**Министерство образования и науки Российской Федерации**

**Федеральное государственное автономное образовательное учреждение   
высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ВЫСШАЯ ШКОЛА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И   
ИНФОРМАЦИОННЫХ СИСТЕМ

Направление: 09.04.04 Программная инженерия

Профиль: Разработка программно-информационных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**АВТОМАТИЗИРОВАННОЕ КОНФИГУРИРОВАНИЕ ПОВЕДЕНИЯ ВЕБ-ПРИЛОЖЕНИЙ .NET CORE НА ОСНОВЕ ДИАГРАММ СОСТОЯНИЙ UML**

**Работа завершена:**

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Студент группы 11-711 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.А. Габидуллин

**Работа допущена к защите:**

Научный руководитель

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Марченко

Директор Высшей школы ИТИС

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Ф. Хасьянов

Казань – 2019 г.

Содержание

[Глоссарий 3](#_Toc1903728)

[Введение 4](#_Toc1903729)

[1 Обзор предметной области 7](#_Toc1903730)

[1.1 Обзор литературы и существующих решений 7](#_Toc1903731)

[1.2 Обзор технологий и инструментов 10](#_Toc1903732)

[2 Проектирование и разработка фреймворка 13](#_Toc1903733)

[2.1 Проектирование фреймворка 13](#_Toc1903734)

[2.2 Файл конфигурации на основе UML диаграммы 15](#_Toc1903735)

[2.3 Разработка фреймворка 18](#_Toc1903736)

[3 Разработка веб-приложения 19](#_Toc1903737)

[Заключение 20](#_Toc1903738)

[Список использованных источников 21](#_Toc1903739)

[Приложение 24](#_Toc1903740)

# Глоссарий

UML – (англ. Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

XML – (англ. eXtensible Markup Language – расширяемый язык разметки) – язык разметки, который определяет набор правил для кодирования документов в формате, удобном для чтения как человеком, так и компьютером.

XMI – XML Metadata Interchange, стандарт OMG для обмена метаданными с помощью языка XML. Наиболее часто XMI применяется как формат обмена UML-моделями.

SCXML – State Chart XML, язык разметки на основе XML, который позволяет описать конечный автомат и UML диаграмму состояний.

Фреймворк – заготовки, шаблоны для программной платформы, определяющие структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных модулей программного проекта.

CASE средство – это инструмент, который позволяет автоматизировать процесс разработки информационной системы и программного обеспечения.

Паттерны разработки – повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Событийно-ориентированная архитектура – шаблон архитектуры программного обеспечения, позволяющий создание, определение, потребление и реакцию на события.

# Введение

Современное программное обеспечение в виде веб-сайтов или приложений для операционных систем становятся всё сложнее. Также это относится и к бизнесу – корпоративные системы также неконтролируемо расширяются. Становится проблематично поддерживать и развивать такую систему. А чем больше функционала, тем в большее количество состояний может перейти программное обеспечение.

На начальной стадии разработки программного обеспечения, считается хорошим тоном использовать такой инструмент как UML [1]. Существуют коммерческие разработки, а также научные статьи по преобразованию UML диаграмм в исходный код программного обеспечения. В чем преимущество этого подхода? В том, что, если программный код будет генерироваться машиной, разработчики будут практически не нужны. В идеальном случае необходимы лишь аналитики, которые описывают программное обеспечение в виде, например, UML диаграмм, а также системные администраторы (или DevOps инженеры), которые разворачивают и поддерживают систему. Но, очевидно, на данный момент такой подход не осуществим. Проблема преобразования UML диаграмм до сих пор не решена, что подробнее будет рассмотрено в главе 1.

В данной работе предлагается несколько иной подход: использовать поведенческую UML диаграмму (диаграмма состояний) для конфигурирования поведения создаваемого приложения. Соответственно для такого подхода необходим инструмент, который даст возможность приложению использовать созданную конфигурацию.

