Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

О. М. Коломейцева, А. А. Шалыто

# Преимущества автоматического синтеза программ на языке *JavaScript* по автоматной спецификации (на примере реализации элемента управления *ToolTip*)

Программирование с явным выделением состояний

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2007

# Оглавление

# Введение

В последнее время программирование с явным выделением состояний все чаще применяется при реализации систем со сложным поведением. Особенно это актуально для платформ с большим числом разнообразных событий. Так, в работах [1, 2, 3] описывается применение конечного автомата для разработки элемента управления *ToolTip* на языке *JavaScript*. В нем за счет применения автомата достаточно просто обеспечивается реализация анимации – медленно всплывающей подсказки. При этом отметим, что этот элемент применяется в различных браузерах.

Описанный в этих работах табличный способ описания автоматов в проектировании обладает недостатками, указанными в следующем разделе. Кроме того, построение кода по модели в этих работах осуществляется вручную. Это может приводить к несоответствию документации и кода, как это имело место, в указанных работах, когда автор изменил код, но не внес все изменения в модель.

Данная работа призвана устранить эти недостатки, что демонстрируется на том же примере. При этом для генерации кода по графу переходов на языке *JavaScript* описан разработанный шаблон *XSLT* для инструментального средства *MetaAuto* [4].

В заключение раздела отметим, что вопрос о применении автоматов для обеспечения рассматриваемого типа анимации был ранее рассмотрен в работе [5].

Отметим также, указанные работы опубликованы на сайте корпорации *IBM* и только в 2007 году, что свидетельствует о том, что применение автоматов в рассматриваемой области не так очевидно, как кажется многим.

# 1. Автоматное программирование в *JavaScript*

Дисциплина, которая диктуется конечными автоматами, придает проекту строгость и наглядность. При этом программы становятся проще в реализации и тестировании. По традиции, конечные автоматы считаются полезными в разработке таких непохожих программ, как сетевые драйверы и компиляторы. Работы на сайте http://is.ifmo.ru/ показывают, что сфера применения автоматов значительно шире. В равной степени они могут быть полезны при разработке приложений на базе браузеров.

В работе [1] показано, как спроектировать поведение динамического элемента управления *ToolTip* при помощи графического и табличного представлений конечного автомата. Приведена реализация табличного представления конечного автомата на языке *JavaScript*.

В данной работе используется другой подход – автомат задается графически в виде графа переходов, изображенного с помощью редактора *MS Visio.* На языке *XSLT* строится шаблон, который позволяет преобразовать указанный граф в скелет программы на языке *JavaScript.* Это преобразование выполняется с помощью инструментального средства *MetaAuto* [4].

Реализация табличного представления имеет ряд недостатков.

1. Графическое представление является более наглядным по сравнению с табличным. Генерация автомата – творческий процесс, поэтому представление автомата должно быть максимально удобным для визуального анализа. В табличном представлении трудно отслеживать циклы и проводить анализ автомата.
2. Дублирование ячеек таблицы. Если у графа переходов есть переход из группового состояния в другое состояние по разным событиям, то у таблицы состояний появится несколько дублирующихся клеток.
3. При большом числе событий таблица становится очень длинной и трудночитаемой.

# 2. Реализация элемента управления *ToolTip*

## 2.1. Постановка задачи

Большинство современных приложений с графическим интерфейсом могут кратковременно отображать маленькие текстовые окна, содержащие полезные описания или инструкции, в случае если курсор оказывается над некоторыми визуальным элементами, такими как картинка, поле ввода или кнопка. Многие web-браузеры, такие как *Microsoft Internet Explorer*, *Opera*, *Mozilla Firefox*, *Netscape Navigator* отображают всплывающие подсказки для любого *HTML*-элемента, который имеет атрибут `title`.

Текстовые подсказки в этих браузерах появляются и исчезают, когда перемещается курсор над элементами. Текстовые окна содержат простой текст без какого-либо форматирования или стиля. Они появляются после короткой паузы в перемещении курсора и внезапно исчезают либо по истечении некоторого произвольного времени, либо после того, как курсор сместится с *HTML*-элемента, либо после нажатия какой-либо клавиши.

Браузер никогда не отображает более одного текстового окна за один раз. Вид и поведение этих подсказок жестко встроены в браузер и изменить их нельзя.

В работе [1] автор с использованием автоматов предлагает реализацию более совершенных подсказок. Они постепенно появляются, а впоследствии постепенно исчезают из виду, а не выскакивают и пропадают. Они содержат текст и изображения, форматирование и стиль. Когда они видимы, они перемещаются вместе с курсором. Эффект плавного исчезновения изменяет направление на обратное (с исчезновения на появление), когда курсор смещается с *HTML*-элемента, и становится обычным, когда курсор возвращается на *HTML*-элемент. Параллельно могут отображаться несколько подсказок, одни из них исчезают, пока другие появляются.

