**Universitatea de Stat din Moldova**

**Facultatea MATEMATICA ŞI INFORMATICA**

**DEPARTAMENTUL DE INFORMATICA**

**PRIJILEVSCHI EVGHENI**

**ГЕНЕРАЦИЯ ПРОГРАММИНОГО КОДА**

**НА ОСНОВЕ ДИНАМИЧЕСКИХ МОДЕЛЕЙ**

**Informatica - 32**

**Teză de licență**

Sef Catedră **\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(semnătura)

Conducător ştiinţific: **\_\_\_\_\_\_\_\_\_\_\_\_\_ Croitor Mihail, Lector universitar**

(semnătura)

Autorul: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura)

**Chişinău-2014**

Содержание (сгенерить)

Введение

a.​ Уникальность работы (15-20%)

b.​ Актуальность работы (70%)

c.​ Цели и задачи работы

Цели – теоретические: исследование, анализ, сравнительный анализ

Задачи – практические: разработка, внедрение, реализация

2)​ Динамические модели информационных систем

a.​ Событийная модель информационных систем

b.​ Динамические модели (Конечные автоматы)

c.​ Генерация кода на основе динамических моделей

d.​ Концепции статического и динамического порождающего программирования

3)​ Существующие генераторы кода

a.​ Примеры приложений генерации кода с помощью конечных автоматов

b.​ Сравнение вышеописанных примеров

c.​ Сценарии использования генерации кода на основе динамических моделей

d.​ Практические примеры

4)​ Описание разработки приложения

a.​ Формулировка проблемы

b.​ Определение функциональных требований (список)

c.​ Инструменты разработки

IDE, язык программирования, специальные библиотеки

d.​ Проектирование информационной системы

e.​ Описание пользовательского интерфейса

f.​ Резюме

5)​ Вывод

a.​ Актуальность (30%)

b.​ Уникальность (80-85%)

c.​ Результаты по целям и задачам

d.​ Дальнейшее развитие проекта

​ •Библиография (книжная + электронная)

​ •Приложение – код программы

# Введение

Предметом этой дипломной работы является генерация кода на основе динамических моделей, называющихся ещё конечными автоматами. Конечные автоматы представляют собой спецификацию последовательности состояний, через которые проходит объект или взаимодействие в ответ на события своей жизни, а также ответные действия объекта на эти события. Конечные автоматы применяются в автоматизации проектирования электронных приборов, протоколах передачи данных, синтаксическом анализе. Между тем, применение конечных автоматов в программировании является привлекательной с точки зрения использования компонентов конечных автоматов. Эти компоненты на практике способствуют минимизации дублирования кода и обеспечивают столько вариантов соединения, сколько потребуется для оптимизации повторного использования.

Несмотря на наличие похожего программного обеспечения, многие из них предназначены для решения узко поставленной задачи в рамках определённого языка программирования. В отсутствии удобного пользовательского интерфейса и богатого инструментария ставится вопрос о разработки нового программного обеспечения, которое оставалось бы актуальным помощником, как для начинающих программистов, так и для опытных разработчиков. Как следствие, уникальность данной работы кроется в сочетании нескольких факторов.

* Во-первых, пользовательский интерфейс представлен в качестве веб-приложения, где пользователь строит динамическую модель на клиентской стороне, а сервер генерирует код на основе полученных данных. Данная клиент-серверная модель делает программу платформонезависимой, доступной и удобной для пользователя.
* Во-вторых, пользователю представлена возможность выбрать для генерации один из нескольких современных языков программирования с реализованной объектно-ориентированной архитектурой. Локализация поведения каждого состояния происходит в каждом отдельном классе благодаря единой модели, которая получается либо проектированием диаграммы состояний, либо импортом/экспортом данных в формате XML.
* В-третьих, генерируемый код реализует паттерн Состояние, что, в отличие от других реализаций, позволит избежать многих хлопот: код будет соответствовать принципу открытости/закрытости (классы должны быть открыты для расширения, но закрыты для реализации). Данный принцип проектирования осуществляется благодаря инкапсулированию переменных аспектов этой архитектуры и конкретизации переменных переходов между состояниями. При огромных масштабах такие состояния не будут «прятаться» в множестве условных конструкций. Таким образом, дальнейшие изменения не приведут к ошибкам в готовом коде.

В наши дни сложно переоценить роль UML-диаграмм в контексте планирования информационных систем. Многочисленные средства разработки позволяют строить графическое описание объектных моделей. Основная причина их появления состоит в том, что языки программирования не обеспечивают нужный уровень абстракции, способный облегчить процесс проектирования. Унифицированный язык моделирования (UML 2.0) описывает 13 официальных типов диаграмм. Хотя эти виды диаграмм отражают различные подходы многих специалистов к моделированию систем, важно понимать, что UML не является реализацией программы, а лишь служит эскизом для дальнейшей разработки. Структуру классов, вариантов использования, компонент можно экспортировать в абстрактный код с реализованным скелетом приложения и пустой реализацией. Особое внимание стоит обратить на диаграмму автоматов, на которой отражены отношения между состояниями и переходами между ними. Диаграмма состояний представляет собой ориентированный граф для конечного автомата, вершины которого обозначают состояния системы, а дуги - события (реакцию), происходящую при переходе из одного состояния в другое. В частности, при разработке сложных систем, основанных на событиях, вызванных действиями пользователя (щелчок мыши, нажатие клавиши), конечные автоматы приходятся как нельзя кстати.

