МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программной и системной инженерии

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
д.н., профессор
_____А.Г. Ивашко
_____2018 г.

МАГИСТЕРСКАЯ ДИСЕРТАЦИЯ

Обнаружение ошибок в mule-приложениях на основе статического анализа кода

09.04.03 Прикладная информатика

Магистерская программа «Прикладная информатика в экономике»

Выполнил работу
Студент 2 курса
очной формы обучения
(Подпись)
Красиков
Виктор
Евгеньевич

Руководитель работы
д.н., профессор
(Подпись)
Бидуля
Юлия
Владимировна

Рецензент
к.т.н., ст. преподаватель
(Подпись)
Григорьев
Андрей
Викторович

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
КАФЕДРА ПРОГРАММНОЙ И СИСТЕМНОЙ ИНЖЕНЕРИИ

ДОПУЩЕНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
д.т.н., профессор
_____А.Г. Ивашко
_____2018 г.

МАГИСТЕРСКАЯ ДИСЕРТАЦИЯ

Обнаружение ошибок в mule-приложениях на основе статического анализа кода
09.04.03 Прикладная информатика
Магистерская программа «Прикладная информатика в экономике»

Выполнил работу Студент 2 курса очной формы обучения	*(Подпись)*	Красиков Виктор Евгеньевич
Руководитель работы д.н., профессор	*(Подпись)*	Бидуля Юлия Владимировна
Рецензент к.т.н., ст. преподаватель	*(Подпись)*	Григорьев Андрей Викторович

Тюмень 2018

РЕФЕРАТ

Выпускная квалификационная работа по теме «Обнаружение ошибок в Mule приложениях на основе статического анализа кода» содержит [34] страницы текстового документа, [4] приложение, [22] использованный источник литературы, [1] иллюстраций.

Объектом разработки выступает инструмент поиска ошибок в Mule-приложениях.

Результаты работы:

В процессе работы был разработан инструмент, выполняющий поиск ошибок в Mule-приложениях при сборке приложения.

Оглавление

РЕФЕРАТ 1

Список сокращений и обозначений 3

Введение 5

Глава 1. Описание предметной области 8

1.1. Постановка задачи 13

Глава 2. Рассмотрение существующих подходов поиска ошибок в многопоточных программах на основе статического анализа 14

Глава 3. Разработка инструмента промежуточного представление исходного текста 17

1.1. Преобразование исходных данных в универсальное представление 19

Глава 4. Разработка инструмента для анализа полученного графа потоков управления из промежуточного представления исходного кода 24

Глава 5. Эксперимент 28

Заключение 31

Список литературы 32

Приложение 35

Список сокращений и обозначений

CMS - (Content management system) информационная система или компьютерная программа, используемая для обеспечения и организации совместного процесса создания, редактирования и управления контентом.

AMQP - (Advanced Message Queuing Protocol) открытый протокол для передачи сообщений между компонентами системы.

HTTP - (HyperText Transfer Protocol) протокол прикладного уровня передачи данных.

LDAP - (Lightweight Directory Access Protocol) протокол прикладного уровня для доступа к службе каталогов X.500.

X.500 — серия стандартов ITU-T для службы распределенного каталога сети.

ITU-T — (International Telecommunication Union - Telecommunication sector) сектор стандартизации электросвязи Международного союза электросвязи.

cURL - инструмент для переноса данных с сервера или на сервер, используя один из поддерживаемых протоколов (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP).

СУБД - система управления базами данных.

MVC – (Model-view-controller) схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные.

API – (application programming interface) набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах.

CRON - классический демон-планировщик задач в UNIX-подобных операционных системах, использующийся для периодического выполнения заданий в определённое время. Регулярные действия описываются инструкциями, помещенными в файлы crontab и в специальные директории.

Введение

Программы всё больше используется как компоненты технических систем. Поэтому проблема повышения качества программного обеспечения стоит достаточно остро. Каждая программа реализует какую-либо логику, зачастую являющейся весьма ответственные и сложные задачи. Но также практически все программы содержат ошибки. **Нефункциональные ошибки** являются одним из видов ошибок в программном обеспечении. Это ошибки, связанные с нарушениями правил какого-либо языка программирования.

Возникновение ошибок может привести к получению неверных результатов, нарушению конфиденциальности хранимой информации, зависанию или сбою работы программы [1].

При разработке программы 50-60% всех затрат уходит на тестирование и отладку. Также, программисты тратят на исправление опечаток гораздо больше времени, чем на непосредственно реализацию [2].

На текущий момент методы автоматизированного обнаружения ошибок разделяются на [3]:

- **статические методы** – для обнаружения ошибок берутся артефакты, созданные в процессе проектирования и разработки программного продукта.
- **динамические методы** — для обнаружения берутся результаты работы разрабатываемого программного продукта
- **гибридные методы** — комбинирование динамических и статических методов

Динамические методы, такие как тестирование, анализ выполнения и мониторинг, обеспечивают получение конкретных результатов. Но основной недостаток динамических методов заключается в невозможности их полной автоматизации и неполноте полученных результатов.

Статические методы включают в себя методы проверки моделей, методы дедуктивной верификация и методы статического анализа. **Методы проверки моделей** [4, 5] гарантируют обнаружение нарушений спецификации, обеспечивают получение полных и точных результатов, позволяют анализировать параллельные программы и обнаруживать ошибки синхронизации. Основными недостатками этих методов являются необходимость ручного построения модели программы и высокая ресурсоемкость алгоритмов. **Методы дедуктивной верификация** [6] позволяют строго доказывать определенные свойства программы и могут применяться для обнаружения программных дефектов. Однако, существующие методы дедуктивной верификации допускают лишь частичную автоматизацию и требуют высокой квалификации пользователя. Обнаружение программных дефектов на основе **методов статического анализа кода** [7, 8] представляется наиболее перспективным подходом. Применение статического анализа кода позволяет обнаруживать все основные типы ошибок, а также обеспечивает получение полных результатов [9]. Также статический анализ кода позволяет полностью автоматизировать процесс ошибок.

Методы статического анализа широко применяется для оптимизации программ, написанных на различных языках, маскирования преобразований и выявления уязвимостей защиты программ, автоматизации обнаружения программных ошибок [10].

Информационные системы содержат в себе программное обеспечение, являющееся программой или набором программ. Отсюда следует, что для

информационных систем так же возможно применять методы статического анализа для обнаружения программных дефектов.

Целью работы данного инструмента является улучшения процента числа найденных ошибок на момент сборки приложения для сокращения времени отладки разрабатываемого приложения.