В настоящей работе предполагается реализовать механизм подсказок, используя программирование с явным выделением состояний с применением автоматической генерации кода. Требуется реализовать механизм подсказок, соответствующий работе [1]: подсказка должна иметь возможность применяться к любому элементу *HTML*-страницы и включать в себя любой *HTML*-код, включая картинки, таблицы и форматированный текст.

## 2.2. Методика разработки элемента управления на языке *JavaScript*

В работе предложен следующий подход к построению элемента управления *ToolTip* на языке *JavaScript*:

1. Постановка задачи.
2. Выделение источников событий и состояний для построения автомата.
3. Построение схемы связей автомата: выделение входных и выходных переменных.
4. Построение графа переходов автомата в графическом редакторе *Visio.*
5. Генерация файла, представляющего автомат, на языке *JavaScript* при помощи инструментального средства *MetaAuto* [4].
6. Реализация входных и выходных переменных на языке *JavaScript.*

Заметим, что для исправления неточностей построенного автомата, возникших при его проектировании, найденных при его тестировании или выявленных в связи с

особенностями работы некоторых браузеров потребуется лишь исправить граф переходов, представленный в формате *Visio* и заново сгенерировать из него файл на языке *JavaScript* при помощи инструментального средства *MetaAuto* [4]. Это потребует минимальные временные затраты и позволит избежать ошибок, которые могут возникать при исправлении больших объемов кода.

Стоит также отметить, что таким образом удобно реализовывать не только элемент управления *ToolTip*, но и другие элементы управления.

## 2.3. Источники событий

В качестве источников событий, как и в статье [1], выступают события `mouseMove`, `mouseOver` и `mouseOut` контролируемого элемента, а также события таймеров – `timeout` и `timetick`.

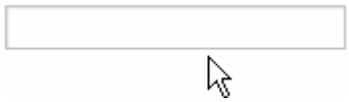
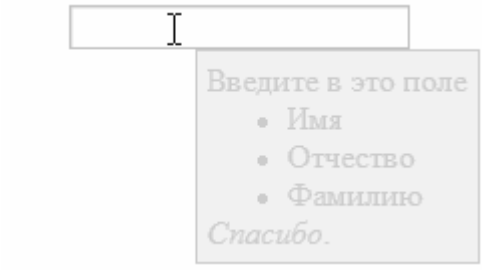
Событие `mouseOver` происходит каждый раз, когда курсор мыши попадает на контролируемый элемент. Событие `mouseOut` происходит при перемещении курсора за определенный объект. Событие `mouseMove` происходит, когда пользователь перемещает курсор.

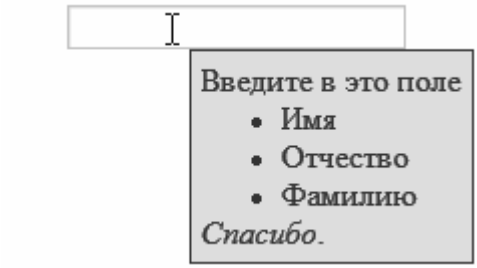
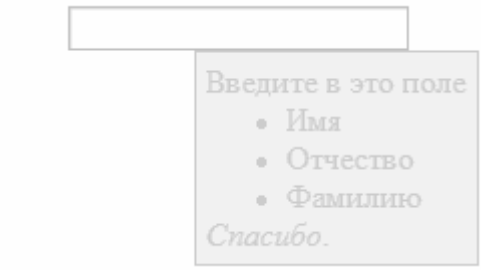
В языке *JavaScript* есть два типа таймеров: разовые таймеры, которые генерируют событие `timeout` по истечении установленного времени, и тикеры, которые генерируют события `timetick` периодически.

## 2.4. Словесное описание автомата *ToolTip*

Автомат *ToolTip* имеет пять состояний, описанных в табл. 1.

**Таблица 1**

| № | Название | Изображение элемента | Описание |
|---|---|---|---|
| 0 | Inactive |  | Состояние, при котором конечный автомат *ToolTip* будет ожидать активации наведением курсора мыши. |
| 1 | Pause |  | Состояние ожидания, пока не поймем, что курсор задержался над контролируемым элементом на достаточно продолжительное время. |
| 2 | Fade in |  | Состояние ожидания, пока выполняется анимация эффекта постепенного появления. |

| | | | |
|---|---|---|---|
| 3 | Display |  | Состояние отображения подсказки. Ожидание выполнения анимационного эффекта постепенного исчезновения. |
| 4 | Fade out |  | Состояние ожидания, пока выполняется анимация эффекта постепенного исчезновения. |

## 2.5. Автомат *ToolTip*

### 2.5.1. Схема связей автомата *ToolTip*

В разд. 2.3 описаны источники событий, изображенные на рис. 1.

