Опубликовано в Научно-техническом вестнике СПбГУ ИТМО. 2008. Выпуск 53. Автоматное программирование, с. 263-273.

УДК 004.432.4

ТЕКСТОВЫЙ ЯЗЫК АВТОМАТНОГО ПРОГРАММИРОВАНИЯ *FSML* ДЛЯ ИНСТРУМЕНТАЛЬНОГО СРЕДСТВА *UNIMOD*

И.А. Лагунов
(Санкт-Петербургский государственный университет информационных технологий, механики и оптики)

Инструментальное средство *UniMod* позволяет проектировать и исполнять автоматные программы. Основное его ограничение – возможность только визуального описания конечных автоматов. В статье рассматриваются текстовый язык автоматного программирования *FSML* (*Finite State Machine Language*) и его редактор, позволяющие описывать автоматы в указанном средстве с помощью текстового языка автоматного программирования.
Ключевые слова: автоматное программирование, UniMod, текстовый язык

Введение

В настоящее время в сфере разработки программного обеспечения широко используются объектно-ориентированные языки программирования. Однако для широкого класса задач управления традиционный объектно-ориентированный подход является не самым оптимальным. В этом случае *SWITCH*-технология, предложенная в работах [1, 2] для поддержки автоматного программирования, является, пожалуй, наиболее естественным решением.

К настоящему моменту было выполнено несколько попыток облегчить применение этой сравнительно молодой парадигмы программирования [3]. Создаются графические среды, позволяющие создавать и редактировать графы переходов автоматов [4, 5]. Одну из таких сред предоставляет проект *UniMod* [4]. Это надстройка для интегрированной среды разработки *Eclipse* [6], позволяющая строить графы переходов автоматов, интерпретировать или компилировать автоматные программы на одном из объектно-ориентированных языков, проверять корректность построения графов переходов, а также отлаживать созданную программу непосредственно в среде разработки. Однако используемый в этом проекте способ описания конечных автоматов с помощью графического редактора весьма трудоемок.

Для решения этой проблемы автором был создан текстовый язык автоматного программирования *FSML* и реализован его редактор (*Editor*), который используется в инструментальном средстве *UniMod 2*. При этом синтаксический анализатор языка и некоторые возможности редактора *FSMLEditor* реализованы на основе *SWITCH*-технологии и инструментального средства *UniMod*.

Описание языка

Язык *FSML* предназначен для текстового представления конечных автоматов и их связей. Его возможности позволяют с использованием синтаксиса, близкого к синтаксису языка программирования *Java*, описать события, состояния, вложенные автоматы, переходы и другие элементы автоматной модели.

Рассмотрим пример реализации на предложенном языке модели пешеходного светофора с таймером (листинг 1). Полное описание синтаксиса языка приведено в [7].

Листинг 1. Код примера на языке *FSML*

```
1 uses trafficlight.provider.TrafficSignalProvider;
2
3 statemachine TrafficLightWithTimer {
4
5 trafficlight.object.RedLamp red;
6 trafficlight.object.GreenLamp green;
7 trafficlight.object.Timer timer;
8
9 initial Init {
10 transitto Inactive;
11 }
12 Active {
13 on switch transitto Inactive;
14
15 initial InitActive {
16 execute red.turnOn, timer.reset
17 transitto Red;
18 }
19 Red {
20 on tick if timer.value == 0
21 execute red.turnOff, green.turnOn, timer.reset
22 transitto Green;
23 on tick else
24 execute timer.decrement;
25 }
26 Green {
27 on tick if timer.value < 5
28 execute green.turnBlinking, timer.decrement
29 transitto GreenBlinking;
30 on tick else
31 execute timer.decrement;
32 }
33 GreenBlinking {
34 on tick if timer.value == 0
35 execute green.turnOff, red.turnOn, timer.reset
36 transitto Red;
37 on tick else
38 execute timer.decrement;
39 }
40 }
41 Inactive {
42 on enter execute red.turnOff, green.turnOff, timer.turnOff;
43 on switch transitto Active;
44 on stop transitto Final;
45 }
46 final Final {
47 }
48 }
```

На листинге 2 приведена вырезка из кода на языке программирования *Java* для поставщика событий `TrafficSignalProvider`, в которой описываются события. На листингах 3 и 4 приведены вырезки из кода для объектов управления `RedLamp` и `Timer` соответственно, в которых описываются действия. Код объекта управления `GreenLamp` аналогичен объекту управления `RedLamp`.