В мире программного обеспечения также существует такой тренд, что программное обеспечение мигрирует в интернет. Многие приложения для операционных систем становятся веб-приложениями, или же изначально созданы для веб использования. Поэтому данная работа посвящена именно разработке веб-приложений.

Для того, чтобы использовать подход, описанный выше, предлагается разработать фреймворк, на основе которого будет разрабатываться необходимое веб-приложения, которое в свою очередь будет неким образом настраивать своё поведение при помощи UML диаграммы состояний.

Актуальность данной разработки в том, что решается проблема сложности процесса разработки и поддержки программного обеспечения. Во-первых, часть разработки будет автоматизирована, а во-вторых, это позволяет повысить удобство поддержки и разработки системы, так как вся информация о поведении системы будет находиться в одном файле.

Объектом исследования является разработка фреймворка, позволяющего разрабатывать веб-приложения, которые используют UML диаграммы состояний в качестве конфигурации поведения.

Предметом исследования является фреймворк, позволяющий разрабатывать веб-приложения, которые используют UML диаграммы состояний в качестве конфигурации поведения.

Таким образом, целью данной работы будет разработка фреймворка, позволяющий разрабатывать веб-приложения, которые используют UML диаграммы состояний в качестве конфигурации поведения.

Для достижения поставленной цели, необходимо выполнить следующие задачи:

1. проанализировать предметную область – какие разработки и исследования существуют в данной сфере;
2. рассмотреть способы преобразования и представления UML диаграммы состояний в машинном формате;
3. спроектировать и разработать фреймворк, который включает в себя инструменты по преобразованию UML диаграммы состояний, а также веб-фреймворк, при помощи которого разрабатывается непосредственно веб-приложение;
4. разработать веб-приложение для апробации рассматриваемого в работе подхода.

# Обзор предметной области

## Обзор литературы и существующих решений

Для работы с UML диаграммами существует большое количество программного обеспечения. Данное программное обеспечение можно объединить термином CASE средства (в текущем контексте, так как CASE средство — это более широкий термин). Некоторые CASE средства помимо того, что позволяют конструировать UML диаграммы, также позволяют частично генерировать из UML диаграмм исходный код программы. CASE средства рассматриваются в работах [2,3,4]. В них описаны основные возможности таких CASE средств, как IBM Rational Rose, ARIS Designer, Microsoft Visio, Enterprise Architect, StarUML и другие.

В работах [2,3] рассматриваются возможности кодогенерации данных средств. Некоторые CASE средства поддерживают генерацию кода, но лишь структурных, например, на основе UML диаграмм классов. В свою очередь из поведенческих диаграмм можно лишь получить каркас исходного кода, а не реализацию бизнес-логики. Также имеет интерес возможность экспорта UML диаграмм в XML файл, большинство CASE средств имеют такой функционал. В работе [4] на практике рассматриваются возможности кодогенерации IBM Rational Rose. Авторы делают вывод, что «программный код оказался сгенерирован не полностью, были созданы только классы, операции и связи. Большинство пришлось дописывать вручную».

Наибольший интерес для текущей работы представляет из себя публикация [5]. В ней описаны 28 работ, посвященных кодогенерации на основе UML диаграмм состояний. Авторы заметили, что интересоваться этой темой начали примерно в 1995 году, но количество статей стало уменьшаться в 2004 году. Несмотря на это, статьи по данной тематике выходят до сих пор, что говорит о том, что проблема до сих пор актуальна и не решена.

В рассматриваемых в [5] работах можно увидеть большое количество подходов. В большинстве статей рассматривают паттерны генерации кода на основе UML, то есть не автоматизированная кодогенерация.

В работе [6] рассматривается проблема оптимизации кода, который генерируется на основе UML. Авторы отмечают, что CASE средства с генерацией кода, создают трудно поддерживаемый, сложный для чтения исходный код программы, имеются проблемы с отладкой программного обеспечения. Помимо этого, полученный исходный код получается плохо оптимизированным.

