Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

(СПбГУ ИТМО)

УДК 004.4'242
№ госрегистрации
Инв. №

УТВЕРЖДАЮ
Ректор СПбГУ ИТМО
д.т.н., профессор

_____Васильев В.Н.

«_14_» _*октября*_ 2006 *г.*

ОТЧЕТ О ПАТЕНТНЫХ ИССЛЕДОВАНИЯХ
№ 2006.10.10-01 от 10 октября 2006 г.

по теме

ТЕХНОЛОГИЯ АВТОМАТНОГО ПРОГРАММИРОВАНИЯ: ПРИМЕНЕНИЕ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА
Шифр ИТ-13.4/004

Ректор СПбГУ ИТМО
д.т.н., профессор _____ _____ В.Н. Васильев
подпись дата

Руководитель темы
зав. каф. Технологий программирования,
д.т.н., профессор. _____ _____ А.А. Шалыто
подпись дата

Санкт-Петербург
2006

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель темы — А.А. Шалыто
Заведующий кафедрой «Технологий программирования»,
д.т.н., профессор

Исполнитель отчета: — И.М. Аничкин
Аспирант

Содержание

Введение

Патентный поиск проводился с целью проверки патентной чистоты полученных результатов опытно-конструкторской работы по теме «Технология автоматного программирования: применение и инструментальные средства», а также для получения сведений об охранных и иных документах, которые могут препятствовать применению результатов данной ОКР в Российской Федерации, и условиях использования таких документов.

Кроме того, требовалось определить рыночную стоимость интеллектуальной собственности, созданной в ходе проведения данной ОКР.

Согласно техническому заданию, создаваемая технология автоматного программирования должна включать следующее.

- Программный комплекс автоматного программирования (ПК АП) для проектирования и реализации сложных программных комплексов в составе:
  - ядро, обеспечивающее создание модели разрабатываемого программного обеспечения, состоящей из диаграмм переходов и схемы связей, интерпретацию модели, генерацию кода и документации по модели, проверку корректности модели, локальную и удаленную отладку модели, сохранение и загрузку модели;
  - встраиваемый модуль для среды разработки *Eclipse* версии 3.1, обеспечивающий визуальное редактирование диаграмм переходов и диаграмм связей, динамическое обнаружение ошибок в модели, запуск модели, визуальную локальную и удаленную отладку, сохранение и загрузку модели с графической информацией;
  - адаптеры, позволяющие запускать модели в различных средах, в том числе запуск модели из командной строки и запуск модели из servlet-контейнера;
  - укладчик диаграмм, обеспечивающий автоматизированную укладку диаграмм для встраиваемого модуля.
- Методику применения автоматного программирования для:
  - задач логического управления;
  - построения процедурных программ;
  - построения объектно-ориентированных программ.

При этом разрабатываемая технология должна обеспечивать следующее.

- Описание сложного поведения программ при создании программного обеспечения на основе схем связей и графов переходов.
- Интерпретацию схем связей и графов переходов с заданием входных функций и выходных воздействий на языке *Java* версии 5.0.
- Компиляцию схем связей и графов переходов в тексты программ на языках:
  - *Microsoft Visual C/C++* 6.0;
  - *GNU C/C++* 2.95.
  - *Java* 1.4.x и 5.0;
- Автоматизацию проверки правильности разрабатываемого программного обеспечения путем проверки:
  - достижимости состояний;
  - полноты и непротиворечивости графов переходов;
  - корректности потоков данных по диаграммам переходов.
- Автоматизацию процесса построения программного обеспечения и создания документации в части:
  - автоматической генерации кода по графам переходов;
  - автоматической проверки корректности графов переходов;
  - автоматической укладки графов переходов;
  - автоматизированного документирования на базе схем связей и графов переходов;
  - автоматического протоколирования во время исполнения программы, использующей разрабатываемый ПК АП.

Инструментальное средство (Программный комплекс автоматного программирования), созданное в результате выполнения темы, получило название *"UniMod"* (Unified Modeling) и состоит из двух частей: *"Java FSM Framework"* (Ядро автоматного программирования) и *"UniMod Eclipse Plug-in"* (Встраиваемый модуль автоматного программирования для среды разработки *Eclipse*). Для обеспечения патентной чистоты именно этого программного комплекса и проводился патентный поиск.

Патентный поиск (приложения А, Б, В) проводился с 10 сентября 2006 г. по 10 октября 2006 г.

1. Исследования патентов

1.1. Выбор поля для исследований

Продуктом, разрабатываемым в ходе данной опытно-конструкторской работы, является технология автоматного программирования, краткое описание которой приведено во введении. Темы, по которым проводился поиск, определялись исходя из требований, предъявляемых к данной работе техническим заданием. Такими темами стали:

- компиляция или интерпретация языков программирования высокого уровня;
- автоматизированное проектирование.

Соответственно этому определились и разделы международной патентной классификации, по которым был проведен поиск патентов. Эти разделы указаны в приложении Б.

Поиск патентной информации проводился в патентных базах данных Федеральной службы по интеллектуальной собственности, патентам и товарным знакам Российской Федерации (Роспатент, www.fips.ru), Бюро по патентам и товарным знакам США (USPTO, www.uspto.gov) и Европейского патентного бюро (EPO, ep.espacenet.com).

В Патентном законе Российской Федерации не признается патентоспособность научных теорий, математических методов, правил и методов интеллектуальной и хозяйственной деятельности, компьютерных программ, и решений, заключающихся только в представлении информации (патентный закон РФ, [1]). Однако, несмотря на это, приоритет международного законодательства позволяет осуществлять и в нашей стране патентование подобного рода «изобретений», и в последнее время в России появляется все больше такого рода таких фиктивных патентов, причем все они имеют иностранное происхождение.

1.2. Особенности патентной системы США

В данном разделе приводится пример патента, выданного патентным бюро США корпорации *IBM*. Факт выдачи данного патента отражают общую ситуацию, сложившуюся в настоящее время в американской патентной системе и получающую все большее распространение в других странах мира, по крайней мере, в отношении патентования изобретений в области программного обеспечения и связанных с ней областях.

1.2.1. Патент на конечные автоматы

Патент США № 5,317,757. «Система и метод для работы конечного автомата с использованием векторов действий». Владелец: *International Business Machines Corporation* (корпорация *IBM*) [2].

Краткое описание патента. *Общий набор стандартных модулей действий, которые производят определенные действия в конечном автомате и являются строго модульными по своей структуре. Набор этих модулей может быть построен из модулей для задач, характерных для типа ресурса и модулей, которые не зависят от типа ресурса. Для каждого типа ресурса создается конечный автомат для управления шагами активации и деактивации ресурса. Каждый автомат уникально определяет новое состояние и произведение действия для каждого типа*

ресурса. Для привязки стандартных модулей действий к каждому конечному автомату для каждого типа ресурсов создаются вектора действия. Вектор действия устанавливает отношение определенного действия, выделенного конечным автоматом, к диспетчеризации одного или более стандартных модулей действий. Вектор действий может включать множество элементов. Каждый из этих элементов идентифицирует модуль действий, которому передается управление, и указатель вызова функции. Указатель на функцию идентифицирует определенную функцию, которую должен выполнить указанный модуль действий. Стандартные модули действий вызываются в порядке следования элементов в векторе действий.

Патентное требование № 1. *Метод обработки данных для управления конечным автоматом в системе обработки данных, которая имеет память для хранения программных инструкций, входные устройства для получения входных данных, процессорный модуль для выполнения программных инструкций в ответ на указанные входные данные, а также выходные устройства для создания выходных данных в ответ на исполнение указанным процессорным модулем указанных программных инструкций. Включает следующие этапы:*

– хранение матрицы переходов первого конечного автомата в указанной памяти, которая имеет множество указателей на следующее состояние и указателей на вектора, к которым осуществляется доступ по значению текущего состояния в указанной памяти и значению входа;

– хранение таблицы векторов, состоящей из векторов действий, в указанной памяти, указанные указатели на вектора идентифицируют вектора действий в указанной таблице векторов, каждый из указанных векторов действий включает последовательность элементов векторов, имеющих указатели на модули и указатели на функции;

– хранение множества модулей действий в указанной памяти, каждый модуль действий включает в себя программные инструкции, организованные в множество сегментов функций, каждый из указанных указателей на модули идентифицирует один из модулей действий указанного множества и указанные указатели на функции идентифицируют одно из сегментов функций указанного множества;

– получение значения входа из указанного входного устройства и доступ к указанной матрице переходов первого конечного автомата с указанным значением входа и указанным значением текущего состояния для получения первого указателя на следующее состояние и первого указателя на вектор;

– доступ к первому вектору действий из указанной таблицы векторов и использованием указанного первого указателя на вектор, для получения первого указателя на модуль и первого указателя на функцию из первого элемента вектора в первой последовательности;

– исполнение программных инструкций в первом сегменте функций, на который указывает указанный первый указатель на функцию, в первом модуле действий, на который указывает указанный первый указатель на модуль, для произведения первого выходного действия указанным выходным устройством.

Сущность патента. Предлагается некий метод использования конечных автоматов для выполнения произвольных действий. Заявленные патентные требования настолько широки, что могут быть отнесены к использованию конечных автоматов практически в любых целях в программировании, сфере телекоммуникаций, и других областях.

Таким образом, использование конечных автоматов при написании программ, для реализации протокольных стеков, а также для иных целей может нарушать данный патент.

1.2.2. Последствия неправильной патентной политики

Недостаток приведенного выше патента, а также многих других патентов, выданных патентным бюро США (таких примеров достаточно много) состоит в том, что патентуются идеи или вполне очевидные, или уже давно и широко применяемые, или же совершенно абстрактные (как и почему такое допускается – это уже отдельный вопрос). Тем самым, вместо того чтобы развивать технологии, поддерживать инновации, как это задумывалось изначально, патентная

система начинает, напротив, препятствовать развитию науки, технологии, созданию и использованию новых идей (в частности, в программировании), заставляя людей бояться придумывать что-то новое и получать за счет этого прибыль из-за опасности судебных исков о нарушении патента.

Многие исследователи и специалисты в соответствующих областях (программисты, системные инженеры, и прочие) выражают свою озабоченность в связи со сложившейся ситуацией в сфере патентного права как в мире в целом, так и в США в частности [3, 4]. Предлагаемые пути выхода из сложившегося кризиса включают полный запрет на выдачу патентов в сфере программного обеспечения или радикальное сокращение сроков их действия.

Следует напомнить, что выдача патентов на программное обеспечение, научные теории и методы интеллектуальной и хозяйственной деятельности запрещена патентными законами России и стран Европы.

1.5. Результаты исследования патентов

Патентный поиск по разделам международной патентной классификации (МПК) «компиляция или интерпретация языков программирования высокого уровня» (*G06F009/45*) и «автоматизированное проектирование» (*G06F017/50*) дал отрицательный результат: в данных разделах МПК не было найдено патентов, аналогичных разрабатываемой в данной опытно-конструкторской работе технологии автоматного программирования.