Этой программе соответствуют диаграмма связей и диаграмма состояний, представленные на рис. 1 и 2 соответственно. Средства *FSML* и *UniMod* позволяют сгенери-

ровать эти диаграммы по коду программы и автоматически расположить их на плоскости. Приведенные диаграммы оптимизированы вручную для большей наглядности.

Листинг 2. Объявление событий в поставщике событий TrafficSignalProvider

```
public class TrafficSignalProvider implements EventProvider {
 /**
 * @unimod.event.descr Full stop signal
 */
 public static final String STOP = "stop";

 /**
 * @unimod.event.descr Switch on/off the traffic-light
 */
 public static final String SWITCH = "switch";

 /**
 * @unimod.event.descr Next tick of system timer happened
 */
 public static final String TICK = "tick";
```

Листинг 3. Объявление действий в объекте управления RedLamp

```
public class RedLamp implements ControlledObject {
 /**
 * @unimod.action.descr switches the lamp on
 */
 public void turnOn(StateMachineContext context) {
 /* Обработка действия */
 }

 /**
 * @unimod.action.descr switches the lamp off
 */
 public void turnOff(StateMachineContext context) {
 /* Обработка действия */
 }

 /**
 * @unimod.action.descr switches the lamp to blinking mode
 */
 public void turnBlinking(StateMachineContext context) {
 /* Обработка действия */
 }
}
```

Листинг 4. Объявление действий в объекте управления Timer

```
public class Timer implements ControlledObject {
 /**
 * @unimod.action.descr current value of timer
 */
 public int value(StateMachineContext context) {
 /* Обработка действия */
 }

 /**
 * @unimod.action.descr decrement timer value
 */
 public void decrement(StateMachineContext context) {
 /* Обработка действия */
 }

 /**
 * @unimod.action.descr reset timer
 */
 public void reset(StateMachineContext context) {
 /* Обработка действия */
 }

 /**
 * @unimod.action.descr turn off
 */
 public void turnOff(StateMachineContext context) {
 /* Обработка действия */
 }
}
```

