования работодателем созданных произведений или передачи исключительного права другому лицу. Размер и порядок выплаты этого определяется договором [2, ст. 1295, п. 2].

5.5 Нарушение прав на программу для ЭВМ и базы данных

Специфика программ для ЭВМ такова, что они очень уязвимы в смысле их незаконного использования (прежде всего, путем копирования и распространения копий). Незаконно изготовленные (скопированные) или используемые экземпляры программы для ЭВМ или базы данных называются контрафактными, а несанкционированное использование чужих программ или баз данных путем опубликования (выпуска в свет), воспроизведения (полного или частичного), распространения, иного использования считается нарушением исключительных прав на программы для ЭВМ или базы данных, т. е. нарушением авторского права.

5.6 Право на официальную регистрацию

В ст. 1262 ГК РФ Ч.4 закреплено право автора или иного правообладателя на государственную регистрацию программы для ЭВМ или базы данных: «Правообладатель в течение срока действия исключительного права на программу для ЭВМ может по своему желанию зарегистрировать такую программу для ЭВМ или такую базу данных в феде-

ральном органе исполнительной власти по интеллектуальной собственности». Исключение составляют программы для ЭВМ и базы данных, в которых содержатся сведения, составляющие государственную тайну.

Предусмотренная регистрация не является правообразующей и носит факультативный характер, т. е. с ней не связано возникновение прав на программу для ЭВМ, однако такая процедура представляется полезной по следующим соображениям.

а) Она является официальным уведомлением общественности о наличии у правообладателей прав в отношении рассматриваемых объектов.

б) Государственная регистрация содействует защите прав в случае возникновения конфликтных ситуаций при нарушении прав или установлении приоритета.

5.7 Процедура официальной регистрации

Процедура официальной регистрации программ для ЭВМ и баз данных в целом определена ст. 1262 ГК РФ Ч.4 и включает подачу заявки в федеральный орган исполнительной власти по интеллектуальной собственности (Роспатент), проверку поданных документов и собственно регистрацию. После поступления заявки на регистрацию в Роспатент проверяется наличие необходимых документов и их соответствие установленным требованиям. При положительном результате проверки сведения о программе для ЭВМ или базе данных вносятся, соответственно, в Реестр программ для ЭВМ или Реестр баз данных под уникальным регистрационным номером и выдается заявителю (здесь заявителем называют правообладателя, подавшего заявку на регистрацию программы или базы данных в Роспатент) свидетельство о государственной регистрации установленной формы, в котором указаны регистрационный номер объекта по Реестру, название программы или базы данных, имя или наименование правообладателя, фамилии авторов и дата регистрации. Сведения о зарегистрированных программах для ЭВМ и базах данных публикуются в официальном бюллетене Роспатента.

5.8 Заявка на официальную регистрацию

Состав заявки на официальную регистрацию программы для ЭВМ или базы данных (далее - Заявка) определен п. 2 ст. 1262 ГК РФ Ч.1, а также в Правилах составления, подачи и рассмотрения заявок на официальную регистрацию программ для электронных вычислительных машин и баз данных (далее - Правила).

Заявка должна относиться к одной программе или одной базе данных. При этом «Программа для ЭВМ, состоящая из нескольких программ для ЭВМ (программный комплекс), которые не могут быть использованы самостоятельно, регистрируется в целом (без регистрации каждой входящей в нее (него) программы для ЭВМ)» [3, п. 5]. Заявка должна содержать следующие документы:

- а) заявление о государственной регистрации;
- б) депонируемые материалы, идентифицирующие программу для ЭВМ, включая реферат;
- в) документ, подтверждающий уплату государственной пошлины в установленном размере или основание для освобождения от уплаты государственной пошлины или уменьшения его размера.

В Правилах подробно описаны требования, предъявляемые к документам заявки.

Заявление на официальную регистрацию представляется отпечатанным на типографском бланке или в виде компьютерной распечатки согласно образцам, приведенным в приложениях к Правилам (формы РП и РП/ДОП).

В состав депонируемых материалов входит также реферат, который представляется в двух экземплярах отдельно от листинга программы для ЭВМ или описания структуры базы данных и не входит в их объем. Реферат должен содержать информацию, определенную в п.п. 18а) — 18и), п.21 и п.23 Правил, в полном объеме. При этом:

- а) аннотация реферата должна содержать сведения, определенные п. 18г) Правил;

б) объем памяти указывается в Кбайтах или Мбайтах и определяется для программ как объем памяти, занимаемый исходным текстом программы (листингом).

5.9 Программный продукт и формы его продажи

Программный продукт — персонифицированная программа для ЭВМ или база данных, которая предназначена для самостоятельного использования конкретным пользователем в личных целях.

Коммерческая реализация (продажа) программного продукта связана с понятием использования программы для ЭВМ третьими лицами (пользователями) и осуществляется на основании лицензионного договора с правообладателем. Договор заключается в письменном виде и может определять следующие условия: способы использования, порядок выплаты вознаграждения и срок действия договора, а также территорию, на которой используется данный продукт [2, ст. 1235, 1236].

Одним из типов лицензионного договора на программу для ЭВМ является традиционный двухсторонний договор правообладателя — лицензиара, с покупателем (пользователем) — лицензиатом, в котором определяется способы, сроки, территория использования программы или базы данных. Такие договоры составляются, как правило, при единичных продажах программного продукта, предназначенного для решения достаточно узких прикладных задач (научных, отраслевых и т. п.), при продажах программного продукта, требующего регулярного обновления и дополнения (некоторые базы данных), а также при передаче прав на тиражирование и распространение программ для ЭВМ или баз данных.

5.10 Договор на использование программы для ЭВМ

Текст договора должен содержать определенную исчерпывающую формулировку лицензионного соглашения между владельцем прав на программу для ЭВМ (далее — объект договора) и покупателем (приобретателем прав на использование объекта договора).

Заключение

В рамках дипломного проекта был проведён обзор последних тенденций на рынке Web приложений, который показал востребованность технологии RIA, в частности Flex приложений. Были сформулированы основные проблемы, возникающие при тестировании взаимодействия Flex приложений с сервером, на основании которых поставлено техническое задание. Распространённость технологии Flex и неотъемлимость фазы тестирования в жизненном цикле программного обеспечения подтверждают актуальность темы дипломного проекта.

Анализ уже существующих решений по тестированию Flex приложений показал, что ни одно из них в полной мере не удовлетворяет сформулированным техническим требованиям, в результате чего было принято решение о разработке собственного программного обеспечения, способного решить поставленную задачу.

Результатом реализации задачи дипломного проекта является программный модуль для тестового фреймворка Apache JMeter, предоставляющий возможность функционального и нагрузочного тестирования взаимодействия Flex приложения с сервером через AMF. В качестве серверной технологии используется BlazeDS.

Были проведены работы по модульному и функциональному тестированию разработанного ПО, показавшие его соответствие заявленным требованиям.

В завершении дипломного проекта было составлено экономическое обоснование и приведен раздел, посвященный защите интеллектуальной собственности.

Планируется дальнейшее развитие программного модуля с целью улучшения визуализации результатов тестов, доработки пользовательского интерфейса и поддержки других функций BlazeDS.

Список использованных источников

1. *Е.А. Симакович Д.В. Гарайс, А.В. Ямшанов.* Обзор современных технологий создания RIA - приложений / А.В. Ямшанов Е.А. Симакович, Д.В. Гарайс // Проект SWorld – международные научно-практические Интернет-конференции. — 2010.
2. *Жирякова, И.А.* FLEX технология разработки интерактивных WEB- приложений / И.А. Жирякова // *Математический практикум для экономистов и инженеров.* — 2011. — Vol. 4. — Рр. 86–91.
3. *А.Н. Иванов Д.В. Кознов, М.Г. Тыжгеев.* Моделирование интерфейса полнофункциональных WEB-приложений, интенсивно работающих с данными / М.Г. Тыжгеев А.Н. Иванов, Д.В. Кознов // *Вестник Санкт-Петербургского университета.* — 2009. — Vol. 3. — Рр. 1–4.
4. *Михальченко, Ю.А.* Применение RIA в системах поддержки принятия решений / Ю.А. Михальченко // Молодые ученые в решении актуальных проблем науки: Всероссийская научно-практическая конференция. Сборник статей студентов и молодых ученых / СибГТУ. — Vol. 4. — г. Красноярск: 2009. — Рр. 279–283.
5. *Савин, Р.* Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. / Р. Савин. — Дело, 2007.
6. *Канер, С.* Тестирование программного обеспечения. Фундаментальные концепции менеджмена бизнес-приложений. / С. Канер. — ДиаСофт, 2001.
7. *Комал Мирчандани, Шрути Уджвал.* Использование IBM Rational Functional Tester для автоматизированного тестирования глобализованных приложений. — 2008. http://www.ibm.com/developerworks/ru/library/0925_mirchandani-ujjwal/index.html.

