Реализация одного класса мультиагентных систем на текстовых языках автоматного программирования (на примере системы дорожного движения)

Н. Н. Красильников
Бакалавр прикладной математики и информатики
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В статье описывается процесс создания на текстовых языках автоматного программирования одного класса мультиагентных систем на примере системы дорожного движения. Данный проект реализован в виде Java-апплета и опубликован на сайте http://is.ifmo.ru в разделе "Проекты".

Агентами в данной системе являются автомобили, которые передвигаются по улицам с соблюдением правил дорожного движения. Управляющая система автомобиля построена на основе автоматного подхода, причем все автомобили оснащены одной и той же системой управления (автоматом).

Система управления автомобилем реализована тремя способами – на текстовом языке автоматного программирования, описанном в среде MPS (Meta Programming System) компании JetBrains, текстовом языке, для которого в данной работе был построен компилятор, и с использованием конструкции `switch` языка Java.

Введение

Автоматное программирование — стиль программирования, основанный на применении конечных автоматов для описания поведения программ.

Конечные автоматы, как правило, представляются в виде графов переходов.

В этой работе рассмотрены различные виды программной реализации автоматов:

- на первом текстовом языке автоматного программирования, описанном в среде *MPS* (*Meta Programming System*) компании *JetBrains*;
- на втором текстовом языке, для которого в данной работе был построен компилятор;
- с использованием конструкции `switch` языка *Java*.

Задача

В работе реализуется мультиагентная система дорожного движения.

Агентами в данной системе являются автомобили, которые передвигаются по улицам с соблюдением правил дорожного движения (ПДД). В данной работе взята упрощенная схема такого движения: дороги имеют по одной полосе движения в каждую сторону, которые разделены двойной полосой (рис. 1).

Управляющая система автомобиля построена на основе автоматного подхода, причем все автомобили оснащены одной и той же системой управления (автоматом).

Система управления автомобилем реализована тремя перечисленными выше способами.