Глава 1. Описание предметной области

Использование информационных систем на сегодняшний день актуально как никогда. Информационные системы не пишут «с нуля», так как существует много готовых решений для этой выполнения необходимых задач. Для того, чтобы сократить стоимость информационной системы, покупают подходящее готовое решение и интегрируют его с уже имеющимися программными продуктами.

Современные информационные системы (ИС) редко изолированны; ИС не может сделать что-либо значимое без взаимодействия с другими ИС. Сервис-ориентированная архитектура интегрирует ИС для совместной работы и ускоряет их работу, разбивая ИС на части, которые могут быть объединены друг с другом. Модель SOA (потребители службы вызывают поставщиков службы) может показаться простой, но возникают две важные проблемы:

- Как потребителю найти провайдера службы, которую он хочет вызвать?
- Как потребитель может быстро и надежно вызвать службу в медленной и ненадежной сети?

Оказывается, существует прямое решение обеих этих проблем – подход, называемый Enterprise Service Bus (ESB – сервисная шина предприятия) [12].

Одной из наиболее популярных ESB в мире является Mule [13].

Mule приложение включает в себя потоки, состоящие из блоков, каждый из которых представляет собой событие в потоке, определяя, как обрабатывается Mule сообщение для следующего этапа в потоке. События этих потоков имеют настраиваемые поля, которые определяют, как будет получены данные, как они будут изменяться и что вернётся, как и большинство функций в других программных продуктах. Каждое событие в потоке Mule приложения

представляет собой модульную часть процесса, по которому передаются Mule сообщения. Mule сообщения принимают коннекторы. **Коннектор** – это модуль, который облегчает взаимодействие между Mule приложением и внешними ресурсами, такими как базы данных или API, через REST, SOAP или Java SDK.

Каждый запрос, поступающий в Mule, обрабатывается в своем потоке. Коннекторы имеет пул потоков с определенным количеством потоков, доступных для обработки запросов на входящих конечных точках, которые используют этот коннектор [14, 15].

Многопоточность - это специализированная форма многозадачности. В основном, выделяют два типа многозадачности: основанную на процессах и основанную на потоках.

Процесс - это по сути запущенная программа. Следовательно, основанная на процессах многозадачность - средство, позволяющее компьютеру выполнять несколько операций (программ) одновременно. Например, основанная на процессах многозадачность предоставляет одновременно редактировать текст в текстовом редакторе и работать с другой запущенной программой.

Поток - это управляемая единица исполняемого кода. В многозадачной среде, основанной на потоках, у всех работающих процессов обязательно имеется основной поток, но их может быть и больше. Это означает, что в одной программе могут выполняться несколько задач асинхронно. К примеру, редактирование текста в текстовом редакторе во время печати, т.к эти две задачи выполняются в различных потоках.

Всего различают две разновидности многозадачности: на основе процессов и на основе потоков. Отличия многозадачности на основе процессов и потоков сводится к следующему: многозадачность на основе процессов организуется для параллельного выполнения программ, а многозадачность на основе потоков - для параллельного выполнения отдельных частей одной программы [16].

Принцип работы Mule приложения схож с принципом работы многопоточной программы, основанной на потоках. Поэтому необходимо применять методы статического анализа допускаемых для многопоточных программ.

Высокая популярность сервисов шины данных Mulesoft ESB, а также отсутствие готового инструмента для поиска ошибок в mule-приложениях, свидетельствуют о том, что задача разработки и реализации методов обнаружения ошибок в mule-приложениях на основе статического анализа является актуальной. В данной работе в рассматриваемом эксперименте использовалось mule-приложения на Muleыщае ESB.

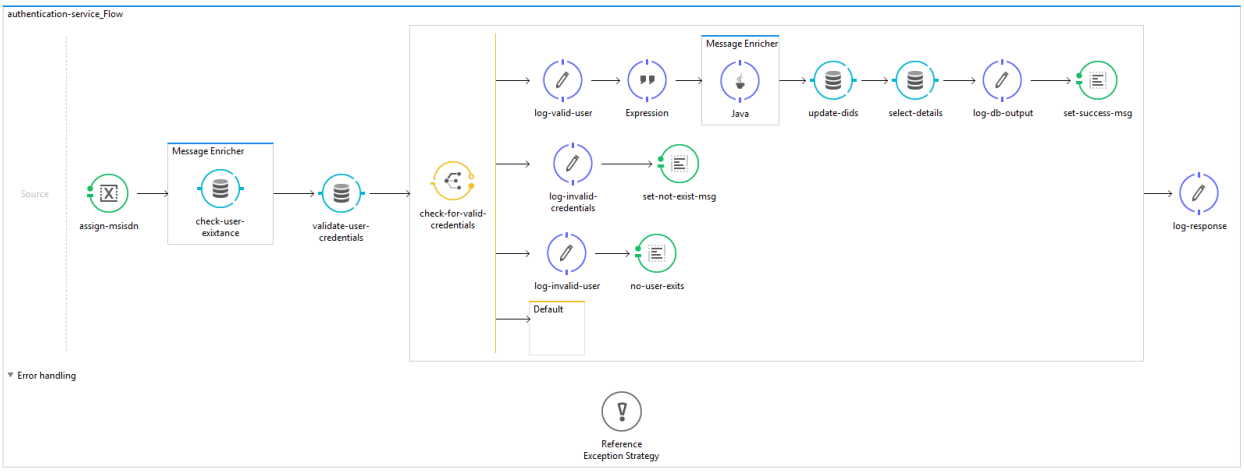
MuleSoft - это компания, предоставляющая платформу Mule. Mule –это платформа интеграции для приложений, данных и API через локальные и облачные вычислительные среды.

Для того, чтобы обеспечить быстродействие как на локальных машинах, так и в облаке, платформа AnyPoint, разработанная MuleSoft, интегрирует приложения с помощью API. Кроме того, платформа имеет сервис-ориентированную архитектуру (SOA)

Ядро среды выполнения платформы AnyPoint, представляет собой легковесную, основанную на Java, корпоративную интеграционную шину (ESB) и платформу интеграции, которая позволяет разработчикам быстро и легко подключать приложения между собой для обмена данными.

Она позволяет легко интегрировать существующие системы, независимо от технологий, которые используются в приложении, в том числе JMS, Web Services, JDBC, HTTP и многое другое. [22].

Mule приложение включает в себя потоки, состоящие из блоков, каждый из которых представляет собой событие в потоке, определяя, как обрабатывается Mule сообщение для следующего этапа в потоке.



События этих потоков имеют настраиваемые поля, которые определяют, как будет получены данные, как они будут изменяться и что вернётся, как и большинство функций в других программных продуктах.