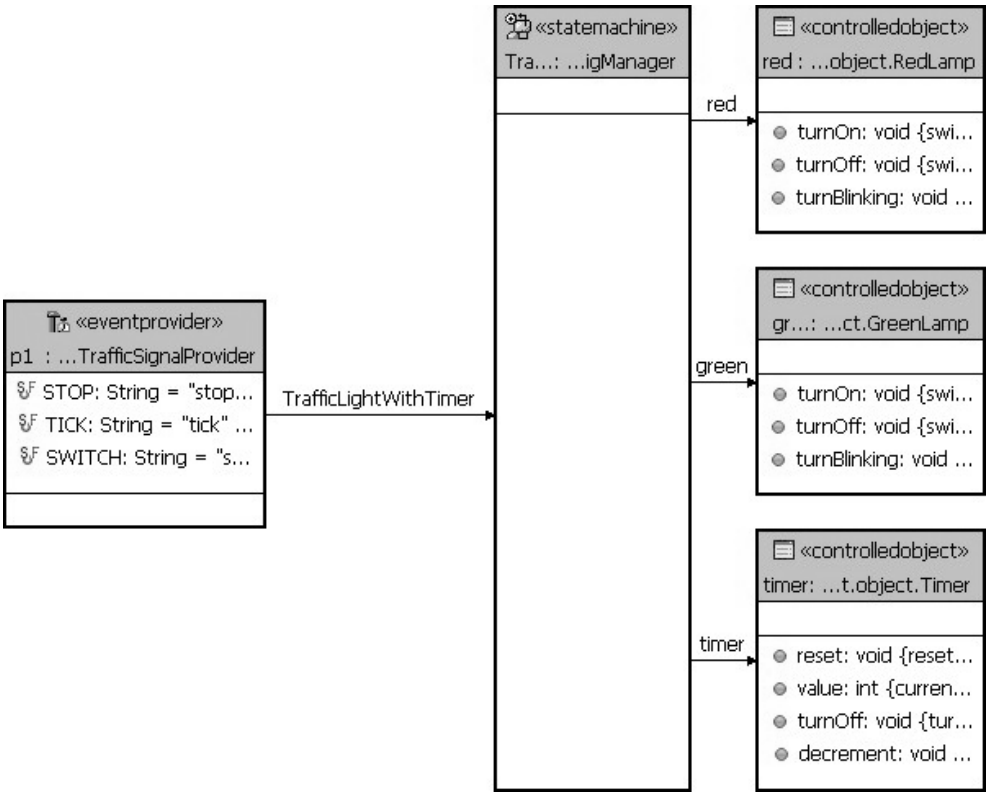

Рис. 1. Диаграмма связей для светофора

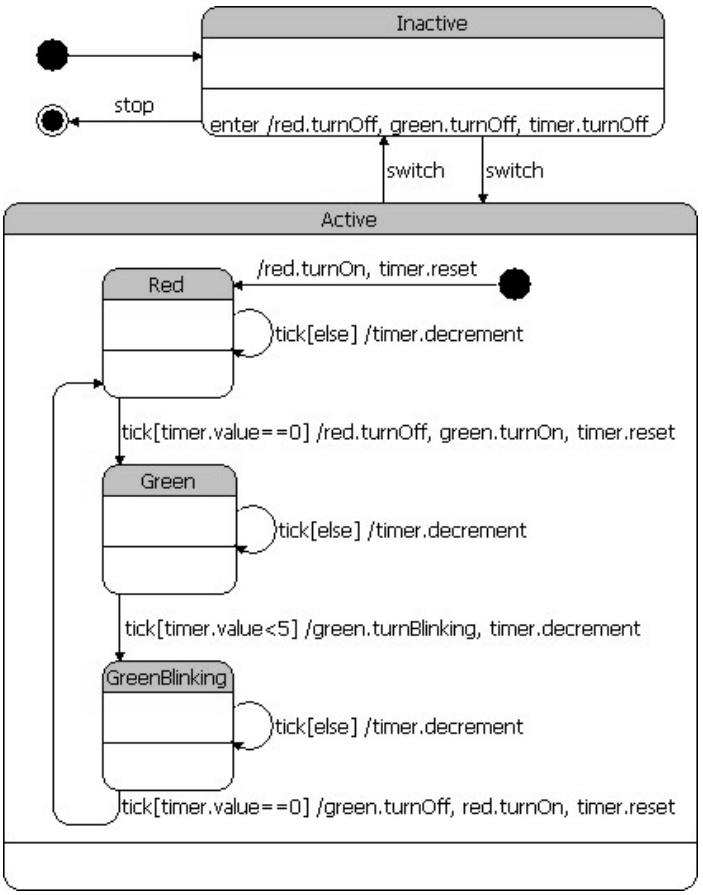

Рис. 2. Диаграмма состояний для светофора

Рассмотрим код программы на языке *FSML* более подробно. Сначала объявляется поставщик событий данной системы:

- `trafficlight.provider.TrafficSignalProvider` – он предоставляет следующие события:
  - `stop` – сигнал завершения работы системы;
  - `switch` – сигнал включения/выключения светофора;
  - `tick` – сигнал от системного таймера.

После этого объявляется автомат `TrafficLightWithTimer` и его объекты управления. Каждой из физических частей светофора соответствует свой объект управления:

- `trafficlight.object.RedLamp red` – красная лампа. Этот объект управления имеет следующий интерфейс:
  - `turnOn - включить лампу;`
  - `turnOff - выключить лампу;`
  - `turnBlinking - включить лампу в режиме мигания;`
- `trafficlight.object.GreenLamp green` – зеленая лампа. Этот объект управления аналогичен предыдущему;
- `trafficlight.object.Timer timer` – табло таймера. Этот объект управления имеет следующий интерфейс:
  - `value - получить текущее значение таймера;`
  - `decrement - уменьшить значение таймера` на единицу;
  - `reset - сбросить таймер к начальному значению;`
  - `turnOff - выключить табло таймера.`

Оставшуюся часть программы занимают описания состояний и переходов между ними. В рассмотренном примере, помимо начального (`initial Init`) и конечного (`final Final`) состояний автомата, присутствуют следующие состояния:

- `Active` – светофор включен. Это сложное состояние, содержащее основной цикл работы светофора, состоящий из четырех состояний:
  - `InitActive` – начальное состояние после включения светофора;
  - `Red` – включен красный сигнал светофора;
  - `Green` – включен зеленый сигнал светофора;
  - `GreenBlinking` – включен зеленый мигающий сигнал светофора;
- `Inactive` – светофор выключен.