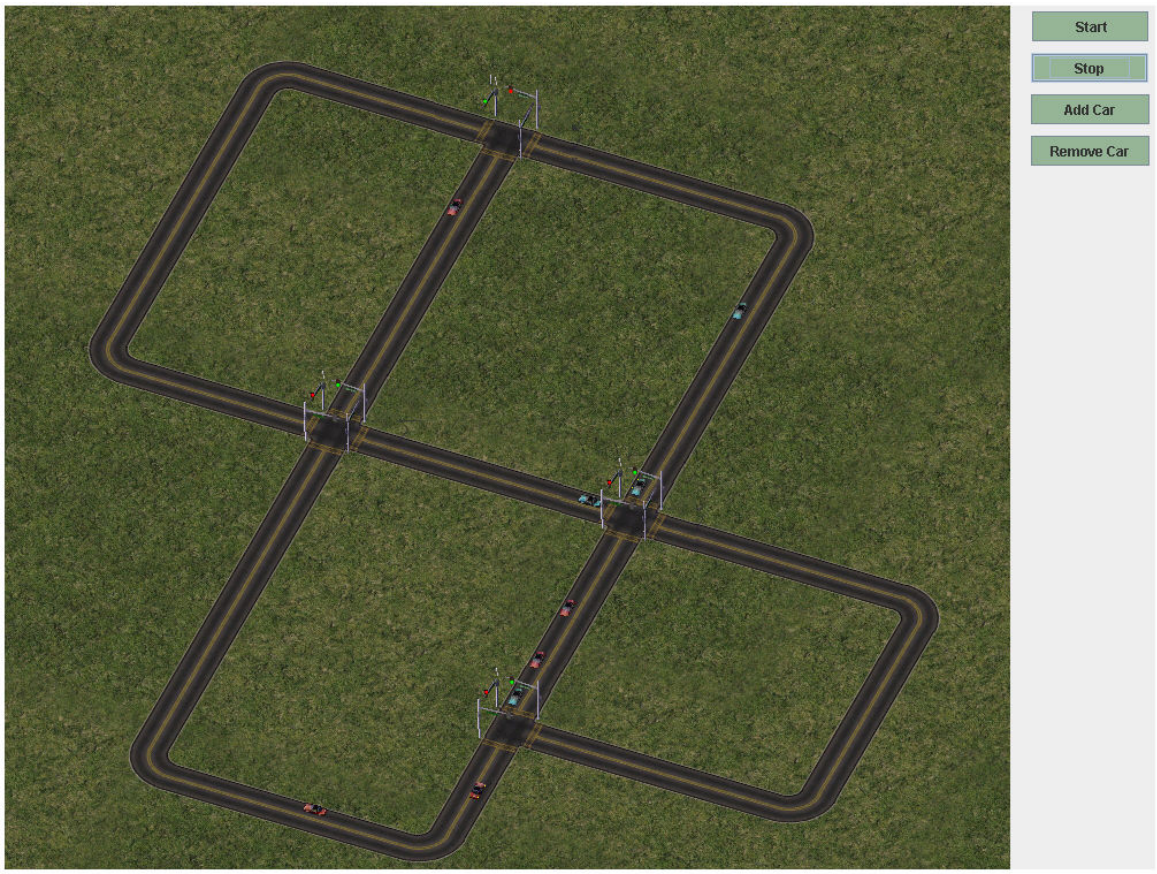

Рис. 1. Машины на дороге

Текстовые языки автоматного программирования

Текстовые язык автоматного программирования, описываемые далее, предназначены для формальной записи автоматов в виде близком к представлению автоматов с использованием конструкции `switch` языка *Java* [1, 2].

При использовании этой конструкции реализация автоматов должна быть изоморфна графу переходов. Однако на практике по коду программы тяжело понять действительно ли это так. Компилятор не способен определить, что во время реализации была допущена ошибка. Например, можно пропустить всего один оператор `break` и программа станет некорректной.

Реализация автомата на рассматриваемых языках автоматного программирования позволяет гарантировать изоморфность между графом переходов и программой, избавляя, тем самым, программиста от многих ошибок. При этом отметим, что можно было и раньше генерировать код автомата по графу переходов (например, с помощью инструментального средства *UniMod* [3]), значительно уменьшая вероятность подобных ошибок. Однако графическое описание автоматов не является единственным, и их текстовое представление может быть весьма полезным для практического использования.

Поскольку рассматриваемые языки автоматного программирования не обладают вычислительной мощностью машины Тьюринга, то писать программы только на нем нельзя. Для того, чтобы написать программу в целом, необходима интеграция с каким-либо универсальным языком программирования. В данном случае это язык *Java*.

Первый текстовый язык автоматного программирования

В последнее время все популярней становятся языки программирования предметной области **(***Domain-Specific Programming Language, Domain-Specific Language (DSL)***)** [4–8]. *DSL* – языки программирования, предназначенные для создания формализованных описаний в терминах предметной области. При этом для решения задач из одной предметной области сначала строится специальный язык программирования, основанный на моделях этой области, а затем на этом языке пишутся программы.

Для создания подобных языков разрабатываются различные инструментальные средства. Корпорация *Microsoft* реализует подобные возможности в проекте *Software Factories*, который входит в состав *SDK* к *Visual Studio 2005* [9, 10]. Корпорация *Intentional Software* также разрабатывает подход к созданию *DSL* [11]. Компания *JetBrains* активно работает над плагином к своей среде разработки *IntelliJ IDEA* – Meta Programming System (MPS) [7, 12], который также предназначен для облегчения процесса создания *DSL*.

Традиционно программа пишется в любом текстовом редакторе, а затем компилируется. Одним из этапов компиляции является синтаксический анализ, в ходе которого строится абстрактное синтаксическое дерево.

В *MPS* фаза синтаксического анализа отсутствует, так как пользователь с помощью специального редактора может строить только синтаксически правильные конструкции. Таким образом, на самом деле пользователь как бы сам строит абстрактное синтаксическое дерево.

Рассмотрим текстовый язык автоматного программирования, созданный В. С. Гуровым и М. А. Мазиным [13].

Как уже говорилось, рассматриваемый язык реализован в среде *MPS* (*Meta Programming System*), разработанной в компании *JetBrains*.

Краткое описание синтаксиса

Опишем синтаксис текстового языка автоматного программирования.

Основные конструкции языка:

- **`event <type> <name> () {<statements>}`**

С помощью этой конструкции описываются возможные события.

Например:

```
event void e0 ( ) {<no statements>}
```

- **`initial state <name> {<statements>}`**

Начальное состояние автомата.

