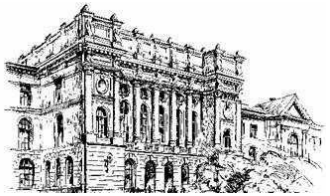
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий



ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Разработка web-frontend для NuSMV**

Студент гр. 43501/3 В.С. Кан

Санкт-Петербург – 2017

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Работа допущена к защите
зав. кафедрой

_____ В.М. Ицыксон

«____» _____ 2017 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Разработка web-frontend для NuSMV**

Направление: 09.03.01 – Информатика и вычислительная техника

Выполнил студент гр. 43501/3 _____ В.С. Кан

Научный руководитель,
к. т. н., доц. _____ В.М. Ицыксон

Санкт-Петербург – 2017

РЕФЕРАТ

Отчет, 63 стр., 12 рис., 10 табл., 18 ист., 1 прил.

ФОРМАЛЬНАЯ ВЕРИФИКАЦИЯ, ПРОВЕРКА МОДЕЛЕЙ, КАЧЕСТВО СИСТЕМ, СТРУКТУРА КРИПКЕ, ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ, ВЕБ-ПРИЛОЖЕНИЕ

В рамках выпускной квалификационной работы разработано веб-приложение, реализующее графический пользовательский интерфейс для верификатора NuSMV. Приложение построено в соответствии с клиент-серверной архитектурой. Рассмотрены существующие решение в области графических интерфейсов для верификаторов NuSMV, Spin, проанализированы их достоинства и недостатки. Сформулирована постановка задачи, требования к разрабатываемому программному обеспечению. Проанализированы технологии, с помощью которых возможна реализация приложения в соответствии с требованиями, обоснован их выбор. Выполнено функциональное и модульное тестирование компонентов приложения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В ходе разработки программных и аппаратных систем предъявляются высокие требования к надежности. Данные системы характеризуются высокой сложностью, что делает практически невозможным реализацию контроля качества методом тестирования [1]. В качестве альтернативы, в случае возможности формализации требований с помощью формул математической логики, представляется возможным верификация модели системы (Model checking).

Model checking – это метод формальной верификации, основанный на представлении системы в виде математической модели (обычно, структуры Крипке) и формализации требований (обычно, в виде формулы темпоральной логики).

Среди инструментов формальной верификации стоит выделить NuSMV. Он основан на проекте Cadence SMV, является свободно-распространяемым программным обеспечением и предоставляет широкие возможности по симуляции и верификации моделей [2]. К недостаткам данной системы относится недостаток поддержки графических интерфейсов пользователя вследствие ориентации на работу с командной строкой. Данный подход сильно ограничивает число пользователей и порог вхождения.

В рамках данной работы рассмотрены следующие вопросы:

- Обзор существующих графических интерфейсов пользователя;
- Формулировка требований к разрабатываемому приложению;
- Проектирование архитектуры приложения;
- Анализ используемых технологий;
- Программная реализация приложения;
- Определение методов тестирования.

1. Анализ существующих решений

Наиболее известными инструментами, реализующими проверку моделей общего назначения, являются NuSMV и SPIN [2]. Рассмотрим существующие графические интерфейсы пользователя для данных систем и оценим их по приведенным ниже критериям:

1. *Платформы* – официально поддерживаемые программные платформы;

2. *Графическое построение модели* – наличие графического редактора моделей;

3. *Визуализация контрпримера* – наличие визуализации вывода NuSMV;

4. *Реализация* – язык и технологии, используемые при реализации программы.

1.1. Графические интерфейсы пользователя для Spin

В сегменте графических пользовательских интерфейсов для Spin представлены две программы: iSpin и jSpin. Рассмотрим их подробнее.

1.1.1. iSpin

iSpin – официально поддерживаемый сообществом Spin пакет, предоставляющий графический пользовательский интерфейс [3]. Поддерживает все основные режимы работы. Имеет окно «Automata view», отображающее сгенерированный на основе описания модели конечный автомат.

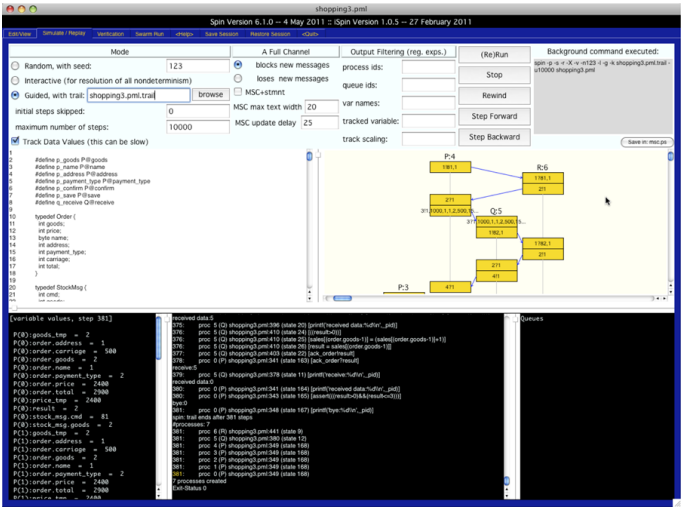


Рисунок 1.1. Окно iSpin

Таблица 1.1. iSpin

Платформы	Linux, Windows, Mac OS X
Графическое построение модели	Отсутствует
Визуализация контрпримера	Отсутствует
Реализация	Написан на языке Tcl с помощью библиотеки Tk.

1.1.2. jSpin

Программа iSpin разработана на языке Java [3]. Программа состоит из ездинственного окна, состоящего из меню и трех текстовых полей:

1. поле редактирования исходного кода Promela,
2. поле отображения вывода Spin,

3. поле отображения предупреждений компилятора.

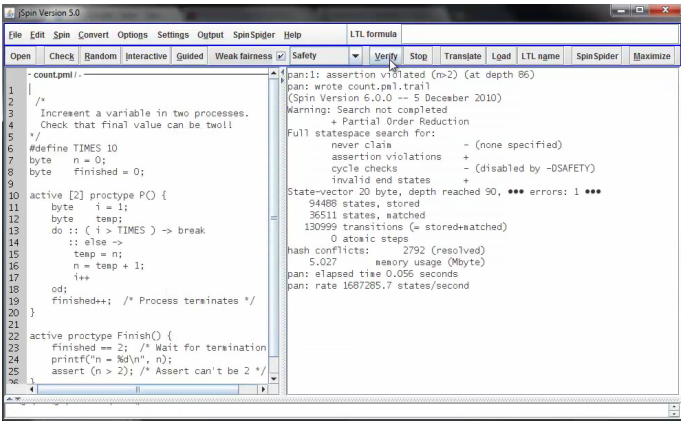


Рисунок 1.2. Окно jSpin

Таблица 1.2. jSpin

Платформы	Linux, Windows, Mac OS X
Графическое построение модели	Отсутствует
Визуализация контрпримера	Отсутствует
Реализация	Реализован с помощью языка Java.

1.2. Графические интерфейсы пользователя для NuSMV

В сегменте графических пользовательских интерфейсов для NuSMV представлены три программы: NuSMV GUI, gNuSMV и NuSeen.

1.2.1. NuSMV GUI

Программа разрабатывается непосредственно сообществом NuSMV, основывается на интерактивном режиме выполнения и предоставляет стандартные возможности редактирования исходного текста описаний моделей *.smv, редактирования спецификаций с помощью инспектора формул темпоральных логик CTL/LTL, а также предоставления вывода NuSMV в текстовом формате [4]. Использует устаревшую версию NuSMV 1.

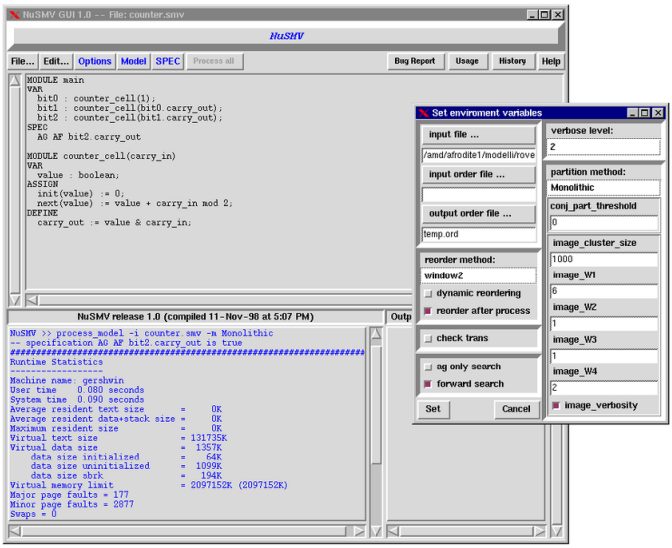


Рисунок 1.3. Окно редактирования исходного кода NuSMV GUI.

1.2.2. gNuSMV

Данный проект находится в разработке, но для пользователей доступна snapshot версия программы [5]. Главным отличием от NuSMV GUI является использование последней мажорной версии NuSMV 2.1.