Интересный подход представлен в работе [7], в ней предполагается трансформирование UML диаграммы состояний в микросервисную событийно-ориентированную архитектуру приложения. Работа предполагает новый подход к использованию UML диаграмм, но не предоставляет каких-либо средств автоматизации для этого подхода. Достоинства работы в том, что в данном случае UML используется практически по прямому назначению, ею описывается поведение системы и уже с помощью неё создается архитектура приложения. Получается, что архитектура приложения основывается сразу на требованиях. Из недостатков, можно отметить, что необходима тщательная валидация UML диаграммы на предмет ошибок.

Работы по генерации кода на основе UML продолжаются. В 2017-2018 годах выпущены несколько статей на эту тему [8,9]. В работе [8] разработан плагин для CASE средства Papyrus, который позволяет генерировать C++ код для UNIX систем. Данное решение позволяет реализовывать бизнес-логику приложений, авторы отмечают, что сгенерированные приложения имеют проблемы с потреблением памяти. Также из недостатков можно отметить генерацию лишь C++ кода, тогда как некоторые другие решения поддерживают большее количество языков программирования.

В работе [9] представлен подход по ограничению возможностей UML диаграмм для того, чтобы достигнуть более оптимизированный исходный код. Соответственно для того, чтобы использовать данный подход, необходимо преобразовывать UML диаграммы в определенный формат.

Существует коммерческое программное решения по генерации кода [10], оно позволяет генерировать исходный код программы на основе UML диаграмм состояний. Недостатками является поддержка лишь одного языка программирования C++, и лишь десктоп приложений. Также некоторый исходный код программы будет дописываться разработчиками. Из достоинств можно выделить то, что для генерации кода используется XMI файл, что позволяет использовать большое количество CASE средств.

Помимо решения [10] существует несколько коммерческих программных решений, которые позволяют генерировать приложение на основе UML диаграмм состояний [11,12]. Решение [11] поддерживает лишь устаревшие технологии (.NET 2.0), поэтому не является актуальным. В свою очередь решение [12] позволяет генерировать рабочий веб-сервер при помощи своего веб UML редактора, но ее имеет возможности импортировать диаграмму из другого CASE средства. Отрицательным фактором также может стать высокая цена лицензии (от 5000 долларов).

Решение [13], которое подробно описано в книге [14], неким образом вдохновило авторов на работу в этом направлении. Фреймворк Spring Web Flow позволяет создавать веб-приложение и конфигурировать его поведение при помощи языка разметки XML. Данный фреймворк разработан для языка программирования Java. Недостатком этого фреймворка является то, что для описания используется собственный формат XML, в нашей работе не используется такое решение и предлагается использовать универсальный язык разметки XML – CSXML, который используется для описания UML диаграмм состояний. Программное решение [15] было разработано из-за того, что фреймворк Spring Web Flow имел некоторые ограничения в работе, для языка программирования Java.

Большое количество работ в направлении преобразования UML диаграмм в исходный код программы показывает, что такой подход имеет место быть в коммерческой разработке, а значит исследования в этой сфере востребованы.

## Обзор технологий и инструментов

В название работы вынесена такая технология как .NET Core, а также язык моделирования UML. Но почему именно они? В данной главе рассмотрим почему был сделан такой выбор.

В сфере разработки программного обеспечения следует выделять следующие аспекты, в которых нужно выбирать те или иные инструменты:

1. программная платформа (фреймворк) и язык программирования;
2. язык моделирования программного обеспечения;
3. CASE среда разработки (IDE);
4. CASE среда проектирования.