Здесь **`<statements>`** – описания переходов и вложенных автоматов**.**

- **`final state <name> {<no statements>}`**

Конечное состояние автомата.

-
```
on <event name> if (this.a.x1( ))
    execute this.a.z1( ),
    execute this.a.z2( )
    transite to <new state name>;
```

Эта конструкция задает переход в состояние **`<new state name>`** при событии **`<event name>`** и входной переменной **`x1.`** Отдельно стоит сказать про конструкцию **`this.a.*( )`**. Предполагается, что в области видимости автомата есть объект **`a`**, в котором реализованы входные и выходные переменные. При этом **`this.a.*( )`** есть обращение к ним.

Создание автомата в среде MPS

Для реализации автомата в среде *MPS* необходимо создать проект и подключить к нему текстовый язык автоматного программирования. После этого в проекте можно создавать новые модели с автоматами внутри.

На рис. 2 показан открытый проект *vehicles* (транспортные средства) с подключенным языком автоматного программирования (*stateMachine).*

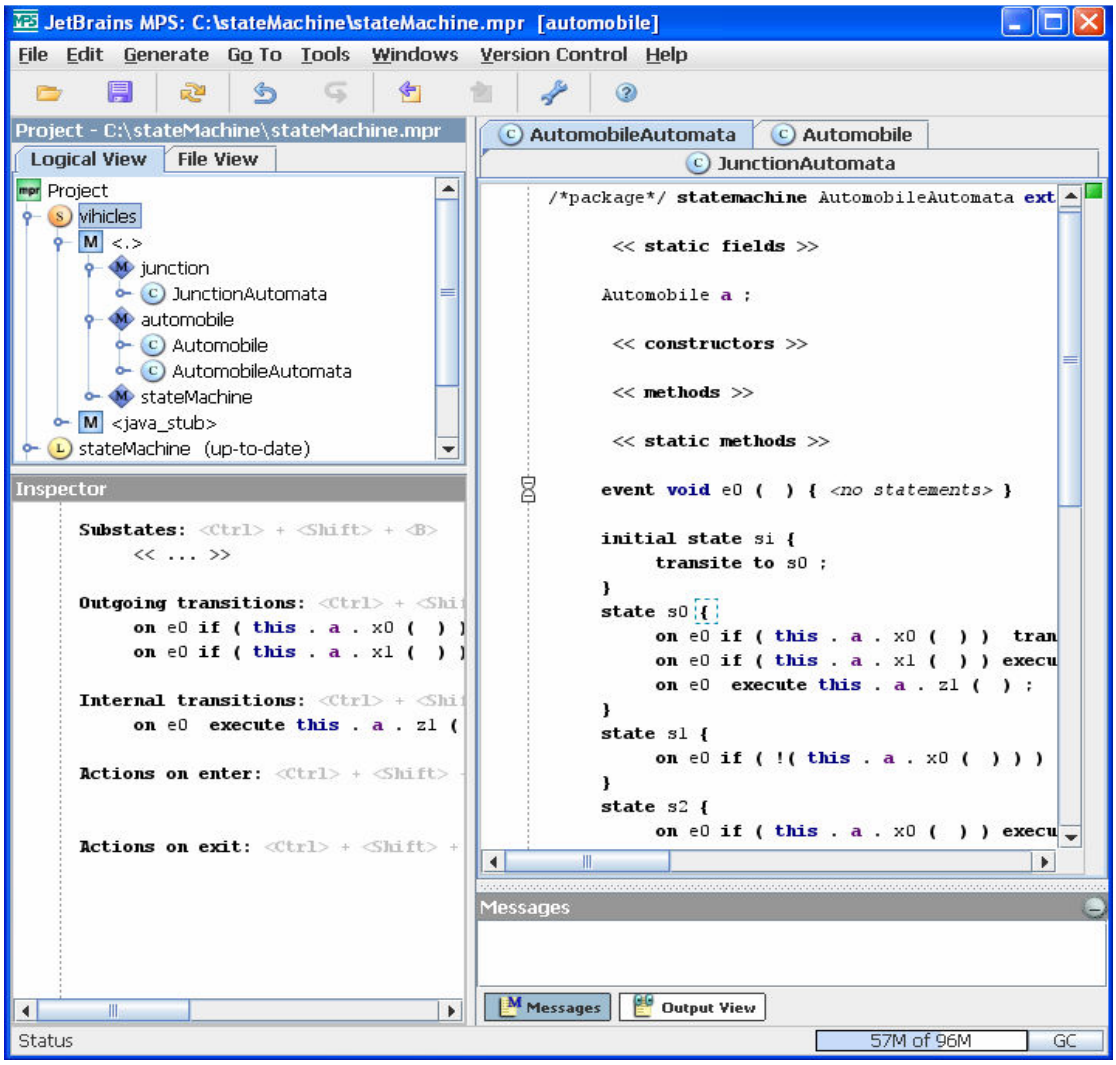

Рис. 2. Создание автомата в среде *MPS*

Моделями являются перекресток (*junction*) и автомобиль (*automobile*). При использовании рассматриваемого языка автомат создается заполнением некоторых шаблонов. Например, в состояние можно добавить конструкцию **“`on * if * execute * transite to *`“** и заполнить соответствующие элементы этого шаблона.

Подробно о создании моделей с помощью *MPS* можно прочесть в работе [4].

Второй текстовый язык автоматного программирования

Приведем лишь краткое описание синтаксиса этого языка, так как с точки зрения семантики он практически полностью дублирует первый.

Краткое описание синтаксиса

В языке предусмотрены следующие конструкции:

- **automata <name>** – объявление автомата, за которым должно следовать его тело, заключенное в фигурные скобки.

- **state <name>** – объявление состояния, за которым в фигурных скобках должно следовать описание его переходов.

- **on <event name> if(<logic statement>) do {<job1>, <job2>,…} go to <state name>;** – описание перехода из текущего состояния в состояние `<state name>` по событию `<event name>`, при выполнении некоторого логического условия `<logic statement>`. При переходе выполняются действия `<job1>,…` В этой конструкции допускается пропуск `if(<logic statement>), do {<job1>, <job2>,…}` и `go to <state name>`.