Заметим, что это далеко не единственная и не лучшая реализация автомата, управляющего светофором. Можно произвести декомпозицию и выделить содержимое состояния `Active` в отдельный автомат, сделав его вложенным в это состояние. Это позволит упростить как программу на языке *FSML*, так и соответствующую диаграмму состояний.

Особенности языка

Выделим особенности данного языка автоматного программирования. Напомним, что он ориентирован на использование в инструментальном средстве *UniMod 2*.

Текстово-визуальный подход к разработке автоматных программ не реализован полноценно ни в одном из существующих инструментальных средств.

Система *MPS* [8] предлагает возможность просмотра диаграммы состояний создаваемого автомата. Однако на данный момент в ней невозможно создание графических редакторов. Это исключает возможность редактирования диаграммы состояний в виде графа. В то же время эта возможность может быть полезна для внесения быстрых изменений в структуру автомата, а текстовый ввод удобен для быстрого первоначального описания автомата. Инструментальное средство *UniMod* [4], наоборот, значительно теряет в эффективности, не позволяя редактировать автомат в текстовом представлении. Именно этот недостаток исправляет язык *FSML*.

Явное задание графа переходов позволяет гарантировать его изоморфность программе, что избавляет программиста от многих ошибок уже на этапе описания автомата. Кроме того, этого значительно облегчает проверку валидности графа переходов.

Интеграция с объектно-ориентированным кодом необходима для использования языка автоматного программирования на практике, поскольку, за исключением языка *TABP* [9], эти языки [10–14] не являются универсальными. Однако большая часть существующих автоматных языков реализует интеграцию с помощью трансляции кода программы в код на одном из языков общего назначения. При этом происходит разбор текста программы и преобразование ее в абстрактное синтаксическое дерево (АСД), по которому генерируется код на языке общего назначения (рис. 3).

Рис. 3. Стандартная схема интеграции автоматного языка с объектно-ориентированным кодом

Такая «непрозрачная» связь затрудняет разработку на этих языках, так как сгенерированный код намного сложнее читать, чем исходный код или диаграмму состояний. Подходом к выполнению этого требования выгодно отличается система *MPS*, на основе которой созданы два языка автоматного программирования. Она позволяет редактировать абстрактное представление программы, а, следовательно, модель конечного автомата в памяти. Затем эта модель может быть транслирована в любой из языков общего назначения или в графическое изображение диаграммы состояний.

Язык *FSML* совместно с инструментальным средством *UniMod* выгодно отличаются от других средств автоматного программирования. Можно выделить два главных отличия схемы на рис. 4 от схемы на рис. 3, соответствующей другим текстовым языкам автоматного программирования:

- с помощью парсера *FSMLParser* сразу строится конкретная автоматная *UniMod*-модель вместо абстрактного синтаксического дерева автоматной программы;
- существует и активно используется обратная связь с объектно-ориентированным кодом на языке *Java* – через поставщики событий и объекты управления автомата.

Эти два отличия в совокупности позволяют работать непосредственно с моделью автомата, запуская его в режиме интерпретации. В этом случае отпадает необходимость трансляции автоматного кода в код на языке общего назначения. Однако такая возможность имеется при необходимости применения компилятивного подхода.

Автоматная модель может модифицироваться через графический редактор в виде диаграмм *UniMod* и через текстовый редактор, использующий парсер языка *FSMLParser* и генератор *fsml*-программ *FSMLGenerator*.

