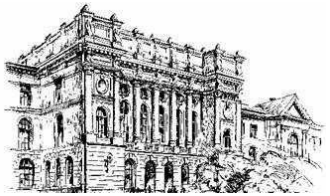
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий



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: 230100 – 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, 35 стр., 1 рис., 5 табл., 3 ист., 1 прил.

ТРАНСФОРМАЦИЯ ПРОГРАММ, СТАТИЧЕСКИЙ АНАЛИЗ,
КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Краткое описание работы

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1. Анализ существующих решений

Наиболее известными инструментами, реализующими проверку моделей общего назначения, являются NuSMV и SPIN. 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. Поддерживает все основные режимы работы. Имеет окно «Automata view», отображающее сгенерированный на основе описания модели конечный автомат.

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. Программа состоит из есдинственного окна, состоящего из меню и трех текстовых полей:

1. поле редактирования исходного кода Promela,

2. поле отображения вывода Spin,

3. поле отображения предупреждений компилятора.

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. Использует устаревшую версию NuSMV 1.

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.

1.2.2. gNuSMV

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

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.

1.2.3. NuSeen

В отличие от вышерассмотренных графических интерфейсов, являющихся самостоятельными приложениями, NuSeen основан на интегрированной среде разработки Eclipse. Отличительной особенностью является построение графов зависимостей переменных и графов зависимостей жестко соединенных наборов переменных (SCV – Strongly Connected Variables set). Поддерживается визуализация контрпримера.

Таблица 1.5. NuSeen

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

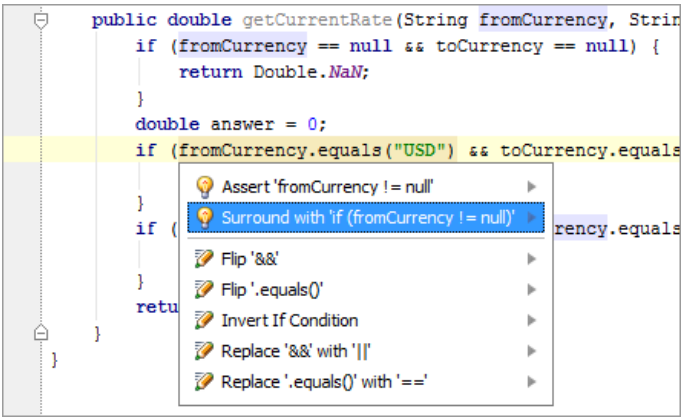


Рисунок 1.1. Предупреждение от статического анализатора в IntelliJ IDEA и всплывающее меню Quick Fix

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

3. Проектирование архитектуры системы

3.1. Архитектура приложения

Реализация веб-приложения подразумевает использование клиент-серверной архитектуры приложения. В качестве протокола прикладного уровня стандартом, де-факто, являются протоколы HTTP и HTTPS. 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. Важно отметить, что RESTful сервисы ориентированы на работу с данными.

3.1.1. REST

REST (Representational State Transfer) – архитектурный стиль взаимодействия распределенных клиент-серверных приложений. Термин «RESTful» применяется к веб-сервисам, работающим в соответствии с заложенными в основу данного стиля принципами. Важнейшей особенностью REST-взаимодействий является соответствие принципам CRUD (create, read, update, delete) – использование четырёх базовых методов работы с персистентными хранилищами данных. 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 1.

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 1. SOAP ориентирован на работу с операциями, т.е. с его помощью возможно построение сложного веб-сервиса.

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.

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, каждая из них 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. 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 байткод языком, что делает программы более высокую скорость исполнения, 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. 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 паттерном MVC (Model-View-Controller). Данный фреймворк

разрабатывался с целью минимизации временный затрат на разработку. 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 выдачу 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. 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 использования 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 рас-

ширения 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. 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;
- Соответствие реактивному паттерну;
- Компонентная структура;
- 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 паттерна 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 огромное 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». 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;
- 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 взамодействия 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 бибилиотеки 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, испольщуемые 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 коротого приведен в листинге 4.2.

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 именнованного 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 фреймфрком, 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 динамичсеких данных происходит с помощью специального синатксиса – 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 интсрументов загрузки статических файлов.

(9) : Загрузка статического файла CSS.

(22) : Загрузка статичсекого файла JavaScript.