8. *С.С. Давыдов, А.Г. Ефимов. IntelliJ IDEA. Профессиональное программирование на Java. Наиболее полное руководство / А.Г. Ефимов С.С. Давыдов. — БХВ-Петербург, 2005.*
9. Action Message Format 3 Specification. — 2006.
http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf.
10. *Alain Abran, James W. Moore. Guide to the Software Engineering Body of Knowledge / James W. Moore Alain Abran. — IEEE, 2004.*
11. *Бек, К. Экстремальное программирование: разработка через тестирование. Библиотека программиста. / К. Бек. — СПб: Питер, 2003. — Р. 224.*
12. *Я.Г. Лило, Е.А. Тюменцев. Повторное использование модульных тестов для организации нагрузочного, стресс-тестирования, тестирования стабильности / Е.А. Тюменцев Я.Г. Лило // Математические структуры и моделирование. — 2011. — Vol. 24. — Рр. 69–74.*

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 }
```

Листинг А.4 — Абстракция прокси интерфейса

```
1 package edu.leti.jmeter.proxy;
2
3 /**
4  * Абстрактный класс для proxy
5  *
6  * @author Tedikova O.
7  * @version 1.0
8  */
9 public abstract class AbstractProxy implements ProxyInterface {
10
11     protected final int localPort;
12
13     public AbstractProxy(int proxyPort) {
14         localPort = proxyPort;
15     }
16
17     @Override
18     public int getLocalPort() {
```



```

19         return localPort;
20     }
21 }

```

Листинг А.5 — Графический интерфейс панели прокси сервера

```

1 package edu.leti.jmeter.proxy;
2
3 import org.apache.jmeter.control.gui.LogicControllerGui;
4 import org.apache.jmeter.gui.UnsharedComponent;
5 import org.apache.jmeter.gui.util.MenuFactory;
6 import org.apache.jmeter.testelement.TestElement;
7 import org.apache.jorphan.gui.JLabeledTextField;
8
9 import javax.swing.*;
10 import java.awt.*;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13 import java.util.Arrays;
14 import java.util.Collection;
15
16 /**
17  * @author Tedikova O.
18  * @version 1.0
19  */
20 public class AMFProxyGui extends LogicControllerGui implements
    UnsharedComponent {
21     private JLabeledTextField proxyPort;
22     private AMFProxyControl amfProxyControl;
23     JButton startButton;
24     JButton stopButton;
25
26     public AMFProxyGui() {
27         super();
28         init();
29     }
30
31     public void init() {
32         setLayout(new BorderLayout());
33         setBorder(makeBorder());
34         add(makeTitlePanel(), BorderLayout.NORTH);
35         JLabeledTextField proxyHost = new JLabeledTextField("AmfProxy
Host:  ");
36         proxyHost.setText("localhost");
37         proxyHost.setEnabled(false);
38         proxyPort = new JLabeledTextField("AmfProxy Port:  ");
39

```

```

40     startButton = new JButton("Start");
41     startButton.addActionListener(new StartListener());
42     stopButton = new JButton("Stop");
43     stopButton.addActionListener(new StopListener());
44
45     JPanel mainPanel = new JPanel(new BorderLayout());
46     JPanel proxyPanel = new JPanel();
47     JPanel buttonPanel = new JPanel();
48
49     proxyPanel.setLayout(new GridBagLayout());
50
51     mainPanel.add(proxyPanel, BorderLayout.NORTH);
52     GridBagConstraints c = new GridBagConstraints();
53     c.fill = GridBagConstraints.HORIZONTAL;
54     c.gridwidth = 2;
55     c.gridx = 0;
56     c.gridy = 0;
57     c.weightx = 1;
58     proxyPanel.add(proxyHost, c);
59
60     c.gridy = 2;
61     proxyPanel.add(proxyPort, c);
62
63
64     proxyPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBo
65     "Settings"));
66
67     buttonPanel.add(startButton, BorderLayout.CENTER);
68     buttonPanel.add(stopButton, BorderLayout.CENTER);
69
70     mainPanel.add(proxyPanel, BorderLayout.NORTH);
71     mainPanel.add(buttonPanel, BorderLayout.SOUTH);
72     add(mainPanel, BorderLayout.CENTER);
73 }
74
75 @Override
76 public String getStaticLabel() {
77     return "AMF Proxy Server";
78 }
79
80 public String getLabelResource() {
81     return getClass().getCanonicalName();
82 }
83
84 /**

```

```

83      * Метод, вызываемый при перерисовке GUI тестового элемента. Отображает в GUI
параметры, содержащиеся в сэмпле
84      *
85      * @param element тестовый элемент экземпляра (сэмплера)
86      */
87      @Override
88      public void configure(TestElement element) {
89          super.configure(element);
90          amfProxyControl = (AMFProxyControl) element;
91          proxyPort.setText(amfProxyControl.getProxyPort());
92      }
93
94      public TestElement createTestElement() {
95          amfProxyControl = new AMFProxyControl();
96          modifyTestElement(amfProxyControl);
97          return amfProxyControl;
98      }
99
100     /**
101     * Передаёт в тестовый элемент из GUI необходимые параметры.
102     *
103     * @param testElement тестовый элемент
104     */
105     public void modifyTestElement(TestElement testElement) {
106         amfProxyControl = (AMFProxyControl) testElement;
107         super.configureTestElement(amfProxyControl);
108         amfProxyControl.setProxyPort(proxyPort.getText());
109     }
110
111     @Override
112     public void clearGui() {
113         super.clearGui();
114         proxyPort.setText("");
115     }
116
117     @Override
118     public Collection<String> getMenuCategories() {
119         return Arrays.asList(MenuFactory.NON_TEST_ELEMENTS);
120     }
121
122     private class StartListener implements ActionListener {
123
124         public void actionPerformed(ActionEvent e) {
125             startProxy();
126         }
127     }

```

```

128
129     private class StopListener implements ActionListener {
130
131         public void actionPerformed(ActionEvent e) {
132             stopProxy();
133         }
134     }
135
136     private void startProxy() {
137         startButton.setEnabled(false);
138         SwingWorker worker = new SwingWorker() {
139             @Override
140             protected Object doInBackground() throws Exception {
141                 modifyTestElement(amfProxyControl);
142                 amfProxyControl.startProxy();
143                 return null;
144             }
145         };
146         worker.execute();
147     }
148
149     private void stopProxy() {
150         amfProxyControl.stopProxy();
151         startButton.setEnabled(true);
152     }
153 }

```

Листинг А.6 — Контроллер прокси сервера

```

1 package edu.leti.jmeter.proxy;
2
3 import edu.leti.amf.MessageDecoder;
4 import edu.leti.jmeter.sampler.AmfRPCSampler;
5 import flex.messaging.io.amf.ActionMessage;
6 import flex.messaging.io.amf.MessageBody;
7 import flex.messaging.messages.RemotingMessage;
8 import org.apache.jmeter.control.GenericController;
9 import org.apache.jmeter.exceptions.IllegalUserActionException;
10 import org.apache.jmeter.gui.GuiPackage;
11 import org.apache.jmeter.gui.tree.JMeterTreeModel;
12 import org.apache.jmeter.gui.tree.JMeterTreeNode;
13 import org.apache.jmeter.protocol.http.sampler.HTTPSamplerBase;
14 import org.apache.jmeter.protocol.http.util.HTTPFileArg;
15 import org.apache.jmeter.samplers.SampleResult;
16 import org.apache.jmeter.testelement.TestElement;
17 import org.apache.jmeter.util.JMeterUtils;
18 import org.apache.jorphan.util.JOrphanUtils;

```