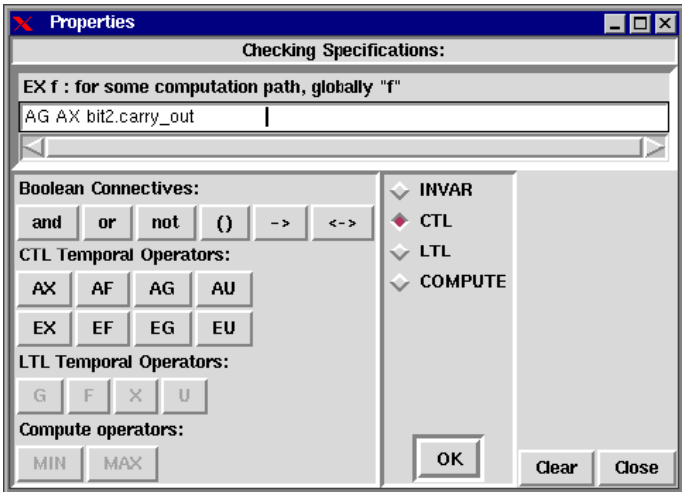


Рисунок 1.4. Окно редактирования спецификаций NuSMV GUI.

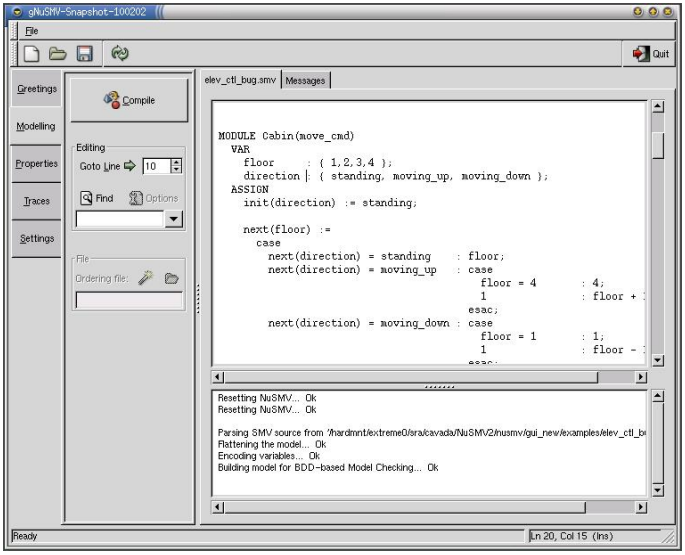


Рисунок 1.5. Окно редактирования исходного кода gNuSMV.

Таблица 1.3. NuSMV GUI

Платформы	Linux, Windows, Mac OS X
Графическое построение модели	Отсутствует
Визуализация контрпримера	Отсутствует
Реализация	Основан на Qt Jambi – библиотеке, представляющей собой Java-обёртку для Qt-компонентов.

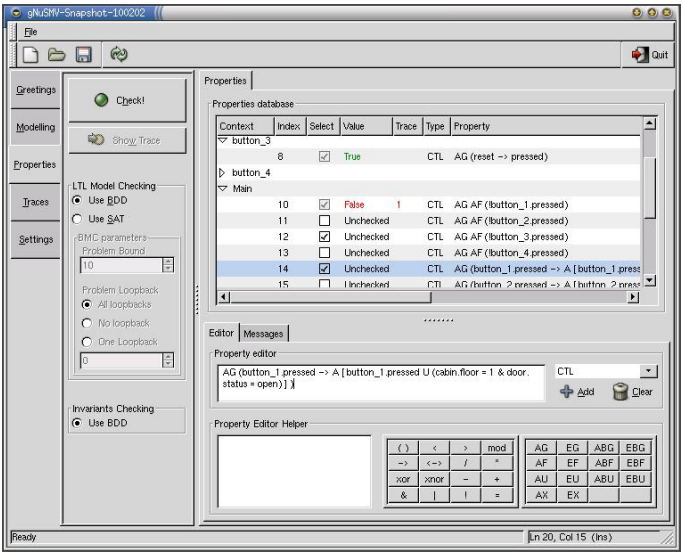


Рисунок 1.6. Окно редактирования спецификаций gNuSMV.

1.2.3. NuSeen

В отличие от вышерассмотренных графических интерфейсов, являющихся самостоятельными приложениями, NuSeen основан на интегрированной среде разработки Eclipse [6]. Отличительной особенностью является построение графов зависимостей переменных и гра-

Таблица 1.4. gNuSMV

Платформы	Linux, Windows
Графическое построение модели	Отсутствует
Визуализация контрпримера	Отсутствует
Реализация	Основывается на pygtk2 – библиотеке для языка python, позволяющей строить графические приложения с помощью библиотеки GTK+2.

фов зависимостей жестко соединенных наборов переменных (SCV – Strongly Connected Variables set). Поддерживается визуализация контрпримера.

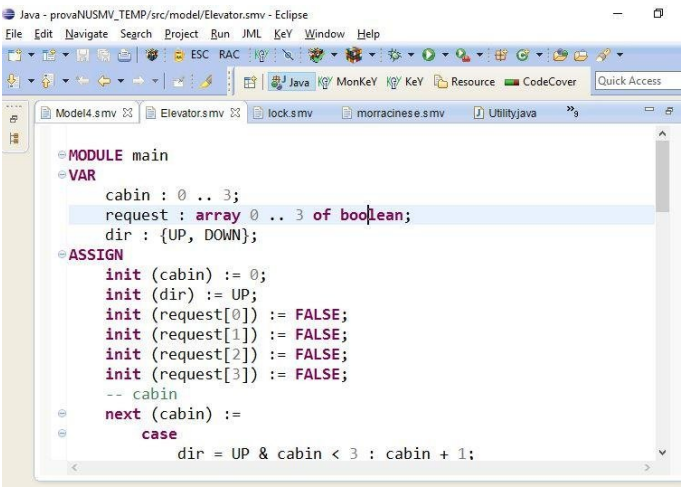


Рисунок 1.7. Окно редактирования исходного кода NuSeen.

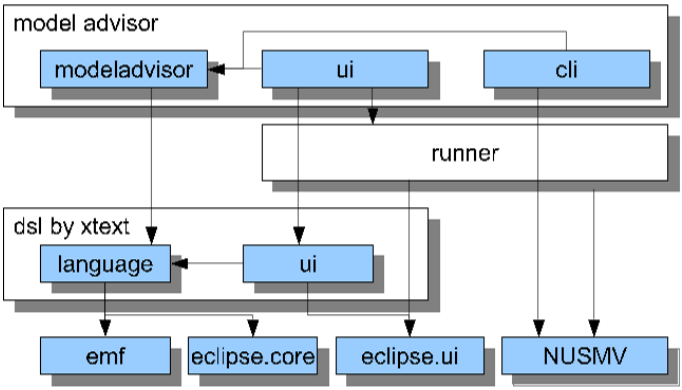


Рисунок 1.8. Пример графа зависимостей переменных NuSeen.

Таблица 1.5. NuSeen

Платформы	Linux, Windows, Mac OS X
Графическое построение модели	Отсутствует
Визуализация контрпримера	Отсутствует
Реализация	Основан на интегрированной среде разработки Eclipse.

1.3. Вывод

Все рассмотренные выше реализации графических пользовательских интерфейсов для верификаторов общего назначения имеют схожие минусы:

- Нет реализации графического редактора верифицируемых моделей;
- Визуализация контрпримера реализована только в NuSeen, но на очень примитивном уровне;

- Для проверки модели необходима установка ПО и зависимых пакетов;

2. Постановка задачи

Основная задача данной выпускной квалификационной работы бакалавра – разработка веб-ориентированного графического интерфейса пользователя для программного обеспечения, реализующего верификацию методом проверки моделей, NuSMV. Программа должна реализовывать основные функциональные возможности NuSMV. Целью реализуемого проекта является использование в качестве инструмента в академических курсах.

Набор используемых технологий не специфицирован, однако они должны отвечать всем нижеперечисленным требованиям.

2.1. Требования

Программа должна представлять собой веб-приложение, построенное в соответствии с архитектурой "клиент-сервер". Взаимодействие частей приложения должно быть реализовано с помощью протокола HTTP. Ниже приведены требования к функциональным частям приложения.

2.2. Клиентская часть приложения

В качестве клиента должен использоваться веб-браузер Google Chrome версии не ниже 50 или Mozilla Firefox версии не ниже 45. Различия в графическом интерфейсе пользователя, отображающемся в различных браузеров должны быть минимизированы.

Клиент должен поддерживать графический ввод верифицируемой модели. Для этого необходимо реализовать поле построение модели, реализующее:

- добавление состояния,

- удаление состояния,
- добавление перехода,
- удаление перехода,
- установка инициализирующего состояния,
- перемещение состояния по полю построения.

Управление переменными состояния должно быть реализовано в отдельном модуле. Для спецификаций также должно быть выделено отдельное окно. Данные модули должны быть реализованы таким образом, чтобы обеспечить максимальный контроль приложением вводимых пользователем данных. Также необходимо организовать окно, динамически отображающее генерируемый код на языке SMV.

2.3. Серверная часть приложения

Сервер должен принимать и обрабатывать HTTP-запросы клиента, хранить и выдавать статические файлы, а также запускать процесс NuSMV. Также серверная часть должна корректно обрабатывать запросы от нескольких пользователей. Целевой операционной системой является Windows 10.

3. Проектирование архитектуры системы

3.1. Архитектура приложения

Реализация веб-приложения подразумевает использование клиент-серверной архитектуры приложения. В качестве протокола прикладного уровня стандартом для веб-сайтов, де-факто, являются протоколы HTTP [7] и HTTPS [8]. Это обусловлено, в первую очередь, тем фактом, что данный протокол поддерживается во всех современных браузерах. Для организации веб-сервиса на основе данного протокола используются два подхода – RESTful и SOAP. Прежде чем рассмотреть конкретные технологии, используемые в разработке подобных приложений, рассмотрим данные архитектурные стили взаимодействия клиента и сервера.