Для данной разработки была выбрана программная платформа .NET Core. Был сделан такой выбор, так как одной из главный целей .NET Core – это мультиплатформенная веб-разработка. Её достоинствами является то, что фреймворк имеет открытый исходный код, благодаря чему разработка имеет большую поддержку .NET сообщества, а также то, что это разработка Microsoft, а значит, данный фреймворк поддерживается компанией. Это позволяет .NET Core быстро развиваться, пополняться новым функционалом, а также исправлять имеющиеся ошибки. Также выбор был продиктован тем, что авторы данной работы имеют большой опыт в этой программной платформе. Версия фреймворка для веб разработки имеет название – ASP .NET Core.

Фреймворка ограничивает выбор языков программирования. В данном случае выбор ограничивается языками C#, Visual Basic, F#. Языком для разработки решения был выбран C#. Так как он является самым популярным языком программирования среди семейства языков .NET Core. По индексу TIOBE (индекс, оценивающий популярность языков программирования) язык программирования C# на ноябрь 2018 года занимает 6 место по популярности [16].

В качестве языка моделирования был выбран язык моделирования UML. Как утверждают авторы статьи [17]: «UML фактически стал «стандартом де-факто» в области разработки программного обеспечения и применяется для решения задач других предметных областей, например, задач бизнес-моделирования». Иные язык моделирования, например, IDEF, или BPMN существенно проигрывают в распространённости языку UML.

В качестве IDE очевидным образом предполагается использовать Visual Studio 2017 Community, так как эта среда создана специально для платформы .NET [18]. Данная среда позволяет быстро и качественно писать код, так как в ней существует большое количество подсказок, которые могут помочь разработчику быстрее и качественнее создавать программный код. В ней удобно следить за исполнением кода. Visual Studio предоставляет множество данных о производительности создаваемой программы, что очень важно в разработке. Удобным средством для работы с программными пакетами является NuGet. Это встроенное в Visual Studio средство для поиска и установки необходимых для программного обеспечения пакетов. Разрабатываемый фреймворк в дальнейшем также можно будет распространять в виде NuGet пакета.

В качестве CASE средства моделирования был выбран Visual Paradigm Community [19]. Данная программа является бесплатной для некоммерческого использования. Программа поддерживает все виды UML диаграмм.

Данное приложение было выбрано из-за того, что поддерживает возможности экспортирования UML диаграмм в XML файл форматов SCXML [20] и XMI [21]. Это важно для данной работы, о чем будет рассказано в дальнейших главах.

В итоге средства для моделирования, проектирования и разработки были выбраны:

1. фреймворк – ASP .NET Core, язык программирования – C#;
2. язык моделирования – UML;
3. среда разработки – Visual Studio 2017;
4. среда проектирования – Visual Paradigm.

# Проектирование и разработка фреймворка

## Проектирование фреймворка

В основу архитектуры фреймворка положен паттерн проектирования Service Layer, описанный Мартином Фаулером в книге «Архитектура корпоративных программных приложений». [22] Архитектура заключается в том, что проект разбивается на несколько независимых служебных слоёв (service layer). Таким образом получается разграниченная по функционалу, бизнес-логике, а также гибкая к модификации структура. Это позволит в дальнейшем расширять фреймворк. Также благодаря тому, что структура имеет общеизвестный вид, разработчики смогут легко разобраться в исходном коде проекта и изменять фреймворк под свои цели.

В проекте существует несколько слоёв:

* Config Engine Layer – движок – слой, который отвечает за работу с файлами конфигурации;
* Flow Manager Layer – слой который управляет Config Engine Layer, и выполняет операции по изменению состояния системы;
* Model Layer – это модель данных.

Опишем Model Layer. Модель данных состоит из одного класса – FlowData. Данный класс содержит поля и свойства, которые могут точно идентифицировать текущее состояние, а также определить в какое следующее состояния перейдет система на следующем шагу. Благодаря социализации, данные из файла конфигурации преобразуются в данные класса FlowData.