```

19 import org.slf4j.Logger;
20 import org.slf4j.LoggerFactory;
21
22 import java.io.File;
23 import java.io.FileInputStream;
24 import java.io.InputStream;
25 import java.util.ArrayList;
26 import java.util.List;
27
28 /**
29  * Класс, отвечающий за управление прокси-сервером—
30  *
31  * @author Tedikova O.
32  * @version 1.0
33  */
34 public class AMFProxyControl extends GenericController implements
    SamplerDeliverer {
35
36     private static final Logger logger =
    LoggerFactory.getLogger(AMFProxyControl.class);
37
38     private static final String PROXY_PORT = "amf_proxy_port";
39     private static final String SERVER_HOST = "amf_server_host";
40     private static final String SERVER_PORT = "amf_server_port";
41
42     /**
43      * Метод создаёт новый {@link AmfRPCSampler} на каждое полученное сообщение
    типа RemotingMessage и отображает
44      * его в GUI
45      *
46      * @param sampler {@inheritDoc}
47      * @param subConfigs {@inheritDoc}
48      * @param result {@inheritDoc}
49      */
50     public void deliverSampler(HTTPSamplerBase sampler, TestElement[]
    subConfigs, SampleResult result) {
51         MessageDecoder decoder = new MessageDecoder();
52         HTTPFileArg[] httpFiles = sampler.getHTTPFiles();
53         for (HTTPFileArg httpFile : httpFiles) {
54             InputStream inputStream = null;
55             try {
56                 inputStream = new FileInputStream(new
    File(httpFile.getPath()));
57                 ActionMessage actionMessage =
    decoder.getActionMessage(inputStream);
58                 MessageBody messageBody = actionMessage.getBody(0);

```

```

59         Object[] data = (Object[]) messageBody.getData();
60         if (data.length > 0 && data[0] instanceof RemotingMessage)
61         {
62             RemotingMessage remotingMessage = (RemotingMessage)
63             data[0];
64             String operation = remotingMessage.getOperation();
65             String destination = remotingMessage.getDestination();
66             List parameters = remotingMessage.getParameters();
67             List<String[]> stringParameters = new
68             ArrayList<String[]>();
69             for (Object parameter : parameters) {
70                 stringParameters.add(new String[] { "",
71                 parameter.toString() });
72             }
73             AmfRPCSampler amfRPCSampler = new AmfRPCSampler();
74             amfRPCSampler.setAmfCall(String.format("%s.%s",
75             destination, operation));
76
77             amfRPCSampler.setEndpointUrl(sampler.getUrl().toString());
78             amfRPCSampler.setParamsTable(stringParameters);
79             amfRPCSampler.setName(sampler.getUrl().toString());
80             JMeterTreeNode myTarget =
81             findFirstNodeOfType(AMFProxyControl.class);
82             try {
83                 JMeterTreeModel treeModel =
84                 GuiPackage.getInstance().getTreeModel();
85                 treeModel.addComponent(amfRPCSampler, myTarget);
86             } catch (IllegalUserActionException e) {
87                 JMeterUtils.reportErrorToUser(e.getMessage());
88             }
89         }
90         } catch (Exception e) {
91             logger.error("Can't open file " + httpFile.getPath(), e);
92         } finally {
93             JOrphanUtils.closeQuietly(inputStream);
94         }
95     }
96 }

/**
 * Находит первый активный узел указанного типа в дереве элементов GUI
 *
 * @param type тип элемента
 * @return первый активный узел указанного типа
 */
private JMeterTreeNode findFirstNodeOfType(Class<?> type) {

```

```

97         JMeterTreeModel treeModel =
GuiPackage.getInstance().getTreeModel();
98         List<JMeterTreeNode> nodes = treeModel.getNodesOfType(type);
99         for (JMeterTreeNode node : nodes) {
100             if (node.isEnabled()) {
101                 return node;
102             }
103         }
104         return null;
105     }
106
107     private AmfHttpProxy amfProxy;
108
109     public String getProxyPort() {
110         return getPropertyAsString(PROXY_PORT);
111     }
112
113     public void setProxyPort(String proxyPort) {
114         setProperty(PROXY_PORT, proxyPort);
115     }
116
117     public String getServerHost() {
118         return getPropertyAsString(SERVER_HOST);
119     }
120
121     public void setServerHost(String serverHost) {
122         setProperty(SERVER_HOST, serverHost);
123     }
124
125     public String getServerPort() {
126         return getPropertyAsString(SERVER_PORT);
127     }
128
129     public void setServerPort(String serverPort) {
130         setProperty(SERVER_PORT, serverPort);
131     }
132
133     public synchronized void startProxy() {
134         if (amfProxy == null) {
135             try {
136                 amfProxy = new
AmfHttpProxy(Integer.parseInt(getProxyPort()), "", 0, this);
137             } catch (NumberFormatException ex) {
138                 JMeterUtils.reportErrorToUser("Couldn't create proxy
server with proxy port= " + getProxyPort() +

```

```

139         "; server host=" + getServerHost() + " and server
    port=" + getServerPort());
140     }
141     } else {
142         amfProxy.stop();
143     }
144     amfProxy.start();
145 }
146
147 public void stopProxy() {
148     if (amfProxy != null) {
149         amfProxy.stop();
150     }
151 }
152 }

```

Листинг А.7 — Семплер amf запросов

```

1 package edu.leti.jmeter.sampler;
2
3 import flex.messaging.io.amf.client.AMFConnection;
4 import flex.messaging.io.amf.client.exceptions.ClientStatusException;
5 import flex.messaging.io.amf.client.exceptions.ServerStatusException;
6 import org.apache.jmeter.samplers.AbstractSampler;
7 import org.apache.jmeter.samplers.Entry;
8 import org.apache.jmeter.samplers.SampleResult;
9 import org.apache.jmeter.testelement.property.CollectionProperty;
10 import org.apache.jmeter.testelement.property.JMeterProperty;
11 import org.apache.jmeter.testelement.property.StringProperty;
12 import org.slf4j.Logger;
13 import org.slf4j.LoggerFactory;
14
15 import java.io.ByteArrayOutputStream;
16 import java.io.PrintStream;
17 import java.util.ArrayList;
18 import java.util.Collections;
19 import java.util.List;
20
21 /**
22  * Реализация сэмплера, отправляющего AMF запросы.
23  * Нагрузочноемногопоточное() тестирование в JMeter организовано так, что каждый
    поток прогоняет тест план,
24  * имитируя работу отдельного пользователя. Данная реализация сэмплера обеспечивает
    создание отдельного соединения
25  * с сервером для каждого пользователя потока() и непрерывность сессии во время
    прохождения тест плана.
26  *

```



```

27  * @author Tedikova O.
28  * @version 1.0
29  */
30  public class AmfRPCSampler extends AbstractSampler {
31
32      private static final Logger logger =
33      LoggerFactory.getLogger(AmfRPCSampler.class);
34
35      private static final String ENDPOINT_URL = "endpoint_url";
36
37      private static final String AMF_CALL = "amf_call";
38
39      private static final String PARAMETERS = "request_parameters";
40
41      private static final String PARAMS_TABLE = "parameters_table";
42
43      {
44          setProperty(AmfRPCSampler.GUI_CLASS,
45          AmfRPCSamplerGui.class.getCanonicalName());
46      }
47
48      /**
49       * Переменная, содержащая свой экземпляр AMFConnection для каждого потока из
50       * ThreadGroup
51       */
52      private static final ThreadLocal<AMFConnection>
53      AMF_CONNECTION_THREAD_LOCAL = new ThreadLocal<AMFConnection>();
54
55      public String getEndpointUrl() {
56          return getPropertyAsString(ENDPOINT_URL);
57      }
58
59      public void setEndpointUrl(String endpointUrl) {
60          setProperty(ENDPOINT_URL, endpointUrl);
61      }
62
63      public void setAmfCall(String amfCall) {
64          setProperty(AMF_CALL, amfCall);
65      }
66
67      public String getAmfCall() {
68          return getPropertyAsString(AMF_CALL);
69      }
70
71      public void setParameters(List<String> parameters) {

```