Основным отличием при использовании данный текстового языка автоматного программирования является то, что он не нуждается в среде *MPS*, а для его использования требуется лишь компактный компилятор.

Компилятор преобразует текст программы на языке автоматного программирования в экземпляр класса языка *Java*.

Автомат управления автомобилем

Алгоритм управления следующий. Исходным является состояние 0 – "*Едем*". В данном состоянии машина перемещается по дороге. Если впереди оказывается другая машина, то автомат переходит в состояние 1 – "*Ждем машину*", и ждет до тех пор, пока она не уедет. За некоторое расстояние до перекрестка автомат переходит в состояние 2 – "*Едем перед перекрестком*", случайным образом выбирается направление поворота, и включаются поворотные огни. Если впереди оказывается другая машина, то автомат переходит в состояние 3 – "*Ждем машину перед перекрестком*", и ждет, пока она не уедет. Так же возможна остановка перед перекрестком, если он занят (состояние 4 – "*Стоим на перекрестке*"). Если перекресток свободен, то машина поворачивает, выбрав состояние соответствующее направлению поворота. После поворота автомат возвращается в исходное состояние 0.

Взаимодействие между автомобилями (мультиагентность) осуществляется через входные переменные `x0` (впереди стоит машина) и `x4` (перекресток свободен). Остальные переменные позволяют автомобилю определять его текущее положение на дороге и определяют направление его дальнейшего движения.

Как отмечено выше, каждый автомобиль управляется автоматом, причем все автоматы являются одинаковыми. Это объясняется ограничениями на движение автомобилей, принятые в данной работе (наличие всего одной полосы движение и необходимость соблюдения ПДД), что делает применение разных автоматов в машинах нецелесообразным.

События

`e0` – прошло время `dt`.

Входные переменные

`x0` – впереди стоит машина.
`x1` – впереди перекресток.
`x2` – перекресток (стоп линия).
`x3` – направление поворота.
`x4` – перекресток свободен.
`x5` – зеленый сигнал светофора.
`x6` – проезд перекрестка закончен.

Выходные воздействия

`z0` – включить (выключить) фары.
`z1` – проехать `dx`.
`z2` – повернуть на `dx`.
`z3` – задать направление поворота.
`z4` – занять перекресток.
`z5` – начать поворот.
`z6` – освободить перекресток.

Граф переходов автомата представлен на рис. 3.