Рассмотрим Config Engine Layer. Config Engine Layer состоит из интерфейса и 2 классов. Данный слой будет реализовывать логику работы с файлами конфигурации. Engine Layer является абстракцией над работой с разными типами файлов. Это позволяет поддерживать языки конфигурации на основе JSON и XML. Основные задачи данного слоя – это чтение файлов конфигурации и возвращение данных из файла в виде класса FlowData.

Flow Manager Layer в свою очередь реализует логику переходов между состояниями. Оперируя Config Engine Layer, а также данными пользователя, слой получает текущее состояние и по конфигурации системы определяет в какое именно состояние перейдет система. С данным слоем взаимодействует контроллер из паттерна MVC.

UML диаграмму компонентов, изображающую архитектуру фреймворка можно увидеть на рисунке 1.

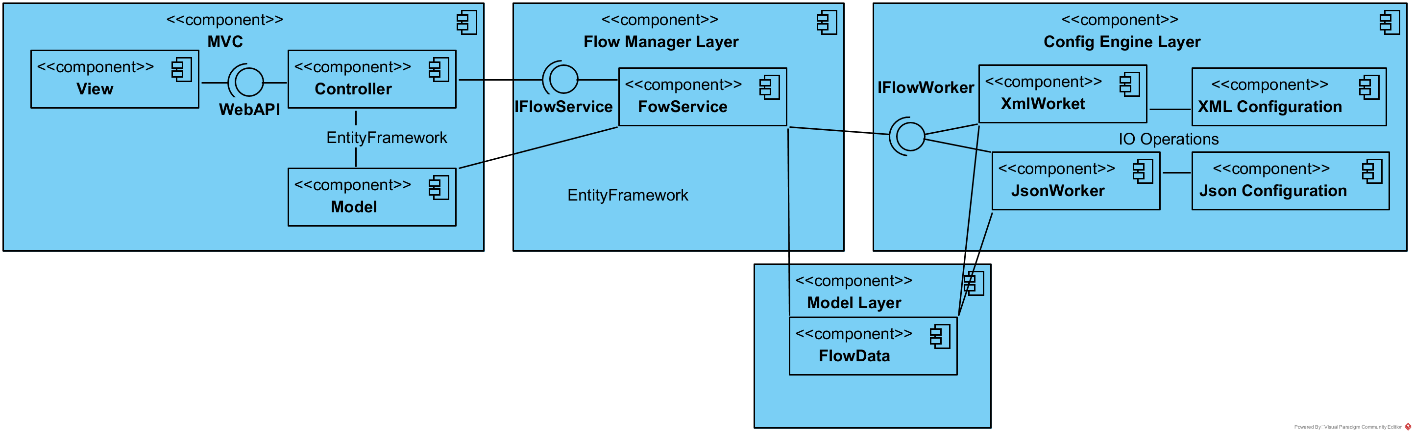


Рис. 1. Структура слоёв фреймворка.

На рисунке 1 можно увидеть каким именно образом взаимодействуют разные слои. Через какие интерфейсы или посредством какого инструмента. Например, связь между компонентами View и Controller в MVC слое осуществляется посредством WebAPI. Операции с базой данных осуществляются инструментом Entity Framework. И работа с конфигурацией, происходит посредством файловых операций.

Отдельной тип связи – это связь через интерфейс. Разные слои взаимодействуют друг с другом через интерфейсы классов (IFlowService и IFlowWorker). Это позволяет подменить или модифицировать непосредственно сами классы, которые выполняют работу. Благодаря этому, например, реализована возможность работы как с XML, так и с JSON конфигурациями.

## Файл конфигурации на основе UML диаграммы

Диаграмма состояний для конечного автомата может использоваться для описания поведения систем. Широко распространено использование диаграмм состояний UML, которые основываются на диаграммах состояний, описанные Дэвидом Харелом [23].