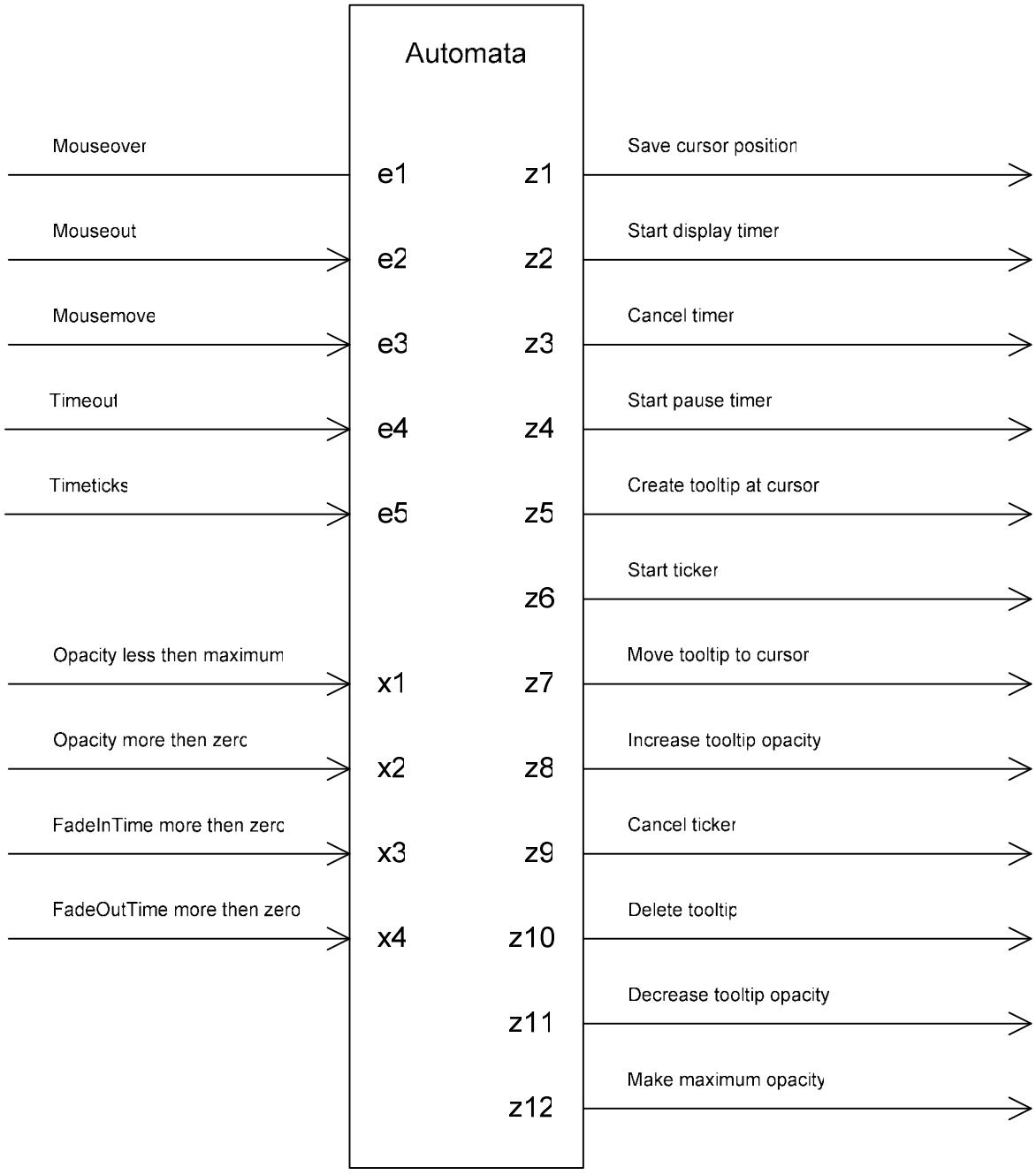



**Рис. 1. Схема связей автомата *ToolTip***

В табл. 2 опишем выходные переменные.