Сказанное, однако, не означает, что в других разделах патентной классификации, не относящихся напрямую к той области, к которой относится разрабатываемый Программный комплекс автоматного программирования, не существует патентов, которые могут потенциально быть нарушенными в процессе эксплуатации этого комплекса. Пример патента *IBM* на конечные автоматы был приведен выше в разделе 1.2.3.

2. Исследования непатентных источников

2.1. Предпосылки к непатентным исследованиям

Согласно Патентному закону Российской Федерации [1], уровень техники, в сравнении с которым выявляется новизна изобретения, определяется не только зарегистрированными в России патентами (это важно в основном для патентной чистоты), но также имеющейся во всем мире общедоступной информацией.

Применительно к данному случаю, такой информацией могут являться сведения о свободно (а также не свободно, в данном случае это не имеет значения) распространяемых в сети Интернет программных продуктах, по своей функциональности аналогичных созданному Программному комплексу автоматного программирования. Таким образом, в силу требования технического задания о получении патентов на результаты интеллектуальной деятельности, созданной в рамках выполнения данной опытно-конструкторской работы, или свидетельств об официальной регистрации программ для ЭВМ, возникает необходимость исследовать аналоги создаваемого программного комплекса и сравнить его с этими аналогами.

Таким образом, данное исследование будет решать следующие задачи.

1. Анализ имеющихся в мире инструментов (средств разработки), в которых для создания программ применяется технология автоматного программирования.
2. Определение критериев, по которым можно сравнивать эти инструменты.
3. Отбор тех инструментов, которые наилучшим образом подходят для создания программного обеспечения, и их последующее сравнение.

Кроме того, сравнение Программного комплекса автоматного программирования с другими аналогичными продуктами, которые можно приобрести на рынке программного обеспечения, необходимо также и для оценки рыночной стоимости этого комплекса.

2.2. Источники информации

В качестве основного источника информации по тематике «автоматное программирование» была выбрана энциклопедия свободного доступа *WikiPedia* (http://www.wikipedia.org/), и конкретно статья этой энциклопедии про конечные автоматы (*Finite state machine*). Эта статья важна тем, что в ней приводится список инструментов (раздел «Tools»), которые либо могут быть использованы для работы с конечными автоматами (КА), либо сами используют конечно-автоматную технологию. Кроме основного были использованы и другие источники информации – различные информационные ресурсы, найденные при помощи поисковых систем, а также ссылки, подсказанные коллегами. Информация, полученная из альтернативных источников, по большей части пересекалась с той, которая имеется в энциклопедии *WikiPedia*. Это говорит о том, что основной источник информации обладает достаточной полнотой, пусть и не абсолютной.

2.3. Общий обзор источников

Вот полный список инструментов, приводимый в энциклопедии *WikiPedia*, по состоянию на 10 октября 2006 года. Каждый пункт этого списка содержит ссылку на сайт, посвященный соответствующей разработке или инструменту.

(Ссылки доступны со страницы http://en.wikipedia.org/wiki/Finite_state_machine#Tools)

- *AsmL* [1]
- *AT&T FSM Library* [2]
- *AutoFSM* [3]
- *Automata* [4]
- *Bandera* [5]
- *Boost Statechart Library* [6]
- *CAZE - FSM-based authorization lib* [7]
- *Cellogica* [8]
- *Concurrent Hierarchical State Machine* [9]
- *Covered* [10]
- *DescoGUI* [11]
- *dRegAut* [12]
- *Dynamic Attachment FSM* [13]
- *Exorciser* [14]
- *Finite State Kernel Creator* [15]
- *Finite State Machine Editor* [16]
- *Finite State Machine Explorer* [17]
- *FIRE Engine, Station and Works* [18]
- *FSA Utilities* [19]
- *FSMGenerator* [20]
- *Grail+* [21]
- *Java Finite Automata* [22]
- *jFAST* [23]
- *JFLAP* [24]
- *jrexx-Lab* [25]
- *JSpasm* [26]
- *Kara* [27]
- *Libero* [28]
- *MetaAuto* [29]
- *MIT Finite-State Transducer Toolkit* [30]
- *Nunni FSM Generator* [31]
- *Petrify* [32]
- *PyFSA* [33]
- *Qfsm* [34]
- *Quantum-Leaps* [35]
- *Ragel* [36]
- *RWTH FSA Toolkit* [37]
- *State Chart XML* [38]
- *SFST Tools* [39]
- *SMC* [40]
- [*state*] *method* [41]
- *Stateflow* [42]
- *Statestep* [43]
- *StateWORKS* [44]
- *Supremica* [45]
- *Vaucanson* [47]
- *Visio2Switch* [48]
- *visualSTATE* [49]
- *WhatOS* [50]
- *Xerox Finite-State Software Tools* [51]
- *XML Transition Network Definition* [52]

Помимо этого списка была найдена ссылка на продукт *Statemate* фирмы *I-Logix* (теперь эта фирма объединилась с фирмой *Telelogic*, для тестирования доступен продукт этой компании, называемый *Rhapsody*), а также коллегами было предложено протестировать продукт *EclipseUML* фирмы *Omondo*. Существуют и другие программные продукты, позволяющие строить программы

с применением технологии конечных автоматов, например, продукт *Together* фирмы *Borland*, пакет *Rose Developer* фирмы *IBM-Rational* и другие инструменты. Однако, большинство из них поставляется на заказ и недоступно для свободного тестирования.

Каждый из приведенных выше сайтов был просмотрен, и был проведен анализ того, что за инструмент или разработка предлагаются авторами. По результатам анализа был составлен краткий обзор всех перечисленных разработок, включающий цитаты со страниц сайта, переведенные на русский язык, и небольшое заключение, отражающее общее впечатление о каждой разработке. В данном отчете полностью приводятся лишь обзоры некоторых наиболее достойных инструментов. В табл. 1 приведен общий результат предварительного анализа – список всех представленных разработок с указанием того, что представляет собой каждая из них.

Таблица 1. Краткое описание разработок

Обозначение	**Предназначение, краткое описание, заключение**
01_AsmL	Проект описания языка автоматного программирования, созданный исследовательским отделом *Microsoft*. *Разработка иного назначения.*
02_ATT_FSMlib	Инструмент для построения распознавателей и преобразователей. Предназначен для языкового моделирования и систем распознавания речи. *Не предназначен для создания ПО общего вида.*
03_AutoFSM	Дополнительный пакет к инструменту для автоматической генерации кода. Консольное *Linux*-приложение. Графического редактора нет. Есть генерация кода по автоматам. При генерации кода используется таблица переходов. При помощи шаблона из файла описания автомата генерирует код на языке *C*. **Инструмент для программирования с применением КА.**
04_Automata	Свободно распространяемый пакет для *Mathematica*. Позволяет работать с КА, преобразовывая их тем или иным образом. *Не предназначен для создания ПО общего вида.*
05_Bandera	Инструменты *Bandera* предназначены для извлечения конечно-автоматной модели из исходного кода на языке *Java* с целью последующей проверки этих моделей на корректность. *Инструмент несколько иного предназначения – не для создания ПО.* *Кроме того, инструмент невозможно скачать и посмотреть в работе.*
06_BoostSCLib	Каркас (framework), предназначенный для создания КА в виде кода на *C++* по диаграммам состояний *UML*. Напоминает *Quantum Leaps*. Код требуется набирать вручную – имеется только описание подхода. *Инструмент специального предназначения.* **Инструмент для программирования с применением КА.**
07_CAZE	Движок (библиотека классов) для реализации логики авторизации для бизнес-приложений. *Разработка иного предназначения.*
08_Cellogica	*Содержание проекта отсутствует. Нечего смотреть.*
09_CHSM	Компилятор автоматов. Консольное *Linux*-приложение. Позволяет компилировать автоматы, задаваемые на языке описания, подобном языку описания грамматик *Yacc*. Графического редактора нет, визуализатора нет. **Инструмент для программирования с применением КА.** *Сам этот инструмент не компилируется.*
10_covered	Инструмент для анализа покрытия кода. Позволяет определять, насколько хорошо диагностический тестовый набор покрывает тестируемый проект. *Разработка иного предназначения.*

Продолжение табл. 1. Краткое описание разработок

Обозначение	**Предназначение, краткое описание, заключение**
11_DescoGUI	**Графическая** оболочка (инструмент) для изображения автоматных графов и сетей Петри. Инструмент работает, позволяет рисовать автоматы, однако не позволяет использовать их для создания программного обеспечения. *Инструмент специального предназначения.*
12_dRegAut	Java-пакет для реализации регулярных выражений и автоматных алгоритмов. *Не предназначен для создания ПО общего вида.*
13_DAFSM	*Предназначение этой разработки непонятно.*
14_Exorciser	По-видимому, это инструмент для построения автоматов по регулярным выражениям, и имеет частное применение – языковой анализ. *Не предназначен для создания ПО общего вида.* *Нет английского варианта.*
15_FSKC	Редактор таблицы переходов автомата. **Графическое** *Linux*-приложение. Редактирование переходов в виде таблицы переходов. Возможность генерации кода на *C*. Возможность интерактивной симуляции (реализовано убого). **Инструмент для программирования с применением КА.**
16_FSMEditor	Редактор графов переходов автомата. **Графическое** *Linux*-приложение. Позволяет редактировать автоматные графы. Прилагается внешняя утилита командной строки для генерации кода на языке *C* или *Python*. **Инструмент для программирования с применением КА.**
17_FSMExplorer	**Графический** редактор графов переходов автомата. Инструмент представляет собой *Java*-апплет. Тематика – распознаватели и преобразователи. Чей-то курсовой проект. *Не предназначен для создания ПО общего вида.*
18_FIRE	Инструмент для построения распознавателей и преобразователей. Предназначен для работы с текстом, индексирования и т.д. *Не предназначен для создания ПО общего вида.*
19_FSAUtilities	Инструмент строит автоматы по регулярным выражениям. Позволяет производить над ними различные операции. Написана на *Prolog* под *Linux*. Есть web-demo. *Не предназначен для создания ПО общего вида.*
20_FSMGenerator	Компилятор автоматов. Консольное приложение. Версии для *Windows* и *Linux*. Генерация кода. Автомат задается в виде текстового файла. Графического редактора нет. **Инструмент для программирования с применением КА.**
21_Grail+	Среда символьных вычислений для конечных автоматов и регулярных выражений, позволяет преобразовывать их из одной формы в другую, минимизировать, детерминировать и т.д. *Не предназначен для создания ПО общего вида.*
22_JavaFA	Ссылка на некую документацию, похожую на документацию библиотек *Java*. *Программы нет, смотреть нечего.*
23_jFAST	Ссылка на статью о том, как сделать домашнюю страничку при помощи *FrontPage* на сайте университета *Villanova*. *Ничего нет, смотреть нечего.*
24_JFLAP	Этот **графический** инструмент представляет собой «помощь при изучении основ теории формальных грамматик и автоматов» *Не предназначен для создания ПО общего вида.*
25_jrexxLab	Этот **графический** инструмент представляет собой «продвинутый визуализатор и графический редактор для регулярных выражений» *Не предназначен для создания ПО общего вида.*