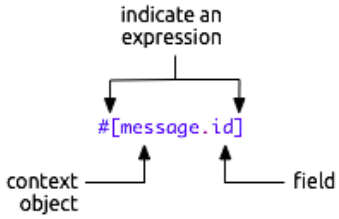
События соответствуют элементам XML в конфигурационном файле XML.

```
<?xml version="1.0" encoding="UTF-8"?>

<mule xmlns:scripting="http://www.mulesoft.org/schema/mule/scripting" xmlns:http="http://www.mulesoft.
http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/current/mule.xsd
http://www.mulesoft.org/schema/mule/xml http://www.mulesoft.org/schema/mule/xml/current/mule-xml.xsd
http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current/mule-http.xs
http://www.mulesoft.org/schema/mule/scripting http://www.mulesoft.org/schema/mule/scripting/current/mu

    <http:listener-config name="HTTP_Listener_Configuration" host="localhost" port="8081" doc:name="HT
    <flow name="Catalog_DownloaderFlow1" >
        <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
        <mulexml:xml-to-object-transformer doc:name="XML to Object"/>
        <scripting:component doc:name="Groovy">
            <scripting:script engine="Groovy" file="myScript.groovy"/>
        </scripting:component>          <logger level="INFO" doc:name="Logger"/>
    </flow> </mule>
```

Рис. Конфигурационный файл в Mule приложении в XML представлении

Каждое событие в потоке Mule приложения представляет собой модульную часть процесса, по которому передаются Mule сообщения.

Эти данные могут быть вызваны и на них можно ссылаться в Mule приложении, используя Mule Expression Language (MEL), который представляет собой специализированный синтаксис Mule.



1.1. Постановка задачи

Для разработки инструмента, позволяющего выполнять поиск ошибок в Mule приложениях во время их сборки необходимо выполнить следующие задачи:

- Разработать получение промежуточного представления из исходного текста
  - Изучить и подобрать существующие подходы поиска ошибок в программном коде;
  - Реализовать приведение к промежуточному представлению из исходного текста Mule приложения для применения выбранного подхода к поиску ошибок в программном коде;
  - Привести полученные промежуточные представления к универсальному классовому представлению.
- Получить граф потока управления в виде xml дерева
  - Визуализация полученного графа потока управления для возможности оценки корректности получаемого промежуточного представления.
- Реализовать поиск ошибок с помощью полученного xml дерева
  - Описание правил обнаружения ошибок в Mule приложении;
  - Реализовать анализ полученного графа потока управления для выявления ошибок;
  - Реализовать инструмент поиска ошибок в виде maven плагина
  - Провести эксперимент для оценки работы разработанного инструмента.

Глава 2. Рассмотрение существующих подходов поиска ошибок в многопоточных программах на основе статического анализа

В работе М.Ю. Моисеева «Автоматическое обнаружение дефектов в многопоточных программах методами статического анализа» рассмотрен подробный анализ существующих подходов поиска ошибок в многопоточных программах на основе статического анализа.

Целью рассмотрения данной работы является определение возможности их применения для поиска ошибок в Mule приложений.

Сложность реализации методов статического анализа зависит от выбранного языка программирования. Для обнаружения дефектов в Mule приложениях необходимо решать большое число специфических задач. Принцип работы Mule приложения схож с принципом работы многопоточных программ, о чём говорилось в первой главе данной работы. В настоящее время уже реализованы инструменты для автоматического поиска ошибок для различных языков программирования, но инструментов, позволяющих выполнять данный поиск в Mule-приложениях, к сожалению, ещё не реализовано. Используемые методы анализа ошибок должны предоставлять возможность получения всей необходимой информации для обнаружения поддерживаемых типов ошибок.

Основными показателями эффективности алгоритмов статического анализа являются полнота и точность обнаружения ошибок. Одним из важных преимуществ методов статического анализа является возможность получения полных результатов (обнаружение всех ошибок поддерживаемых типов, которые имеются в программе).

В качестве итога сформулируем основные требования к методам анализа многопоточных программ на Mule ESB:

- Поддержка всего множества конструкций Mule ESB (возможность анализа произвольных программ без ограничений).
- Анализ состояний обычных объектов программы (определение значений переменных-указателей и интервальных переменных).
- Извлечение необходимой информации для обнаружения широкого класса программных дефектов.
- Анализ произвольного числа потоков программы.
- Анализ произвольного числа объектов синхронизации поддерживаемых типов с идентификацией этих объектов в разных потоках.
- Совместное выполнение алгоритмов анализа с обменом результатами между этими алгоритмами.
- Получение результатов с полнотой равной или близкой к 100% (обнаружение всех или почти всех дефектов поддерживаемых типов, имеющихся в программе).

Критериями оценки рассматриваемых методов и средств обнаружения ошибок является удовлетворение перечисленным выше требованиям. Для средств обнаружения ошибок дополнительным критерием является наличие открытой информации о методах и алгоритмах, используемых в этих средствах, а также об эффективности работы этих средств на открытых тестовых наборах или программах с открытым исходным кодом [17].

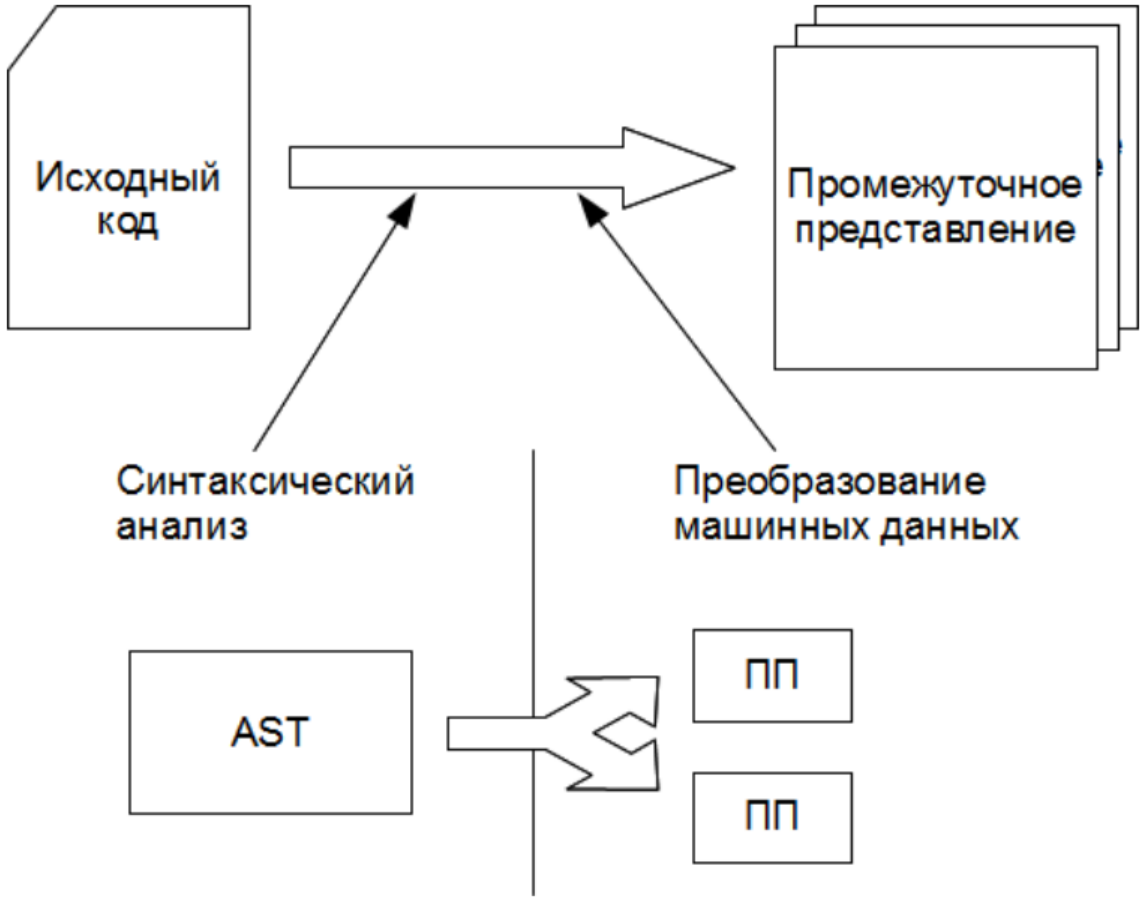
В работе [18] в качестве модели программы используется расширенный граф потока управления, который содержит дуги переходов, дуги синхронизации и дуги, обозначающие наличие конфликтов — CCFG (Concurrent CFG). Вершинами CCFG являются блоки последовательных конструкций программы.