**Таблица 2**

| | | |
|---|---|---|
| z1 | `function(){`<br>`  this.lastCursorX = this.htmlEvent.clientX;`<br>`  this.lastCursorY = this.htmlEvent.clientY;`<br>`}` | Сохраняет новую позицию курсора при его перемещении. |
| z2 | `function(){`<br>`  var self = this;`<br>`  this.currentTimer = setTimeout(`<br>`  function() { self.handleEvent(4, null ); },`<br>`  this.displayTime*1000);`<br>`}` | Запуск таймера ожидания выполнения анимационного эффекта постепенного исчезновения. |
| z3 | `function(){`<br>`  if (this.currentTimer)`<br>`  clearTimeout(this.currentTimer);`<br>`  this.currentTimer = null;`<br>`}` | Сброс таймера. |
| z4 | `function(){`<br>`  var self = this;`<br>`  this.currentTimer = setTimeout(`<br>`  function() { self.handleEvent(4, null ); },`<br>`  this.pauseTime*1000);`<br>`}` | Запуск таймера ожидания выполнения анимационного эффекта постепенного появления. |
| z5 | `function(){`<br>`  this.tooltipDivision =`<br>`  document.createElement("div");`<br>`  this.tooltipDivision.innerHTML =`<br>`  this.tooltipContent;`<br>`  if (this.tooltipClass) {`<br>`  this.tooltipDivision.className =`<br>`  this.tooltipClass;`<br>`  } else {`<br>`  this.tooltipDivision.style.minWidth = "25px";`<br>`  this.tooltipDivision.style.maxWidth =`<br>`  "350px";`<br>`  this.tooltipDivision.style.height = "auto";`<br>`  this.tooltipDivision.style.border =`<br>`  "thin solid`<br>`  black";`<br>`  this.tooltipDivision.style.padding = "5px";`<br>`  this.tooltipDivision.style.backgroundColor =`<br>`  "#CFCFCF";`<br>`  }`<br><br>`  this.tooltipDivision.style.position =`<br>`"absolute";`<br>`  this.tooltipDivision.style.zIndex = 101;`<br>`  this.tooltipDivision.style.left =`<br>`this.lastCursorX`<br>`  + (document.body.scrollLeft`<br>`  - document.body.clientLeft)`<br>`  + this.tooltipOffsetX;`<br>`  this.tooltipDivision.style.top =`<br>`this.lastCursorY`<br>`  + (document.body.scrollTop`<br>`  - document.body.clientTop)`<br>`  + this.tooltipOffsetY;`<br>`  this.currentOpacity = 0;`<br>`  this.tooltipDivision.style.opacity = 0;`<br>`  if (this.tooltipDivision.filters)`<br>`  this.tooltipDivision.style.filter =`<br>`  "alpha(opacity=0)"; // for MSIE only` | Создает элемент *ToolTip*. |

| | | |
|---|---|---|
| | document.body.appendChild(this.tooltipDivision);<br>} | |
| z6 | function(){<br>  var self = this;<br>  this.currentTicker = setInterval(<br>  function() { self.handleEvent( 5, null ); },<br>  1000/this.fadeRate);<br>  } | Запускает тикер. |
| z7 | function(){<br>  this.tooltipDivision.style.left =<br>  this.htmlEvent.clientX<br>  + (document.body.scrollLeft<br>  - document.body.clientLeft)<br>  + this.tooltipOffsetX;<br>  this.tooltipDivision.style.top =<br>  this.htmlEvent.clientY<br>  + (document.body.scrollTop<br>  - document.body.clientTop)<br>  + this.tooltipOffsetY;<br>} | Меняет расположение элемента *ToolTip* при перемещении курсора. |
| z8 | function(){<br>  this.fadeTooltip(<br>  +this.tooltipOpacity<br>  /(this.fadeinTime*this.fadeRate));<br>} | Увеличивает насыщенность элемента *ToolTip*. |
| z9 | function(){<br>  if (this.currentTicker)<br>  clearInterval(this.currentTicker);<br>  this.currentTicker = null;<br>} | Сброс тикера. |
| z10 | function(){<br>  if (this.tooltipDivision)<br><br>document.body.removeChild(this.tooltipDivision);<br>  this.tooltipDivision = null;<br>} | Удаляет элемент *ToolTip*. |
| Z11 | function(){<br>  this.fadeTooltip(<br>  -this.tooltipOpacity<br>  /(this.fadeoutTime*this.fadeRate));<br>} | Уменьшает насыщенность элемента *ToolTip*. |
| Z12 | function(){<br>  this.fadeTooltip(+this.tooltipOpacity);<br>} | Увеличивает насыщенность элемента *ToolTip* до максимума. |

### 2.5.2. Граф переходов автомата *ToolTip*

На рис. 2 изображен граф переходов автомата *ToolTip*, построенный по словесному описанию из работы [1] .