Продолжение табл. 1. Краткое описание разработок

Обозначение	**Предназначение, краткое описание, заключение**
26_JSpasm	*JSpasm* представляет собой не редактор автоматов, а набор классов на *Java* (framework), при помощи которого можно создавать автоматы, определяя состояния, события и переходы, а затем запускать их. Приводятся примеры. **Инструмент для программирования с применением КА.**
27_Kara	Программа – эмулятор поведения божьей коровки. *Разработка иного предназначения.* *Только по-шведски, нет английского варианта.*
28_Libero	Графическое GUI-приложение, автоматы редактируются не в виде графа, а в виде отдельных переходов. Проект прекратил развитие в 1999 году. **Инструмент для программирования с применением КА.**
29_MetaAuto	Инструментальное средство, позволяет по графам переходов, построенным с помощью редактора *MS Visio*, автоматически генерировать изоморфные им исходные коды программ на разных языках программирования, для каждого из которых предварительно должен быть создан шаблон. **Инструмент для программирования с применением КА.**
30_MIT_FSTT	Документация на тему "набор конечно-автоматных преобразователей для обработки речи и языка". *Не предназначен для создания ПО общего вида.*
31_Nunni	Компилятор автоматов. Позволяет по описанию таблицы переходов автомата генерировать код на *C*, *C++* или *Java*, реализующий этот автомат. Таблица переходов должна быть полной (например, 9 состояний * 5 событий = 45 переходов должно быть определено). **Инструмент для программирования с применением КА.**
32_Petrify	Инструмент для синтеза сетей Петри и асинхронных схем. *Разработка иного предназначения.*
33_PyFSA	Инструмент для манипулирования с конечными автоматами: минимизация, детерминирование, регулярные выражения, распознаватели. *Не предназначен для создания ПО общего вида.*
34_Qfsm	Инструмент для синтаксического анализа. **Графическое** приложение для *Linux*. Позволяет редактировать графы переходов автоматов. Позволяет проводить интерактивную симуляцию. Генерации кода нет. Условия перехода – регулярные выражения. Предназначен для создания сканеров, а не ПО общего вида. Интерфейс удобный. *Не предназначен для создания ПО общего вида.*
35_QuantumLeaps	*Quantum Platform* представляет собой не редактор автоматов, а некий движок, при помощи которого можно создавать программы для встроенных систем. Кроме того, подробно и качественно излагается методология реализации иерархических КА на языках *C* и *C++*. *Инструмент специального предназначения.* **Инструмент для программирования с применением КА.**
36_Ragel	Консольное приложение. Распознаватель текста по регулярным выражениям. Редактора нет. Визуализатора нет. По описанию автомата в стиле *Yacc* генерирует код на *C++*. *Не предназначен для создания ПО общего вида.*
37_RWTH	Инструментарий для языковых задач (распознавания речи, переводов и т.д.). Кроме того, плохо документирован, не разобраться. *Не предназначен для создания ПО общего вида.*

Продолжение табл. 1. Краткое описание разработок

Обозначение	**Предназначение, краткое описание, заключение**
38_SCXML	*StateChart XML*. Приводится нотация языка описания автоматов. *Разработка иного предназначения.* Однако это может быть интересно, как стандарт, если его примут в *W3C* в качестве стандарта.
39_SFST	Инструмент для морфологического анализа языка. *Не предназначен для создания ПО общего вида.*
40_SMC	Компилятор автоматов. Консольное *Java*-приложение. Компилирует автомат, задаваемый в виде текстового файла, в код на языке *Java*, *C++* или других языках. Графического редактора нет. Визуализатора нет. **Инструмент для программирования с применением КА.**
41_[state]method	**Графический** инструмент, представляет собой надстройку над *Visual Studio .NET*, которая предназначена для создания ПО при помощи конечных автоматов. Позволяет изображать графы переходов. Точное назначение непонятно, возможно, предназначен для построения кода программ. *Коммерческое ПО.* *Скачать можно только на немецком. Не тестировался.*
42_Stateflow	Довольно серьезный инструмент фирмы *MathWorks*. Поддерживает все, что нужно – **графическое** представление, компиляция, отладка. Реализован как часть пакета *MatLab*. *Коммерческое ПО. Не тестировался.* **Инструмент для программирования с применением КА.**
43_Statestep	**Графическое** приложение. Написан на *Java*. Имеется документация. Состояние системы описывается набором переменных. Редактирование переходов не в виде графа переходов, а в виде довольно хитрой таблицы. Отдельная утилита для генерации кода на *C++* или *Java*. **Инструмент для программирования с применением КА.**
44_stateworks	**Графическое** *Windows*-приложение. Редактор автоматов в виде графа переходов. Предоставляется временная лицензия на 30 дней. По описанию, все делается способом, похожим на *Switch*-технологию. *Коммерческое ПО. Временную лицензию получить не удалось.* **Инструмент для программирования с применением КА.**
45_Supremica	Инструмент для создания ПО для встроенных систем, для программирования контроллеров. Данный инструмент позволяет редактировать автоматные графы переходов. Не тестировался. *Инструмент специального предназначения.*
46_Vaucanson	Хотя написано по-английски, совершенно не ясен смысл описания. Писали французы. На серьезный инструмент не похож. *Документации нет. Ничего не понятно.*
48_Visio2Switch	Инструментальное средство, которое позволяет по графам переходов, построенным с помощью редактора *MS Visio*, автоматически генерировать изоморфные им исходные коды программ на языке *C* в соответствии со *Switch*-технологией. **Инструмент для программирования с применением КА.**
49_visualSTATE	**Графическое** *Windows*-приложение. Коммерческое. Данная программа была взломана хакерами. Это говорит о том, что данный инструмент, вероятно, довольно широко применяется. Генерация кода на *C*, отладка. *Коммерческое ПО.* **Инструмент для программирования с применением КА.**

Продолжение табл. 1. Краткое описание разработок

Обозначение	**Предназначение, краткое описание, заключение**
50_WhatOS	Программа для узкой области применения (встроенные системы), Графического интерфейса нет, генерации кода нет, автоматы пишутся прямо в виде исходного кода на языке *Python*. Это даже не приложение, а некий framework – скелет, предназначенный для написания встроенных приложений для микропроцессоров. *Инструмент специального предназначения.*
51_Xerox	В лаборатории *Xerox* исследуют естественные языки, создаваемые там инструменты предназначены именно для этого (распознавание, преобразование, морфологический анализ). Инструменты недоступны для скачивания, есть только их описание. *Не предназначены для создания ПО общего вида.* *Кроме того, инструмент невозможно скачать и посмотреть в работе.*
52_XTND	Это проект документа *W3C*. Данный документ определяет формат (язык) взаимообмена для Интернет, описанный на *XML*. *Разработка иного предназначения.*
StateMate	Графическое *Windows*-приложение. Коммерческое. Скачать не удается. В обзоре представлен продукт *Rhapsody* той же фирмы. *Коммерческое ПО.* **Инструмент для программирования с применением КА.**
Rhapsody	Очень серьезный продукт. Удалось получить лицензию на 30 дней. Позволяет строить диаграммы состояний, по которым можно генерировать код на различных языках, в частности, на *Java*. Возможности этого инструмента весьма обширны, имеется возможность строить и другие *UML* диаграммы, а не только диаграммы состояний, однако непосредственно к созданию работающей программы (генерации кода) имеют отношение в основном диаграммы классов и диаграммы состояний, а также диаграммы деятельностей. Имеется возможность отладки ПО (модели) в графическом режиме (анимация диаграмм состояний). *Коммерческое ПО.* **Инструмент для программирования с применением КА.**
EclipseUML	Плагин для *Eclipse* фирмы *Omondo*. Помимо прочего, позволяет создавать *UML* диаграммы состояний, однако эти диаграммы никак не привязаны к остальной части программы, а являются только картинкой. Генерации кода по диаграммам состояний нет. Документация на более чем 500 страниц не содержит почти никакой существенной информации. *Коммерческое ПО.* *Разработка иного предназначения.*

2.4. Выбор критериев сравнения

Из изложенного в разд. 2.3 следует, что среди перечисленных в табл. 1 разработок встречаются как инструменты, предназначенные для работы с конечными автоматами, так и приложения или иные проекты, в которых лишь применяется конечно-автоматная технология. Сразу отметим, что примеры использования технологии автоматного программирования в конкретных проектах не представляют интереса, и потому не будут являться объектом данного исследования.

Инструменты для программирования с применением конечных автоматов, перечисленные в данной таблице, имеют различные возможности и обладают различными свойствами. Например, среди них есть такие, которые имеют графический интерфейс, и такие, которые работают из

командной строки. Функциональность некоторых инструментов ограничена возможностью создания распознавателей текстовых строк, другие же применимы для создания программного обеспечения общего вида. Инструменты, предназначенные для создания распознавателей, и другие средства языкового анализа также не представляют интереса для данного исследования, так как они имеют довольно узкую область применения.

Основными объектами рассмотрения будут те инструменты, которые применимы для создания программного обеспечения общего вида с использованием конечных автоматов. Критерием сравнения этих инструментов между собой может быть наличие и отсутствие тех или иных свойств у каждого инструмента.

После анализа содержимого тех сайтов, на которых представлены перечисленные инструменты для программирования с применением конечных автоматов, и первичного опробования в работе доступных для скачивания инструментов, был определен набор критериев, по которым можно проводить сравнение инструментов между собой. Вот эти критерии.