В работах [19] и [20] до некоторой степени решается задача определения порядков выполнения конструкций в потоках программы, однако при этом не выполняется анализ и учет состояний обычных объектов программы (интервальных переменных и переменных-указателей). Отсутствие совместного анализа значений всех объектов программы негативно влияет на эффективность подходов, рассматриваемых в этих работах, и не позволяет применять их для анализа реальных многопоточных программ на Mule ESB.

Итог данной главы можно считать то, что для решения поставленных задач наиболее подходящим способом реализации инструмента для поиска ошибок представляет преобразование многопоточного приложения к последовательному виду посредством промежуточного преобразование исходного текста для сокращения затрат ресурсов при последующей обработке, в первую очередь трудозатрат на разработку инструментов.

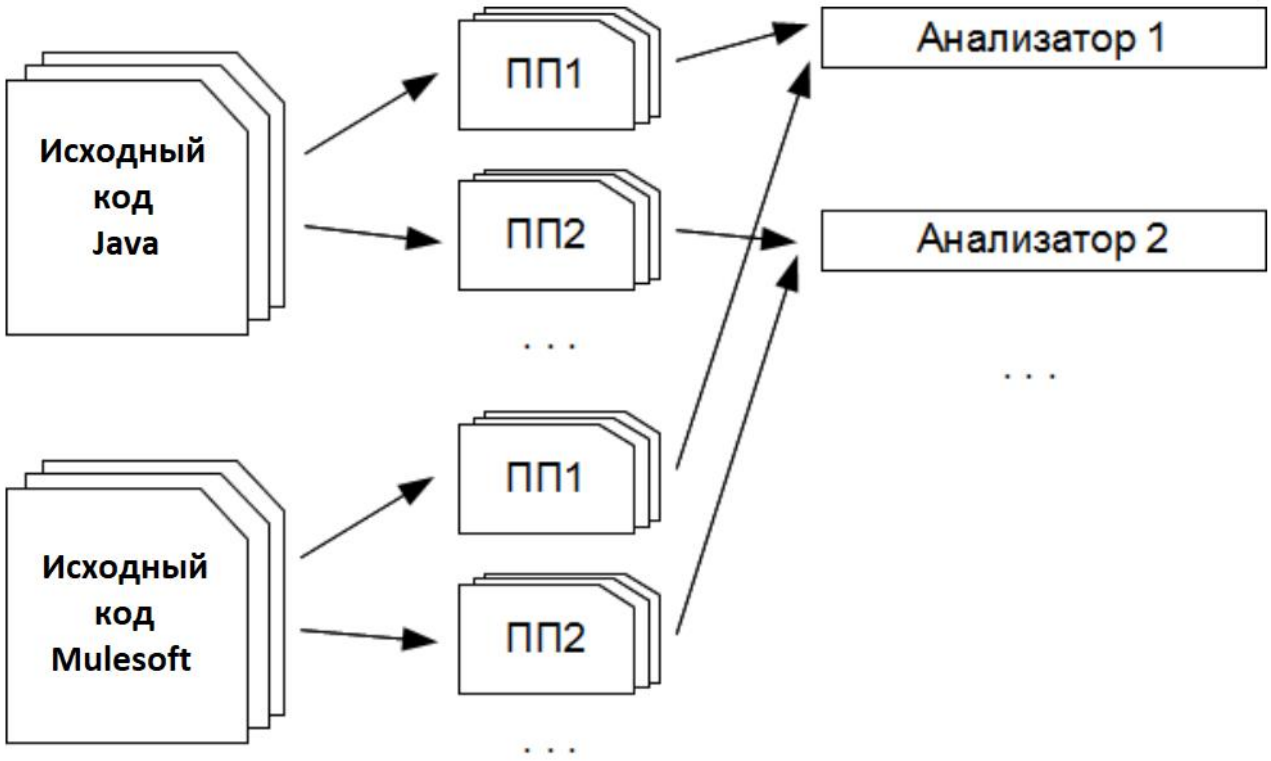
Глава 3. Разработка инструмента промежуточного представление исходного текста

Промежуточное представление исходного текста - это текстовый набор данных, над которым выполняется анализ. ПП получается из исходного кода с целью сокращения затрат ресурсов при последующей обработке, в первую очередь трудозатрат на разработку инструментов.



При ПП необходимо выбрать формат представления структурированных данных наиболее распространёнными являются XML, JSON и YAML. При этом выбор XML позволяет сократить затраты на создание инструментов анализа и построения эквивалентных представлений благодаря большого числа инструментов по извлечению знаний из исходных текстов для XML.