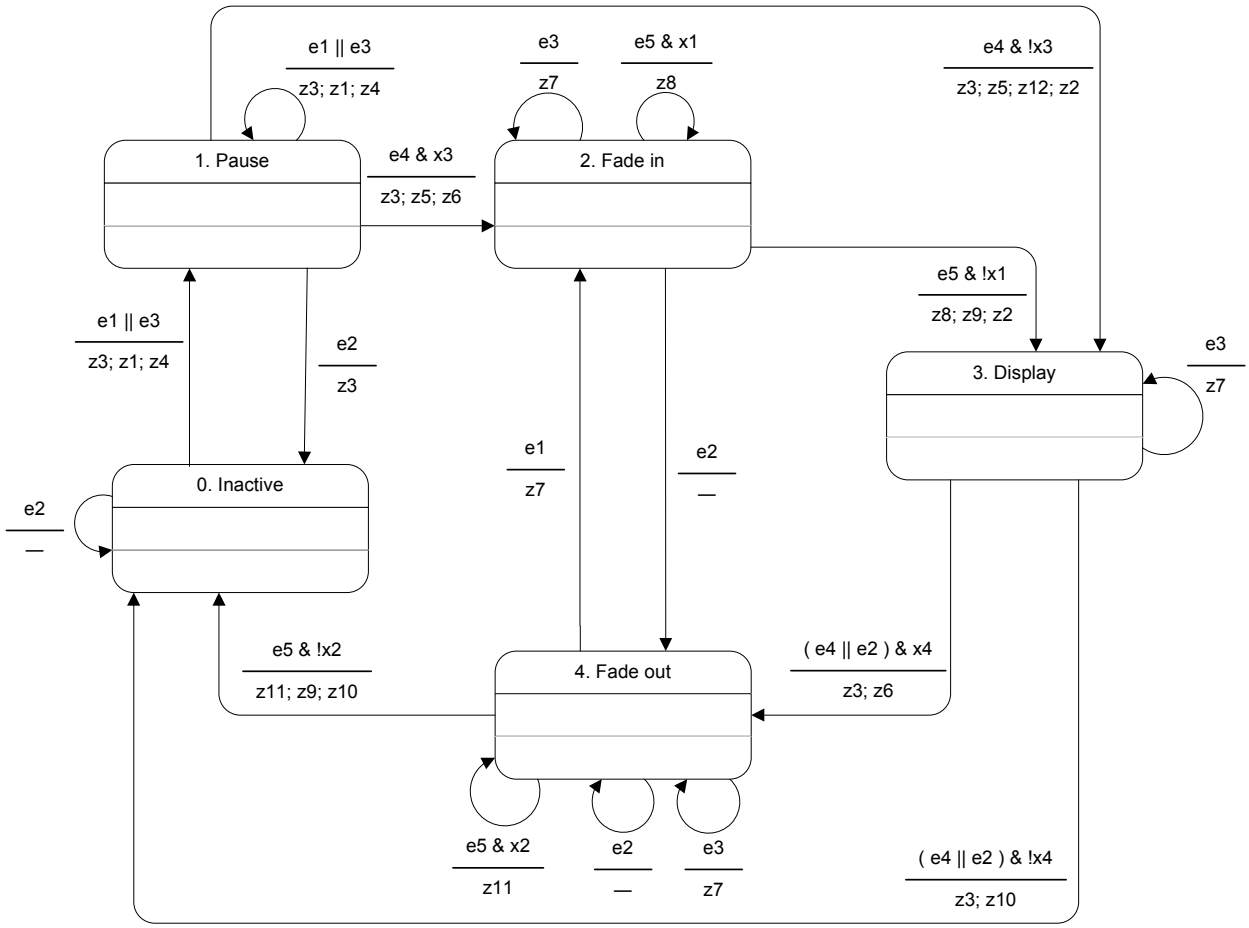



**Рис. 2. Граф переходов автомата *ToolTip***

# 3. Преимущества автоматического синтеза

Достоинством предлагаемого подхода является автоматическая генерация кода. При тестировании автором работы [3] автомата *ToolTip*, у графа переходов, построенного в работе [1], появились дополнительные переходы, связанные с особенностями работы некоторых браузеров, при этом автор изменял код программы, а затем уже обновлял документацию. Эти изменения частично были отражены на графе переходов в работе [3].

При использовании *MetaAuto* требуется лишь добавить новые переходы в граф переходов и сгенерировать новый файл.

На рис. 3 пунктиром показаны те переходы, которые были отображены в работе [3] после тестирования динамического элемента *ToolTip*, и не были отображены в работе [1] на этапе проектирования. Прерывистой линией показаны те переходы, которые не были отображены автором работы [3] в документации, но были использованы непосредственно в коде.