3.1.1. REST

REST (Representational State Transfer) – архитектурный стиль взаимодействия распределенных клиент-серверных приложений [9]. Термин «RESTful» применяется к веб-сервисам, работающим в соответствии с заложенными в основу данного стиля принципами. Важно отметить, что RESTful сервисы ориентированы на работу с данными. Важнейшей особенностью REST-взаимодействий является соответствие принципам CRUD (create, read, update, delete) — использование четырёх базовых методов работы с персистентными хранилищами данных [9]. Как следствие, при построении приложений возможно использование стандартных методов протокола HTTP, что упрощает разработку и уменьшает объем передаваемого сообщения. Пример запроса и ответа в RESTful стиле приведен в листинге 3.1.

Листинг 3.1. Пример запроса и ответа в RESTful стиле.

```
1 /* Запрос */
```

```
GET /index.php HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Linux i686; ru) Firefox/3.0b5
Accept: text/html
Connection: close

/* Ответ */
HTTP/1.0 200 OK
Server: nginx/0.6.31
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close

/* HTML page */
```

3.1.2. SOAP

SOAP - Simple Object Access Protocol, протокол обмена структурированными сообщениями в распределенном приложении. Главным преимуществом данного стиля является строгая типизация данных. Данные передаются в формате XML, пример запроса и ответа приведен в листинге 3.2. SOAP ориентирован на работу с операциями, т.е. с его помощью возможно построение сложного веб-сервиса [10].

Для передачи SOAP-сообщений наиболее часто используется протокол HTTP, но, в отличие от RESTful сервисов, не реализует принципы CRUD, т.к. все сообщения передаются с помощью HTTP-метода POST.

Листинг 3.2. Пример запроса и ответа в SOAP стиле.

```
/* Запрос */
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:ns1="http://example.com/soap"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:enc="http://www.w3.org/2003/05/soap-encoding">
<env:Body>
```

```
<ns1:getSmth env:encodingStyle="http://www.w3.org/2003/05/soap-
    encoding">
</ns1:getSmth>
</env:Body>
</env:Envelope>

/* Ответ */
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:ns1="http://example.com/soap"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://xml.apache.org/xml-soap"
xmlns:enc="http://www.w3.org/2003/05/soap-encoding">
<env:Body xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
<ns1:getSmthResponse env:encodingStyle="http://www.w3.org/2003/05/
    soap-encoding">
<rpc:result>return</rpc:result>
<return xsi:type="ns2:Map">
<item>
<key xsi:type="xsd:string"> Something </key>
<value xsi:type="xsd:float"> 0 </value>
</item>
</ns1:getSmthResponse>
</env:Body>
</env:Envelope>
```

3.1.3. Вывод

Наиболее оптимальным решением, в соответствии с требованиями к приложению, является использование REST стиля. Проектируемое приложение в рамках клиент-серверного взаимодействия сводится к выполнению двух операций клиента и двух операций сервера:

- Со стороны клиента:
  - Запрос статических файлов (HTML, CSS, JavaScript);
  - Отправка исходного кода описания верифицируемой модели;
- Со стороны сервера:

– Выдача статических файлов;

– Отправка результата верификации описанной в запросе модели.

Таким образом, клиентской части приложения необходимы удаленные вызовы двух методов. Очевидно, что SOAP, в данном случае, избыточен вследствие своей направленности на работу с операциями, в то время как построение REST сервиса на основе HTTP-методов GET (для запроса статики) POST (для отправки серверу описания модели) является оптимальным.

3.2. Серверная часть приложения

В рамках инструментов реализации веб-серверов представлен широкий спектр технологий. Все они имеют существенные отличия друг от друга, каждая ориентирована на определенную характеристику предоставляемых сервисов. Ниже рассмотрим основные технологии, используемые в разработке веб-серверов.

3.2.1. Java Sockets

Сокет – программный интерфейс, представляющий собой средство межпроцессного взаимодействия в распределенном программном обеспечении. Java Sockets – реализация сокетов в языке Java. С помощью сокетов становится возможна передача данных между двумя удаленными узлами компьютерной сети. Для передачи данных используются потоки ввода/вывода, привязанные к объекту сокета, с помощью которых возможна передача как текстовых, так и бинарных данных. В качестве транспортного протокола возможно использование как TCP, так и UDP [11]. В ходе написания веб-сервера с помощью Java Sockets необходимо решить следующие задачи:

- Разбор HTTP-заголовков запроса, формирование HTTP-заголовков ответа;
- Параллельное обслуживание клиентов;
- Выдача данных статических файлов.

Статическая типизация в языке Java предоставляет больший уровень контроля исполнения кода разработчиком (по сравнению с динамически типизированными языками). Java является компилируемым в байт-код языком, что делает программы быстрее, в отличие от интерпретируемых языков.

3.2.2. Node.js

Node.js – программная платформа, предназначенная для трансляции JavaScript в машинный код, тем самым превращая его в язык общего назначения. Основан на движке V8, разработанным корпорацией Google. Разработчик node.js заложил в основу платформы принципы событийно-ориентированно и асинхронного программирования [12]. Платформа предоставляет разработчику инструменты разработки сервисов параллельного обслуживания клиентов, не обременяя его реализацией механизмов синхронизации доступа к данным благодаря использованию единственного потока. Данный поток способен поддерживать большое количество параллельных соединений, что возможно вследствие использования неблокирующего ввода/вывода. Однако этот факт вводит ограничение на сложность вычислений при обработке запроса. В ходе написания веб-сервера с помощью node.js необходимо решить следующие задачи:

- Выдача данных из статических файлов;
- Разработка механизма минимизации времени простоя запросов, необходимого вследствие времязатратной обработки.

3.2.3. Django Framework

Django – веб-фреймворк, написанный на языке Python, позволяющий разработчику создавать приложения в соответствии с архитектурным шаблоном MVC (Model-View-Controller). Данный фреймворк разрабатывался с целью минимизации временный затрат на разработку [13]. Среди особенностей необходимо выделить концепцию разделения функциональности сервера на независимые Django-приложения, что предоставляет возможность переиспользования кода. В ходе написания веб-сервера с помощью Django необходимо решить следующие задачи:

- Выдача статических файлов;
- Запуск процесса NuSMV и выдача результатов выполнения.

Для реализации RESTful сервиса в соответствии с описанными выше задачами в Django существует библиотека REST Framework, предоставляющая доступ к декораторам, упрощающим обработку запросов методов API.

3.2.4. Вывод

В результате анализа инструментов разработки веб-серверов, наиболее оптимальным для данной задачи был выбран Django Framework. Его использование позволяет существенно сократить количество кода, что сокращает как время разработки, так и количество потенциальных ошибок. Концепция разделения функциональности на приложения, что в нашем случае позволяет разделить сервер на функциональные блоки: блок выдачи статических файлов и блок запуска процесса верификации модели.

3.3. Клиентская часть приложения

Клиентская часть приложения представляет собой веб-страницу, при разработке которой стандартом является использование HTML (HyperText Markup Language) для разметки страницы, CSS (Cascading Style Sheets) для стилизации элементов и JavaScript для обеспечения интерактивного взаимодействия с пользователем. Наиболее интересным для рассмотрения является выбор технологий JavaScript в силу огромного количества фреймворков, библиотек и паттернов.

Клиентскую часть приложения функционально разделим на рендеринг графического интерфейса и реализацию модели и логики. Графический интерфейс, в свою очередь, разделим на две части:

- Блок стандартных элементов управления;
- Блок построения диаграмм.

Рассмотрим технологии, предоставляющие возможность разработки вышеперечисленных элементов системы.

3.3.1. Инструмент рендеринга блока стандартных элементов управления

React.js

React – библиотека JavaScript, предназначенная для создания пользовательских интерфейсов, разрабатываемая компанией Facebook и сообществом индивидуальных разработчиков [14]. Отличительной особенностью является гибкость: написанные с помощью библиотеки компоненты могут использоваться вместе с любой другой технологией. Компоненты соответствуют принципам реактивного программирования – при любом изменении данных происходит рендеринг, что позволяет строить отзывчивые приложения. Также следует отметить высокое быстродействие, достигнутое с помощью ис-

пользования виртуальной модели DOM. Также библиотека предоставляет простой механизм создания: наследование от базового класса *React.Component* и переопределение необходимых методов жизненного цикла.

Главный метод жизненного цикла компонента, реализующий рендеринг – *Component.render()*. React предоставляет синтаксическое расширения языка JavaScript – JSX – с помощью которого реализуется комбинирование тегов HTML и кода JavaScript. С помощью JSX реализуется реактивное поведение компонентов. Пример JSX кода приведен в листинге.

Angular

Angular – JavaScript фреймворк, предназначенный для построения одностраничных приложений. Отличительной особенностью данной технологии является расширение стандартного HTML директивами, необходимыми для динамического изменения отображения в соответствии с данными [15]. Технология реализует паттерн MVC и использует двустороннее связывание – изменение содержимого элемента влечет изменение модели и наоборот. Пример HTML кода с Angular директивами приведен в листинге.

Vue

Vue – библиотека JavaScript, во многом схожая с React, например:

- Использование виртуального DOM [16];
- Соответствие реактивному шаблону;
- Компонентная структура [16];
- Реализация исключительно слоя View паттерна MVC;

Рассмотрим также отличия библиотеки Vue от React:

- Использование более легковесной модели виртуального DOM;
- Использование шаблонов HTML и CSS вместо JSX и CSS-in-JS в React;
- Независимость от версии JavaScript.