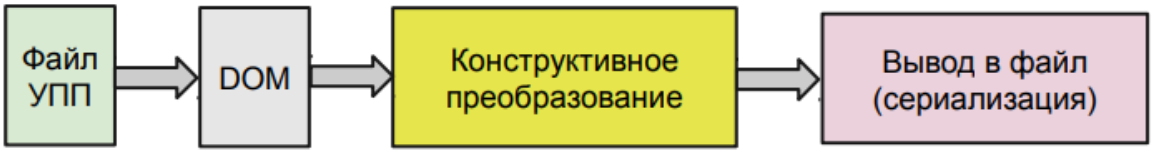
Универсальные многоуровневые представления в анализе исходного текста - промежуточные представления, описывающие родственные синтаксические конструкции разных языков наиболее близким, то есть общим способом. Строятся из представлений предыдущего уровня путем сохранения только необходимые для конкретного анализа данных Предназначены для упрощения конкретного анализа «прореживанием» предыдущего ПП Универсальный (единый) формат позволяет использовать единый программный инструмент для исходных текстов на разных языках.



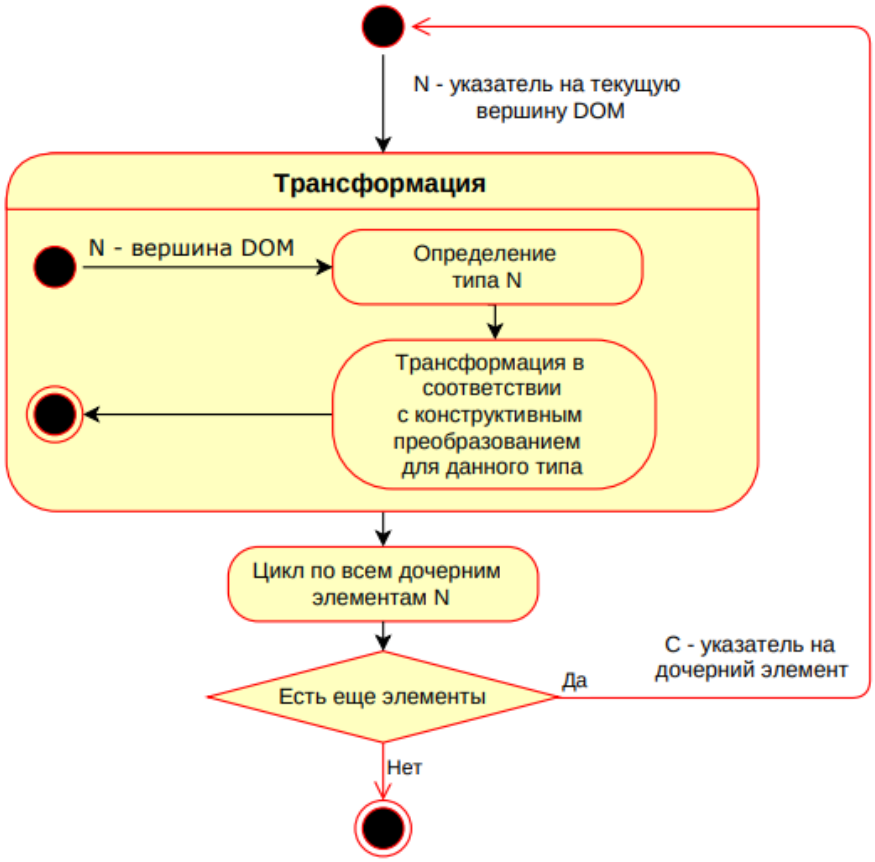
Для преобразования универсальное классовое представление (UCR) для Mulesoft ESB и Java предназначено для построения и анализа диаграммы классов по исходному тексту проекта. Позволяет применять унифицированный инструмент анализа диаграммы классов для исходного текста Mulesoft ESB и Java. Данная библиотека доступна в открытом доступе.

1.1. Преобразование исходных данных в универсальное представление

Работа алгоритм по получения из исходного кода mule-приложения универсального промежуточного представления заключается в преобразовании в абстрагированный набор операций, обёрнутых в операторы потока управления.



Алгоритм преобразования в CFG



Ограничения, принятые для построение CFG:

- Граф строится только для функции или метода, с возможностью связывания отдельных частей друг с другом
- Вызовы из метода пока не учитываются
- Участки последовательного кода без ветвления представляются одним узлом
- Представление - ориентированный граф, предназначенный для анализа известными методами
- Текстовая форма основана на открытом формате GraphML (graphml.graphdrawing.org), позволяющем описывать в XML графы любого вида.

Реализация инструмента для построения графа потока управления был выбран плагин для IDE Eclipse, поскольку в данной IDE происходит разработка как Java программ, так и проектов, использующих Mulesoft ESB.

Для визуализации получаемого графа используется формат dot из пакета graphviz (www.graphviz.org). а каждый метод создается отдельное изображение.

Далее приведён пример на псевдокоде, который идентичен приведённому исходному коду для mule-приложения и полученный для данного куска кода, результат.

- *Псевдокод:* **Если x больше y то вернуть x;**

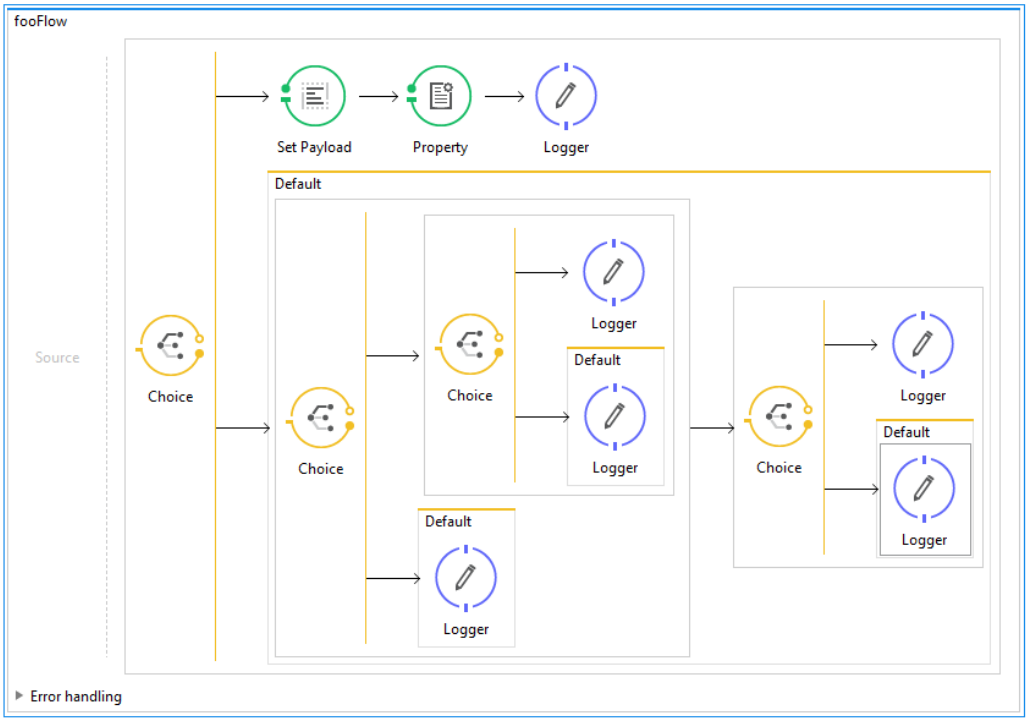
- *Исходный текст:*

```
<choice doc:name="">
    <when
        expression="#[flowVars.x > flowVars.y]">
        <set-payload value="flowVars.x"></set-payload>
    </when>
    <otherwise>
    </otherwise>
</choice>
```