1. **Применение**. Возможности применения инструмента: для создания программного обеспечения общего вида; только для построения языковых инструментов; либо для иных целей. Многие из инструментов ориентированы на создание только приложений для встроенных систем.
2. **Операционная система**. Та операционная система, для которой написан конкретный инструмент – обычно это *Windows*, *Unix* (*Linux*), или виртуальная машина *Java*.
3. **Исходные коды** – наличие открытых исходных кодов инструмента, и то, на каком языке они написаны.
4. **Интерфейс** – графический или интерфейс командной строки.
5. **Графическое представление**. Способ графического представления автомата (применимо для инструментов с графическим интерфейсом) – в виде графа переходов, таблицы переходов или другим способом.
6. **Формат файла** – в каком виде сохраняется автомат на диске – обычно это простой текстовый файл, двоичный или *XML*. Некоторые инструменты требуют описания автомата в виде кода на языке программирования.
7. **Метод генерации кода**.
8. **Язык генерации кода**. Если инструмент позволяет генерировать по автомату исходный код на языке программирования, то указывается, на каком языке и в каком виде это делается – по *Switch*-технологии (используется конструкция `switch` языка *C* или ее аналоги в других языках), с использованием таблицы переходов (функции вызываются по адресам, хранящимся в таблице) или еще как-то. Также указывается, в каком виде представлены автоматы, состояния, обработчики событий – в виде классов, методов, и т.д.
9. **Отладка, интерпретация**. Возможность производить исполнение (отладку) построенного автомата средствами, предоставляемыми самим инструментом.
10. **Поиск ошибок, верификация**. Возможность верификации создаваемой при помощи данного инструмента программы – проверка полноты и непротиворечивости графов переходов и т.д.
11. **Удобство интерфейса**. Субъективная количественная оценка по 10-балльной шкале.
12. **Ясность документации**. Субъективная количественная оценка по 10-балльной шкале.

2.5. Сравнение инструментов

В табл. 2 приводится список инструментов с их оценкой по перечисленным критериям. Для полноты картины здесь приведены все перечисленные ранее разработки, а не только те инструменты, которые предназначены для создания программ общего вида.

Таблица 2. Оценка инструментов по выбранным критериям

	Применение [1]	Операционная система	Исходные коды	Интерфейс [2]	Графическое Представление [3]	Формат файла [4]	Метод генерации кода	Язык генерации кода	Отладка, интерпретация	Поиск ошибок, верификация	Удобство интерфейса	Ясность докуметации	Оценка
01_AsmL	O												
02_ATT_FSMlib	G	*Unix*	Нет	Cmd	-	Txt	Нет	Нет	Нет		-	9	
03_AutoFSM	All	*Unix*	*C*	Cmd	-	Txt	Вызов внешней функции для определения следующего состояния	*C*	Нет	Нет	-	6	
04_Automata	O												
05_Bandera	O												
06_BoostSCLib	All	*Mathematica*	*C++*	Cmd	-	Code	Состояние – адрес функции Событие – выбор по `switch`. Код пишется вручную.	*C++*	Нет	Нет	-	8	
07_CAZE	O												
08_Cellogica	?												
09_CHSM	All	*Unix*	*C*	Cmd	-	*Yacc*	Непонятно	*C++*, *Java*	Нет	Нет	-	9	
10_covered	O												
11_DescoGUI	O	*Windows*	Нет	Gra	Gra	Bin	Оболочка для *Supremica*	нет	Нет		4	4	
12_dRegAut	O												
13_DAFSM	?												
14_Exorciser	?												
15_FSKC	All	*Unix*	*C*	Gra	Tabl	Bin	Таблица переходов	*C*	+	Нет	3	9	
16_FSMEditor	All	*Unix*	*C++*	Gra	Gra	*Xml*	*Switch*-технология	*C++*, *Python*	+	Нет	8	8	
17_FSMExplorer	G	*Java*	*Java*	Gra	Gra	Txt	Нет	нет	+		7	7	
18_FIRE	G,O												
19_FSAUtilities	G	*Unix*	*Prolog*	Cmd	-	Txt	Непонятно	*C*, *C++*, *Java*, *Prolog*	Нет		-	8	
20_FSMGenerator	All	*Win*/*Unix*	*C++*	Cmd	-	Txt	Аналог *Switch*-технологии	*C*, *C++*, *Java*	Нет	Нет	-	8	

Продолжение табл. 2. Оценка инструментов по выбранным критериям

	Применение [1]	Операционная система	Исходные коды	Интерфейс [2]	Графическое Представление [3]	Формат файла [4]	Метод генерации кода	Язык генерации кода	Отладка, интерпретация	Поиск ошибок, верификация	Удобство интерфейса	Ясность докуметации	Оценка
21_Grail+	G	*DOS*/*Unix*	*C++*	Cmd	-			*C++*	Нет		-	7	
22_JavaFA	-												
23_jFAST	-												
24_JFLAP	G	*Java*	Нет	Gra	Gra	*Xml*	Нет	Нет	+		9	9	
25_jrexxLab	G	*Java*	*Java*	Gra	Gra	Bin	Нет	Нет	+		9	6	
26_JSpasm	All	*Java*	*Java*	Cmd	-	FW	Нет	Нет	Нет	Нет	-	6 jd	
27_Kara	?												
28_Libero	All	*Win*/*Unix*/...	*C*/*C++*	Gra	Tabl	Txt	Таблица переходов	*C*, *C++*, *Java*...	Нет	Нет	3	8	
29_MetaAuto	All	*Win*, *Visio*	*C#*	-	-	Bin	*Switch*-технология	*C#*, *C++* и др.	Нет	Нет	5	7	
30_MIT_FSTT	O(-)												
31_Nunni	All	*Java*	*Java*	Cmd	-	Txt	Вариант таблицы переходов	*C*, *C++*, *Java*	Нет	Нет	-	8	
32_Petrify	O												
33_PyFSA	O	*Unix*	*Python*	Cmd									
34_Qfsm	G	*Unix*	*C++*	Gra	Gra	*Xml*	Нет	Нет	+		9	0	
35_QuantumLeaps	All	*C* compiler	нет	Cmd	-	Code	Состояние – адрес функции Событие – выбор по `switch`. Код пишется вручную.	*C++*	Нет	Нет	-	9	
36_Ragel	G	*Unix*/*Win*	*C++*	Cmd	-	*Yacc*	Непонятно	*C*, *C++*	Нет		-	7	
37_RWTH	G	*Unix*	*C++*									2	
38_SCXML	O												
39_SFST	G												
40_SMC	All	*Java*	*Java*	Cmd	-	Txt	Аналогично *QuantumLeaps*	*C*, *C++*, *Java*, *C#*, *Python*, *Tcl*, *VB*	Нет	Нет	-	6	
41_[state]method	All	*Win*, *MSVS*	Нет	Gra	Gra	Txt	?	?	?	?	-	-	

Продолжение табл. 2. Оценка инструментов по выбранным критериям

	Применение [1]	Операционная система	Исходные коды	Интерфейс [2]	Графическое Представление [3]	Формат файла [4]	Метод генерации кода	Язык генерации кода	Отладка, интерпретация	Поиск ошибок, верификация	Удобство интерфейса	Ясность докуметации	Оценка
42_Stateflow	All	*MatLab*	Нет	Gra	Gra	Txt	Есть, но не исследовалось	*C*	+	?	9	10	
43_Statestep	All	*Java*	Нет	Gra	Tabl	*Xml*	Таблица переходов	*C*, *C++*, *Java*	Нет	+	5	8	
44_stateworks	All	*Windows*	Нет	Gra	Gra	?	Не тестировалось	?	+	?	?	?	
45_Supremica	O												
46_Vaucanson	O?												
48_Visio2Switch	All	*Win*, *Visio*	Нет	-	-	Bin	*Switch*-технология	*C++*	Нет	Нет	5	9	
49_visualSTATE	All	*Windows*	Нет	Gra	Gra	Txt	Непонятно, вероятно, таблица вызова.	*C*	+	Нет	10	10	
50_WhatOS	O	*Unix*	*Python*	Cmd	-	Code	-	-	Нет		-	4	
51_Xerox	G												
52_XTND	O												
Statemate	All	*Windows*	Нет	Gra	Gra	?	?	?	?	?	?	?	
Rhapsody	All	*Windows*	Нет	Gra	Gra	*Xml*	С использованием библиотеки функций.	*C*, *C++*, *Java*...	+	+	9	9	
EclipseUML	O	*Eclipse*	Нет	Gra	Gra	*Xml*	Нет	Нет	Нет	Нет	5	7	

Примечания к табл. 2.

1. All – применим для построения программ общего вида.
   G – применим для построения распознавателей, преобразователей, и прочих языковых инструментов.
   O – другое применение.
2. Cmd – интерфейс командной строки.
   Gra – графический интерфейс.
3. Tabl – таблица переходов.
   Gra – граф переходов.
4. Txt – текстовый файл произвольного формата.
   Xml – текстовый файл формата Xml.
   Yacc – текстовый файл, подобный описанию грамматики для Yacc.
   Bin – двоичный файл.
   FW – автомат описывается с использованием библиотек framework'а.
   Code – автомат кодируется вручную на обычном языке программирования.

Замечание к приведенной таблице. Жирным шрифтом в таблице выделены инструменты, которые будут исследоваться и тестироваться более подробно. Все оценки, даваемые в этой таблице и далее, в обзорах каждого инструмента, являются до некоторой степени субъективными, и основаны на мнении авторов данного исследования.

Как следует из табл. 2, среди всех перечисленных разработок насчитывается всего 19 инструментов для программирования с применением конечных автоматов, предназначенных для создания программного обеспечения общего вида. Из них только девять имеют графический интерфейс, остальные 10 работают из командной строки.

В табл. 3 приведен список упомянутых десяти инструментов с графическим интерфейсом.

Таблица 3. Инструменты с графическим интерфейсом

Название	**Полное название**
FSKC	*Finite State Kernel Creator*
FSM Editor	*Finite State Machine Editor*
Libero	*iMatrix Libero*
Stateflow	*Mathworks Stateflow*
Statestep	*Statestep*
Stateworks	*StateWORKS*
VisualState	*IAR VisualState*
Statemate	*I-Logix Statemate*
Rhapsody	*I-Logix Rhapsody*

В следующем разделе приводится краткий обзор каждого из этих инструментов.

2.6. Обзоры избранных инструментов

2.6.1. *FSKC*

2.6.1.1. Выдержки со страницы на сайте производителя

«Что делает этот инструмент?

Создатель ядер конечных автоматов (*Finite State Kernel Creator*, *FSKC*) – это инструмент компьютеризированной разработки приложений (CASE-инструмент), который позволяет:

1. Графически проектировать КА.
2. Легко добавлять состояния и события.
3. Интерактивно выполнять ранее созданные КА.
4. Распечатывать таблицу переходов КА.
5. Распечатывать описание состояний и событий.
6. Генерировать исходный код, реализующий ядро КА.
7. Включать в программу круговой буфер настраиваемой длины, сохраняющий предыдущие переходы для отладочных целей.
8. Находить бесконечные состояния и предупреждать о наличии таковых.
9. Определять безопасные действия для неопределенных комбинаций «состояние-событие».

CASE-средство для создания КА позволяет разработчикам указывать все действительные и недействительные комбинации «состояние-событие» программного обеспечения посредством простого графического интерфейса пользователя (GUI). Полная таблица состояний может быть в любой момент распечатана в форме *ASCII* для обзоров проекта или презентаций. Одним щелчком мыши CASE-средство автоматически создаст исходный код, реализующий ядро конечных автоматов. Функции, вызываемые ядром КА, могут быть написаны в виде программного кода и собраны вместе с ядром КА, образуя конечный автомат. Новые состояния и события могут быть добавлены или убраны в несколько раз быстрее, чем это делалось ранее вручную. Инженеры могут интерактивно запускать автомат в пошаговом режиме для проверки его корректности и полноты».