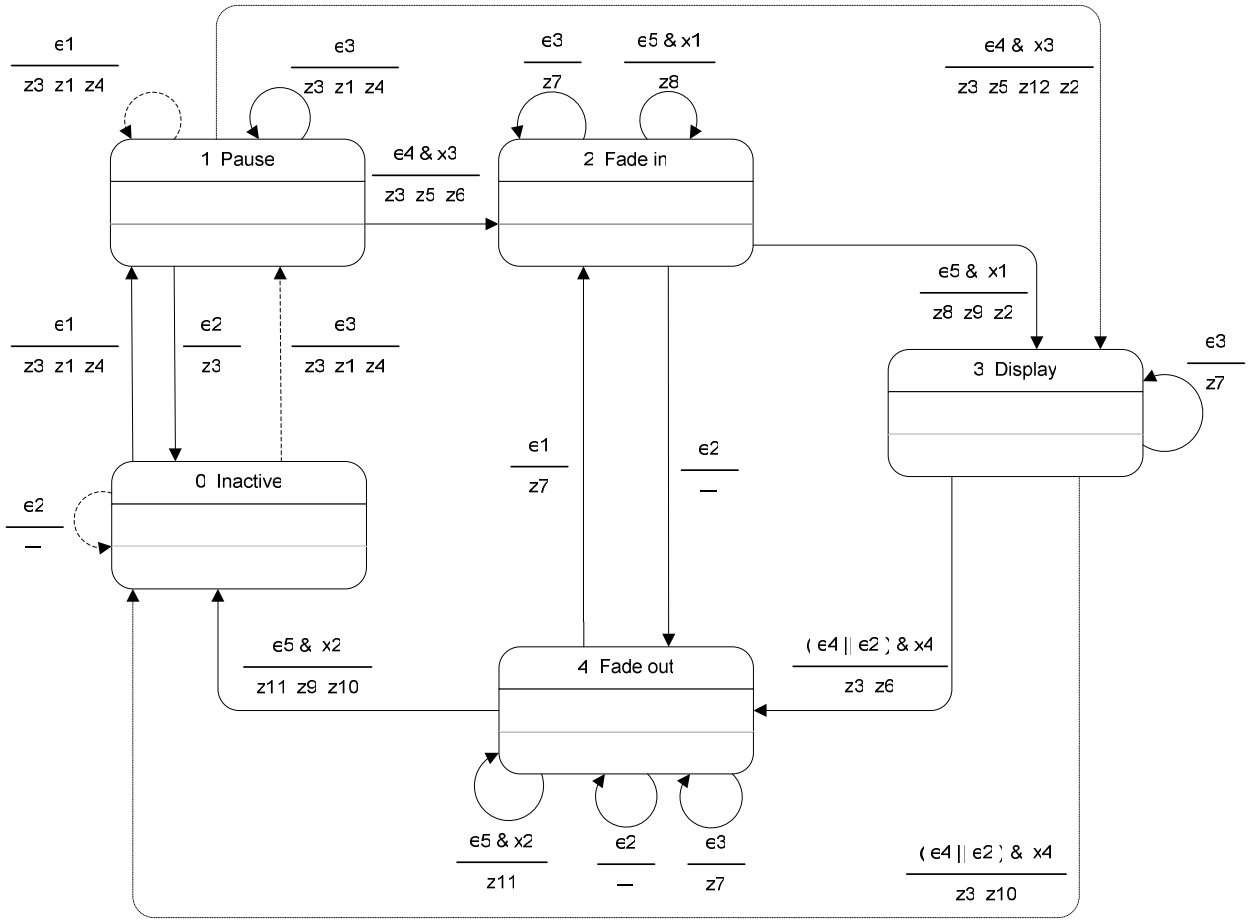



**Рис. 3. Изменения графа переходов**

Таким образом, после тестирования автором работы [1] графа переходов, в документации появились переходы, которые на рис. 3 изображены штриховым пунктиром:

- из состояния ***0. Inactive*** в состояние ***1. Pause*** по событию ***e3. Mousemove****;*
- из состояния ***0. Inactive*** в состояние ***0. Inactive*** по пустому событию;
- из состояния ***1. Pause*** в состояние ***1. Pause*** по событию ***e1. Mouseover***.

Кроме того, в коде, опубликованном в работе [1], произошли изменения, не отображенные в документации [3]. При этом были добавлены переходы, которые на рис. 3 изображены точечным пунктиром:

- из состояния ***1***. ***Pause*** в состояние ***3***. ***Display*** по событию ***e4***. ***TimeOut*** при условии ***!x3***. ***Not*** ( ***FadeInTime more than zero***);
- из состояния ***3***. ***Display*** в состояние ***0***. ***Inactive*** по одному из событий ***e4. TimeOut*** или ***e2. MouseOut*** при условии ***!x***4. ***Not ( FadeOutTime more than zero***).

Это достаточно распространенная ситуация. На этапе тестирования становится необходимым добавление нового перехода в граф переходов, однако, так как документация независима от кода, то изменения зачастую затрагивают лишь код. В нашем подходе при использовании автоматической генерации кода такая ситуация не может произойти.