```

68         List<JMeterProperty> propertyList = new
ArrayList<JMeterProperty>();
69         for (String parameter : parameters) {
70             propertyList.add(new StringProperty(parameter, parameter));
71         }
72         CollectionProperty property = new CollectionProperty(PARAMETERS,
propertyList);
73         setProperty(property);
74     }
75
76     public void setParamsTable(List<String[]> paramsTable) {
77         List<JMeterProperty> propertyList = new
ArrayList<JMeterProperty>();
78         for (String[] row : paramsTable) {
79             propertyList.add(new StringProperty(row[0], row[1]));
80         }
81         CollectionProperty property = new CollectionProperty(PARAMS_TABLE,
propertyList);
82         setProperty(property);
83     }
84
85     public List<String> getParameters() {
86         List<String> result = new ArrayList<String>();
87         JMeterProperty property = getProperty(PARAMETERS);
88         if (property instanceof CollectionProperty) {
89             CollectionProperty collectionProperty = (CollectionProperty)
property;
90             for (int i = 0; i < collectionProperty.size(); i++) {
91                 result.add(collectionProperty.get(i).getStringValue());
92             }
93             return result;
94         } else {
95             return Collections.emptyList();
96         }
97     }
98
99     public List<String[]> getParamsTable() {
100         List<String[]> result = new ArrayList<String[]>();
101         JMeterProperty property = getProperty(PARAMS_TABLE);
102         if (property instanceof CollectionProperty) {
103             CollectionProperty collectionProperty = (CollectionProperty)
property;
104             for (int i = 0; i < collectionProperty.size(); i++) {
105                 result.add(new
String[]{ collectionProperty.get(i).getName(),
collectionProperty.get(i).getStringValue() });

```

```

106         }
107         return result;
108     } else {
109         return Collections.emptyList();
110     }
111 }
112
113 public SampleResult sample(Entry entry) {
114     AmfRPCSamplerResult sampleResult = new AmfRPCSamplerResult(this);
115     sampleResult.setSampleLabel(getName());
116
117     sampleResult.setSuccessful(true);
118     /*
119     * Проверка того, существует ли для данного потока AMFConnection. Если
для данного потока AMFConnection не
120     * существует, то создаётся новый экземпляр AMFConnection.
121     */
122     AMFConnection amfConnection = AMF_CONNECTION_THREAD_LOCAL.get();
123     if (amfConnection == null) {
124         try {
125             amfConnection = new AMFConnection();
126             amfConnection.connect(getEndpointUrl());
127             AMF_CONNECTION_THREAD_LOCAL.set(amfConnection);
128         } catch (ClientStatusException e) {
129             logger.error("Couldn't connect to " + getEndpointUrl(), e);
130         }
131     }
132     try {
133         Object result = amfConnection.call(getAmfCall(),
getParameters().toArray());
134         sampleResult.setResponseData(result.toString(), "UTF-8");
135
136     } catch (ClientStatusException cse) {
137         handleException(cse, sampleResult);
138     } catch (ServerStatusException sse) {
139         handleException(sse, sampleResult);
140     }
141     return sampleResult;
142 }
143
144 private void handleException(Exception e, SampleResult sampleResult) {
145     ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
146     PrintStream stream = new PrintStream(outputStream);
147     e.printStackTrace(stream);
148     logger.error("Send request to " + getAmfCall() + "failed", e);

```

```

149         sampleResult.setResponseData(new
String(outputStream.toByteArray()), "UTF-8");
150         sampleResult.setSuccessful(false);
151     }
152
153 }

```

Листинг А.8 — Графический интерфейс семплер amf запросов

```

1 package edu.leti.jmeter.sampler;
2
3 import org.apache.jmeter.samplers.gui.AbstractSamplerGui;
4 import org.apache.jmeter.testelement.TestElement;
5 import org.apache.jorphan.gui.JLabeledTextField;
6
7 import javax.swing.*;
8 import java.awt.*;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 /**
13  * Графический интерфейс AMF sampler
14  *
15  * @author Tedikova O.
16  * @version 1.0
17  */
18 public class AmfRPCSamplerGui extends AbstractSamplerGui {
19     private JLabeledTextField endpointUrlField;
20     private JLabeledTextField amfCallField;
21     private ParametersPanel parametersPanel;
22
23     public AmfRPCSamplerGui() {
24         super();
25         init();
26     }
27
28     private void init() {
29         setLayout(new BorderLayout());
30         setBorder(makeBorder());
31
32         add(makeTitlePanel(), BorderLayout.NORTH);
33
34         endpointUrlField = new JLabeledTextField("Endpoint URL");
35         amfCallField = new JLabeledTextField("AMF Call");
36         parametersPanel = new ParametersPanel();
37
38         JPanel mainPanel = new JPanel(new BorderLayout());

```

```

39     JPanel amfRequestPanel = new JPanel();
40     amfRequestPanel.setLayout(new GridBagLayout());
41
42     GridBagConstraints c = new GridBagConstraints();
43     c.fill = GridBagConstraints.HORIZONTAL;
44     c.gridwidth = 2;
45     c.gridx = 0;
46     c.gridy = 0;
47     c.weightx = 1;
48     amfRequestPanel.add(endpointUrlField, c);
49
50     c.gridy = 2;
51     amfRequestPanel.add(amfCallField, c);
52
53     c.gridy = 4;
54     c.weightx = 6;
55     c.gridheight = 4;
56     amfRequestPanel.add(parametersPanel, c);
57
58
59     amfRequestPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchBorder,
60     "AMF Request"));
61
62     mainPanel.add(amfRequestPanel, BorderLayout.NORTH);
63
64     add(mainPanel, BorderLayout.CENTER);
65 }
66
67 @Override
68 public String getStaticLabel() {
69     return "AMF RPC Sampler";
70 }
71
72 public String getLabelResource() {
73     return getClass().getCanonicalName();
74 }
75
76 /**
77  * Метод, вызываемый при перерисовке GUI тестового элемента. Отображает в GUI
78  * параметры, содержащиеся в сэмпле
79  *
80  * @param element тестовый элемент экземпляра( сэмплера)
81  */
82 @Override
83 public void configure(TestElement element) {
84     super.configure(element);

```

```

82         AmfRPCSampler amfSender = (AmfRPCSampler) element;
83         endpointUrlField.setText(amfSender.getEndpointUrl());
84         amfCallField.setText(amfSender.getAmfCall());
85         parametersPanel.setTableData(amfSender.getParamsTable());
86     }
87
88     public TestElement createTestElement() {
89         AmfRPCSampler amfSender = new AmfRPCSampler();
90         modifyTestElement(amfSender);
91         return amfSender;
92     }
93
94     /**
95      * Метод передаёт в тестовый элемент из GUI необходимые параметры.
96      *
97      * @param testElement тестовый элемент
98      */
99     public void modifyTestElement(TestElement testElement) {
100         AmfRPCSampler amfSender = (AmfRPCSampler) testElement;
101         super.configureTestElement(amfSender);
102         amfSender.setEndpointUrl(endpointUrlField.getText());
103         amfSender.setAmfCall(amfCallField.getText());
104
105         amfSender.setParameters(getParametersList(parametersPanel.getTableData()));
106         amfSender.setParamsTable(parametersPanel.getTableData());
107     }
108
109     @Override
110     public void clearGui() {
111         super.clearGui();
112         endpointUrlField.setText("");
113         amfCallField.setText("");
114         parametersPanel.setTableData(new ArrayList<String[]>());
115     }
116
117     private java.util.List<String> getParametersList(List<String[]>
118     params) {
119         List<String> result = new ArrayList<String>();
120         for (String[] row : params) {
121             result.add(row[1]);
122         }
123         return result;
124     }
125 }

```

Листинг А.9 — Результат семплирования amf запросов

```

1 package edu.leti.jmeter.sampler;
2
3 import org.apache.jmeter.samplers.SampleResult;
4
5 import java.util.Arrays;
6
7 /**
8  * @author Tedikova O.
9  * @version 1.0
10 */
11 public class AmfRPCSamplerResult extends SampleResult {
12
13     private AmfRPCSampler amfRPCSampler;
14
15     public AmfRPCSamplerResult(AmfRPCSampler amfRPCSampler) {
16         this.amfRPCSampler = amfRPCSampler;
17     }
18
19     @Override
20     public String getSamplerData() {
21
22         StringBuilder samplerData = new StringBuilder();
23         samplerData.append("Endpoint Url: ");
24         samplerData.append(amfRPCSampler.getEndpointUrl());
25         samplerData.append("\n");
26         samplerData.append("Amf Call: ");
27         samplerData.append(amfRPCSampler.getAmfCall());
28         samplerData.append("\n");
29         samplerData.append("Request parameters: ");
30
31         samplerData.append(Arrays.toString(amfRPCSampler.getParameters().toArray()));
32
33         return samplerData.toString();
34     }
35
36     @Override
37     public String getDataType() {
38         return SampleResult.TEXT;
39     }
40 }

```

Листинг А.10 — Графический интерфейс панели параметров прокси сервера