Когда разработчик сталкивается с необходимостью построения контекстно-свободной грамматики, у него есть несколько вариантов работы. Вполне возможно использование формальной грамматики на основе некоторого конечного алфавита, детерминированного конечного автомата или регулярных выражений. Первичной целью для моей работы является обеспечение возможностью построения чётких и ясных переходов между состояниями в доступной форме одновременно как для опытных специалистов, так и для начинающих программистов.

# Цели и задачи работы

Одним из важнейших условий разработки успешного приложения служит понимание теоретических принципов, служащих основой программы. В данном случае, генерация осуществляется за счёт конечных автоматов. Для того, чтобы все компоненты органично взаимодействовали между собой, передо мной была поставлена цель ознакомиться со структурой и теоретической базой динамических моделей, а также иметь чёткое представление о концепциях статического и порождающего программирования. Сочетание данных знаний позволит лучше понять событийную модель информационных систем, а также генерацию кода на основе динамических моделей.

По мере развития сети Интернет, технологии программирования не стоят на месте, постоянно расширяясь и дополняясь новыми фреймворками. Пришло время и нам идти в ногу со временем, поэтому мой взгляд привлекли Java EE технологии, а точнее Java-сервлеты вместе с контейнером сервлетов (также именуемым веб-контейнером). Следующей целью служит ознакомление с современными методами разработки веб-приложений, взяв за основу известный и активно используемый фреймворк Spring MVC, а также следуя принятым методам разработки на основе системы контроля версий, Apache Maven для автоматизации сборки проектов, а также применяя паттерны проектирования в проекте.

Конечная задача данной дипломной работы – использовать возможные преимущества на практике и уменьшить многократно ручной труд, заменив его на автоматизированный, посредством использования конечных автоматов. На основе вышеупомянутых целей можно выделить задачи перед разрабатываемым приложением:

* Строить графическую модель конечного автомата, где множество состояний описывает состояние системы в момент бездействия пользователя, а дуги – события и реакцию системы.
* Преобразовывать графическую модель конечного автомата в программное представление в виде xml-документа.
* Преобразовывать динамическую модель в необходимый конкретный язык.
* Обеспечить использование паттернов проектирования в генерируемый код при необходимости.
* Возможность загружать/сохранять как весь проект, так и xml-структуру динамической модели в отдельности.
* Обеспечить возможность добавления новых языков.
* Добавить необходимый справочный материал для быстрого ознакомления пользователя с информационной системой.

# Динамические модели информационных систем

Событийная модель информационных систем

wiki:Автоматное программирование

wiki: Switch-технология

http://habrahabr.ru/post/173103/

http://habrahabr.ru/post/128772/

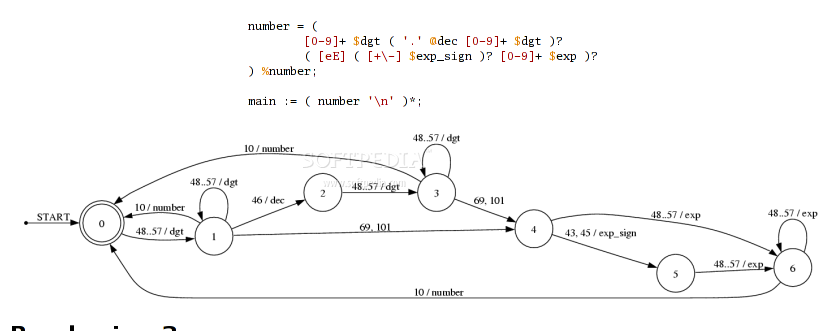
# Исследование предметной области

Сегодня ни одна из существующих IT-компаний не может обходиться без инструментов проектирования систем различного назначения и различных уровней сложности. Внедрение проектирования в жизненный цикл разработки является одним из важнейших условием динамичного развития.