2.6.1.2. Обзор инструмента

Программа представляет собой графическое приложение для *Unix*. У инструмента *FSKC* есть довольно приличная документация. И хотя в ней приводятся графы переходов автоматов, они, по-видимому, получены не в самой программе, а сделаны во внешнем графическом редакторе исключительно как иллюстрация к документации. Сама программа позволяет редактировать автомат в виде таблицы переходов, что не особенно удобно, в случае, когда состояний и событий много. Невозможно задавать дополнительные условия переходов. Почему авторы называют такой способ задания переходов «графическим проектированием», не совсем понятно. Сгенерированный код ядра автомата – «состояние * событие = новое состояние» – в виде таблицы переходов.

2.6.1.3. Заключение

Редактор довольно простой, если не сказать примитивный. Переходы реализованы в виде таблицы переходов. Генерация кода на языке *C*. Отладка реализована неудовлетворительным образом.

Оценка: 2/10.

2.6.2. *FSM Editor*

2.6.2.1. Выдержки со страницы на сайте производителя

«Написание хорошей программы – это непростой процесс. Поведение программы трудно предсказуемо, таким образом, для получения большего контроля над исполнением программы рекомендуется использовать инструменты, позволяющие делать это графически. Мой шеф получил кое-какую информацию про конечные автоматы и посоветовал мне прочитать книгу А.А. Шалыто «Swicth-технология». В ней я увидел путь использования конечных автоматов для написания программ общего вида. Я написал первую версию редактора конечных автоматов для системы *Flora*. Он был ужасный. Он был глючный. Но создаваемые при помощи него программы работали стабильно. Потом я переписал редактор, используя автоматную технологию. Потом я начал проект с открытым исходным кодом. Он использует похожие принципы, но написан на другом языке и работает в другом окружении, чем его коммерческий аналог.

Перед вами первый выпуск *FSME*. Еще нужно бы реализовать множество идей, но и сейчас его вполне можно использовать.

Некоторые части *FSME* написаны при помощи него самого.

К версии 1.0 *FSME* вырос в три ветки:

- *FSME — Finite State Machine Editor* (редактор конечных автоматов) Используется для проектирования конечных автоматов. Описание КА сохраняется в *XML*-файле, который не зависит от языка и платформы.
- *FSMC — Finite State Machine Compiler* (компилятор КА). Используется для получения исходных кодов из XML описания. На данный момент существуют генераторы для языков *C++* и *Python*.
- *FSMD — Finite State Machine Debugger/Tracer* (отладчик/трассировщик КА). Используется для мониторинга КА в реальном времени. Просто перенаправьте поток вывода программы в *FSMD*, и наблюдайте все, что происходит, в графическом виде».

2.6.2.2. Обзор инструмента

Программа написана российским программистом.

Редактор представляет собой графическое приложение для *Unix*, все сделано красиво, графический интерфейс, переходы изображаются в виде графа переходов. В самом начале

«документации» приведена ссылка на книгу А.А. Шалыто «Switch-технология». Из этого можно догадаться, что программа написана в соответствии с рекомендациями автора книги.

Генерация кода на языке *C++* в соответствии со *Switch*-технологией. Также имеется отладчик (трассировщик) автоматов. Документация снабжена иллюстрациями и содержит пример написания программы с использованием редактора.

2.6.2.3. Заключение

Редактор довольно толковый, редактирует граф перехода автомата, генерация кода – по *Switch*-технологии. Имеется графический отладчик автоматов. Документация построена на разборе примера. Хотя это и частный проект одного программиста, выглядит он вполне достойно.

Оценка: 6/10.

2.6.3. *Libero*

2.6.3.1. Выдержки со страницы на сайте производителя

«Вы программист? Вы, правда, хотите писать программы лучше? Тогда взгляните на *Libero*, свободно распространяемый программный инструмент от *iMatrix*.

Как использовать *Libero*?

1. Спроектируйте свою программу визуально в виде диаграммы состояний.
2. Выберите свой язык программирования.
3. Сгенерируйте каркас для своей программы.
4. Заполните каркас, перейдите от быстро сделанного прототипа к работающей программе.
5. Повторите шаги 1-4, пока программа не станет идеальной.

Libero – это инструмент программиста и генератор кода. Вы определяете высокоуровневую логику задачи в виде диаграммы, а *Libero* генерирует код, реализующий эту логику. Приложения становится более легкими в написании, более надежными, более понятными. В качестве основной модели *Libero* использует конечный автомат».

2.6.3.2. Обзор инструмента

Инструмент имеет графический интерфейс, позволяет редактировать конечные автоматы. Редактирование автоматов выполняется в форме редактирования каждого отдельного перехода между состояниями. Переход из состояния в состояние осуществляется по событиям, однако «сторожевые» условия на переходе не задаются.

Компиляция в код на языке программирования выполняется по специальному шаблону. Таких шаблонов имеется достаточно много для различных языков. Код строится в виде таблицы переходов, что делает понятным отсутствие возможности задать сторожевые условия.

На сайте производителя имеются примеры использования *Libero*, однако эти примеры весьма примитивны и дают возможность убедиться в убогости данного инструмента.

2.6.3.3. Заключение

Автомат задается в виде таблицы переходов. При этом редактируются отдельные переходы между состояниями. Имеется документация и встроенный *help*. Прилагаются примеры использования инструмента, однако они достаточно убогие. Возможна генерация кода на разных языках программирования. Отладка или интерпретация автомата отсутствуют.

Инструмент не представляет интереса для разработки серьезных приложений.

Оценка: 3/10.

2.6.4. *Stateflow*

2.6.4.1. Выдержки со страницы на сайте производителя

«Stateflow – это интерактивный инструмент для проектирования и моделирования событийных систем. *Stateflow* предоставляет языковые элементы, требуемые для описания сложной логики в естественной, читабельной и понятной форме. Он тесно интегрирован с *Matlab* и *Simulink*, предоставляя эффективное окружение для разработки встроенных систем, которые включают в себя логику управления, наблюдения и режимов.

Диаграммы *Stateflow* позволяют создавать графическое представление иерархических и параллельных состояний и переходов между ними, которые происходят по событиям. *Stateflow* реализует преимущество традиционных диаграмм состояний вместе с инновационными возможностями потока управления, функциями *Matlab*, графическими функциями, таблицами истинности, темпоральными операторами, широковещательным распространением направленных событий и поддержкой интеграции кода на языке *C*, написанного вручную.

Возможна автоматическая генерация кода на языке *C* из диаграмм *Stateflow* при использовании *Stateflow Coder* (приобретается отдельно).

Ключевые моменты:

- предоставляет языковые элементы, иерархичность, параллелизм и семантику детерминированного исполнения для описания сложной логики в естественной и понятной форме;
- определяет функции графически, с использованием диаграмм потоков; процедурно, с использованием языка *MatLab*, а также в табличной форме, с таблицами истинности.
- планирует переходы и события с использованием темпоральной логики;
- поддерживает автоматы Мура и Мили;
- поддерживает код на языке *C* с входными и выходными аргументами;
- поддерживает сигналы на шине, векторные и матричные данные, и данные с фиксированной точкой;
- производит статическую проверку, включая проверку неправильно заданных таблиц истинности;
- производит проверку конфликтов переходов, проблем с циклами, противоречивости состояний, нарушение диапазонов задания данных и условий переполнения;
- анимирует диаграммы *Stateflow* и протоколирует данные в процессе моделирования для улучшения понятности системы и возможности ее отладки;
- включает встроенный отладчик для установки графических точек останова, пошагового исполнения диаграмм состояний и просмотра данных».

2.6.4.2. Обзор инструмента

Инструмент является дополнением (подключаемым модулем) к *Mathworks MatLab*. Позволяет редактировать автоматные графы переходов. Поддерживает вложенные автоматы. Имеются базовые возможности по обнаружению ошибок, например, пропущенных событий. Имеется встроенный графический отладчик, позволяющий исследовать поведение автоматов, отображая их работу прямо на графах переходов, подсвечивая активные состояния и пройденные переходы. Отладчик позволяет задавать точки останова на графах. Имеется возможность вести протокол работы автоматов.

Некоторые ошибки, такие как конфликты переходов и бесконечные циклы, могут быть обнаружены только в процессе отладки при запуске автомата на исполнение.

2.6.4.3. Заключение

Хороший, довольно мощный закрытый коммерческий инструмент для работы с конечными автоматами. Графический редактор, отладчик-визуализатор, возможность генерации кода. Валидация на этапе компиляции не на уровне. Хорошая документация и примеры, что обычно для *MathWorks*.

Оценка: 9/10.

2.6.5. *Statestep*

2.6.5.1. Выдержки со страницы на сайте производителя

«*Statestep* – это инструмент с уникально мощным интерфейсом пользователя, предназначенный для исследования и управления очень большим числом возможностей систематическим и эффективным образом.

Это особенно полезно при разработке программного обеспечения, где основная проблема заключается в «угловых ситуациях» – непредвиденных комбинациях условий, где могут вкрасться серьезные ошибки. *Statestep* поможет предвидеть это и указать корректное ответное действие для миллионов возможных случаев, позволяя быть уверенным в качестве создаваемой системы и сократить время разработки.

В некомпьютерных областях *Statestep* потенциально полезен в тех задачах (банковская деятельность, страхование, юриспруденция, производство, медицина и т.д.) где какие-либо процедуры или выходы должны быть определены для многих комбинаций условий. В этом контексте *Statestep* может использоваться для преодоления ограничений таблицы принятия решений.

В разработке программного обеспечения *Statestep* в основном создан как интерактивный инструмент специфицирования. Однако, потенциально он может использоваться в связке с автоматической генерацией кода, например, при создании прототипов, для реализации логики пользовательского интерфейса или при тестировании».

2.6.5.2. Обзор инструмента

Довольно сложный инструмент для редактирования состояний и переходов. Имеется исполняемый вариант на *Java*, документация. Также имеется утилита стороннего производителя для преобразования модели *StateStep* в исходный код на *C*/*C++* или *Java*.

Состояние системы определяется не одной, а набором переменных, каждая из которых может принимать несколько возможных значений, которые заранее определены. Редактирование не в виде графов переходов автомата, а в виде довольно хитрой таблицы переходов.

В таблице для каждого события определяется ряд привил (одно или более правил). Каждое правило состоит в том, чтобы для данного события в зависимости от значений каждой из множества переменных состояния изменить значение этих переменных требуемым образом, или оставить их без изменения (более подробное описание приведено в документации). При этом программа показывает, какие правила противоречат друг другу, а для каких событий набор правил определен не полностью.