Переиспользование компонентов кода позволяет решить проблему дублирования кода и является необходимым требованием при создании сложных систем. Однако в случае языка автоматного программирования это достаточно сильное требование, полноценная реализация которого может свести почти на нет преимущества такого языка. Такая реализация использована в языке *State Machine* [14]. В результате он является очень громоздким, требуя для каждого состояния создание отдельного класса, объявление и инициализацию дополнительных переменных. Поэтому требуется найти компромисс между удобством проектирования и удобством кодирования.

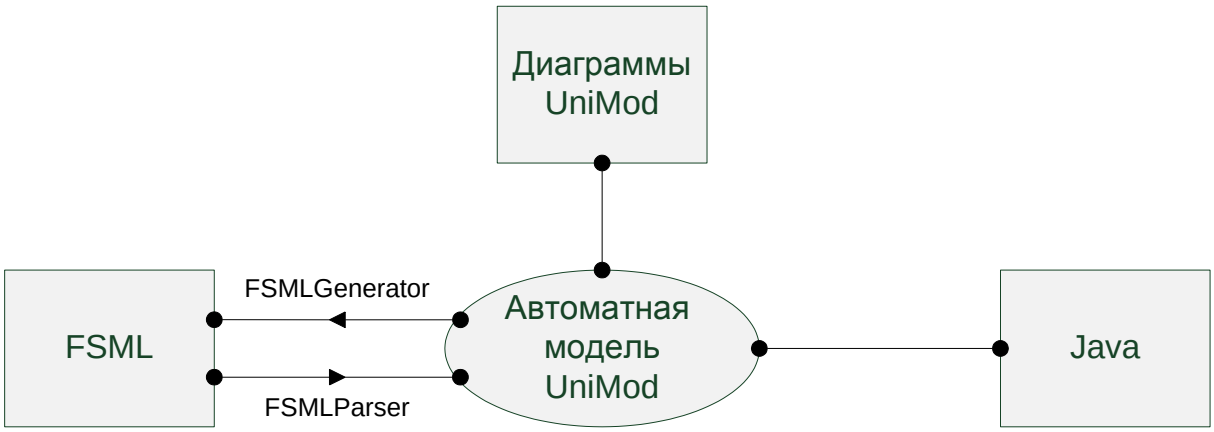

Рис. 4. Схема интеграции языка *FSML* с объектно-ориентированным кодом

В языке *FSML* существует два варианта повторного использования кода:

- вложенные автоматы позволяют повторно использовать компоненты логики;

- поставщики событий и объекты управления на языке *Java* позволяют использовать стандартные техники объектно-ориентированного программирования для повторного использования кода.

Краткость и понятность синтаксиса языка является простым, но немаловажным фактором. В целом, от него зависит эффективность использования языка автоматного программирования.

Описание редактора языка

Редактор состоит из лексического и синтаксического анализаторов, генератора объектных автоматных моделей, систем валидации и автоматического завершения ввода. В перспективе планируется добавить отладчик кода программы и генератор *fsml*-программ из объектной автоматной модели.

Рассмотрим более подробно каждый элемент. *Лексический анализатор* (*FSMLLexer*) осуществляет чтение входной цепочки символов и их группировку в элементарные конструкции, называемые лексемами. *Синтаксический анализатор* (автомат *FSML*) выполняет разбор исходной программы, используя поступающие лексемы, а также семантический анализ программы. *Генератор объектных автоматных моделей* (*FSMLToModel*) строит конечное представление автомата, сохраненного в *fsml*-программе. *Система валидации* проверяет код *fsml*-программы на наличие синтаксических и семантических ошибок. *Система автодополнения* (автоматического завершения ввода) предоставляет пользователю список строк, при добавлении которых редактируемая программа будет синтаксически верна. *Отладчик* предоставляет средства для интерактивного поиска нетривиальных семантических ошибок. *Генератор fsml*-программ осуществляет обратную связь, преобразуя отображаемые визуально объектные модели автоматов в программы на языке *FSML*. Это позволит пользователю полноценно редактировать конечные автоматы как в текстовом представлении, так и в визуальном.

Теперь рассмотрим функции редактора языка *FSML* и соответствующие им проектные решения. Структура редактора изображена на рис. 5.