Однако процессу генерации кода не уделяется должного внимания. Среди факторов, препятствующих использованию инструментов, порождающих код, наиболее значимыми являются следующие: нежелание компаний подстраивать команду разработчиков под отдельно взятую программу, несоответствие со специфичными требованиями разработчика, а также отсутствием возможности повторного использования сгенерированного кода. Существующие генераторы кода на основе динамических моделей можно разделить по следующим требованиям, являющимися критериями оценки выбранных приложений:

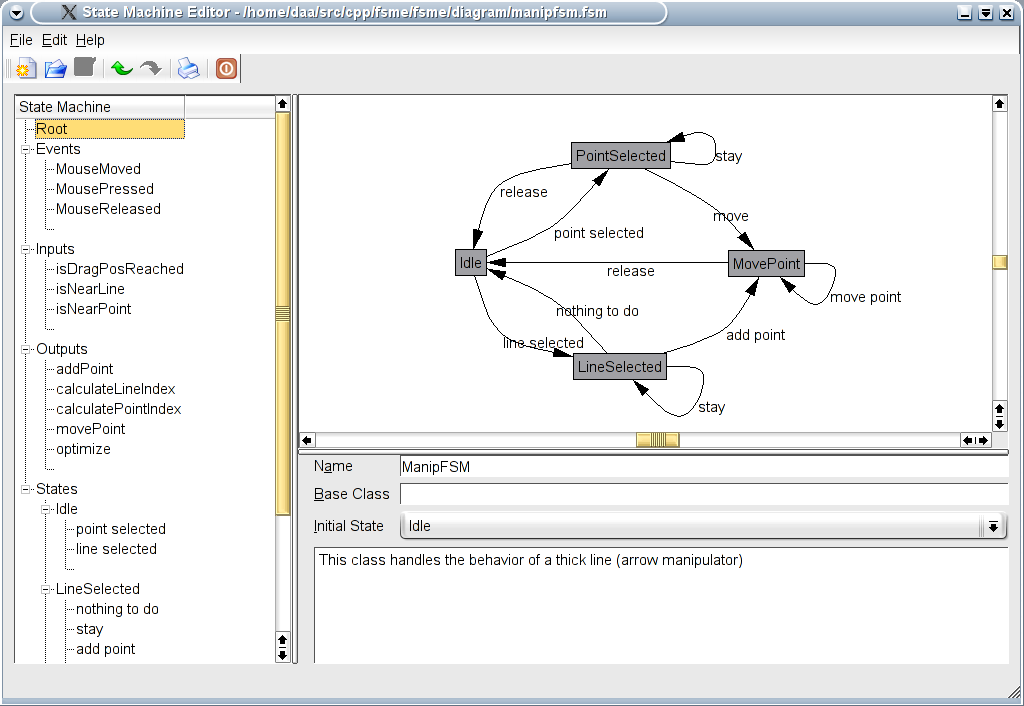
1. **Удобство интерфейса.** На какую целевую аудиторию нацелено данное приложение, какой оно требует квалификации, насколько оно накладывает дополнительные требования на скорость работы интерфейса.
2. **Универсальность.** Предназначено ли данное приложение для решения узкого круга задач или является общим для обширного рода задач на разных языках программирования.
3. **Возможность хранения и и повторного использования данных.** Загрузка и сохранение данных позволяет повторно использовать наработанные материалы.
4. **Поддержка.** Как часто проводятся обновления, существует ли руководство пользователя, является ли приложение платным/бесплатным.

В качестве примеров существующих приложений возьмём несколько примеров и сравним их по вышеуказанным характеристикам:

**Ragel State Machine Compiler.** 

Ragel компилирует выполняемые динамические модели из регулярных выражений. Ragel ориентирован на C, C++, Objective-C, C#, D, Java, Ruby, OCaml and Go. Ragel применяется для построений трудоёмких реализаций протоколов, парсинга данных, лексического анализа грамматик языков программирования, проверки ввода с клавиатуры.

**FSME**

****

FSME (Finite State Machine Editor) – графическое приложение для разработки конечных автоматов. Созданные динамические модели хранятся в файлах совместимых с Qt Designer IDE, а также могут быть конвертированы в код на C++.

**QM**

<http://www.state-machine.com/qm/index.php>

FSM Creator

<http://www.fsmcreator.de.gg/>

Stateflow

<http://www.mathworks.com/products/stateflow/>

Rational Rose RealTime

<ftp://ftp.software.ibm.com/software/rational/docs/documentation/manuals/rosert.html>

Enterprise Architect

<http://www.sparxsystems.com/support/faq/code_generation.html>

<http://www.sparxsystems.com/enterprise_architect_user_guide/9.2/software_engineering/code_generated_from_state_mach.html>

Visio2SWITCH

<http://www.softcraft.ru/auto/switch/v2s.shtml>

Visio2Switch - инструментальное средство [Visio2Switch](http://is.ifmo.ru/progeny/visio2switch/) позволяет по графу переходов, построенному в определенной нотации и изображенному с помощью редактора Visio, автоматически реализовать его в виде изоморфной программы на языке С.

MetaAuto - инструментальное средство [MetaAuto](http://is.ifmo.ru/projects/metaauto/) позволяет по графу переходов, построенному в той же нотации и изображенному с помощью того же редактора, автоматически реализовать его в виде изоморфной программы на любом языке программирования, для которого предварительно построен шаблон.

UniMod - инструментальное средство [UniMod](http://unimod.sourceforge.net/) предназначено для поддержки автоматного программирования и построения и реализации не только автоматов, но и программ в целом.