Вывод

В процессе выбора технологии разработки стандартных элементов управления на первом этапе отборе был исключен Angular Framework. Несмотря на то, что он, в отличие от React и Vue, предоставляет инструменты создания как слоя View, так и Model шаблона MVC, он остается HTML-ориентированным, что ограничивает функциональные возможности по сравнению с JavaScript-ориентированными React и Vue.

При сравнении React и Vue решающим фактором был тот факт, что для React существует большое количество готовых компонентов, что открывает возможности их переиспользования и, как следствие, уменьшение времени разработки.

3.3.2. Инструмент рендеринга блока построения диаграмм

Pixi.js

Разработчики Pixi позиционируют свой продукт как «быстрый, гибкий и бесплатный движок создания HTML5 рендеров» [17]. Pixi позволяет создавать быстрые приложения с плавной анимацией. Библиотека работает только с 2D графикой, поддерживает высокий уровень абстракции, что позволяет разработчикам создавать приложения, не вникая в детали реализации. Рендеринг по умолчанию основан на WebGL, но, в случае, если в браузере отсутствует его поддержка, происходит автоматическое переключение на canvas. Как известно, в

canvas объекты после прорисовки становятся частью точечного полотна, что затрудняет связывание объекта с обработчиком событий, однако в Pixi реализован механизм определения объекта события.

D3.js

D3 – библиотека для визуализации данных. В отличие от Pixi, D3 работает с SVG элементами. Построение элементов происходит в соответствии с привязанными данными. К особенностям разработки с помощью данной библиотеки относят:

- Использование выборок элементов [18];
- Активное использование функторов для определения обработчиков событий и установки атрибутов элементов;
- «Текучий интерфейс» - использование цепочек методов, каждый следующий из которых использует возвращаемое значение предыдущего;
- Использование связанных множеств при создании элементов.

Вывод

В разработке блока построения диаграмм более оптимальным выбором является использование библиотеки D3. Разрабатываемое приложение ориентировано на данные, что соответствует принципам данной библиотеки, т.к. главной ее целью является построение приложений визуализации данных. Также использование SVG позволяет изменять стиль элементов с помощью CSS, что уменьшает количество JavaScript кода.

3.4. Реализация модели

Для реализации модели, с учетом выбранного стека технологий, стандартом является соответствие паттерну «однонаправленный поток данных». Для реализации данного шаблона проектирования используется библиотека Redux, три основополагающих принципа которого:

- Использование единственного источника данных, объединяющего состояние всех компонентов в один объект;
- Состояние является объектом, доступным только для чтения;
- Использование чистых функций для описания мутаций.

В Redux оперируют такими понятиями, как **действие**, **хранилище**, **редьюсер**, **представление**. Схема взаимодействия элементов потока данных выглядит следующим образом (см. рис.):

1. Текущее состояние хранилища отображается в представлении;
2. Воздействие пользователя на представление вызывает действие;
3. Действие и текущее состояние передается редьюсеру;
4. Редьюсер создает новое состояние системы и передает его в хранилище;

4. Разработка системы

Прежде чем приступить к описанию процесса разработки, вернемся к рассмотрению выбранных технологий для реализации системы и обозначим решаемые с их помощью задачи.

В качестве архитектуры взаимодействия клиента и сервера выбран паттерн REST. Он позволяет использовать стандартные HTTP-методы для вызова методов API. Сервер разрабатывался на языке Python с помощью веб-фреймворка Django. Выбор обусловлен высоким уровнем модульности, благодаря чему возможно функциональное разделение сервера и, как следствие, независимая техническая поддержка частей, а также большой выбор билиотек, решающих рутинные задачи.

Клиентская часть, помимо HTML и CSS, содержит код JavaScript, необходимый для обеспечения интерактивности взаимодействия клиента с приложением. Здесь применен подход рендеринга со стороны клиента, что уменьшает нагрузку на веб-сервер. С помощью библиотеки React были построены основные элементы управления, Redux позволил реализовать паттерн «однонаправленный поток данных», что позволяет структурировать данные и обеспечить их актуальность для предоставления представлениям. Графическое построение конечных автоматов реализовано с использованием библиотеки D3, ориентированной на визуализацию данных через SVG.

4.1. Сервер

Сервер состоит из двух Django-приложений:

- *model_ checker* – приложение, реализующее выдачу статических файлов;
- *api* - приложение, реализующее обработку запросов к API, вклю-

чая запуск процесса NuSMV и выдачу результата выполнения.

Также в разработке использовались сторонние приложения: *rest_framework*, предоставляющий инструменты создания Web API в соответствии с принципами REST, и *webpack_loader*, необходимый для загрузки JavaScript bundle-файлов. Фрагмент конфигурационного файла приведен в листинге 4.1.

Листинг 4.1. Фрагмент конфигурационного файла Django.

```
1 INSTALLED_APPS = [
2   'django.contrib.auth',
3   'django.contrib.contenttypes',
4   'django.contrib.staticfiles',
5   'webpack_loader',
6   'model_checker',
7   'api',
8   'rest_framework',
9 ]
```

4.1.1. Приложение *api*

Данное приложение, помимо генерируемых Django модулей, содержит две директории, необходимые для реализации обработки запросов к методам API:

- */bin/* – содержит исполняемый файл модел-чекера NuSMV;
- */sandbox/* – содержит сгенерированные файлы исходного кода описания верифицируемой модели.

Обработка запроса происходит в модуле *views.py*. Данную рекомендацию дают разработчики библиотеки Django REST Framework. В данном модуле реализовано 5 функций:

- *run_simulation(request, flags)* – запускает процесс NuSMV в интерактивном режиме и, в зависимости от флагов, передает в

стандартный поток ввода процесса необходимые команды. По завершении возвращает информацию, предоставленную стандартным потоком вывода. Параметр *request* представляет собой строку, содержащую переданный клиентом в теле запроса исходный код модели на языке SMV. Параметр *flags* представляет собой флаги, используемые при запуске NuSMV.

- *run_ verification(request)* – запускает процесс NuSMV в обычном режиме, т.е. в режиме верификации модели. По завершении возвращает информацию, предоставленную стандартным потоком вывода. Параметр *request* представляет собой строку, содержащую переданный клиентом в теле запроса исходный код модели на языке SMV.

- *create_ file(source_ code)* – функция, создающая файл исходного кода SMV для последующей передачи процессу NuSMV. Параметр *source_ code* – строка, содержащая описание модели. Для генерации названия файла используется шаблон вида *"smv<hash>.smv"*, где *<hash>* – хэш-сумма параметра *source_ code*, вычисленная по алгоритму SHA-256. Возвращает путь к файлу в виде строки.

- *create_ command(filename, flags)* – возвращает массив строк, которые впоследствии будут переданы процессу NuSMV через именованный канал.

- *run_ in_ command_ line(commands, nusmv_ commands)* – функция, создающая новый процесс, в котором происходит запуск NuSMV. Параметр *commands* является массивом строк, полученным с помощью функции *create_ command(filename, flags)*. Параметр *nusmv_ commands* – массив комманд интерактивного режима NuSMV.

К функциям *run_ simulation* и *run_ verification* подключены декораторы *@api_ view* с параметром *'POST'*. Данный декоратор является частью библиотеки Django REST Framework и необходим при вызове обработчиков запросов. Для разделения методов симуляции и верификации модели используется механизм роутинга, который предполагает использование разных URL для вызываемых методов. Привязка обработчиков к URL происходит в модуле urls.py, фрагмент которого приведен в листинге А.8.

Листинг 4.2. Фрагмент файла urls.py.

```
1 urlpatterns = [
2   url(r'^simulate/$', run_simulation, name='run_simulation'),
3   url(r'^verify/$', run_verification, name='run_verification')
4 ]
```

Как упоминалось выше, создание процесса NuSMV и взаимодействие с ним происходит в методе *run_ in_ command_ line*. В нем используется класс *Popen* пакета *subprocess*. Данный класс предоставляет возможность создания процесса и привязки к его стандартным потокам ввода/вывода именованного канала, благодаря которому мы получаем информацию о выполнении NuSMV в процессе сервера.

4.1.2. Приложение *model_ checking*

Данное приложение отвечает за выдачу статических файлов клиенту. В корневой директории приложения, помимо сгенерированных фреймворком, содержатся две папки:

- */static/*, содержащая статические файлы JavaScript и CSS, которые будут рассмотрены подробнее в разделе 4.2;
- */templates/*, содержащая файлы шаблонов.

Фреймворк Django дополняет синтаксис HTML, что позволяет строить шаблоны страниц, в которых загрузка динамических данных происходит с помощью специального синтаксиса – DTL (Django

Template Language). Разрабатываемое приложение содержит всего одну страницу, шаблон которой приведен в листинге 4.3. Особо выделим строки, содержащие синтаксис DTL, и опишем выполняемые ими функции:

(1) : Инициализация приложения, необходимого для загрузки JavaScript файла, сгенерированного системой сборки Webpack (подробнее в разделе 4.2).

(2) : Инициализация инструментов загрузки статических файлов.

(7) : Загрузка статического файла CSS.

(20) : Загрузка статического файла JavaScript.