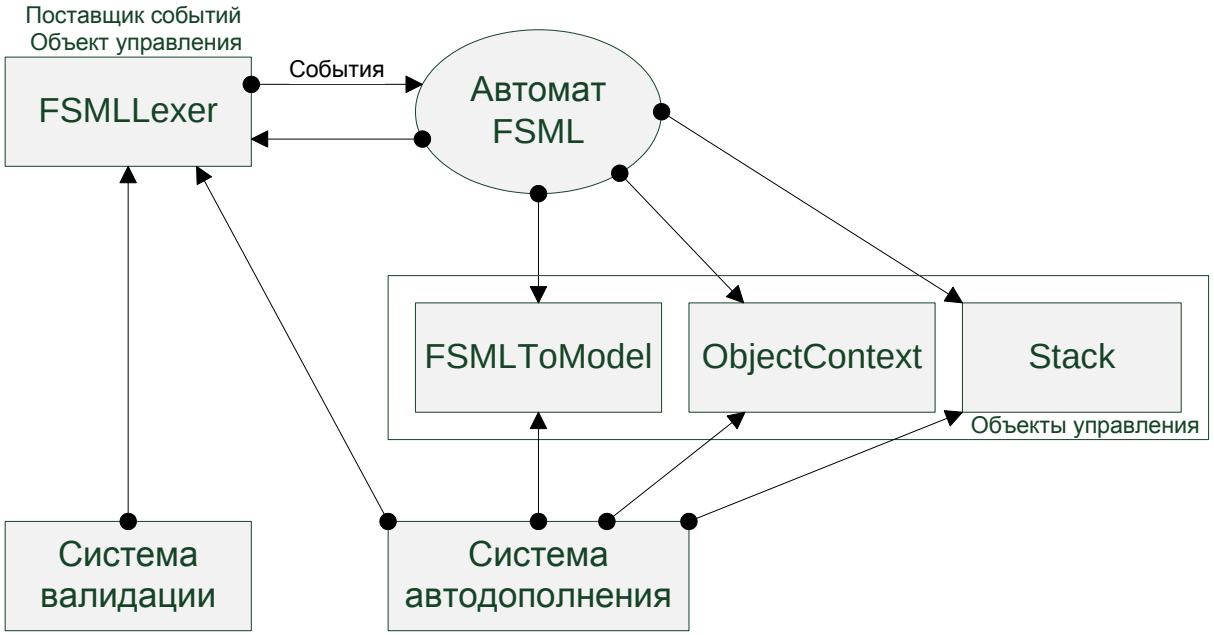

Рис. 5. Структура редактора языка *FSML*

При построении редактора языка автоматного программирования использовались автоматы.

1. Основной функцией данного редактора является *построение автоматной модели по программе на языке FSML*. Автоматная модель – это внутреннее представление диаграмм состояний автоматов и схемы их связей с поставщиками событий и объектами управления. Для реализации этой функции редактор языка *FSML* включает в себя синтаксический анализатор (автомат *FSML*) и генератор объектной автоматной модели (*FSMLToModel*). Эти средства естественно представляются в виде событийной системы. Поэтому было решено реализовать их на основе автоматного подхода с использованием пакета *UniMod* [4]. Синтаксический анализатор – это система, которая получает события в виде поступающих лексем и управляет генератором объектной модели. Таким образом, синтаксический анализатор реализован в виде конечного автомата *FSML*, лексический анализатор – в виде поставщика событий-лексем *FSMLLexer*, генератор объектной автоматной модели – в виде объекта управления *FSMLToModel* для этого автомата. Для выполнения рассматриваемой функции автомату также требуются вспомогательные данные, реализованные в виде объектов управления *ObjectContext* и *Stack*.
2. Дополнительная функция редактора – *валидация программы*. Она позволяет автоматически находить ошибки в программе. Эта система использует данные, генерируемые лексическим и синтаксическим анализаторами, поэтому для нее достаточно единственной зависимости – от объекта *FSMLLexer*.
3. Система автодополнения обеспечивает *автоматическое завершение ввода пользователя*. Автоматный подход значительно упрощает реализацию этой функции. Для этого используется построенный автомат, который позволяет получить набор ожидаемых лексем в каждом состоянии. Поэтому система автодополнения имеет зависимости от объектов управления автомата, хранящих необходимые данные. Кроме того, для обработки ошибок автомат дополняется всеми недостающими переходами с помощью алгоритма, предложенного в работе [15].