Диаграмма состояний UML представляет из себя ориентированный граф, который состоит из следующих базовых элементов:

1. Круг – это начальное состояние, в свою очередь конечное состояние представляет окружность с маленьким кругом внутри;
2. Набор состояний, в нотации UML изображаются в виде скругленного прямоугольника;
3. Стрелками обозначают переходы между состояниями. Над стрелкой пишут событие, при котором происходит переход в другое состояние.

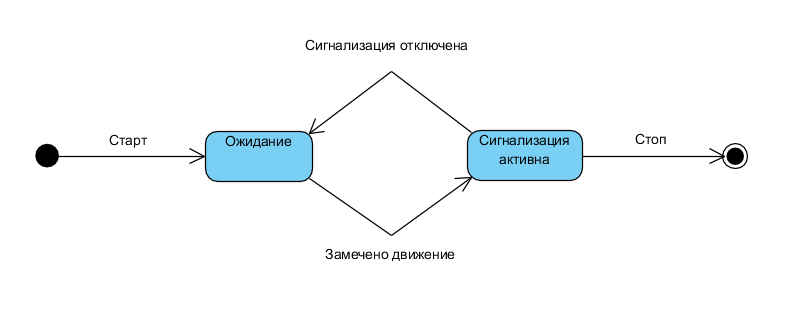


Рис. 2. Пример диаграммы состояний, созданной в программе Visual Paradigm.

Рассмотрим пример диаграммы состояний на рисунке 2. После старта система будет находиться в состоянии «Ожидание». Когда система заметит движение, произойдет переход в состояние «Сигнализация активна». В свою очередь это состояние может вернуться в «Ожидание». Или же система может быть выключена, при этом произойдет переход в конечное состояние.

Выше описано то, каким образом UML диаграмма состояний визуально отображается. Для хранения же в машинном виде используются разные форматы XML:

1. XMI – XML Metadata Interchange [21], стандарт OMG для обмена метаданными с помощью языка XML. Наиболее часто XMI применяется как формат обмена UML моделями.
2. SCXML – State Chart XML [20]. Это нотация, основанная на XML, которая позволяет описывать UML диаграммы состояний.

Большинство CASE средств поддерживают файлы форматов XMI и SCXML [3]. Для данной работы был выбран SCXML, так как он имеет более дружелюбный формат. Например, диаграмма из рисунка 2, изображена в формате SCXML на рисунке 3. Можно заметить, что диаграмма в формате SCXML всё также легко читается, и её вполне могут написать разработчики. В свою очередь XMI хранит большое количество технической информации, что делает её громоздкой.

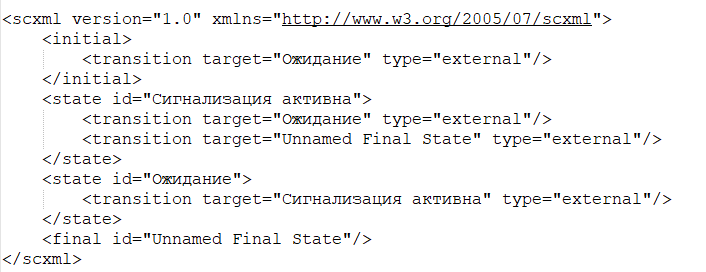


Рис. 3. Диаграмма состояний с рисунка 3, в формате SCXML файла.

Для работы с данным файлом разработан класс ScxmlWorker, который считывает и обновляет необходимый файл конфигурации. Данный класс будет описан подробно в следующей главе.

Рассмотрим пример с файлом конфигурации, который больше подходит к веб-приложениям. На рисунке 4 изображена диаграмма состояний в виде SCXML. Она описывает переходы между страницами на сайте.