Листинг 4.3. Файл main.html – шаблон страницы.

```
1  {% load render_bundle from webpack_loader %}
2  {% load staticfiles %}
3  <!DOCTYPE html>
4  <html lang="en">
5    <head>
6      <meta charset="UTF-8">
7      <link rel="stylesheet" href="{% static 'css/websmv.css' %}">
8      <title> WebSMV </title>
9    </head>
10   <body>
11     <section>
12       <div class="app-container">
13         <div id="dropdown-container"></div>
14         <div id="state-editor-container"></div>
15         <div id="workspace-container"></div>
16         <div id="run-config-container"></div>
17         <div id="source-code-editor-container"></div>
18         <div id="results-container"></div>
19       </div>
20       {% render_bundle 'main' %}
21     </section>
22   </body>
23 </html>
```

4.2. Клиент

Разработчики React придерживаются разделения компонентов на два типа: ***«глупые»*** *компоненты* – сущности, занимающиеся исключительно рендером элементов HTML и вызывающие обработку событий с помощью механизма обратных вызовов, ***«умные»*** *компоненты* – инкапсулируют в себе «глупые» компоненты и передают им ссылки на функции и данные.

Здесь и далее примем *«умные» компоненты* как *контейнеры*, а *«глупые»* как *компоненты*.

4.2.1. Компоненты

StateEditor – реализует работу с множеством состояний конечного автомата. Активируется при выборе конкретного состояния. Состоит из поля ввода названия переменной, раскрывающегося списка типов переменных и кнопки добавления.Стоит отметить, что состав компонента StateEditor изменяется при изменении типа переменной. Так, для переменной типа массив компонент отображает также диапазон используемых индексов и раскрывающийся список возможных типов элементов массива, а для типа перечисление отображается поле ввода нового элемента перечисления, кнопка его добавления и список возможных значений, поддерживающий возможность удаления.

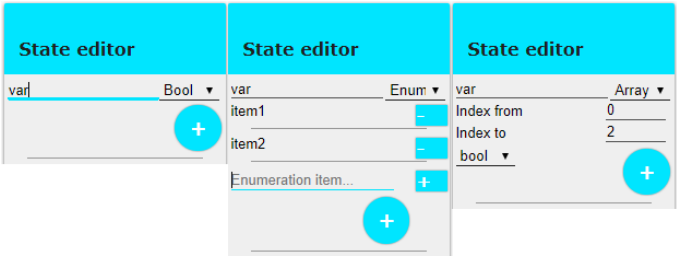


Рисунок 4.1. Вид компонента StateEditor для трех типов переменных.

StateItems – реализует отображение и редактирование списка атомарных высказываний, соответствующих выбранному состоянию. Состоит из списка атомарных высказываний, каждый элемент которой имеет текстовое поле, отображающее название переменной и ее тип, раскрывающийся список возможных значений переменной и кнопку удаления переменной из множества состояний. Для переменной типа массив компонент отображает множество выпадающих списков, содержащих допустимые для элементов массива значения.

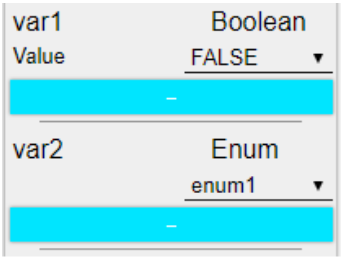


Рисунок 4.2. Вид компонента StateItem.

TransitionView – отображает отношение перехода, т.е. имена измененных после перехода переменных и их новые значения. Состоит из списка текстовых полей.

SimFlagsBox – реализует возможность настройки флагов, используемых при симуляции.

CtlBox/LtlBox – компоненты, реализующие создание и отображение спецификаций. В зависимости от выбранного типа темпоральной логики изменяются доступные кнопки.

4.2.2. Контейнеры

В каждом из контейнеров реализовано две функции Redux:

- *mapStateToProps* – устанавливает соответствие между объектами-свойствами контейнера и объектами хранилища;

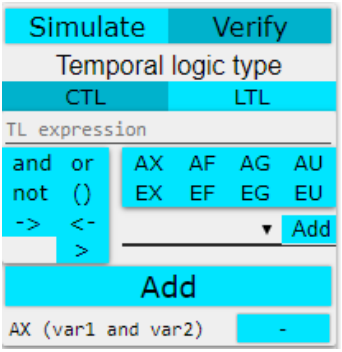


Рисунок 4.3. Вид компонента CtlBox.

- *mapDispatchToProps* – устанавливает соответствие между объектами-функциями контейнера и функциями создателями действий.

Таким образом происходит присоединение представления к схеме однонаправленного потока данных. Рассмотрим подробнее разработанные контейнеры.

StateEditorContainer – контейнер, инкапсулирующий компоненты отображения и редактирования состояний и переходов. При выборе состояния отображает *StateEditor* и *StateItem*, при выборе перехода отображает *EdgeView*. Использует следующие редьюсеры:

- WorkspaceReducer,
- StateItemReducer,
- StateEditorReducer.

WorkspaceContainer – контейнер, инкапсулирующий в себе главный SVG элемент блока построения конечных автоматов. Содержит объект D3Graph (подробнее в разделе 3.3.2). Работает с WorkSpaceReducer.

TlContainer – контейнер, отображающий CtlBoc или LtlBox в зависимости от полученного свойства.

RunConfigContainer – контейнер, содержащий в себе кнопки переключения режима (симуляция/верификация). В режиме верификации отображает кнопки переключения типа темпоральной логики и *TlContainer*.

SourceCodeContainer – текстовое поле, отображающее сгенерированный исходный код на языке SMV.

ExecutionResultContainer – текстовое поле, содержащее вывод программы NuSMV, полученный из тела ответа сервера.

4.2.3. Модель редьюсеров

В приложении используется 6 редьюсеров.

StateItemReducer – редьюсер, необходимый для создания новой переменной состояния. Описание полей объекта приведено в таблице 4.1.

Листинг 4.4. Инициализация состояния StateItemReducer.

```
1 const initialState = {
2   stateItem:{
3     stateName: '',
4     type: 'bool',
5     value: '',
6     range: [],
7     arrayType: ''
8   }
9 };
```

Таблица 4.1. Описание полей StateItemReducer.

Название	Тип	Описание
stateName	Строка	Название переменной
type	Строка	Тип переменной
value	Строка / массив	Значение переменной; массив значений для типа array
range	Массив	Диапазон возможных значений; нижний и верхний индексы для типа array
arrayType	Строка	Тип элементов массива; пустая строка, если $type \neq' array'$

StateEditorReducer – работает с массивом переменных состояния, каждый элемент которого является объектом StateItem.

Листинг 4.5. Инициализация состояния StateEditorReducer.

```
1 const initialState = {
2   states:[]
3 };
```

WorkspaceReducer – основной редьюсер, работающий моделью конечного автомата, представленной в виде графа, генерируемым исходным кодом описания, а также генераторами названий состояний и переходов. Подробнее поля объекта состояния рассмотрены в таблице 4.2.

Листинг 4.6. Инициализация состояния WorkspaceReducer.

```
1 const initialState = {
2   graph: {
3     vertices:[],
4     edges: [],
5     initVertex: {},
6     selected: ''},
7   enums: [],
8   vertexGenerator: 0,
9   edgeGenerator: 0,
```

```
10    sourceCode: '',
11  };
```

Таблица 4.2. Описание полей WorkspaceReducer.

Название	Тип	Описание
graph	Объект	Объектное представление модели конечного автомата
vertices	Массив	Массив объектов вершин графа
edges	Массив	Массив объектов ребер графа
initVertex	Объект	Инициализирующее состояние конечного автомата
selected	Строка	Название текущей выбранной вершины или ребра
enums	Массив	Массив объектов переменных, типом которых является *enumeration*
vertexGenerator	Целое число	Генератор номеров вершин
edgeGenerator	Целое число	Генератор номеров ребер
sourceCode	Строка	Исходный код описания модели на языке SMV

TlReducer – работает с типом выбранной пользователем темпоральной логики.

Листинг 4.7. Инициализация состояния TlReducer.

```
1  const initialState = {
2    tlType:'ctl'
3  };
```

TlBoxReducer – работает с вводимой пользователем в CtlBox/LtlBox формулой.

Листинг 4.8. Инициализация состояния TlBoxReducer.

```
1 const initialState = {
2   formula: ' '
3 };
```

RunConfigReducer – редьюсер, используемый в контейнере *RunConfigContainer*. Описание полей приведено в таблице 4.3.

Листинг 4.9. Инициализация состояния RunConfigReducer.

```
1 const initialState = {
2   runMode: 'sim',
3   simFlags: {},
4   ctlFormulas: [],
5   ltlFormulas: [],
6   result: ''
7 };
```

Таблица 4.3. Описание полей RunConfigReducer.

runMode	Строка	Тип операции NuSMV: симуляция или верификация
simFlags	Объект	Объект, содержащий флаги NuSMV и их значения
ctlFormulas	Массив	Массив введенных пользователем CTL-формул, представленных в виде строк
ltlFormulas	Массив	Массив введенных пользователем LTL-формул, представленных в виде строк
result	Строка	Полученный от сервера вывод программы NuSMV

4.2.4. Создатели дейтсвий

Создатели действий (Action Creators) – чистые функции, принимающие измененную часть хранилища состояния и возвращающие новый объект, содержащий тип действия и полученный параметр. В таблице 4.4 описаны основные типы используемых действий.

Таблица 4.4. Основные типы действий.