```

1 package edu.leti.jmeter.sampler;
2
3 import javax.swing.*;

```

```

4 import javax.swing.table.AbstractTableModel;
5 import java.awt.*;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 /**
12  * Панель параметров прокси сервера.
13  *
14  * @author Tedikova O.
15  * @version 1.0
16  */
17 public class ParametersPanel extends JPanel {
18
19     private ParamsTableModel tableModel;
20     private JTable paramsTable;
21
22     public ParametersPanel() {
23         init();
24     }
25
26     private void init() {
27         setLayout(new BorderLayout(0, 4));
28         tableModel = new ParamsTableModel();
29         paramsTable = new JTable(tableModel);
30
31         JScrollPane scrollPane = new JScrollPane(paramsTable);
32
33         JButton addButton = new JButton("Add");
34         addButton.addActionListener(new AddRowListener());
35         JButton deleteButton = new JButton("Delete");
36         deleteButton.addActionListener(new DeleteRowListener());
37
38         JPanel buttonPanel = new JPanel();
39         buttonPanel.add(addButton, BorderLayout.CENTER);
40         buttonPanel.add(deleteButton, BorderLayout.CENTER);
41
42         add(scrollPane, BorderLayout.CENTER);
43         add(buttonPanel, BorderLayout.SOUTH);
44
45         setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(),
46             "Request Parameters"));
47
48     }
49
50     public List<String[]> getTableData() {

```



```

48         return tableModel.getRows();
49     }
50
51     public void setTableData(List<String[]> rows) {
52         tableModel.setRows(rows);
53     }
54
55     private class AddRowListener implements ActionListener {
56
57         public void actionPerformed(ActionEvent e) {
58             tableModel.addRow();
59         }
60     }
61
62     private class DeleteRowListener implements ActionListener {
63
64         public void actionPerformed(ActionEvent e) {
65             tableModel.deleteRow();
66         }
67     }
68
69     private class ParamsTableModel extends AbstractTableModel {
70
71         private String[] columnNames = new String[] { "name", "value" };
72         private java.util.List<String[]> rows = new ArrayList<String[]>();
73
74         public void addRow() {
75             rows.add(new String[] { "", "" });
76             int firstRow = rows.size() - 1;
77             fireTableRowsInserted(0, firstRow);
78         }
79
80         public void deleteRow() {
81             int selectedRow =
ParametersPanel.this.paramsTable.getSelectedRow();
82             rows.remove(selectedRow);
83             fireTableRowsDeleted(selectedRow, selectedRow);
84         }
85
86         public int getRowCount() {
87             return rows.size();
88         }
89
90         public int getColumnCount() {
91             return columnNames.length;
92         }

```