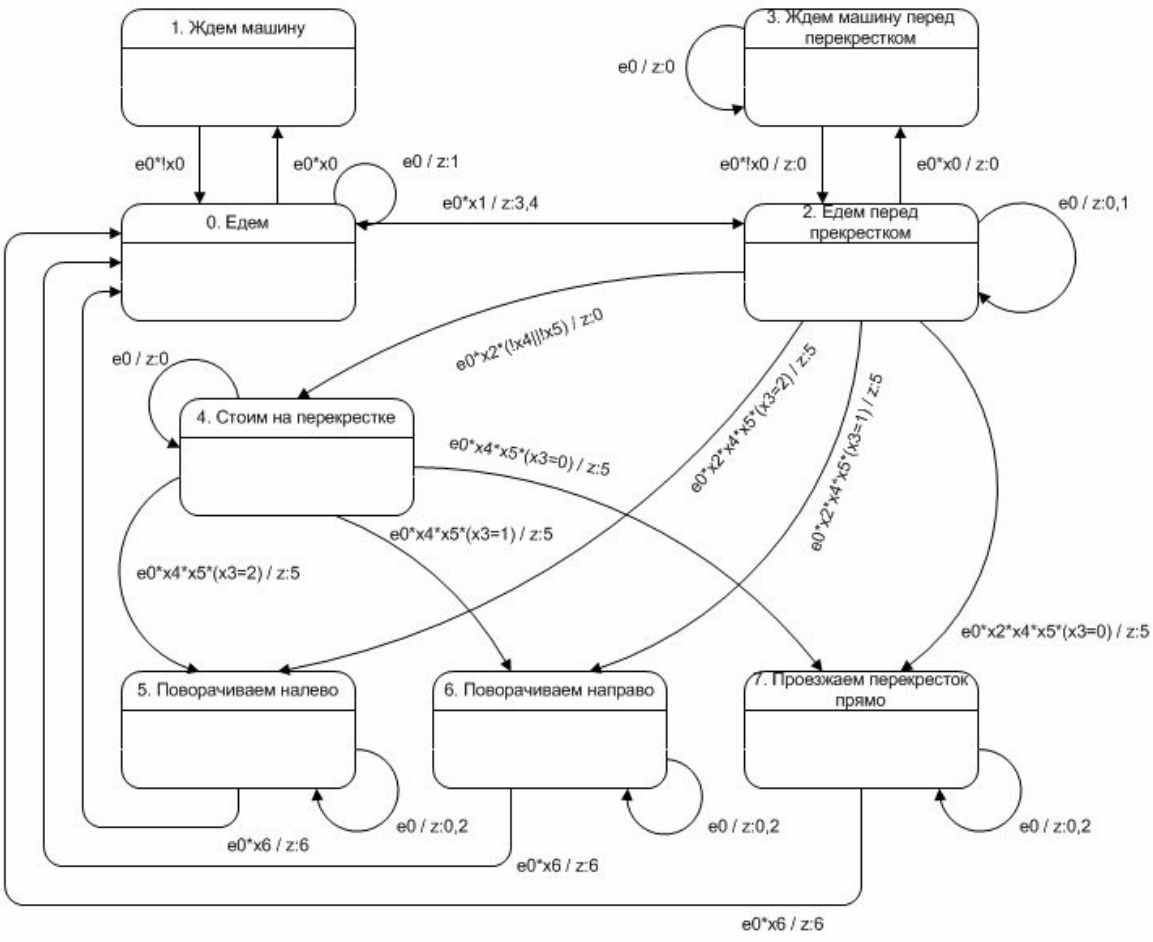

Рис. 3. Граф переходов автомата управления автомобилем

Реализация этого графа переходов на первом текстовом языке автоматного программирования имеет вид:

```
event void e0 ( ) {<no statements>}

initial state si {
    transite to s0;
}
state s0 {
    on e0 if (this.a.x0( )) transite to s1;
    on e0 if (this.a.x1( )) execute this.a.z3( ), execute this.a.z4( )
             transite to s2;
    on e0 execute this.a.z1( );
}
state s1 {
    on e0 if (!(this.a.x0( ))) transite to s0;
}
state s2 {
    on e0 if (this.a.x0( )) execute this.a.z0( ) transite to s3;
    on e0 if (this.a.x2( ) && ((!(this.a.x4( ))) || (!(this.a.x5( )))))
              this.a.z0( ) transite to s4;
    on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 2)) this.a.z5( ) transite to s5;
    on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 1)) this.a.z5( ) transite to s6;
    on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 0)) this.a.z5( ) transite to s7;
}
state s3 {
    on e0 if (!(this.a.x0( ))) execute this.a.z0( ) transite to s2;
    on e0 execute this.a.z0( );
}
state s4 {
    on e0 if ((this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 2)) this.a.z5( ) transite to s5;
    on e0 if ((this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 1)) this.a.z5( ) transite to s6;
    on e0 if ((this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 0)) this.a.z5( ) transite to s7;
    on e0 execute this.a.z0( );
}
state s5 {
    on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
    on e0 execute this.a.z0( ), execute this.a.z2( );
}
state s6 {
    on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
    on e0 execute this.a.z0( ), execute this.a.z2( );
}
state s7 {
    on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
    on e0 execute this.a.z0( ), execute this.a.z2( );
}
```

Граф переходов на втором текстовом языке автоматного программирования реализуется следующим образом:

```
automata MyAuto
{
      state s0
      {
            on e0 if(x0) go to s1;
            on e0 if(x1) do{z3,z4} go to s2;
```

```
            on e0 do{z1};
        }
        state s1
        {
            on e0 if(!x0) go to s0;
        }
        state s2
        {
            on e0 if(x0) do{z0} go to s3;
            on e0 if(x2 && (!x4 || !x5)) do{z0} go to s4;
            on e0 if(x2 && x4 && x5 && (x3 == 2)) do{z5} go to s5;
            on e0 if(x2 && x4 && x5 && (x3 == 1)) do{z5} go to s6;
            on e0 if(x2 && x4 && x5 && (x3 == 0)) do{z5} go to s7;
            on e0 do{z0,z1};
        }
        state s3
        {
            on e0 if(!x0) do{z0} go to s2;
            on e0 do{z0};
        }
        state s4
        {
            on e0 if(x2 && x4 && x5 && (x3 == 2)) do{z5} go to s5;
            on e0 if(x2 && x4 && x5 && (x3 == 1)) do{z5} go to s6;
            on e0 if(x2 && x4 && x5 && (x3 == 0)) do{z5} go to s7;
            on e0 do{z0};