Тип действия	Описание
CHANGE_FORMULA	Изменение формулы в редакторе спецификаций
ADD_LTL	Добавление LTL формулы к списку спецификаций
ADD_VERTEX	Добавление новой вершины к графу
MOVE_VERTEX	Перенос вершины в поле построения
ADD_EDGE	Добавление нового ребра к графу
SET_INIT	Установка инициализирующей вершины графа
CHANGE_SELECTED	Изменение выбранного пользователем элемента графа
ADD_STATE_TO_GRAPH	Добавление новой переменной состояния в граф

4.2.5. Поле построения конечных автоматов

Для отрисовки поля построения конечных автоматов и взаимодействия с ним используется библиотека D3. Реализован класс D3graph, список методов и их описание приведены в таблице 4.5.

Построение происходит путем добавления SVG элементов средствами библиотеки D3. Основой для представления вершины графа служит SVG circle, для представления ребра – SVG line. Как было сказано в разделе 3.3.2, библиотека ориентирована на визуализацию данных, которые связываются с элементами с помощью метода *data(array)*, где *array* является массивом данных. Последующая обработка происходит в определяемых пользователем библиотеки функциях посредством передачи ссылок на обработчики данных или лямбда-выражений. В листинге 4.10 приведен фрагмент кода, реализующего добавление линий разметки поля построения.

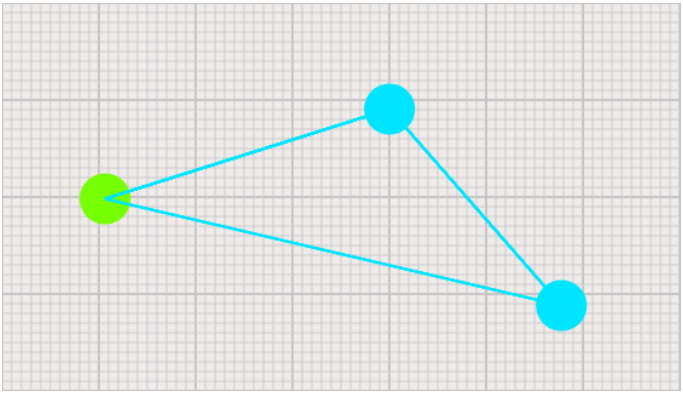


Рисунок 4.4. Вид поля построения конечных автоматов.

Листинг 4.10. Добавление разметки поля построения конечных автоматов.

```
let lines = root.append('g').selectAll('line')
  // steps — массив координат отступов между линиями разметки
  // по оси ординат
  .data(steps)
  .enter()
  .append('line')
  .attr('class', 'markdown')
  .attr('x1', (d)=>{return d;})
  .attr('x2', (d)=>{return d;})
  .attr('y1', 0)
  // height — высота поля построения
  .attr('y2', height);
```

4.2.6. Взаимодействие React и D3

Совместное использование библиотек React и D3 осложнено реализацией совместного использования и изменения объектного представления конечного автомата. Компонент React реализован придерживаясь концепции элементов с жизненным циклом, в то время как D3 выполняет непосредственную вставку элементов в корневой элемент SVG. Для актуализации отображаемой информации был реа-

лизован метод *D3graph.update(props)*. Данный метод, как описано в 4.2.5, производит переотрисовку всех элементов конечного автомата в соответствии с принятым как параметр объектом. Данный метод вызывается в одном из методов жизненного цикла React-компонента – *React.Component.componentDidUpdate()*.

4.2.7. Сборка фронэнд-части проекта

Прежде чем приступить к сборке проекта, рассмотрим структуру проекта в файловой системе (рисунок 4.11).

Листинг 4.11. Дерево директорий проекта.

```
|---bundles
| |---main<hashcode>.js
|
|---css
| |---main.css
|
|---js
    |---actions
    |---components
    |---containers
    |---d3
    |---reducers
    |---store
    |---util
    |---app.js
```

Рассмотрим подробнее приведенные выше директории:

- *bundles* содержит главный JavaScript файл, генерируемый системой сборки. Данный файл передается сервером клиенту.
- *css* содержит CSS файл, в котором описаны все используемые классы элементов.
- *js* содержит JavaScript файлы исходного кода для разработчиков.

- *actions* содержит функции, реализующие создание действий Redux.
- *components* – содержит классы React компонентов.
- *containers* – содержит классы React контейнеров.
- *d3* – содержит класс D3graph, реализующий отрисовку поля построения конечного автомата средствами библиотеки D3.
- *reducers* – содержит описания редьюсеров Redux.
- *store* – содержит конфигурационный файл хранилища Redux.
- *util* – содержит вспомогательные файлы, например, ArrayAdditions.js, в котором реализован поиск элемента в массиве объектов по одному из полей.

Данная структура проекта предоставляет большую наглядность, что упрощает разработку проекта. Однако такое разбиение проекта на модули усложняет процесс подключения скриптов в HTML файле. Также, вследствие отсутствия поддержки браузерами, использование стандарта ECMA Script 6 требует трансляцию кода в соответствии со стандартом EMCA Script 5. Использование различных библиотек влечет за собой большое количество внешних зависимостей, подключение которых вручную – достаточно трудоемкая задача. Для решения данных проблем используется система сборки Webpack. Данная система позволяет транслировать код, написанный с помощью разных стандартов как JavaScript, так и CSS, в код, соответствующий поддерживаемым большинством браузером стандартам. Webpack использует механизм загрузчиков: с помощью их конфигурации (листинг 4.12) система автоматически выполняет необходимые операции по трансляции.

Листинг 4.12. Фрагмент конфигурационного файла Webpack.

```
loaders: [
  //a regexp that tells webpack use the following loaders on all
  //.js and .jsx files
  {
    test: /\.jsx?$/,
    loader: 'babel-loader',
    query: {
      //specify that we will be dealing with React code
      presets: ['react', 'es2015', 'stage-0'],
      env: {
        development: {
          presets: ["react-hmre"]
        }
      }
    }
  }
],
```

Таблица 4.5. Описание методов класса D3graph.

Сигнатура метода	**Описание**
constructor(props)	Конструктор класса. Параметр *props* – объект, содержащий представление конечного автомата, а также ссылки на создателей дейтсвий.
update(props)	Отвечает за обновление визуализации конечного автомата в соответствии с входными данными. Параметр *props* – объект того же вида, что и в конструкторе класса. Данный метод вызывается каждый раз при обновлении Redux-хранилища.
createMarkup()	Создает разметку поля построения. Вызывается один раз при инициализации инкапсулирующего поле контейнера *WorkspaceContainer*.
on<Elem>Click()	Обработчики клика на элемент SVG, где *<Elem>* – целевой элемент SVG клика, может принимать три значения: *Root* – корневой SVG элемент, *Vertex* – группа элементов, представляющих вершину графа, *Edge* – группа элементов, представляющих ребро графа.
on<Elem>ContextClick()	Обработчики контекстного клика на элемент SVG, где *<Elem>* соответствует описанному в группе методов *on<Elem>Click()*.
onDrag()/onDrop()	Обработчики событий перемещения элемента вершины графа.
on<Action>Click()	Обработчики клика на элемент контексного меню.

5. Тестирование системы

5.1. Тестирование клиентской части

В общем случае, тестирование графических интерфейсов пользователя производится ручным методом. Однако для приложений, написанных с помощью библиотеки React, удобно использовать в качестве инструмента тестирования Jest. Jest реализует метод тестирования с помощью слепков, также поддерживается модульное тестирование с возможностью исключения зависимостей методом подстановки заглушек. Использование библиотеки рассмотрим на примере компонента StateItem, представляющего собой элемент массива переменных состояния. Создадим рендер данного элемента, используя в качестве функций обратного вызова заглушки *jest.fn()*. Далле создадим два слепка рендеров: до вызова метода *onChange()*и после (листинг 5.1). Результат выполнения теста представлен в листинге 5.2.

Листинг 5.1. Фрагмент теста для компонента StateItem.

```
const state = {stateName: 'test', type: 'bool', value: 'false'},
  vertex = {},
  enums = [],
  removeState = jest.fn(),
  changeSelect = jest.fn(),
  changeSelectArr = jest.fn();

const component = renderer.create(
<StateItem
  state={state} vertex={vertex}
  enums={enums} removeState={removeState}
  changeSelect={changeSelect}
  changeSelectArr={changeSelectArr} />
);

let tree = component.toJSON();
expect(tree).toMatchSnapshot();
tree.props.onChange();
tree = component.toJSON();
expect(tree).toMatchSnapshot();
```

Листинг 5.2. Фрагмент вывода сиситмы тестирования Jest.

```
> jest

FAIL  js\__tests__\StateItem-test.js
+ Select value change test

TypeError: tree.props.onChange is not a function

at Object.<anonymous> (js/__tests__/StateItem-test.js:28:16)
at Promise.resolve.then.el (node_modules/p-map/index.js:42:16)

- Select value change test (18ms)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   1 passed, 1 total
```

Помимо тестирования с помощью слепков, в ходе работы использовалось ручное тестирование. Было реализовано несколько разных сценариев использования.

- Проверка соответствия графически введенной модели текстовому описанию;
- Проверка поведения программы при количестве переменных состояния равному 50;
- Проверка поведения программы при количестве спецификаций равному 50;
- Проверка поведения программы при количестве состояний равному 50 и количеству переходов равному 50;
- Тестирование вводимых в формы названий переменных состояния;
- Проверка корректности отображения модели при случайных кликах;

- Проверка возможности добавления переменных с одинаковым именем;

В результате тестирования выявился ряд дефектов, например, определенный набор кликов по полю построения проекта порождал неверное отображение SVG элементов. Данный дефект был локализован и устранен.

Также было проведено тестирование функции трансляции объектного представления модели в текстовую. Производилось оно путем создания графического представления и оценки генерируемого кода на соответствие данной модели.