```

93
94     public String getColumnName(int columnIndex) {
95         return columnNames[columnIndex];
96     }
97
98     public Class<?> getColumnClass(int columnIndex) {
99         return getValueAt(0, columnIndex).getClass();
100     }
101
102     public boolean isCellEditable(int rowIndex, int columnIndex) {
103         return true;
104     }
105
106
107     public Object getValueAt(int rowIndex, int columnIndex) {
108         return rows.get(rowIndex)[columnIndex];
109     }
110
111     public void setValueAt(Object aValue, int rowIndex, int
columnIndex) {
112         rows.get(rowIndex)[columnIndex] = (String) aValue;
113         fireTableCellUpdated(rowIndex, columnIndex);
114     }
115
116     public List<String[]> getRows() {
117         return rows;
118     }
119
120     public void setRows(List<String[]> rows) {
121         this.rows = rows;
122     }
123 }
124 }

```

Приложение Б План приёмочных испытаний приложения

Б.1 Объект тестирования

Объектом тестирования является программный модуль для тестирования взаимодействия Flex приложений с сервером через протокол AMF. Модуль является дополнением к тестовому фреймворку Apache JMeter.

Б.2 Цели и приоритеты тестирования

Целью тестирования является проверка соответствия разработанного программного обеспечения требованиям, установленным в техническом задании.

Б.3 Стратегии тестирования

Функциональное тестирование — проверка соответствия требованиям, указанным в техническом задании, применяется к функционалу, за который отвечает разрабатываемый модуль.

Б.4 Последовательность работ

- а) Проектирование набора тестовых случаев;
- б) Подготовка тестового окружения;
- в) Проведение тестирования;
- г) Анализ результатов;

Б.5 Критерии начала тестирования

- а) законченность разработки требуемого функционала;
- б) наличие задокументированных требований к программному обеспечению;
- в) наличие набора тестовых случаев;
- г) готовность тестового окружения;

Б.6 Критерии окончания тестирования

- а) соответствие продукта функциональным требованиям;
- б) стабильность работы продукта на протяжении обозначенного периода;

Б.7 Тестовое окружение

Для проведения тестирования необходимо следующее программное обеспечение:

- а) сервер BlazeDS;
- б) скомпилированное Flex приложение, использующее AMF канал для передачи сообщений;
- в) web-браузер с установленным Flash Player;
- г) Apache JMeter;
- д) Apache Maven;

Приложение В Сценарии тестирования приложения

В.1 Настройка тестового окружения

Перед проведением тестов необходимо настроить тестовую среду следующим образом:

- а) установить программное обеспечение, указанное в разделе "Тестовое окружение" плана приёмочных испытаний. Место установки JMeter далее будем называть JMETER_HOME;
- б) подключить к JMeter тестируемый модуль amf-translator — добавить jar-файл приложения в каталог JMETER_HOME/lib/ext;
- в) развернуть Flex приложение на сервере BlazeDS, запустить его и убедиться, что оно функционирует;