Заключение

Итак, создан текстовый язык *FSML* для представления конечных автоматов и реализован редактор этого языка *FSMLEditor*, предназначенный для использования совместно с инструментальным средством *UniMod*. В совокупности эти две компоненты имеют существенные преимущества перед существующими средствами автоматного программирования:

- текстово-визуальный подход к описанию конечных автоматов;
- естественное сочетание автоматного и объектно-ориентированного подходов при разработке программ;
- современные средства для работы с кодом, такие как валидация ошибок и автоматическое завершение ввода.

Отметим, что текстовый редактор *FSMLEditor* реализован с использованием инструментального средства *UniMod*. Таким образом, на основе автоматного подхода создан эффективный инструмент автоматного программирования.

Литература

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – Режим доступа: http://is.ifmo.ru/books/switch/1

2. Шалыто А.А., Туккель Н.И. SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. – 2001. – № 5. – С. 45–62. – Режим доступа: http://is.ifmo.ru/works/switch
3. Шалыто А.А. Парадигма автоматного программирования / Международная научно-техническая мультиконференция «Проблемы информационно-компьютерных технологий и мехатроники». Материалы международной научно-технической конференции «Многопроцессорные вычислительные и управляющие системы». Т. 1. – Таганрог: НИИМВС, 2007. – С. 191–194.
4. Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А. UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. – 2004. – № 6. – С. 12–17. – Режим доступа: http://is.ifmo.ru/works/UML-SWITCH-Eclipse.pdf
5. Решетников Е. О. Инструментальное средство для визуального проектирования автоматных программ на основе Microsoft Domain-Specific Language Tools. Бакалаврская работа / СПбГУ ИТМО. – 2007.
6. Решетников Е.О. Инструментальное средство для визуального проектирования автоматных программ на основе Microsoft Domain-Specific Language Tools. Бакалаврская работа. СПбГУ ИТМО. 2007. – Режим доступа: http://is.ifmo.ru/download/StateMachineDesigner2/doc/StateMachineDesigner2.pdf
7. Среда разработки Eclipse. – Режим доступа: http://www.eclipse.org
8. Язык программирования FSML. – Режим доступа: http://unimod.sourceforge.net/wiki/index.php/FSML
9. Дмитриев С. Языково-ориентированное программирование: следующая парадигма // RSDN Magazine. – 2005. – № 5.
10. Дмитриев С. Языково-ориентирвоанное программирование: следующая парадигма. – Режим доступа: http://www.rsdn.ru/article/philosophy/LOP.xml
11. Цымбалюк Е.А. Текстовый язык автоматного программирования TABP. Магистерская диссертация / СПбГУ ИТМО. 2008.
12. Язык AsmL. – Режим доступа: http://research.microsoft.com/fse/asml
13. Язык SMC. – Режим доступа: http://smc.sf.net
14. Степанов О.Г., Шалыто А.А., Шопырин Д.Г. Предметно-ориентированный язык автоматного программирования на базе динамического языка Ruby // Информационно-управляющие системы. – 2007. – № 4. – С. 22–27. – Режим доступа: http://is.ifmo.ru/works/_2007_10_05_aut_lang.pdf
15. Гуров В. С., Мазин М. А., Шалыто А. А. Текстовый язык автоматного программирования // Тезисы докладов международной научной конференции, посвященной памяти профессора А.М. Богомолова «Компьютерные науки и технологии». – Саратов: СГУ, 2007. – С. 66–69. – Режим доступа: http://is.ifmo.ru/works/_2007_10_05_mps_textual_language.pdf
16. Шамгунов Н.Н. Разработка методов проектирования и реализации поведения программных систем на основе автоматного подхода. Диссертация … канд. техн. наук / СПбГУ ИТМО. – 2004. – Режим доступа: http://is.ifmo.ru/disser/shamg_disser.pdf
17. Гуров В.С., Мазин М.А. Создание системы автоматического завершения ввода с использованием пакета UniMod // Вестник II Межвузовской конференции молодых ученых. Т.1. – СПбГУ ИТМО, 2005. – С. 73–87.