Разобраться в работе программы непросто – для сложных систем достаточно трудно бывает понять, как правильно создать такой набор правил, чтобы он быть полным и при этом непротиворечивым. Однако, редактор «подсказывает» разработчику возможные пути решения.

Приводятся примеры построения событийных систем, например, система управления наручными электронными часами при помощи четырех кнопок.

Трудно оценить, насколько удобен интерфейс редактирования у этого инструмента.

2.6.5.3. Заключение

Редактор таблицы переходов построен по весьма интересной концепции. Документация достаточно подробная и содержит поясняющие примеры. Однако генерация кода средствами самого редактора невозможна. Отладка или интерпретация автомата также отсутствуют.

Оценка: 7/10.

2.6.6. *StateWORKS*

2.6.6.1. Выдержки со страницы на сайте производителя

«*StateWORKS Studio* – это профессиональная среда для разработки и исполнения программного обеспечения. Она содержит:

- редактор для разработки и построения систем КА;
- симулятор для исполнения разработанной системы;
- монитор для тестирования запущенной системы;
- клиент-терминал для тестирования автоматизации с использованием скриптов;
- стандартный исполнитель для запуска созданных приложений.

StateWORKS Studio работает под *WindowsNT*/2000/*XP*. Метод моделирования – виртуальные конечные автоматы (*VFSM*). Можно описывать конечные автоматы, определять зависимости между ними и устанавливать любые свойства спецификации. Результатом этого является полная спецификация всех аспектов управления системой. Имеются несколько выходных файлов, которые могут быть использованы для позднейшей реализации или документирования. Система исполнения, предусмотренная для нескольких операционных систем (как *WindowsNT*/2000/*XP*, *Linux* и другие *Unix*-подобные операционные системы) исполняет окончательную спецификацию в высшей степени быстро и надежно.

Редактор используется для моделирования системы управления, основанной на виртуальных конечных автоматах. Некоторые из его встроенных функций:

- имеет возможность отображать в виде графа всю систему КА, показывая зависимости между всеми используемыми автоматами;
- имеет встроенный редактор виртуальных КА, при помощи которого можно создавать автоматы и определять их поведение;
- имеет встроенный менеджер проектов для управления проектом, который требует разработки спецификаций по отдельным автоматам, по конфигурации модуля ввода-вывода, и по связям между различными автоматами;
- может генерировать несколько выходов для целей документирования и реализации, например:
    1. *XML*-представление поведения всей системы.
    2. КА и систему КА в виде графа.
    3. Таблицы объектов и поведения для текстовых процессоров.
    4. Исполняемые файлы для симулятора или системы исполнения (система *StateWorks*).
    5. Файлы *C++*, необходимые для генерации определенных типов встроенных систем».

2.6.6.2. Обзор инструмента

По-видимому, *StateWORKS* – это довольно сложный инструмент для редактирования состояний и переходов. Можно скачать, но для регистрации нужен временный ключ на 30 дней, получить который не удается. Без ключа программа не устанавливается. Генераторов ключей для взлома этой программы найдено не было.

Автоматы представлены в виде графов переходов. Какие именно атрибуты можно задать состояниям и переходам, а также детали реализации системы автоматов трудно понять без документации, а она недоступна для скачивания (поставляется вместе с самим программным

пакетом, установка которого оказалась невозможна). Однако по скриншотам видно, что переходы осуществляются по событиям, есть входные и выходные действия в состояниях.

По виду редактируемые графы отдаленно напоминает то, что описывается в работе А.А. Шалыто и Н.И. Туккеля «Система управления дизель-генератором» (http://is.ifmo.ru/projects/), однако схема связей автоматов не упоминается, и, видимо, она отсутствует.

2.6.6.3. Заключение

Коммерческое ПО. Ключ для запуска недоступен. Заключение сделать невозможно.
Оценка: (не оценивается).

2.6.7. *VisualState*

2.6.7.1. Выдержки со страницы на сайте производителя

«*VisualSTATE* – это пакет графических инструментов для проектирования встроенных и событийных систем. Он применим для любых 8, 16 и 32-битных микроконтроллеров, и даже для приложений *Microsoft Windows*. При помощи *VisualSTATE* каждая фаза разработки становится намного легче.

- Проектируйте встроенные приложения, рисуя объекты, события, действия и т.д., используя мощную нотацию иерархических диаграмм состояний и автоматов *UML*.
- Обсуждайте проект и обменивайтесь идеями с другими благодаря модельному подходу к проектированию и графическому представлению проекта.
- Моделируйте и визуализируйте Ваши приложения, чтобы найти ошибки на ранней стадии проектирования.
- Проверяйте логическую целостность рабочей модели Вашего проекта при помощи очень мощного формального верификатора.
- Проводите экстенсивные тесты постоянно и итеративно на протяжении всего процесса разработки: валидацию поведения приложения, регрессионное тестирование и автоматический отчет о полноте тестирования.
- Используйте мощь исполняемого *UML* и *RealLink*, чтобы исполнять и визуализировать поведение Ваших приложений на целевой системе.
- Автоматически генерируйте безошибочный и очень компактный код на *C*/*C*++, на 100% совместимый с Вашим проектом.
- Пошаговая разработка, повторное использование кода и прототипирование.
- Добавление новой функциональности в существующее приложение в соответствии с изменениями требований рынка.
- Поддержка существующих приложений благодаря точной, структурированной документации, которая всегда синхронизируется с конечным проектом».

2.6.7.2. Обзор инструмента

Данный коммерческий продукт предназначен в первую очередь для разработки встроенных систем. Имеет редактор графов перехода автомата. Генерация кода на *C*. Имеется также встроенный отладчик автоматов, который позволяет наблюдать происходящее также и в графическом виде.

Особенности. Поддержка иерархии состояний автомата (некий аналог вложенных автоматов в *Switch*-технологии). Действия при входе в состояние, при выходе из состояния и в самом состоянии, а также на переходах. Несмотря на возможность генерации кода на языке *C*, код, реализующий сам автомат (переходы), по-видимому, не генерируется, генерируются только заголовочные файлы, определения констант и внешних функций.

2.6.7.3. Заключение

Коммерческий продукт для разработки программ с использованием диаграмм состояний. Редактор не очень удобен. Генератор кода более чем странный, по сути, он не работает. Отладчик понравился. Документация не меньше 1000 страниц.

Оценка: 8/10.

2.6.8. *Statemate*

2.6.8.1. Выдержки со страницы на сайте производителя

«*Statemate* позволяет инженерам быстро проектировать и проверять сложные продукты системного уровня, используя уникальную комбинацию графического моделирования, симуляции, генерации кода, генерации документации, и определения плана тестирования. В результате *Statemate* развился до уровня стандарта разработки встроенных систем уровня *high-end* в области медицины, автомобилестроения, аэрокосмической и оборонной промышленности.

- Позволяет полностью проектировать системы на высочайшем уровне.
- Устраняет неоднозначности, часто встречающиеся в рукописных спецификациях.
- Проверяет поведение системы на ранней стадии процесса проектирования.
- Позволяет «захватывать» тестовые сценарии на этапах разработки требований и создания спецификации.
- Обнаруживает и устраняет ошибки спецификации перед реализацией.
- Генерирует исполняемую спецификацию системы.
- Связывает проектировщиков, разработчиков, и пользователей для совместной работы над проектом, повышая уровень коммуникации и кооперации.
- Упрощает понимание действий благодаря анимации графической модели во время исполнения кода.
- Ускоряет процесс быстрого прототипирования, предоставляя код на языке *C* или *Ada* для виртуальных или физических прототипов.
- Производит качественную генерацию кода по проектной модели.
- Позволяет проводить совместную спецификацию железа и софта.
- Автоматически генерирует полную, непротиворечивую и формальную документацию.
- Сокращает время разработки как минимум на 30%».

2.6.8.2. Обзор инструмента

Ни программу, ни документацию к ней скачать не удалось, поэтому обзора нет. Можно лишь сказать, что *Statemate* похож на серьезный продукт для разработки встроенных систем. По-видимому, графическое редактирование, генерация кода и отладка имеются.

2.6.8.3. Заключение

Коммерческий продукт для разработки программ. Скачать невозможно, опробовать не удалось. Документацию тоже скачать невозможно, по краткому обзору (брошюра в *PDF*-файле) ничего толком не понять.

Оценка: (не оценивается).

2.6.9. *Rhapsody*

2.6.9.1. Выдержки со страницы на сайте производителя

«*Rhapsody* – это передовая промышленная среда разработки на основе модели для систем, программного обеспечения и тестирования. *Rhapsody* позволяет системным инженерам и

разработчикам ПО достигать значительного прироста производительности по сравнению с традиционным подходом, давая пользователю возможность графически определять системы и проекты программного обеспечения, моделировать и автоматически проверять систему сразу после ее сборки, для того чтобы в итоге произвести полный выходной код по модели встроенной системы.

Используя *Rhapsody* с ее пакетами, созданными отдельно для каждой из ролей системного инженера или разработчика ПО, пользователь получает следующие преимущества:

- Цельную среду для системной или программной разработки с полной переносимостью проектов.
- Гибкие среды проектирования, поддерживающие *SysML*, *UML* 2.0, *DoDAF* и специализированные языки (*Domain Specific Languages*):
  - импорт из *Rational Rose* и поддержка *XMI*.
- Встроенное моделирование требований, возможность пошаговой отладки и анализа:
  - моделирование требований на *SysML*;
  - пошаговая отладка и анализ на всем жизненном цикле.
- Совместная работа в малой и большой команде при помощи:
  - нахождения различий в модели и слияния моделей;
  - автоматизированной генерации документации.
- Проектирование и возможность тестирования, которая включает:
  - моделирование;
  - тестирование на основе требований;
  - автоматическое создание тестов;
  - встроенную отладку на уровне целевой модели.
- Полная генерация приложений для 8, 16, 32 и 64-битных приложений:
  - *C*, *C++*, *Java* и *Ada;*
  - кодогенерация, основанная на правилах;
  - генерация для *COM*/*CORBA;*
  - визуализация кода и обратный инжиниринг;
  - легко настраивается для любой коммерческой ОС реального времени;
  - тесная интеграция и IDE, работающих на основе *Eclipse*».

2.6.9.2. Обзор инструмента

Пожалуй, данный инструмент позволяет делать все, что можно потребовать от инструмента для программирования с применением конечных автоматов. Имеется графический редактор диаграмм состояний, отладчик, генератор кода. Кроме диаграмм состояний *Rhapsody* позволяет также строить диаграммы классов и связывать их с диаграммами состояний, для того чтобы на основе этого генерировать объектно-ориентированный код программы на языках *C++*, *Java* и т.д.

Кроме этих двух типов диаграмм инструмент позволяет строить и другие диаграммы *UML*, однако они, по сути, является просто картинками, иллюстрирующими модель, и не участвуют в генерации кода работающей программы.