- г) запустить JMeter — приложение запускается с помощью файла jmeter.bat или ApacheJMeter.jar, находящихся в каталоге JMETER_HOME/bin.

В.2 Функциональные тесты

В.2.1 Проверка записи тестовых запросов

Предусловия:

- а) Добавить элемент AMF Proxy Server (WorkBench > Add > Non-Test Elements > AMF Proxy Server);
- б) 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;
- в) Запустить браузер и указать там точно такие же настройки прокси-сервера. Также стоит убедиться, что указанный Вами порт уже не занят другим приложением;
- г) Далее открыть в браузере Flex приложение;

Шаги теста:

- а) Запустить прокси-сервер, нажав кнопку "Start"

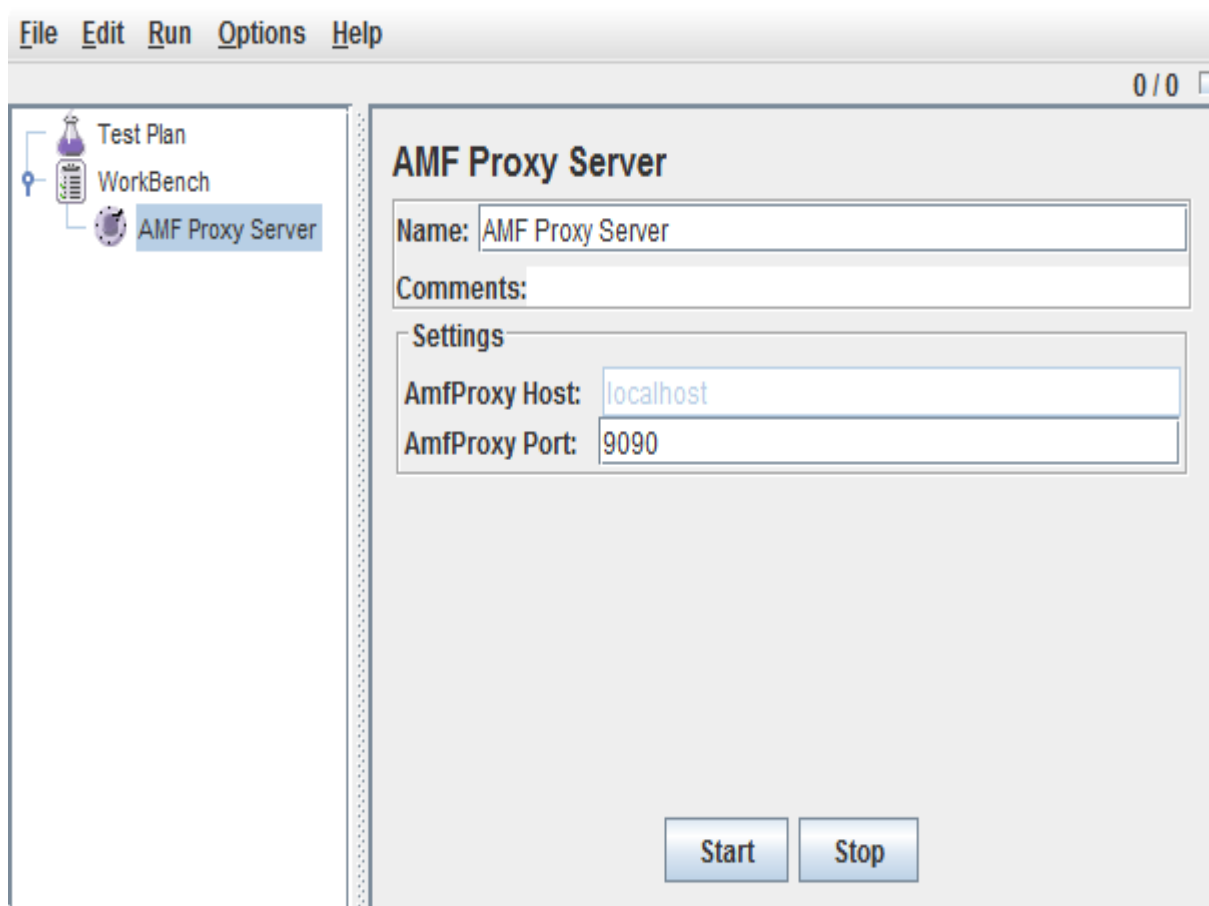


Рисунок В.1 — Настройка прокси-сервера

б) С помощью браузера ввести данные во Flex приложение и вызвать метод удалённого объекта.

в) Завершить запись тестовых запросов, нажав кнопку "Stop"

Ожидаемый результат:

а) После завершения записи тестов, в дереве элементов JMeter в качестве дочернего элемента AMF Proxy Server появился компонент типа AMF RPC Sampler;

б) В AMF RPC Sampler записаны верные параметры AMF запроса, а именно:

- 1) Endpoint Url - URL, по которому отправлялся запрос;
- 2) AMF Call - имя удалённого объекта и процедуры(Например, если мы хотим вызвать у объекта registrationDestination метод registerUser, в этом поле следует написать registrationDestination.registerUser);

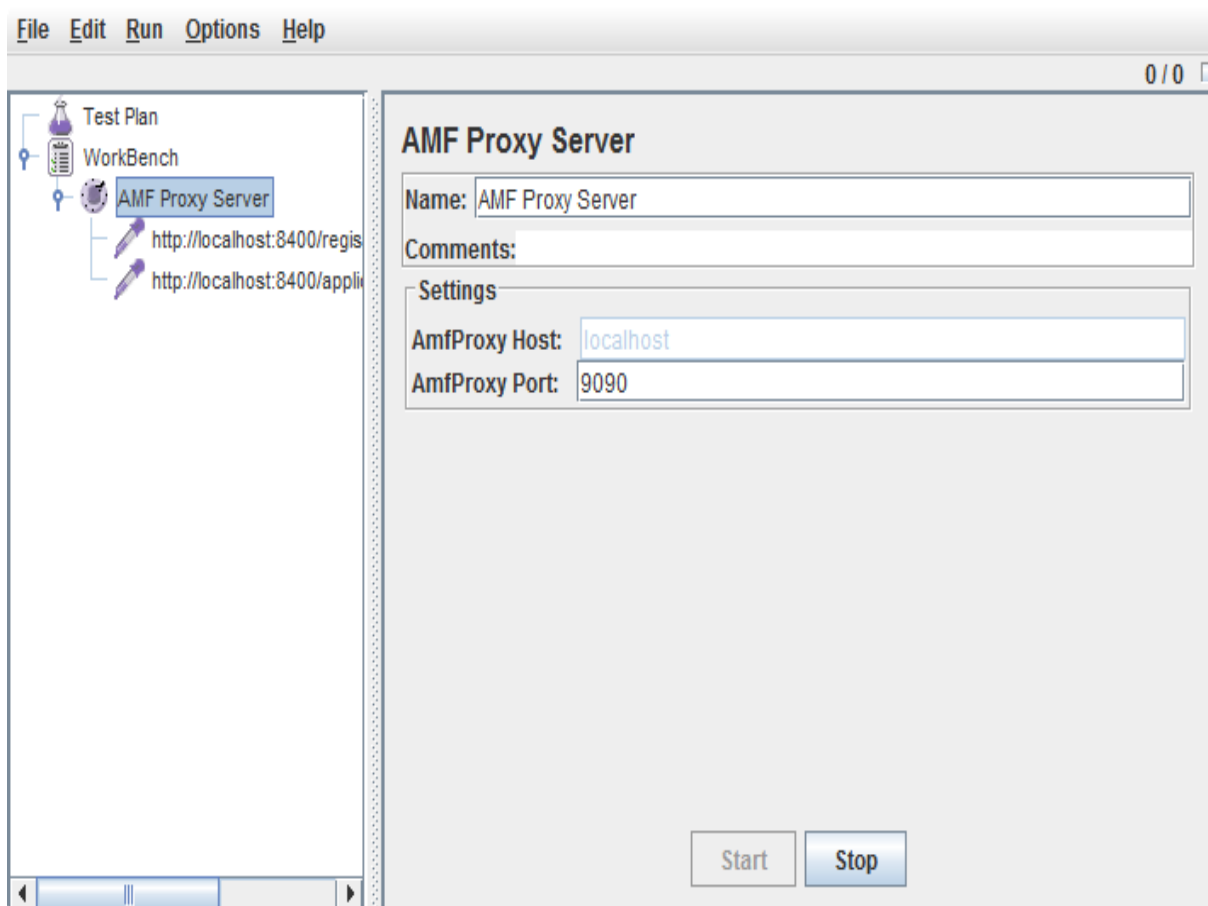


Рисунок В.2 — Запуск прокси-сервера

3) Request Parameters - параметры, переданные методу.

В.2.2 Проверка отправки корректного AMF запроса через AMF RPC Sampler

Предусловия:

- а) Добавить в Test Plan группу потоков - Thread Group (Test Plan > Threads (Users) > Thread Group).
- б) Задать для Thread Group следующие параметры:
 - 1) Действие, которое будут производиться в случае, если в тест выполняется с ошибкой (Action to be taken after a Sampler error) — Continue ;
 - 2) число потоков, в которое будут запускаться шаги тест-плана (Number of Threads) установить равным единице;

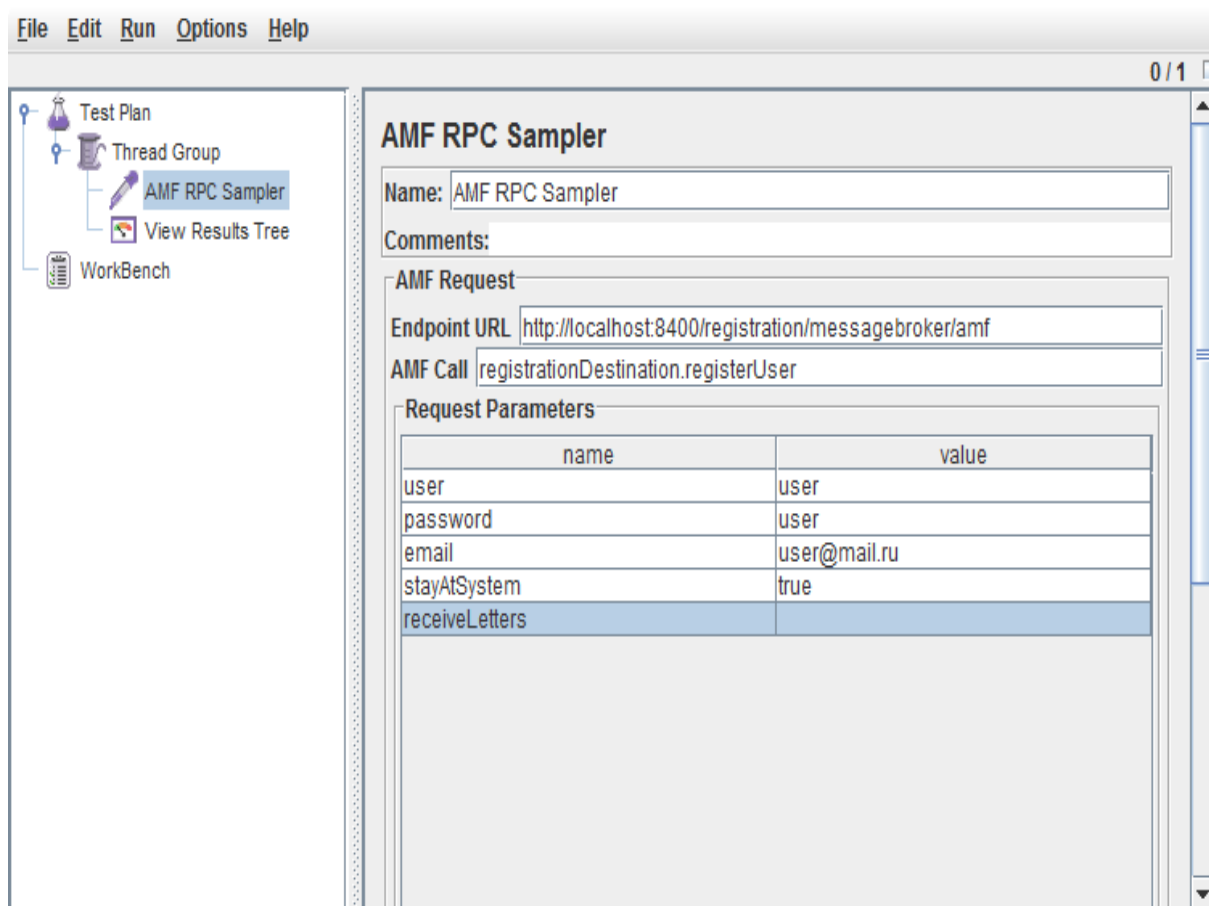


Рисунок В.3 — Интерфейс элемента AMF RPC Sampler

- 3) Интервал, в течение которого будет запущено указанное в предыдущем параметре число потоков (Ramp-Up Period) установить равным единице;
- 4) Число повторений набора тестов (Loop Count) установить равным единице;
- в) добавить визуалайзер результатов, чтобы иметь возможность отслеживать ход выполнения теста (Thread Group > Add > Listener > View Results Tree)

Шаги теста:

- а) В Thread Group в качестве дочернего элемента добавить AMF RPC Sampler;
- б) Ввести в AMF RPC Sampler все необходимые корректные параметры и запустить содержимое элемента Test Plan (Run > Start);

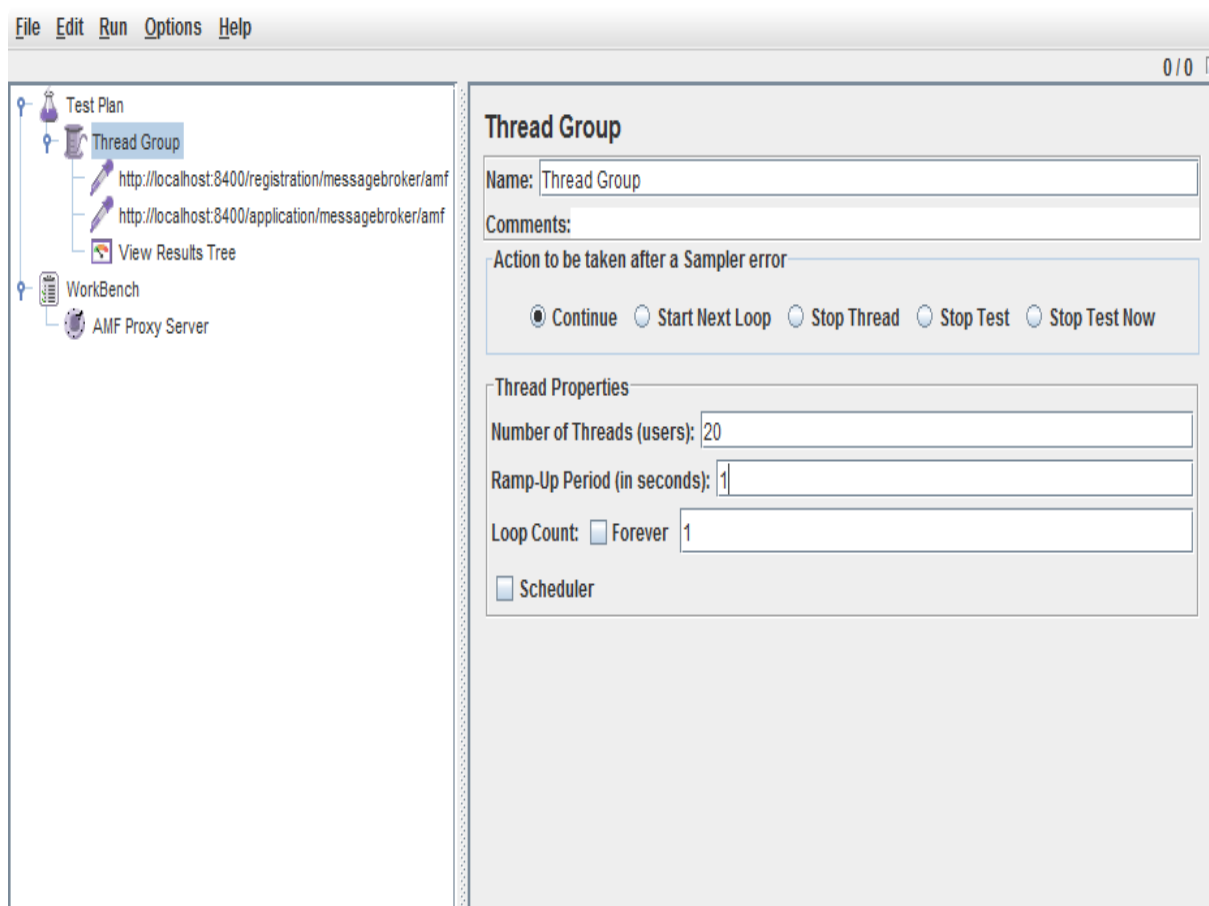


Рисунок В.4 — Создание группы потоков

в) С помощью браузера ввести те же самые данные во Flex приложение и вызвать метод удалённого объекта.

Ожидаемый результат:

а) После завершения прогона тестов во View Results Tree в дереве элементов тест плана элемент AMF RPC Sampler подсвечен зелёным цветом (выполнен успешно);