- *Эквивалентное представление:*

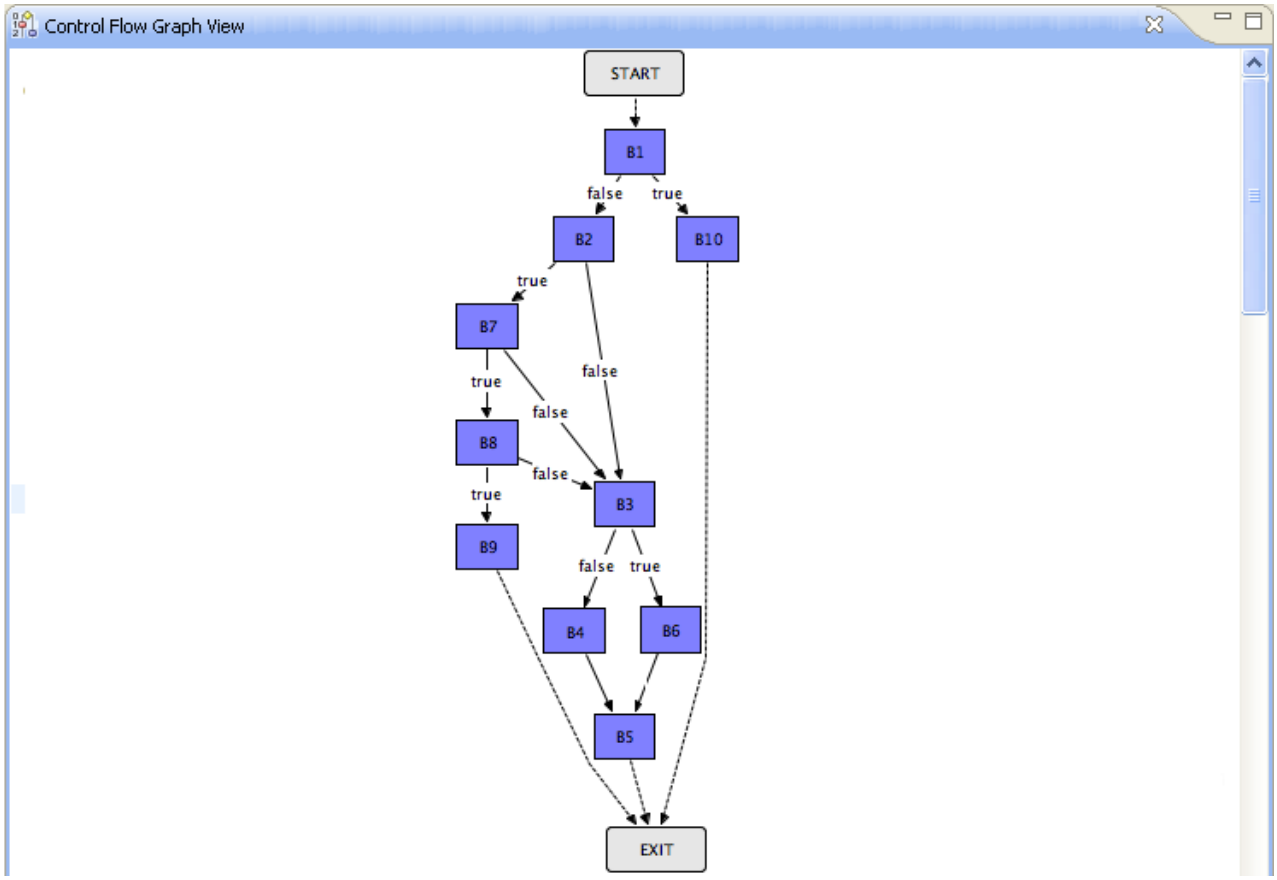
```
<If>
 <Condition>
   <Read name="y" id="2"/>
   <Read name="x" id="1"/>
 </Condition>
 <Then>
   <Return><Copy name="x" id="1"/></Return>
 </Then>
</If>
```

Пример приложения на Mulesoft ESB:




```
<flow name="fooFlow">
    <choice doc:name="Choice">
        <when expression="#[message.properties['isCond1']]">
            <set-payload doc:name="Set Payload"/>
            <properties-transformer doc:name="Property"/>
            <logger level="INFO" doc:name="Logger"/>
        </when>
        <otherwise>
            <choice doc:name="Choice">
                <when expression="#[message.properties['isCond2']]">
                    <choice doc:name="Choice">
                        <when expression="#[message.properties['isCond3']]">
                            <logger level="INFO" doc:name="Logger"/>
                        </when>
                        <otherwise>
                            <logger level="INFO" doc:name="Logger"/>
                        </otherwise>
                    </choice>
                </when>
                <otherwise>
                    <logger level="INFO" doc:name="Logger"/>
                </otherwise>
            </choice>
            <choice doc:name="Choice">
                        <when expression="#[message.properties['isCond4']]">
                            <logger level="INFO" doc:name="Logger"/>
                        </when>
                        <otherwise>
                            <logger level="INFO" doc:name="Logger"/>
                        </otherwise>
                    </choice>
        </otherwise>
    </choice>
</flow>
```

Полученная визуализация графа потока управления



Глава 4. Разработка инструмента для анализа полученного графа потоков управления из промежуточного представления исходного кода

Для анализа полученного графа потоков управления был написан инструмент, позволяющий по заданным правилам производить поиск ошибок в прилагаемых файлах проверяемого проекта.