# 4. Создание шаблона

Шаблон для инструментального средства *MetaAuto* был разработан на основе шаблонов для языка *C++*, описанного в работе [4].

В листинге 1 приведено начало шаблона, отвечающее за объявление требуемых ключей и вывода «шапки» файла.

**Листинг 1. Шаблон для *JavaScript* кода**

```
<?xml version='1.0' ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method='text' indent="no" />
  <xsl:key name="distinctInputVariables"
           match="//conditionNode[@type='INPUT_VARIABLE']"
           use="parameter[@name='name']/@value" />
  <xsl:key name="distinctOutputActions"
           match="//actionNode[@type='OUTPUT_ACTION']"
           use="parameter[@name='name']/@value" />
  <xsl:key name="distinctAutomataCalls"
           match="//actionNode[@type='AUTOMATA_CALL']"
           use="parameter[@name='automata']/@value" />

  <xsl:template match="/model">
    //--- this file is machine generated ---
    //Model: <xsl:value-of select="@name" />

    <xsl:apply-templates select="stateMachine"></xsl:apply-templates>
```

В листинге 2 приведена часть шаблона, отвечающая за генерацию `switch`-блока автомата.

**Листинг 2. Шаблон для *JavaScript* кода**

```
  </xsl:template>
  <!--================================================================-->
  <!--Automata processing-->
  <xsl:template match="stateMachine">
    /// <xsl:value-of select="@description" />
    function A_<xsl:value-of select="@name" />(t, e)
    {
      switch (t.y)
      {
      <xsl:apply-templates select="state//state[count(state) = 0]"
                           mode="SWITCH_BLOCK">
        <xsl:sort select="@name" data-type="text" />
      </xsl:apply-templates>
      }
    }
  </xsl:template>
  <!--End Automata processing-->
  <!--================================================================-->
  <xsl:template match="state//state" mode="SWITCH_BLOCK">
    case <xsl:value-of select="@name" />:
    <xsl:apply-templates
      select="ancestor::stateMachine/transition
              [current()/ancestor-or-self::state/@name = @sourceRef]"
      mode="SWITCH_BLOCK">
```

```
      <xsl:sort select="count(condition)"
                data-type="number"
                order="descending" />
      <xsl:sort select="string-length(@priority) = 0"
                data-type="text"
                order="ascending" />
      <xsl:sort select="@priority" data-type="number" />
      <xsl:sort select="@targetRef" data-type="text" />
    </xsl:apply-templates>
    break;
  </xsl:template>
  <!--================================================================-->
  <!-- Transitions processing for creating switch block -->
  <xsl:template match="transition[not(count(./condition) = 0)]"
                mode="SWITCH_BLOCK">
    <xsl:choose>
      <xsl:when test="position() = 1">
        if (<xsl:apply-templates select="condition"
                                 mode="SWITCH_BLOCK" />)
        {<xsl:apply-templates select="outputAction"
          mode="SWITCH_BLOCK" />
          t.y = <xsl:value-of select="@targetRef" />;
        }
      </xsl:when>
      <xsl:otherwise>
        else if (<xsl:apply-templates select="condition"
                                      mode="SWITCH_BLOCK" />
                )
        {<xsl:apply-templates select="outputAction"
          mode="SWITCH_BLOCK" />
          t.y = <xsl:value-of select="@targetRef" />;
        }
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template match="transition[count(./condition) = 0]"
                mode="SWITCH_BLOCK">
    <xsl:choose>
      <xsl:when test="position() = 1">
        if (true)
        {<xsl:apply-templates select="outputAction"
                              mode="SWITCH_BLOCK" />
          t.y = <xsl:value-of select="@targetRef" />;
        }
      </xsl:when>
      <xsl:otherwise>
        else if (true)
        {<xsl:apply-templates select="outputAction"
                              mode="SWITCH_BLOCK" />
          t.y = <xsl:value-of select="@targetRef" />;
        }
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <!-- End Transitions processing for creating switch block -->
```

Входные и выходные переменные, а также логические операции, входящие в состав условий перехода, описываются в листинге 3.