Листинг 4.3. Файл main.html – шаблон страницы.

```
1  {% load render_bundle from webpack_loader %}
2  {% load staticfiles %}
3
4  <!DOCTYPE html>
5
6  <html lang="en">
7    <head>
8      <meta charset="UTF-8">
9      <link rel="stylesheet" href="{% static 'css/websmv.css' %}">
10     <title> WebSMV </title>
11   </head>
12   <body>
13     <section>
14     <div class="app-container">
15       <div id="dropdown-container"></div>
16       <div id="state-editor-container"></div>
17       <div id="workspace-container"></div>
18       <div id="run-config-container"></div>
19       <div id="source-code-editor-container"></div>
20       <div id="results-container"></div>
21     </div>
22     {% render_bundle 'main' %}
23     </section>
24   </body>
25 </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.

рис.

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.

рис.

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

рис.

SimFlagsBox – реализует возможность настройки флагов, используемых при симуляции.

рис.

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

рис.

4.2.2. Контейнеры

В каждом из контейнеров реализовано две функции Redux:

- *mapStateToProps* – устанавливает соответствие между объектами-свойствами контейнера и объектами хранилища;
- *mapDispatchToProps* – устанавливает соответствие между объектами-функциями контейнера и функциями содателями 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 слудующие редьюсеры:

- WorkspaceReducer,
- StateItemReducer,
- StateEditorReducer.

WorkspaceContainer – контейнер, инкапсулирующий в себе главный SVG элемент блока построения конечных автоматов. Содержит объект D3Graph (подробнее в разделе **??**). 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 8 редьюсеров.

StateEditorReducer –

Листинг 4.4. Инициализация состояния StateEditorReducer.

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

StateItemReducer –

WorkspaceReducer –

Листинг 4.5. Инициализация состояния WorkspaceReducer.

```
const initialState = {
  graph: {
    vertices:[],
    edges: [],
    numberOfEdges: 0,
    initVertex: {},
    selected: ''},
  enums: [],
  vertexGenerator: 0,
  edgeGenerator: 0,
  sourceCode: '',
};
```

TlReducer –

Листинг 4.6. Инициализация состояния TlReducer.

```
const initialState = {
  tlType:'ctl'
};
```

TlBoxReducer –

Листинг 4.7. Инициализация состояния TlBoxReducer.

```
const initialState = {
  formula: ' '
};
```

TransitionEditorReducer –

Листинг 4.8. Инициализация состояния TransitionEditorReducer.

```
const initialState = {
  transition:{
    name: '',
    value: ''
  }
};
```

RunConfigReducer –

Листинг 4.9. Инициализация состояния RunConfigReducer.

```
const initialState = {
```

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

SourceCodeEditorReducer –

Листинг 4.10. Инициализация состояния SourceCodeEditorReducer.

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

4.2.4. Создатели дейтсвий

4.2.5. Поле построения конечных автоматов

4.2.6. Взаимодействие React-Redux и D3

4.2.7. Система сборки Webpack

4.2.8. Стили эелементов

5. Тестирование системы

Пара слов о тестировании системы

ЗАКЛЮЧЕНИЕ

Выводы по работе

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пупкин Василий, Пупкина Василиса, Пупкинский В.П. Русская статья // Научно-технические ведомости НИИ ЧАВО. — 2000. — № 16. — С. 32–64.
2. ANTLR [Электронный ресурс], Веб-сайт проекта ANTLR. — URL: `http://www.antlr.org/` (дата обращения: 18.06.2015).
3. Нимейер П., Леук Д. Программирование на Java. — Эксмо, 2014. — ISBN: 9785457661004. — URL: `https://books.google.ru/books?id=zVpUBQAAQBAJ`.

ПРИЛОЖЕНИЕ А
ЛИСТИНГИ

Листинг А.1. Код на Java

```
1  private static boolean changesLine(final GenericTreeNode patternTree
       , int reportLine) {
2    LineNumberFetcher fetcher = new LineNumberFetcher();
3    try {
4      fetcher.visit(patternTree);
5      return fetcher.lines.contains(reportLine);
6    } catch (Exception e) {
7      return false;
8    }
9  }
10
11 private static class LineNumberFetcher implements TreeVisitor {
12   Set<Integer> lines = new TreeSet<>();
13
14   @Override
15   public void visit(GenericTreeNode genericTreeNode) throws
         Exception {
16     for (GenericTreeNode matchingNode : genericTreeNode.
           getMatchNodes()) {
17       lines.add(matchingNode.getLine());
18     }
19     for (GenericTreeNode child : genericTreeNode.getChildren()) {
20       child.acceptVisitor(this);
21     }
22   }
23 }
```