5.2. Тестирование серверной части

Сервер реализует следующую функциональность: обработка клиентских запросов, запуск процесса NuSMV, создание файла SMV, выдача клиенту ответа. Тестирование сервера преимущественно сводится к тестированию REST-сервиса. Для этого были использованы различные дополнения к браузерам Chrome и Firefox. В качестве передаваемых описаний конечных автоматов выдавались примеры из пакета NuSMV. Тестирование создания файла производилось с помощью модульного тестирования функции *create_file(source_code)*. Все тестовые сценарии прошли успешно.

ЗАКЛЮЧЕНИЕ

В ходе данной выпускной квалификационной работы было реализовано веб-приложение, реализующее проверку моделей с помощью верификатора NuSMV. были рассмотрены существующие решения в области графических интерфейсов пользователя для верификаторов, проанализированы преимущества и недостатки. Также была спроектирована архитектура приложения, обоснован выбор используемых в проекте технологий. С помощью JavaScript библиотеки React были реализованы основные элементы управления, библиотека Redux облегчила управление потоком данных в клиентской части приложения. Было разработано поле графического построения моделей с помощью библиотеки D3 и технологии SVG. Серверная часть приложения поддерживает обработку нескольких клиентов, также был реализован API, с помощью которого происходит взаимодействие клиентов и сервера.

Данный проект можно использовать в академических курсах с целью ознакомления с формальной верификацией систем. Использование автоматического перевода модели из графического представления в текстовое описание на языке SMV, а также инспекторов переменных состояния и спецификаций, предоставляют возможность верификации модели без необходимости изучения синтаксиса SMV. При доработке проекта возможно более широкое использование в промышленных масштабах.

Тем не менее, у проекта есть проблемы, решение которых планируется в будущем. Приложение поддерживает только самые примитивные типы данных: перечисления, булевы переменные и массивы. Динамически сгенерированный код недоступен для правки, т.к. для этого необходима функция обратной генерации (из текстового представления в графическое). Также не реализована интерпретация ответа от сервера; в данном контексте наиболее интересной функцией

является визуализация контрпримера. Также планируется расширить возможности приложения аутентификацией пользователей и возможностью сохранять разработанные модели в облачном сервисе или непосредственно на сервере.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Миронов А. М. Верификация программ методом Model Checking. — Москва, 2012. — С. 84.
2. Карпов Ю. Г. Верификация параллельных и распределенных программных систем. — БХВ-Петербург, 2010. — С. 560.
3. SPIN [Электронный ресурс], SPIN. — URL: `http://www.spinroot.com/` (дата обращения: 18.06.2017).
4. NUSMV: A New Symbolic Model Verifier / Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, Marco Roveri // Proceedings of the 11th International Conference on Computer Aided Verification. — CAV '99. — London, UK, UK : Springer-Verlag, 1999. — P. 495–499. — URL: `http://dl.acm.org/citation.cfm?id=647768.733923`.
5. SPIN [Электронный ресурс], NuSMV. — URL: `http://www.spinroot.com/` (дата обращения: 18.06.2017).
6. NuSeen [Электронный ресурс], NuSeen. — URL: `http://nuseen.sourceforge.net/` (дата обращения: 18.06.2017).
7. RFC 2616 HTTP/1.1 [Электронный ресурс], RFC. — URL: `https://tools.ietf.org/html/rfc2616` (дата обращения: 18.06.2017).
8. RFC 2818 HTTPS [Электронный ресурс], RFC. — URL: `https://tools.ietf.org/html/rfc2818` (дата обращения: 18.06.2017).
9. Wilde Erik, Pautasso Cesare. REST: From Research to Practice. — 1st edition. — Springer Publishing Company, Incorporated, 2011. — ISBN: 1441983023, 9781441983022.
10. SOAP [Электронный ресурс], W3. — URL: `https://www.w3.org/TR/2000/NOTE-SOAP-20000508/` (дата обращения: 18.06.2017).
11. Stevens W. Richard, Fenner Bill, Rudoff Andrew M. UNIX Network Programming, Vol. 1. — 3 edition. — Pearson Education, 2003. —

ISBN: 0131411551.

12. Surhone Lambert M., Tennoe Mariam T., Henssonow Susan F. Node.Js. — Mauritius : Betascript Publishing, 2010. — ISBN: 6133180196, 9786133180192.
13. Django [Электронный ресурс], Django. — URL: `https://www.djangoproject.com/` (дата обращения: 18.06.2017).
14. Abel Todd. ReactJS: Become a Professional in Web App Development. — USA : CreateSpace Independent Publishing Platform, 2016. — ISBN: 1533118825, 9781533118820.
15. Green Brad, Seshadri Shyam. AngularJS. — 1st edition. — O'Reilly Media, Inc., 2013. — ISBN: 1449344852, 9781449344856.
16. VueJS [Электронный ресурс], VueJS. — URL: `https://vuejs.org/` (дата обращения: 18.06.2017).
17. PixiJS [Электронный ресурс], PixiJS. — URL: `http://www.pixijs.com/` (дата обращения: 18.06.2017).
18. Zhu Nick Qi. Data Visualization with D3.Js Cookbook. — Packt Publishing, 2013. — ISBN: 178216216X, 9781782162162.

ПРИЛОЖЕНИЕ А
ЛИСТИНГИ

Листинг А.1. Обработка запроса на верификацию.

```
from rest_framework.decorators import api_view
from rest_framework.response import Response
import platform
import os
import hashlib
from subprocess import Popen, PIPE, STDOUT


SUBPROCESS_TIMEOUT = 10
SANDBOX_RELATIVE_PATH = '.\\api\\sandbox\\'
PATH = os.getcwd()
MODE_VERIFICATION = 'verification'
MODE_SIMULATION = 'simulation'
NUSMV_EXE = '.\\api\\bin\\nusmv.exe'


@api_view(['POST'])
def run_verification(request):
if platform.system() == 'Windows':
filename = create_file(request.body.decode('utf-8'))
commands = create_command(filename, flags)
output = run_in_command_line(commands, None)
return Response(output)


def create_file(source_code):
path = PATH + SANDBOX_RELATIVE_PATH
hashcode = str(hashlib.sha256().hexdigest())
filename = 'smv' + hashcode + '.smv'
with open(path + filename, 'wt', encoding='utf-8') as f:
f.write(source_code)
return SANDBOX_RELATIVE_PATH + filename


def create_command(filename, flags):
commands = []
commands.append(NUSMV_EXE)
commands.append(filename)
```

```
for f in flags:
commands.append(f)
return commands


def run_in_command_line(commands, nusmv_commands):
p = Popen(commands, stdout=PIPE, stdin=PIPE, stderr=STDOUT)
stdout = p.communicate()[0].decode()
if nusmv_commands is not None:
for c in nusmv_commands:
stdout += p.communicate(c)[0].decode()
return stdout
```

Листинг А.2. Пример сгенерированного SMV-кода.

```
MODULE main
VAR
  VDSSD : boolean;
  VDSVSD  : boolean;
SPEC VDSSD
ASSIGN
  init(VDSSD) := TRUE;
  init(VDSVSD) := TRUE;
  next(VDSSD):=
    case
      VDSSD = TRUE & VDSVSD = TRUE  : FALSE;
      VDSSD = TRUE & VDSVSD = FALSE  : FALSE;
      TRUE : TRUE;
    esac;
  next(VDSVSD):=
    case
      VDSSD = TRUE & VDSVSD = TRUE  : FALSE;
      VDSSD = TRUE & VDSVSD = FALSE  : TRUE;
      TRUE : TRUE;
    esac;
```

Листинг А.3. Создатели действий WorkspaceActions.

```
import {store} from '../app'

export function addVertex(vertex) {
  vertex.states = store.getState().stateEditorReducer.states.slice()
      ;
  return {
    type: "ADD_VERTEX",
```

```
    payload: vertex
  }
}

export function removeVertex(vertex) {
  return {
    type: "REMOVE_VERTEX",
    payload: vertex
  }
}

export function moveVertex(vertex, x, y) {
  return {
    type: "MOVE_VERTEX",
    payload: {vertex: vertex, x: x, y: y}
  }
}

export function addEdge(vertex1, vertex2) {
  return {
    type: "ADD_EDGE",
    payload: [vertex1, vertex2]
  }
}

export function removeEdge(vertices) {
  return {
    type: "REMOVE_EDGE",
    payload: vertices
  }
}

export function changeInitVertex(vertex) {
  return {
    type: 'SET_INIT',
    payload: vertex
  }
}
```