У *Rhapsody* достаточно хорошая документация, а также имеются примеры, позволяющие разобраться в возможностях данного инструмента и приемах работы с ним.

2.6.9.3. Заключение

Лучший инструмент из всех протестированных. Умеет делать все. Есть недостаток – при некоторых условиях не обнаруживает конфликт переходов в автоматных графах на этапе компиляции (переходы с одновременно выполняющимися условиями). Также данный инструмент является закрытым и платным.

Оценка: 9/10.

2.7. Результаты непатентных исследований

Исследование инструментов для разработки программ с использованием конечных автоматов показало, что в мире существует как минимум десять таких инструментов, созданных различными фирмами, коллективами или отдельными программистами. Все эти инструменты предназначены для создания программного обеспечения с использованием конечно-автоматной технологии и имеют графический интерфейс пользователя, позволяющий задавать состояния и переходы автомата в виде графа переходов или в виде таблицы. Практически все эти инструменты имеют возможность генерации кода, реализующего автомат, на текстовом языке программирования, таком как *C++* или *Java*, а также встроенные средства отладки.

Некоторые инструменты так и не удалось опробовать в работе, так как они предназначены только для корпоративных клиентов, и не доступны для свободного скачивания даже в «демонстрационном» варианте (например, *Statemate*, *TAU* и ряд других продуктов). Однако некоторые коммерческие продукты, такие как *Rhapsody* или *VisualState*, доступны для тестирования. Также существуют инструменты, распространяемые свободно и бесплатно, которые доступны для скачивания вместе со своими исходными кодами (*FSKC*, *FSM Editor*).

Технологии, применяемые в рассмотренных инструментах для создания программ, также различаются. Некоторые из них (например, *FSM Editor*) используют *Switch*-технологию, другие инструменты работают иначе, программный код в них строится при помощи таблицы переходов, либо же используется встроенная библиотека функций, реализующая автоматы. Среди программ, не вошедших в основную часть исследования (например, имеющих только интерфейс командной строки), также есть интересные идеи или реализации. Например, в руководстве по программированию на языке *C* с использованием платформы *Quantum* (ссылка 35 в табл. 1) предлагается подход для реализации конечного автомата на языке программирования, отличный от *Switch*-технологии. Проект *JSpasm* (ссылка 26) предлагает библиотеку для программной реализации конечных автоматов на языке *Java*, при помощи которой можно создавать автоматы прямо по ходу выполнения программы, вводя новые состояния и переходы, и после этого выполнять их. Есть и другие более или менее интересные проекты.

Появление инструментов и библиотек для работы с конечными автоматами свидетельствует о том, что применение конечно-автоматной технологии в последнее время представляет большой интерес для сообщества программистов. Причем, в последнее время ощущается бурное развитие этого направления в программировании

3. Программный комплекс автоматного программирования и его отличия

Как отмечалось выше, в мире существуют такие инструментальные средства и методологии создания программного обеспечения, функциональность которых во многом аналогична функциональности создаваемой технологии.

Эти инструментальные средства (и методологии), будучи в значительной степени ориентированы на создание приложений реального времени, предназначенных для встроенных систем, тем не менее, могут достаточно эффективно использоваться и для создания программного обеспечения общего вида – для так называемых универсальных вычислений.

Одна из таких методологий описана в книге Самека М. «Практическое использование диаграмм состояний на языках *C* и *C++*» [5]. Автор книги предлагает способ реализации иерархических конечных автоматов в виде программного кода на языке *C++* без использования каких-либо редакторов графов переходов и визуальных сред проектирования. Программный код по этой методологии строится несколько иначе, чем в *Switch*-технологии, однако предлагаемый там способ, как будто бы, ничем не хуже. Конечно, отсутствие визуальной среды проектирования может быть очень существенным недостатком.

Похожая методология предлагается и в проекте *Boost Statechart Library*, который упоминается в данном обзоре.

Говоря о визуальных инструментальных средствах, предназначенных для создания программного обеспечения на основе конечных автоматов, можно упомянуть как коммерческие, так и некоммерческие продукты.

Одним из таких коммерческих инструментов является протестированный в данном исследовании программный продукт *VisualState*. Этот инструмент, однако, не смог сгенерировать исходные коды, реализующие автомат на языке программирования *C++*. Инструменты *Statemate* и *StateWORKS*, судя по их описанию, также предназначены для создания программного обеспечения по конечно-автоматной технологии, но протестировать их не удалось.

Очень достойными инструментами оказались *Stateflow* фирмы *MathWorks* и *Rhapsody* фирмы *I-Logix* (объединившейся теперь с фирмой *Telelogic*). Данные инструменты предоставляют пользователю практически полный набор функциональных возможностей, требуемых для программирования с использованием конечно-автоматной технологии.

Среди некоммерческих инструментов достойным упоминания является свободно распространяемый продукт *FSME*. Этот инструмент предоставляет значительно урезанную по сравнению со *Switch*-технологией функциональность, позволяя редактировать только один автомат, однако имеет как встроенный генератор исходных кодов, так и графический отладчик автомата.

Также существуют инструменты, предназначенные для создания моделей программного обеспечения на унифицированном языке моделирования *UML* (*Unified Modeling Language*), среди которых имеются как коммерческие, так и некоммерческие программные продукты. К числу коммерческих продуктов относятся инструмент *Together* фирмы *Borland*, *TAU* фирмы *Telelogic*, *Rational Software Architect* фирмы *IBM*, *EclipseUML* от *Omondo* и ряд других. Из некоммерческих наиболее известен *ArgoUML*.

Эти инструменты позволяют создавать модели программного обеспечения, описывая их на языке моделирования *UML*. Однако их функциональность в смысле создания на базе модели реально работающих программ в основном ограничивается генерацией программного кода, реализующего диаграммы классов *UML*. И лишь немногие инструменты позволяют генерировать код программы по диаграммам состояний. К числу таковых относится, в частности, продукт *TAU* фирмы *Telelogic*. Сюда же можно отнести и рассмотренный в данном обзоре продукт *I-Logix Rhapsody*.

В ходе проведенной опытно-конструкторской работы был создан Программный комплекс автоматного программирования. Его название – *UniMod*.

Преимуществами *UniMod* перед другими программными продуктами являются:

- возможность проектирования и исполнения программы в целом, а не только графов переходов;
- возможность моделирования как статической составляющей программы (диаграмм классов в форме схем связей автомата), так и ее динамической составляющей (графов переходов). Как говорил Гради Буч, один из авторов языка *UML*, «нашим традиционным текстовым языкам просто не хватает средств для того, чтобы оценивать и понимать поведение программных систем. Это заставляет нас искать другие пути, а те ведут к абстракции моделирования» [6]. *UniMod* предоставляет пользователю именно такую возможность – моделировать поведение разрабатываемых программ в динамике;
- динамическое обнаружение ошибок в модели в процессе ее редактирования; автоматизация проверки правильности разрабатываемого программного обеспечения путем проверки корректности графов переходов:
  - достижимости состояний;
  - полноты и непротиворечивости графов переходов;
- визуальная локальная и удаленная отладка модели;
- возможность запускать модели в различных средах при помощи адаптеров;
- автоматизированная укладка автоматных графов переходов;
- возможность наряду с компиляцией схем связей и графов переходов в тексты программ проводить также их интерпретацию;

- автоматизированное документирование на базе схем связей и графов переходов;
- автоматическое протоколирование во время исполнения программы;
- инструментальное средство открытое и бесплатное.

Фактически, *UniMod* реализует концепцию исполняемого *UML* (*executable UML* [7]), что означает возможность исполнения создаваемой модели программного обеспечения непосредственно из среды моделирования, как в режиме интерпретации, так и в режиме компиляции [8].

Документация на *UniMod* соответствует стандарту ГОСТ ЕСПД.

4. Оценка стоимости Программного комплекса автоматного программирования

Согласно требованиям технического задания требуется оценить стоимость созданного «Программного комплекса автоматного программирования» (*UniMod*) в соответствии с «Методическими рекомендациями», утвержденными Министерством имущественных отношений 26 ноября 2002 г. № СК-4/21297. Эти рекомендации предлагают следующие способы оценки рыночной стоимости продукта, являющегося интеллектуальной собственностью:

1. Выяснение потенциальной выгоды, которую может дать использование продукта.
2. Сравнение с аналогами продукта, имеющимися на рынке.
3. Затратный подход, описание которого весьма туманно и неясно.

При оценке стоимости будет использован главным образом второй способ. Первый подход, основанный на потенциальной выгоде от использования, лучше всего применим либо при отсутствии аналогов продукта, либо при условии, что продажа аналогов производится не массово (как, например, ОС *Microsoft Windows*), а исключительно через сделку непосредственно между продавцом и покупателем (например, ПО для системы учета энергоресурсов конкретного промышленного предприятия). В рассматриваемом случае у созданного «Программного комплекса автоматного программирования» на рынке имеются аналоги, наличие которых позволяет сравнивать с ними созданный продукт.

Однако, многие качественные аналоги созданного программного комплекса не находятся в открытой продаже, а поставляются производителем клиенту по специальному заказу. При этом отпускная цена продукта исходно не указывается [9], а назначается производителем в процессе оформления сделки. В этом случае производитель, ориентируясь на финансовые возможности клиента, может достаточно произвольно устанавливать цену на свое программное обеспечение в каждом конкретном случае, мотивируя это различием функциональности, настройкой под нужды заказчика и т. п. (*Telelogic TAU*, *I-Logix Rhapsody*). Данное обстоятельство (малая распространенность продуктов такого типа и их уникальность) часто не позволяет прямо сравнивать цены с достаточной точностью. Можно говорить лишь о порядке величины.

Так, по разным оценкам цена программного продукта *I-Logix Rhapsody*, составляет примерно от 1000 до 5000 долларов США за рабочее место, в зависимости от варианта поставки и количества покупаемых лицензий. Оценить стоимость продукта *Telelogic TAU* вообще не удалось. Однако существует и другая практика. Например, фирма *Omondo*, производящая продукт *EclipseUML*, устанавливает цены на свой продукт явно и открыто. На данный момент цена этого продукта (коммерческой версии) варьируется от 1990 евро при покупке одного рабочего места до 950 евро при лицензировании сразу 1000 рабочих мест [10].

Существуют и другие продукты, которые в данном обзоре не рассматривались ввиду их недоступности для тестирования. Например, можно назвать такие продукты, как *Borland Together Designer* (3500$), *IBM-Rational Rose Developer* (2500$), которые могут непосредственно использоваться при создании программного обеспечения в качестве инструментов автоматного программирования. Наконец, можно упомянуть продукты типа *Microsoft Visio Professional* (500$), которые не имеют такой возможности, но позволяют рисовать диаграммы состояний *UML* (графы

переходов автоматов). Эти диаграммы в дальнейшем могут быть преобразованы в коды программ при помощи внешних утилит. (Приведенные цены на продукты указаны согласно [9]).