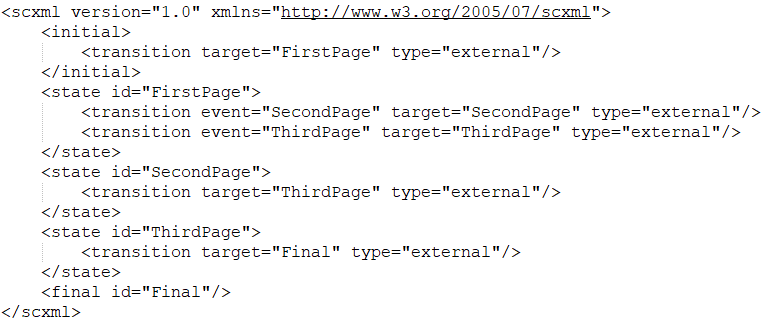


Рис. 4. Диаграмма состояний, описывающая переходы между страницами.

Для начала разберем синтаксис SCXML языка. Так как это XML, то многие вещи описываются тегом. Для начального и конечного состояния существуют отдельные теги – initial и final. Это сделано для того, чтобы чётко отделить обычные состояния от начального и конечного. Обычные состояния в свою очередь определяются тегом state. У тегов состояний должны быть атрибуты id. Это идентификатор состояния. У каждого тега состояния (кроме конечного), существуют дочерние теги – transition. Эти теги описывают то, какие переходы можно осуществить из текущего состояния. Теги transition имеют два основных атрибута – target и event. Target отвечает за то, в какое состояние переходит система. Этому атрибуту присваивается идентификатор нужного состояния. Атрибут event отвечает за то, при каком событии происходит переход в состояние, описанное в атрибуте target текущего тега transition. Теперь рассмотрим систему на рисунке 4.

Благодаря диаграмме состояний можно гибко настроить поведение системы. С состояния Initial можно перейти лишь в состояние FirstPage. По аналогии с веб-сайтом, пользователь, например, после входа в систему попадает на страницу FirstPage. В свою очередь со страницы FirstPage существуют переходы на страницы SecondPage и ThirdPage. И при разных событиях, например нажатие двух разных кнопок на странице, произойдет переход в другое состояние и соответственно страницу. И соответственно на странице ThirdPage существует лишь один путь – на страницу Final. Это финальное состояние, с которого уже невозможно выйти. При этом пользователь переходит опять в состояние Initial. Описанный на рисунке 4 процесс может быть чем угодно. Это последовательные действия. Например, прохождение теста, оформление билета, оформление заказа в интернет-магазине, и многое другое, что может требовать от пользователя какой-либо линейной последовательности действий. Эти все сценарии, это как раз то для чего может использоваться разработанный фреймворк.