В данном проекте в формате xml-разметки указаны правила, которые необходимо проверять при анализе полученного графа потоков.

```
<rules>
        <rule category="Security">
                <checks name="Passwords">
                        <check match="//@password" pattern="external-property"/>
                        <check match="//@storePassword" pattern="external-property" />
                </checks>
                <checks name="Check secured-properties">
                        <check match="//secure-property-placeholder:config/@key" pattern="external-property" />
                        <check match="//secure-property-placeholder:config/@location" pattern="external-property" />
                </checks>
        </rule>

        <rule category="Flow and sub-flow names">
                <checks name="Check for lowercase, hyphenated names">
                        <check match="//:flow/@name" pattern="lowercase-hyphenated-words"/>
                        <check match="//:sub-flow/@name" pattern="lowercase-hyphenated-words"/>
                </checks>
        </rule>

        <rule category="Variable names">
                <checks name="Check for camel-case">
                        <check match="//:set-variable/@variableName" pattern="lowercase-camel-case-property"/>
                </checks>
        </rule>
        <rule category="HTTP Inbound-endpoint">
                <checks name="Check for hard-coded values">
                        <check match="//http:inbound-endpoint/@host" pattern="external-property"/>
                        <check match="//http:inbound-endpoint/@port" pattern="external-property"/>
                </checks>
```

Рис. Пример описания правил для анализа Mule-приложения

Также была добавлена возможность описать дополнительные файлы в формальном виде:

```
version '0.0.1'

element 'logger' hasAttribute 'category'

element 'logger' hasParent 'until-successful'

element 'http:request' hasChild 'User-Agent'

element 'http:request' hasChild 'http:header' withAttribute 'headerName' havingValue 'User-Agent' withAttribute
```

Данные инструмент реализован в виде Maven-плагина. На входе данный плагин получает:

- путь до файла с дополнительными правилами
- путь до файла Mule-приложения
- путь до файла, в который будет сохранён результат проверки Mule-приложения.

Далее приведён пример на псевдокоде, который идентичен приведённому УПП для mule-приложения и полученный для данного УПП результат.

- *Псевдокод:*

  Определить переменной y значение "#[payload.Uetr]"

  Если x больше y то вернуть x;

- *УПП:*

```
<flow name="testPluginFlow">
    <Deft name="y" id="1" value="#[payload.Uetr]"/>
    <If>
        <Condition>
            <Read name="y" id="1"/>
            <Read name="x" id="2"/>
        </Condition>
        <Then>
            <Return><Copy name="x" id="2"/></Return>
        </Then>
    </If>
</flow>
```

- *Правило:*

```
<rule catefory="Variable initialize">
    <check match="//:deft/@variableName" pattern="variables"/>
    <check match="//:read/@variableName" pattern="variables"/>
    <check match="//:modf/@variableName" pattern="variables"/>
</rule>
```

- *Результат:*

- *{*
- *"countErrors": 1,*
- *"findings": [*
- *"No signature of flowVariable: value: ['x']"*
- *],*
- *"version": "0.0.2"*
- *}*

Пример настройки плагина:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.krasikov</groupId>
      <artifactId>mule-lint-maven-plugin</artifactId>
      <version>0.1.0</version>
      <configuration>
        <rules>rules.txt</rules>
        <sources>src/main/app</sources>
        <output>mule-lint-results.json</output>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Пример вызова настроенного плагина:

```
mvn org.krasikov:mule-lint-maven-plugin:analyze-mule
```

Глава 5. Эксперимент

Для проверки работы плагина по указанным выше количественным показателям было реализовано Mule-приложение со следующими ошибками:

- Использование неинициализированной переменной. Количество: 5.
- Получение данных из конфигурационных файлов приложения. Количество: 4.
- Вызов потока, не описанного в приложении. Количество: 1.

Для проверки эффективности разработанного инструмента будут использоваться следующие количественные показатели [19]:

- **полнота** - доля обнаруженных дефектов среди всех дефектов, имеющихся в программе,
- **точность** - доля истинных дефектов среди всех обнаруженных дефектов,
- **степень автоматизации** - трудоемкость и сложность применения метода,
- **ресурсоемкость** - количество выполняемых операций и необходимый объем памяти.

Полнота определяется по следующей формуле: ПОД = 100 * (R / C).

В рамках этой формулы литерой R обозначается число обнаруженных ошибок, литерой C – общее количество ошибок.

Точность определяется по следующей формуле: ТОД = 100 * (F / P).

В рамках этой формулы литерой F обозначается число корректно обнаруженных ошибок, литерой P – общее количество ошибок.

Всего в тестовое приложение было внесено 6 ошибок. Далее привожу результаты по указанным количественным показателям сборки приложения для его дальнейшего запуска без разработанного плагина и вместе с ним:

Количественный показатель	Без плагина	С плагином
полнота	50%	100%
точность	100%	100%
степень автоматизации	Полностью автоматически	Полностью автоматически
ресурсоемкость	0,33 ошибки в секунду Время сборки: 15 сек. Оперативная память: 20 mb RAM Загруженность процессора: 0,01089 CPU Время запуска после сборки: 12 сек.	0,5 ошибок в секунду Время сборки: 20 сек. Оперативная память: 40 mb RAM Загруженность процессора: 0,022 CPU Время запуска после сборки: 12 сек.

Результат работы плагина:


```
{
    "countErrors": 6,
    "findings": [
        "No signature of flow: value: ['testNotFoundSubFlow']",
        "No signature of flowVariable: value: ['testNotFoundVariable1']",
        "No signature of flowVariable: value: ['testNotFoundVariable2']",
        "No signature of flowVariable: value: ['testNotFoundVariable3']",
        "No signature of configProperties: value: ['missedProperties1']",
        "No signature of configProperties: value: ['missedProperties2']"
    ],
    "version": "0.0.2"
}
```


При сборке приложения с использованием разработанного плагина были найдены все заданные ошибки, в отличии от сборки, в которой он не использовался. Поскольку были найдены все ошибки, больше нет необходимости искать их вручную при работе приложения.

Заключение

Разработанный инструмент позволил определить ошибки, которые были пропущены при старте сборки приложения. Это позволит сократит время на отладку приложения, поскольку мы исключаем шаг запуска приложения после сборки для определения ошибки вручную.

Данный инструмент разработан и размещен в публичном репозитории по адресу https://github.com/vekrasikov/csu-code-analysis .

Я считаю использование подобного плагина является целесообразным в рамках поставленной мною задачи, по результатам эксперимента, поскольку разработанный инструмент позволяет сократит время разработки Mule приложения на t, где t – время запуска приложения после сборки.