        }
        state s5
        {
            on e0 if(x6) do{z6} go to s0;
            on e0 do{z0,z2};
        }
        state s6
        {
            on e0 if(x6) do{z6} go to s0;
            on e0 do{z0,z2};
        }
        state s7
        {
            on e0 if(x6) do{z6} go to s0;
            on e0 do{z0,z2};
        }
}
```

Из сравнения приведенных текстов программ с графом переходов следует, что текстовое описание в обоих случаях полностью ему изоморфно, если не считать начальный переход для первого языка в состояние `s0`.

Также для сравнения приведем реализацию данного автомата на основе третьего подхода – применения конструкции `switch` языка *Java*:

```
switch (a.y0) {
    case 0:// Едем
        if ((e == 0) && a.x0()) {
            a.y0 = 1;
        } else if ((e == 0) && a.x1()) {
            a.z3();
            a.z4();
            a.y0 = 2;
        } else if ((e == 0)) {
            a.z1();
        }
        break;
```

```
case 1:// Ждем машину
    if ((e == 0) && !a.x0()) {
        a.y0 = 0;
    }
    break;

case 2:// Едем перед  прекрестком
    if ((e == 0) && a.x0()) {
        a.z0();
        a.y0 = 3;
    } else if ((e == 0) && a.x2() && (!a.x4() || !a.x5())) {
        a.z0();
        a.y0 = 4;
    } else if ((e == 0) && a.x2() && a.x4() && a.x5() &&
              (a.x3() == 0)) {
        a.z5();
        a.y0 = 7;
    } else if ((e == 0) && a.x2() && a.x4() && a.x5() &&
              (a.x3() == 1)) {
        a.z5();
        a.y0 = 6;
    } else if ((e == 0) && a.x2() && a.x4() && a.x5() &&
              (a.x3() == 2)) {
        a.z5();
        a.y0 = 5;
    } else if ((e == 0)) {
        a.z0();
        a.z1();
    }
    break;

case 3:// Ждем машину пред перекрестком
    if ((e == 0) && !a.x0()) {
        a.z0();
        a.y0 = 2;
    } else if ((e == 0)) {
        a.z0();
    }
    break;

case 4:// Стоим на перекрестке
    if ((e == 0) && a.x4() && a.x5() && (a.x3() == 0)) {
        a.z5();
        a.y0 = 7;
    } else if ((e == 0) && a.x4() && a.x5() && (a.x3() == 2)) {
        a.z5();
        a.y0 = 5;
    } else if ((e == 0) && a.x4() && a.x5() && (a.x3() == 1)) {
        a.z5();
        a.y0 = 6;
    } else if ((e == 0)) {
        a.z0();
    }
    break;

case 5:// Поворачиваем налево
    if ((e == 0) && a.x6()) {
        a.z6();
        a.y0 = 0;
    } else if ((e == 0)) {
        a.z0();
        a.z2();
    }
```

```
            break;

        case 6:// Поворачиваем направо
            if ((e == 0) && a.x6()) {
                a.z6();
                a.y0 = 0;
            } else if ((e == 0)) {
                a.z0();
                a.z2();
            }
            break;

        case 7:// Проезжаем перекресток прямо
            if ((e == 0) && a.x6()) {
                a.z6();
                a.y0 = 0;
            } else if ((e == 0)) {
                a.z0();
                a.z2();
            }
            break;

        default:

    }
```