б) на вкладках View Results Tree присутствуют данные запроса и полученного от сервера ответа;

в) Ответ сервера должен быть тем же, что был получен во время вызова процедуры через браузер.

В.2.3 Проверка отправки некорректного AMF запроса через AMF RPC Sampler

Предусловия:

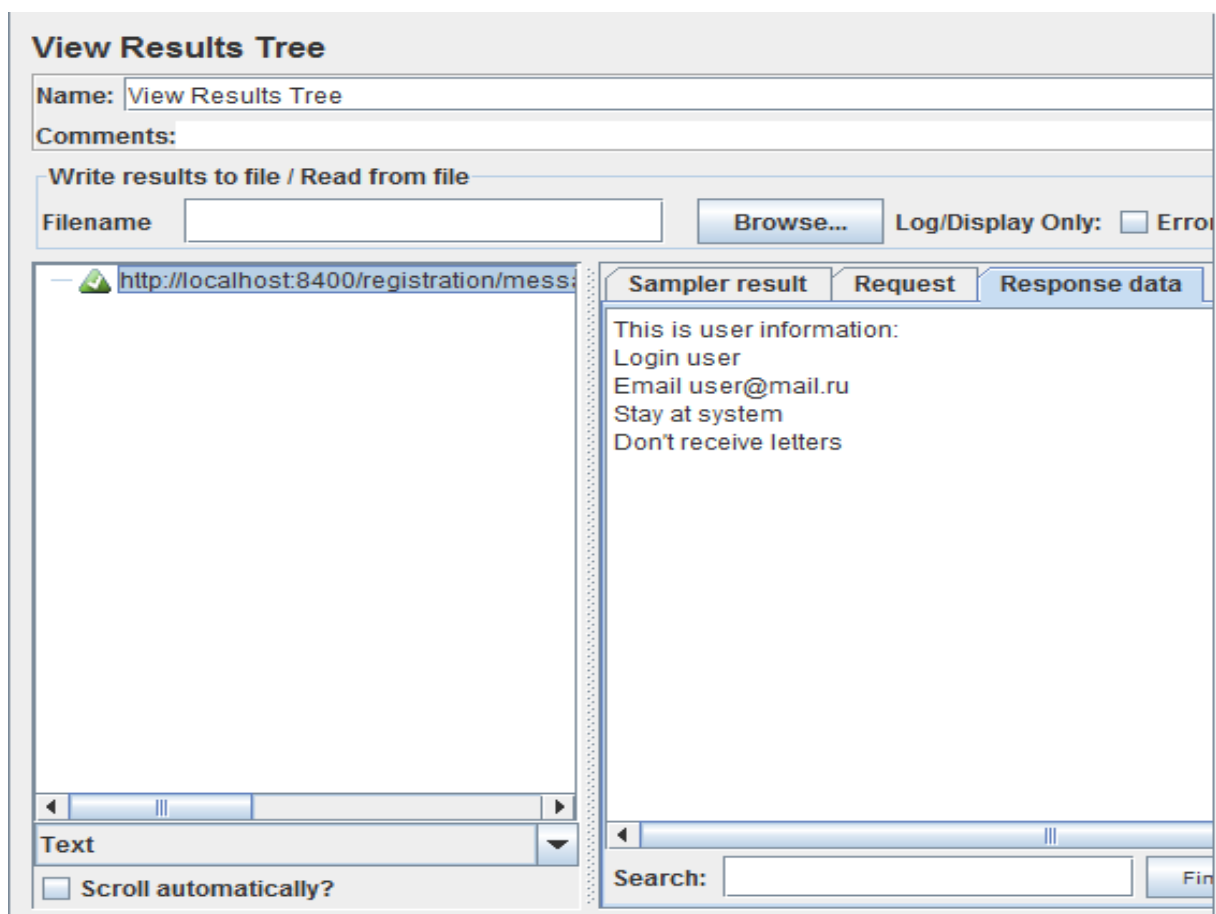


Рисунок В.5 — Результаты корректного теста

- а) Добавить в Test Plan группу потоков - Thread Group (Test Plan > Threads (Users) > Thread Group).
- б) Задать для Thread Group следующие параметры:
 - 1) Действие, которое будут производиться в случае, если в тест выполняется с ошибкой (Action to be taken after a Sampler error) — Continue ;
 - 2) число потоков, в которое будут запускаться шаги тест-плана (Number of Threads) установить равным единице;
 - 3) Интервал, в течение которого будет запущено указанное в предыдущем параметре число потоков (Ramp-Up Period) установить равным единице;
 - 4) Число повторений набора тестов (Loop Count) установить равным единице;

в) добавить визуалайзер результатов, чтобы иметь возможность отслеживать ход выполнения теста (Thread Group > Add > Listener > View Results Tree)

Шаги теста:

а) В Thread Group в качестве дочернего элемента добавить AMF RPC Sampler;

б) Ввести в AMF RPC Sampler неверное имя метода удалённого объекта и запустить содержимое элемента Test Plan (Run > Start);

Ожидаемый результат:

а) После завершения прогона тестов во View Results Tree в дереве элементов тест плана элемент AMF RPC Sampler подсвечен красным цветом (тест не пройден);

б) на вкладках View Results Tree присутствуют данные запроса и полученного от сервера ответа;

в) Ответ сервера должен содержать сообщение о соответствующей ошибке.

В.2.4 Проверка запуска содержимого Test Plan в несколько потоков

Предусловия:

а) Добавить в Test Plan группу потоков - Thread Group (Test Plan > Threads (Users) > Thread Group).

б) Задать для Thread Group следующие параметры:

- 1) Действие, которое будут производиться в случае, если в тест выполняется с ошибкой (Action to be taken after a Sampler error) — Continue ;
- 2) число потоков, в которое будут запускаться шаги тест-плана (Number of Threads) установить равным десяти;
- 3) Интервал, в течение которого будет запущено указанное в предыдущем параметре число потоков (Ramp-Up Period) установить равным единице;

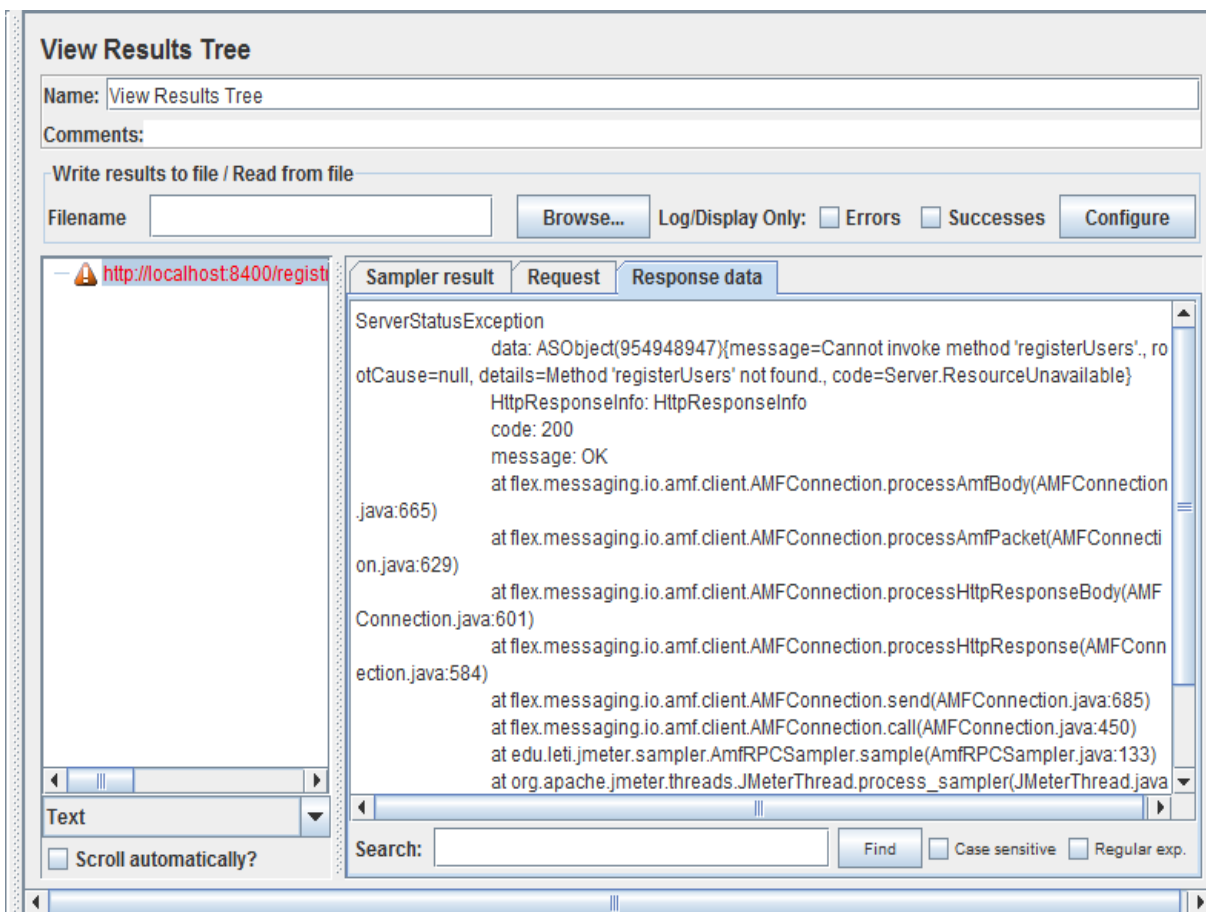


Рисунок В.6 — Результаты некорректного теста

- 4) Число повторений набора тестов (Loop Count) установить равным единице;
- в) добавить визуализатор результатов, чтобы иметь возможность отслеживать ход выполнения теста (Thread Group > Add > Listener > View Results Tree)

Шаги теста:

- а) В Thread Group в качестве дочерних элементов добавить около пяти элементов AMF RPC Sampler;
- б) Ввести в элементы AMF RPC Sampler все необходимые корректные параметры и запустить тест план;

Ожидаемый результат:

После завершения прогона тестов во View Results Tree в дереве элементов все тесты должны быть пройдены — корректный ответ от сервера должен быть получен для каждого элемента.