Листинг А.4. Реализация компонента LtlBox.

```
import React from 'react'

export class LtlBox extends React.Component {

constructor(props) {
```

```
6   super(props);
7   this.selected = '';
8   this.binary = ['and', 'or', 'not', '()', '->', '<->'];
9   this.ctl = ['G','F','X','U'];
10  this.addState = this.addState.bind(this);
11  this.changeFormula = this.changeFormula.bind(this);
12  this.onAppend = this.onAppend.bind(this);
13  this.changeState = this.changeState.bind(this);
14  this.addLtl = this.addLtl.bind(this);
15 }
16
17 static propTypes = {
18   formula: React.PropTypes.string.isRequired,
19   states: React.PropTypes.array.isRequired,
20   addLtl: React.PropTypes.func.isRequired,
21   changeFormula: React.PropTypes.func.isRequired
22 };
23
24 render() {
25   let _this = this;
26   if ((_this.props.states.length != 0) && this.selected == ''){
27     this.selected = _this.props.states[0].stateName;
28   }
29   return (
30     <div className="tl-box">
31     <input className="tl-expr"
32       onChange={_this.changeFormula}
33       value={_this.props.formula}
34       placeholder="TL expression"/>
35     <div className="boolean-operators-box">
36     {
37       _this.binary.map((b) => {
38       return (
39         <button className="btn-boolean"
40         key={b}
41         value={b}
42         onClick={_this.onAppend}>
43         {b}
44         </button>
45         )
46       })
47     }
48     </div>
49     <div className="ltl-operators-box">
50     {
51     _this.ctl.map((b) => {
```

```
52        return (
53          <button className="btn-ctl"
54            key={b}
55            value={b}
56            onClick={_this.onAppend}>
57          {b}
58          </button>
59          )
60      })
61      }
62      </div>
63      <div className="state-insert-box">
64      <select className="state-select" onChange={_this.changeState}>
65      {
66        _this.props.states.map( (st) => {
67          return (
68            <option key={st.stateName} value={st.stateName}>
69            {st.stateName}
70            </option>
71            )
72        })
73      }
74      </select>
75      <button className="btn-add-state-tl"
76        onClick={_this.addState}>
77        Add
78      </button>
79      </div>
80      <button className="btn-add-tl-expr" onClick={this.addLtl}>
81      Add
82      </button>
83      </div>
84    );
85  }
86
87  addLtl(){
88    this.props.addLtl(this.props.formula);
89    this.props.changeFormula('');
90  }
91
92  addState() {
93    this.props.changeFormula(this.props.formula.concat(this.selected+'
          '));
94  }
95
96
```

```
97  changeFormula(event) {
98    this.props.changeFormula(event.target.value);
99  }
100
101 onAppend(event) {
102   this.props.changeFormula(this.props.formula.concat(event.target.
          value+' '));
103 }
104
105 changeState(event) {
106   this.selected = event.target.value;
107 }
108 }
```

Листинг А.5. Реализация контейнера WorkspaceContainer.

```
1   import React from 'react'
2   import { bindActionCreators } from 'redux'
3   import { connect } from 'react-redux'
4   import D3graph from '../d3/D3graph'
5   import * as wsActions from '../actions/WorkspaceActions'
6
7   export class WorkspaceContainer extends React.Component {
8
9   constructor(props) {
10    super(props);
11    console.log(props);
12    this.d3graph = new D3graph(props);
13    this.isInit = true;
14  }
15
16  static propTypes = {
17    graph: React.PropTypes.object.isRequired,
18    addVertex: React.PropTypes.func.isRequired,
19    removeVertex: React.PropTypes.func.isRequired,
20    addEdge: React.PropTypes.func.isRequired,
21    removeEdge: React.PropTypes.func.isRequired,
22    moveVertex: React.PropTypes.func.isRequired,
23    changeInitVertex: React.PropTypes.func.isRequired,
24    changeSelectedVertex: React.PropTypes.func.isRequired
25  };
26
27  render() {
28    return (
29    <div className="container-svg">
30      <svg className="main-svg" id="main-svg"/>
```

```
31    </div>
32    );
33  }
34
35  componentDidMount() {
36    if (this.isInit){
37      this.d3graph.createMarkup();
38      this.isInit = false;
39    }
40  }
41
42  componentDidUpdate() {
43    this.d3graph.update(this.props);
44  }
45  }
46
47  function mapStateToProps (state) {
48    return {
49      graph: state.workspaceReducer.graph
50    }
51  }
52
53  function mapDispatchToProps(dispatch) {
54    return {
55      addVertex: bindActionCreators(wsActions.addVertex, dispatch),
56      removeVertex: bindActionCreators(wsActions.removeVertex,
            dispatch),
57      addEdge: bindActionCreators(wsActions.addEdge, dispatch),
58      removeEdge: bindActionCreators(wsActions.removeEdge, dispatch),
59      moveVertex: bindActionCreators(wsActions.moveVertex, dispatch),
60      updateD3: bindActionCreators(wsActions.updateD3, dispatch),
61      changeInitVertex: bindActionCreators(wsActions.changeInitVertex,
             dispatch),
62      changeSelectedVertex: bindActionCreators(wsActions.
            changeSelectedVertex, dispatch)
63    }
64  }
65
66  export default connect(mapStateToProps, mapDispatchToProps)(
        WorkspaceContainer);
```

Листинг А.6. Конструктор класса D3graph.

```
1  constructor(props){
2    this.isVertexClicked = false;
3    this.isContextActive = false;
```

```
4     this.edgeFrom = '';
5     this.action = 'select';
6
7     this.graph = props.graph;
8     this.activeTool = props.activeTool;
9     this.addVertex = props.addVertex;
10    this.removeVertex = props.removeVertex;
11    this.moveVertex = props.moveVertex;
12    this.addEdge = props.addEdge;
13    this.removeEdge = props.removeEdge;
14    this.changeState = props.changeState;
15    this.updateD3 = props.updateD3;
16    this.changeInitVertex = props.changeInitVertex;
17    this.changeSelectedVertex = props.changeSelectedVertex;
18
19    this.update = this.update.bind(this);
20    this.createMarkup = this.createMarkup.bind(this);
21    this.onRootClick = this.onRootClick.bind(this);
22    this.onRootContextClick = this.onRootContextClick.bind(this);
23    this.onVertexClick = this.onVertexClick.bind(this);
24    this.onVertexContextClick = this.onVertexContextClick.bind(this);
25    this.onEdgeClick = this.onEdgeClick.bind(this);
26    this.onEdgeContextClick = this.onEdgeContextClick.bind(this);
27    this.onDrag = this.onDrag.bind(this);
28    this.onDrop = this.onDrop.bind(this);
29    this.onInitClick = this.onInitClick.bind(this);
30    this.onConnectClick = this.onConnectClick.bind(this);
31    this.onDeleteClick = this.onDeleteClick.bind(this);
32  }
```

Листинг А.7. Пример сгенерированного SMV-кода.

```
1   const initialState = {
2   stateItem:{
3   stateName: '',
4   type: 'bool',
5   value: '',
6   range: [],
7   arrayType: ''
8   }
9   };
10
11  export default function StateItemReducer(state = initialState,
        action) {
12  switch (action.type) {
13    case 'CHANGE_TYPE':{
```

```
14      let s1 = {...state.stateItem, type: action.payload};
15      if(s1.type == 'array'){
16        s1.range = [0,0];
17        s1.arrayType = 'bool';
18        s1.value = [];
19      }
20      else
21        s1.range = [];
22      return {...state, stateItem: s1};
23    }
24    case 'CHANGE_NAME':{
25      let s2 = {...state.stateItem, stateName: action.payload};
26      return {...state, stateItem: s2};
27    }
28    case 'ADD_ENUM':{
29      let range = state.stateItem.range.slice();
30      if(range.includes(action.payload)){
31        return state;
32      }
33      range.push(action.payload);
34      let tmp1 = {...state.stateItem, range: range};
35      return {...state, stateItem: tmp1};
36    }
37    case 'REMOVE_ENUM':{
38      let range = state.stateItem.range.slice();
39      range.splice(action.payload - 1, 1);
40      let tmp2 = {...state.stateItem, range: range};
41      return {...state, stateItem: tmp2};
42    }
43    case 'SET_RANGE': {
44      let tmp3 = {...state.stateItem, range: action.payload};
45      return {...state, stateItem: tmp3};
46    }
47    case 'SET_ARRAY_TYPE': {
48      let tmp4 = {...state.stateItem, arrayType: action.payload};
49      return {...state, stateItem: tmp4};
50    }
51    case 'RESET': {
52      return initialState;
53    }
54      default: return state;
55    }
56  }
```

Листинг А.8. Точка входа приложения app.js.

```
1  import React from 'react'
2  import {render} from 'react-dom'
3  import configureStore from './store/configureStore'
4  import StateEditorContainer from './containers/StateEditorContainer'
5  import DropdownContainer from './containers/DropdownContainer'
6  import WorkspaceContainer from './containers/WorkspaceContainer'
7  import SourceCodeEditorContainer from "./containers/
       SourceCodeEditorContainer";
8  import {Provider} from 'react-redux'
9  import RunConfigContainer from "./containers/RunConfigContainer";
10 import ResultsContainer from "./containers/ResultsContainer";
11
12 export const store = configureStore()
13
14 render(
15   <Provider store={store}>
16     <div className="dropdown">
17       <DropdownContainer />
18     </div>
19   </Provider>,
20   document.getElementById('dropdown-container')
21 );
22
23 render(
24   <Provider store={store}>
25     <div className="state-editor">
26       <StateEditorContainer />
27     </div>
28   </Provider>,
29   document.getElementById('state-editor-container')
30 );
31
32 render(
33   <Provider store={store}>
34     <div className="workspace">
35       <WorkspaceContainer/>
36     </div>
37   </Provider>,
38   document.getElementById('workspace-container')
39 );
40
41 render(
42   <Provider store={store}>
43     <div className="source-code">
44       <SourceCodeEditorContainer/>
```

```
45      </div>
46    </Provider>,
47    document.getElementById('source-code-editor-container')
48  );
49
50  render(
51    <Provider store={store}>
52      <div className="run-config">
53        <RunConfigContainer/>
54      </div>
55    </Provider>,
56    document.getElementById('run-config-container')
57  );
58
59  render(
60    <Provider store={store}>
61      <div className="results">
62        <ResultsContainer/>
63      </div>
64    </Provider>,
65    document.getElementById('results-container')
66  );
67
68  export default store;
```