Список литературы

1. Автоматическое обнаружение дефектов в многопоточных программах методами статического анализа [Электронный ресурс]. — URL: http://netess.ru/3informatika/290014-1-avtomaticheskoe-obnaruzhenie-defektov-mnogopotochnih-programmah-metodami-staticheskogo-analiza.php (дата обращения: 11.01.2018)
2. Тестирование информационных систем [Электронный ресурс]. URL: http://aplana.ru/services/testing/testirovanie-sistem (Дата обращения: 11.01.2018)
3. Кулямин В.В. Методы верификации программного обеспечения // Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению "Информационно-телекоммуникационные системы", 2008. — 117 с.
4. Карпов Ю.Г. Model Checking. Верификация параллельных и распределенных программных систем. — СПб.: БХВ-Петербург, 2010. 552 с.
5. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. — М.: МЦНМО, 2002. — 416 с.
6. Peled D. Software Reliability Methods — Berlin: Springer-Verlag, 2001. — 331 p.
7. Chess B., West J. Secure Programming with Static Analysis. — Addison- Wesley, 2007. —619 p.
8. Nielson F., Nielson H.R., Hankin C. Principles of Program Analysis. — Berlin: Springer, 2005. — 452 p.
9. Gotsman A., Berdine J., Cook B., Sagiv M. Thread-Modular Shape Analysis // Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation. — New York: ACM, 2007. —p. 266-277.
10. Гайсарян С.С., Чернов А.В., Белеванцев А.А. и др. О некоторых задачах анализа и трансформации программ // Труды Института системного программирования РАН, 2004.

11. Моисеев, Михаил Юрьевич Автоматическое обнаружение дефектов в многопоточных программах методами статического анализа : автореферат дис. ... кандидата технических наук : 05.13.11 Санкт-Петербург 2011
12. Зачем разработчикам нужна Enterprise Service Bus? [Электронный ресурс]. URL: https://www.ibm.com/developerworks/ru/library/ws-whyesb/ (Дата обращения: 22.03.2017)
13. Data Integration Software List [Электронный ресурс]. URL: https://www.g2crowd.com/categories/data-integration/products (Дата обращения: 22.03.2017)
14. About Threads in Mule [Электронный ресурс]. URL: https://docs.mulesoft.com/mule-user-guide/v/3.6/tuning-performance Дата обращения: 22.03.2017)
15. What is a Connector? [Электронный ресурс]. URL: https://docs.mulesoft.com/anypoint-connector-devkit/v/3.7/ (Дата обращения: 22.03.2017)
16. INTUIT.ru: Курс: Основы параллельного ..: Лекция №4: Многопоточность в .NET Framework [Электронный ресурс]. URL: http://www.intuit.ru/studies/courses/4807/1055/lecture/16374 (Дата обращения: 22.03.2017)
17. Emrath P., Ghosh S., Padua D. Detecting Nondeterminacy in Parallel- Programs // In IEEE Software, v.9, n.l. — Washington: IEEE Computer Society, 1992—p. 69-77.
18. Lai A., Reps T. Reducing Concurrent Analysis Under a Context Bound to Sequential Analysis // Formal Methods in System Design. — New York: ACM, 2009. —p. 73-97.

19. Jackson D., Rinard M. Software Analysis: A Roadmap // Proceedings of the Conference on The Future of Software Engineering. — New York: ACM, 2000, — p. 133-145.
20. Emrath P., Ghosh S., Padua D. Detecting Nondeterminacy in Parallel- Programs // In IEEE Software, v.9, n.l. — Washington: IEEE Computer Society, 1992—p. 69-77.
21. Ladkin P., Simons B. Static Analysis of Interprocess Communication. Lecture Notes in Computer Science. — Berlin: Springer, 1995. — 145 p.
22. What is Mule ESB? [Электронный ресурс]. URL: https://www.mulesoft.com/resources/esb/what-mule-esb (Дата обращения: 25.12.2016)

Приложение

MuleSoft Certified Developer

Viktor Krasikov

Has successfully completed the requirements to be recognized as a

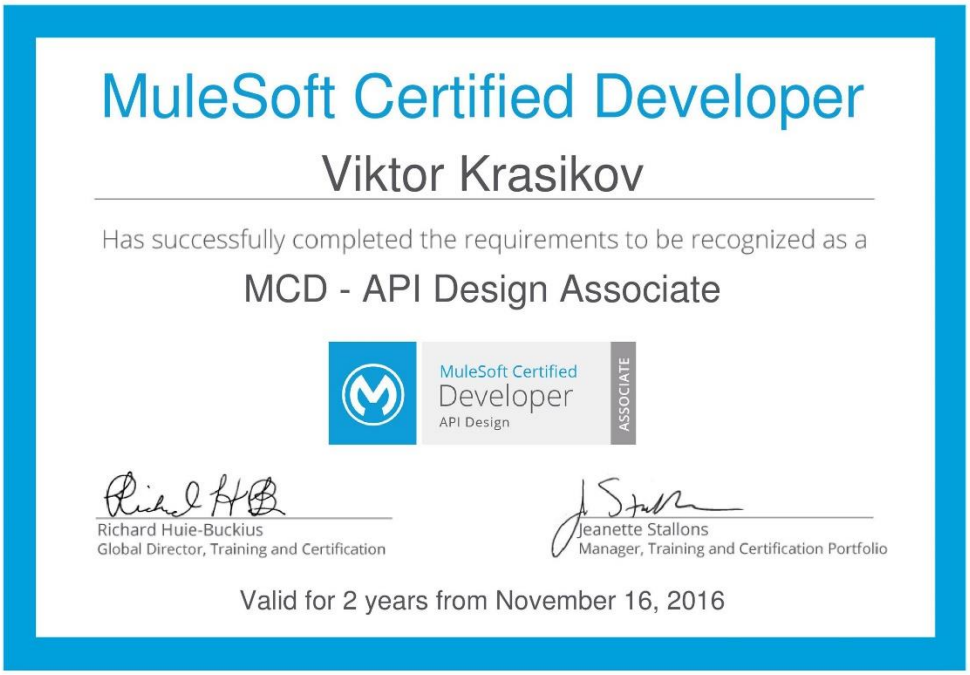
MCD - Integration Professional (Mule 3.8)

MuleSoft Certified Developer Integration PRO

Richard Huie-Buckius
Global Director, Training and Certification

Jeanette Stallons
Manager, Training and Certification Portfolio

Valid for 2 years from January 30, 2017



MuleSoft Certified Developer

Viktor Krasikov

Has successfully completed the requirements to be recognized as a

MCD - API Design Associate

MuleSoft Certified Developer API Design ASSOCIATE

Richard Huie-Buckius
Global Director, Training and Certification

Jeanette Stallons
Manager, Training and Certification Portfolio

Valid for 2 years from November 16, 2016

MuleSoft Certified Developer

Viktor Krasikov

Has successfully completed the requirements to be recognized as a

MCD - Connector Specialist

Richard Huie-Buckius
Global Director, Training and Certification

Jeanette Stallons
Manager, Training and Certification Portfolio

Valid for 2 years from November 11, 2016

MuleSoft Certified Developer

Viktor Krasikov

Has successfully completed the requirements to be recognized as a

MCD - Integration and API Associate

Richard Huie-Buckius
Global Director, Training and Certification

Jeanette Stallons
Manager, Training and Certification Portfolio

Valid for 2 years from October 31, 2016