## Разработка фреймворка

# Разработка веб-приложения

# Заключение

# Список использованных источников

1. Unified Modeling Language [электронный ресурс] Режим доступа http://www.uml.org/ свободный. Дата обращения 25.11.2018
2. Варганов Д. Обзор средств генерации программного кода на основе UML // Альманах научных работ молодых ученых университета ИТМО. Материалы XLVI научной и учебно-методической конференции, Санкт-Петербург, Россия, 31 января-03 февраля 2017, с. 38-41.
3. ИНТУИТ. Лекция 8: Обзор CASE-средств для построения диаграмм UML [электронный ресурс] Режим доступа https://www.intuit.ru/studies/courses/941/229/lecture/5963/ свободный. Дата обращения 18.11.2018
4. Мухин М.С., Хоружева Ю.И. Проектирование приложения на основе прямого преобразования UML моделей в Java, на примере Rational Software Architect // Информационные Технологии. Радиоэлектроника. Телекоммуникации, 2017, №7, с. 379-394.
5. Domínguez, E., Pérez, B., Rubio, Á.L., Zapata, M.A. A systematic review of code generation proposals from state machine specifications // Information and Software Technology, October 2012, Volume 54, Issue 10, pp. 1045-1066.
6. Charfi, A., Mraidha, C., Gérard, S., Terrier, F., Boulet, P. Does code generation promote or prevent optimizations? // 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing – ISORC 2010, Carmona, Sevilla, Spain, 5-6 May 2010, pp. 75-79.
7. Petrasch, R. Transformation of state machines for a microservice-based event-driven architecture: A proof-of-concept // Advances in Intelligent Systems and Computing, 2019, Volume 769, pp. 327-336.
8. Pham, V.C., Radermacher A., Gerard S., Li S. Complete code generation from UML state machine // 5th International Conference on Model-Driven Engineering and Software Development, Porto, Portugal, 19-21 Feb. 2017, pp. 208-219.
9. Sunitha, E.V., Samuel, P. Object constraint language for code generation from activity models // Information and Software Technology, Volume 103, November 2018, pp. 92-111.
10. SinelaboreRT [электронный ресурс]. Режим доступа www.sinelabore.com/ свободный. Дата обращения 18.11.2018
11. The North State Framework [электронный ресурс]. Режим доступа https://www.northstatesoftware.com/ свободный. Дата обращения 18.11.2018
12. Workflow Engine [электронный ресурс]. Режим доступа https://workflowengine.io/ свободный. Дата обращения 18.11.2018
13. Spring Web Flow [электронный ресурс]. Режим доступа http://projects.spring.io/spring-webflow/ свободный. Дата обращения 18.11.2018
14. Vervaet E. The Definitive Guide to Spring Web Flow. Berkeley: Apress, 2008, pp. 380.
15. Навигация: вариант реализации для корпоративного приложения [электронный ресурс]. Режим доступа https://habr.com/post/175507/ свободный. Дата обращения 18.11.2018
16. TIOBE Index [электронный ресурс] Режим доступа https://www.tiobe.com/tiobe-index/ свободный. Дата обращения 18.11.2018
17. Буй Д., Шишацкая Е. UML: история, спецификация, библиография // XIII-th International Conference Knowledge-Dialogue-Solution, Varna, Bulgaria, 18-24 June 2007, pp. 309-316.
18. Visual Studio [электронный ресурс] Режим доступа https://visualstudio.microsoft.com/ свободный. Дата обращения 25.11.2018
19. Visual Paradigm [электронный ресурс] Режим доступа https://www.visual-paradigm.com/ свободный. Дата обращения 25.11.2018
20. State Chart XML (SCXML): State Machine Notation for Control Abstraction [электронный ресурс] Режим доступа https://www.w3.org/TR/scxml свободный. Дата обращения 20.11.2018
21. About the XML Metadata Interchange Specification Version 2.5.1 [электронный ресурс] Режим доступа https://www.omg.org/spec/XMI свободный. Дата обращения 25.11.2018
22. Мартин Фаулер, Архитектура корпоративных программных приложений / Мартин Фаулер – Изд-во Вильямс, 2006. — 544 c.
23. Harel, D. Statecharts: A visual formalism for complex systems // Science of Computer Programming, Volume 8, 1987, pp. 231-274.
24. Эндрю Троелсен, Язык программирования C# 6.0 и платформа .NET 4.6 7-е издание: книга для разработчиков С# / Эндрю Троелсен – Изд-во Вильямс, 2016. – 1440 с.

# Приложение

Справка о принятии статьи в печать:

ООО «ГАЗПРОМ ТРАНСГАЗ КАЗАНЬ»

ООО «КОНВЕРТ»

Оргкомитет конференции «Фундаментальные и прикладные разработки в области технических и физико-математических наук»

СПРАВКА

Дана авторам: Габидуллин Ибрагим Анварович, Марченко Антон Александрович о регистрации в качестве участника в научной конференции «**Фундаментальные и прикладные разработки в области технических и физико-математических наук**»: круглый стол № 7 – вопросы теории и практики, статья: «Преобразование UML диаграммы состояний в конфигурацию поведения веб-приложения» принята в печать и будет опубликована в сборнике конференции с последующей загрузкой в систему РИНЦ, в декабре 2018 года.

Справка дана для предъявления по месту требования.

«29» ноября 2018 г.

Генеральный директор

OOO «Конверт» Соловьева О.В.