С другой стороны, существуют и свободно распространяемые инструменты с открытым исходным кодом, некоторые из которых были рассмотрены в данном отчете.

Сравнение возможностей созданного продукта *UniMod* с возможностями аналогов позволяет говорить о том, что *UniMod* не уступает продуктам многих известных фирм, а в некотором плане даже превосходит их. Однако, назначение цены на инструментальное средство *UniMod* сравнимой с ценами на эти продукты в силу известных обстоятельств не представляется рациональным. Предлагается распространять *UniMod* по тому же принципу, по какому распространяются продукты с открытым исходным кодом (лицензия *GPL*), взимая при этом денежное вознаграждение за настройку продукта под нужды конкретного пользователя, обучение пользователей работе с данным продуктом и другие подобные услуги.

Заключение

Поиск в патентных базах данных различных стран мира дал отрицательный результат: не было найдено патентов, описывающих аналоги технологии автоматного программирования. Однако, как показало исследование, в мире уже существуют средства разработки программ на основе конечных автоматов. Существуют альтернативные методологии преобразования этих автоматов в код программы, аналогичные по своей функциональности *Switch*-технологии. Тем не менее, ни в российских, ни в зарубежных патентных документах, ни среди непатентованных технологий – нигде не встречается именно такого способа построения программного кода по конечным автоматам, какой применяется в *Switch*-технологии. Поэтому можно утверждать, что патентная чистота предлагаемого метода генерации исходного кода обеспечена с достаточной степенью достоверности.

Следует сказать, что международные договоры, подписанные Российской Федерацией, в силу приоритета международных правовых актов перед российским законодательством оставляют возможность для зарубежных компаний регистрировать в России патенты, которые были до этого зарегистрированы за рубежом, причем так, что эти патенты могут не соответствовать Российскому патентному закону. Парадоксальность сложившейся ситуации в том, что патент на программное обеспечение в России может быть выдан иностранному владельцу интеллектуальной собственности, но не российскому изобретателю. Это приводит к дискриминации граждан России и Российских фирм, давая преимущество гражданам других стран и иностранным компаниям.

Поэтому, имея в виду возможность регистрации в России зарубежного патента на программное обеспечение, способ интеллектуальной деятельности или математический метод, отметим, что абсолютной гарантии патентной чистоты разрабатываемой технологии автоматного программирования в будущем дать невозможно.

Для защиты интеллектуальной собственности авторами инструментального средства *UniMod* были поданы две заявки на получение свидетельств об официальной регистрации программ для ЭВМ. По первой из них получено свидетельство № 2006613249 от 14.09.2006 «Ядро автоматного программирования». Таким образом, данная программа зарегистрирована в реестре программ для ЭВМ Российской Федерации. По второй заявке № 2006612475 от 17.07.2006 «Встраиваемый модуль автоматного программирования для среды разработки *Eclipse*» в настоящее время проводится регистрация.

Литература и источники

1. *Патентный закон РФ*. http://www.legal-support.ru/information/laws/intellect/patent-law.html
2. *US Patent Database* Number Search. http://patft.uspto.gov/netahtml/PTO/srchnum.htm, 5,317,757
3. *Stallman R.M.* The Danger of Software Patents. http://www.gnu.org/philosophy/stallman-mec-india.html
4. *Irlam G., Williams R.* Software Patents: An Industry at Risk. http://lpf.ai.mit.edu/Patents/industry-at-risk.html
5. *Samek M.* Practical Statecharts in C/C++. CMP Books, 2002.
6. Будущее моделирования выглядит оптимистично //PC WEEK / RE. 2005. № 42, с. 23, 38
7. *Mellor S.J.*, *Balcer M.J.* Executable UML. A Foundation for Model-Driven Architecture, Boston, Addison Wesley, 2004.
8. *Гуров В.С., Нарвский А.С., Шалыто А.А.* Исполняемый UML из России //PC Week/RE. 2005. № 26, с.18,19. http://is.ifmo.ru/works/_umlrus.pdf
9. *UML Products by Price*. http://www.objectsbydesign.com/tools/umltools_byPrice.html
10. *Omondo products pricing*. http://www.eclipsedownload.com/pricing.html

Приложение А

УТВЕРЖДАЮ
заведующий кафедрой
«Технологии программирования»
д.т.н., профессор

_____Шалыто А.А.

«10» *сентября* 2006 *г.*

ЗАДАНИЕ № 2006.09.10-01
на проведение патентных исследований

Наименование работы (темы) *Технология автоматного программирования: применение и инструментальные средства*

Шифр работы (темы) ИТ-13.4/004

Этап работы *Обеспечение патентной чистоты и оценка рыночной стоимости результатов ОКР*

Сроки его выполнения *Сентябрь* 2006 *г. – Октябрь* 2006 *г.*

Задачи патентных исследований

1. *Обеспечить патентную чистоту результатов ОКР, представить сведения об охранных и иных документах, которые будут препятствовать применению результатов ОКР в Российской Федерации и условия их использования.*

2. *Оценить рыночную стоимость созданного программного комплекса автоматного программирования*

КАЛЕНДАРНЫЙ ПЛАН

Виды патентных исследований	Подразделения-исполнители (соисполнители)	Ответственные исполнители (Ф.И.О.)	Сроки выполнения патентных исследований. Начало. Окончание	Отчетные документы
Проверка на патентную чистоту		*Аничкин И.М.*	*Сентябрь 2006 – Октябрь 2006*	*Отчет о патентных исследованиях*

Руководитель патентного подразделения _____ личная подпись _____ расшифровка подписи _____ дата

Руководитель подразделения исполнителя работы _____ личная подпись _____ расшифровка подписи _____ дата

Приложение Б Регламент поиска № 2006.09.11-01

«11» *сентября* 2006 *г.*
дата составления регламента

Наименование работы (темы) *Технология автоматного программирования: применение и инструментальные средства*

Шифр работы (темы) ИТ-13.4/004

Номер и дата утверждения задания 2006.09.10-01 , «10» *сентября* 2006 *г.*

Этап работы *Обеспечение патентной чистоты результатов ОКР*

Цель поиска информации 1. *Создание списка документов, которые потенциально могут нарушить патентную чистоту результатов ОКР и препятствовать их применению в Российской Федерации*
2. Оценка возможной рыночной стоимости созданной в ходе проведения ОКР интеллектуальной собственности

Обоснование регламента поиска

Начало поиска 10 *сентября* 2006 *г.* Окончание поиска 10 *октября* 2006 *г.*

Предмет поиска (объект исследования, его составные части, товар)	Страна поиска	Источники информации, по которым будет проводиться поиск								Ретро-спектив-ность	Наименование информационной базы (фонда)
		патентные		НТИ*		конъюнктурные		другие			
		Наименование	Классификационные рубрики: МПК (МКИ)*, МКПО*, НКИ* и другие	Наименование	Рубрики УДК* и другие	Наименование	Код товара: ГС*, СМТК*, БТН*	Наименование	Классификационные индексы		
1	2	3	4	5	6	7	8	9	10	11	12
1. Компиляция или интерпретация языков программирования высокого уровня; 2. Автоматизированное проектирование	Россия, США, Европа.	База данных по изобретениям Федеральной службы по интеллектуальной собственности, патентам и товарным знакам (Роспатент) Патентная база данных Бюро по патентам и товарным знакам США (USPTO) Патентная база данных Европейского патентного бюро (EPO)	G06F00 9/45 G06F017/50	-	-	-	-	-	-	20 лет	Роспатент www.fips.ru USPTO, www.uspto.gov EPO, ep.espacenet.com

Руководитель подразделения исполнителя работы _____ (личная подпись) _____ (расшифровка подписи) _____ (дата)

Руководитель патентного подразделения _____ (личная подпись) _____ (расшифровка подписи) _____ (дата)

Приложение В Отчет о поиске № 2006.10.10-01

Поиск проведен в соответствии с заданием зав. каф. технологий программирования Шалыто А.А.

№ 2006.09.10-01 от «10» *сентября* 2006 *г.*

и Регламентом поиска № 2006.09.11-01 от «11» *сентября* 2006 *г.*

Этап работы *Обеспечение патентной чистоты результатов ОКР*

Начало поиска 10 *сентября* 2006 *г.* Окончание поиска 10 *октября* 2006 *г.*

Сведения о выполнении регламента поиска

Поиск выполнен полностью. Документов, которые могут препятствовать применению результатов ОКР в Российской Федерации, обнаружено не было. Созданный «Программный комплекс автоматного программирования» рекомендовано распространять свободно, с открытым исходным кодом.

Предложения по дальнейшему проведению поиска и патентных исследований

Патентный закон России и некоторых других стран (в основном стран Европы) запрещает патентование научных теорий, математических методов, правил и методов интеллектуальной и хозяйственной деятельности, компьютерных программ, и решений, заключающихся только в представлении информации. В то же время патентные системы других стран (например, США, Японии и стран Азии) значительно расширяют перечень инноваций, которые могут считаться изобретениями и быть признаны патентоспособными, включая сюда также и компьютерные программы. Получить патент на изобретение, непосредственно относящееся к программному обеспечению, можно и в тех странах, где это формально запрещено, но для этого требуется в патентной заявке описать компьютерную программу не как таковую, а как систему, метод или устройство, предназначенное для определенных действий или решения каких-то задач. Соответствующие примеры можно найти.

Количество патентов, выданных, например, в США, и имеющих то или иное отношение к компиляции программ на языках высокого уровня или к системам автоматизированного проектирования (к той функциональности, которую должна обеспечивать разрабатываемая технология автоматного программирования согласно техническому заданию), измеряется сотнями. И, хотя среди исследованных патентов не нашлось аналогов создаваемой технологии автоматного программирования, в других разделах патентной классификации могут найтись патенты, в которых заявляются требования,

охватывающие какие-то существенные детали разрабатываемой технологии.

При этом нельзя сказать определенно, будет ли тот или иной патент на программное обеспечение, выданный в одной из тех стран, где это разрешено, когда-либо действовать на территории России. Международные договоры, подписанные Российской стороной, оставляют такую возможность в силу приоритета международных правовых актов перед российским законодательством. Парадоксальность сложившейся ситуации в том, что патент на программное обеспечение в России может быть выдан владельцу иностранного патента, но не российскому изобретателю, что приводит к дискриминации граждан России и Российских компаний, давая преимущество гражданам других стран и иностранным компаниям.

По итогам патентных исследований можно утверждать, что патентная чистота предлагаемого метода генерации исходного кода обеспечена с достаточной степенью достоверности. Однако абсолютной гарантии патентной чистоты разрабатываемой технологии автоматного программирования в силу указанных выше причин дать невозможно.

Материалы, отобранные для последующего анализа

Отсутствуют.