Сравнение реализаций автомата управления автомобилем

Реализации с использованием первого текстового языка автоматного программирования требует наличия *MPS*, и тем самым требует обязательного использования строго определенной среды разработки программ.

Реализация при помощи второго текстового языка автоматного программирования требует лишь небольшого компилятора, разработанного автором.

Достоинством этих реализаций является изоморфность графу переходов, что позволяет избавиться от ряда синтаксических ошибок.

Реализация с использованием конструкции `switch` языка *Java* более подвержена ошибкам кодирования, однако она более платформонезависима и позволяет проводить модификацию без привлечения дополнительного инструментария.

Заключение

Традиционно в программировании при использовании текстовых языков программы пишутся непосредственно на них без использования графических моделей. В настоящей работе предлагается качество программ обеспечивать за счет формального и изоморфного перехода от графов переходов, описывающих поведение программ, к их тексту.

На сайте http://is.ifmo.ru/ в разделе “Проекты” можно найти приложение и его исходные коды [14].

Источники

1. *Туккель Н. И., Шалыто А. А.* Switch-технология – автоматный подход к созданию программного обеспечения "реактивных" систем // Программирование. 2001. № 5. http://is.ifmo.ru/works/switch/1
2. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/alg_log.
3. *Инструментальное средство Unimod*. http://unimod.sourceforge.net/intro.html
4. *Wikipedia*. Domain-specific programming language.
http://en.wikipedia.org/wiki/Domain-specific_programming_language,
http://ru.wikipedia.org/wiki/Язык_программирования_предметной_области
5. *Фаулер М.* Языковой инструментарий: новая жизнь языков предметной области (Domain Specific Languages) // Клиент-сервер. 2005. № 3.
http://www.optim.su/cs/2005/3/fowler/fowler.asp
6. *Чернецки К., Айзенкер У.* Порождающее программирование: методы, инструменты, применение. СПб.: Питер, 2005.
7. *Дмитриев С.* Языково-ориентированное программирование: следующая парадигма // RSDN Magazine. 2005. № 5. http://www.rsdn.ru/article/philosophy/LOP.xml
8. *Кириллов Д*. Ориентация на язык //Компьютера. 2006. № 10.
http://offline.computerra.ru/2006/630/258015/
9. *Microsoft Software Factories*. http://msdn.microsoft.com/vstudio/teamsystem/workshop/sf
10. *Гринфилд Дж., Шорт К., Кук С.* Фабрики разработки программ: Потоковая сборка типовых приложений, моделирование, структуры и инструменты. М.: Диалектика, 2006.
11. *Intentional Software.* http://intentsoft.com/
12. *MPS от JetBrains*. http://www.jetbrains.com/mps/
13. *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. http://is.ifmo.ru/works/_2007_10_05_mps_textual_language.pdf
14. *Красильников Н. Н., Шалыто А. А.* Мультиагентная система дорожного движения. СПбГУ ИТМО. 2007. http://is.ifmo.ru/projects/vehicles2/