**Листинг 3. Шаблон для *JavaScript* кода**

```
<!--=============================================================-->
<xsl:template match="condition" mode="SWITCH_BLOCK">
  <xsl:apply-templates mode="SWITCH_BLOCK" />
</xsl:template>
<xsl:template match="outputAction" mode="SWITCH_BLOCK">
  <xsl:apply-templates mode="SWITCH_BLOCK" />
</xsl:template>
<!--=============================================================-->
<!-- Action nodes -->
<xsl:template match="actionNode[@type='OUTPUT_ACTION']"
    mode="SWITCH_BLOCK">
  t.z<xsl:value-of select="@name" />();
</xsl:template>
<xsl:template match="actionNode[@type='AUTOMATA_CALL']"
    mode="SWITCH_BLOCK">
  Call_<xsl:value-of
      select="parameter[@name='automata']/@value" />(
  <xsl:value-of select="parameter[@name='event']/@value" />);
</xsl:template>
<xsl:template match="actionNode" priority="0" mode="SWITCH_BLOCK">
  /* ERROR: Unknown action Node type:
  '<xsl:value-of select="@type" />';*/
</xsl:template>
<!-- End Action nodes -->
<!--=============================================================-->

<!-- Condition nodes and operations -->
<xsl:template match="conditionNode[@type='INPUT_VARIABLE']"
  mode="SWITCH_BLOCK">
  t.x<xsl:value-of select="@name" />()
</xsl:template>
<xsl:template match="conditionNode[@type='EVENT']"
  mode="SWITCH_BLOCK">
  e == <xsl:value-of select="@name" />
</xsl:template>
<xsl:template match="conditionNode" priority="0"
  mode="SWITCH_BLOCK">
  /* ERROR: Unknown action Node type: '
  <xsl:value-of select="@type" />'; Please, correct the xslt file;*/
</xsl:template>
<xsl:template match="binaryOperation[@type='AND']"
  mode="SWITCH_BLOCK">
  (
  <xsl:apply-templates select="child::*[position()=1]"
    mode="SWITCH_BLOCK" />) &amp;&amp;
  (<xsl:apply-templates select="child::*[position()=2]"
   mode="SWITCH_BLOCK" />)
</xsl:template>
<xsl:template match="binaryOperation[@type='OR']"
  mode="SWITCH_BLOCK">
  (<xsl:apply-templates
      select="child::*[position()=1]" mode="SWITCH_BLOCK" />) ||
  (<xsl:apply-templates select="child::*[position()=2]"
   mode="SWITCH_BLOCK" />)
</xsl:template>
<xsl:template match="unaryOperation[@type='NOT']"
  mode="SWITCH_BLOCK">
  !(<xsl:apply-templates select="child::*[position()=1]"
  mode="SWITCH_BLOCK" />)
</xsl:template>
<xsl:template match="binaryOperation" priority="0"
  mode="SWITCH_BLOCK">
```

```
    /*Error: Unknown binary operation type:
    '<xsl:value-of select="@type" />';
    Please, correct the xslt file; */
  </xsl:template>
  <xsl:template match="unaryOperation" priority="0"
    mode="SWITCH_BLOCK">
    /*Error: Unknown unary operation type:
    '<xsl:value-of select="@type" />';
    Please, correct the xslt file; */
  </xsl:template>
  <!-- End Condition nodes and operations -->
  <!--================================================================-->
</xsl:stylesheet>
```

# 5. Использование инструментального средства *MetaAuto* для реализации сложной логики, описываемой автоматами, на языке *JavaScript*

В разд. 2.2 была описана методика разработки элемента управления. Опишем подробнее процесс генерации файла, представляющего автомат, на языке *JavaScript* при помощи инструментального средства *MetaAuto* [4] на примере элемента управления *ToolTip*.

## 5.1. Шаг 1. Изображение графа переходов и схемы связей

Изобразим граф переходов спроектированного автомата в редакторе *MS Visio*. Граф переходов автомата *ToolTip* расположим на странице с названием *ToolTip* (рис. 2). На следующей странице расположим схему связей автомата *ToolTip* (рис. 1). При этом название страницы значения не имеет. Сохраним граф переходов и схему связей в файл *tooltip.vsd*.

## 5.2. Шаг 2. Настройка конфигурационного файла

Конфигурационный файл (*Makefile*) отвечает за последовательность действий, необходимых для преобразования изображения автомата в исходный код. Необходимость редактирования этого файла вызвана тем, что в некоторых проектах может быть применено более одного автомата. Указанный файл предназначен для удобства использования инструментального средства *MetaAuto* и не является необходимым условием применения этого средства. Все действия, описанные в этом файле, можно выполнить вручную.

В настоящей работе используется *Makefile*, представленный ниже.

```
tooltip.js : tooltip.xml javascript.xslt
"Utils\XSLTransform.exe" tooltip.xml javascript.xslt tooltip.js

tooltip.xml : tooltip.vsd
"Visio2Xml\Visio2Xml.exe" tooltip.vsd tooltip.xml
```

Файл *tooltip.js* генерируется с помощью утилиты *"Utils\XSLTransform.exe"* из файлов *tooltip.xml* и *javascript.xslt*. Файл *javascript.xslt* является *XSLT*-шаблоном, описанным в разделе 4. Файл *tooltip.xml* генерируется с помощью утилиты *"Visio2Xml\Visio2Xml.exe"* из файла *tooltip.vsd.*

## 5.3. Шаг 3. Получение исходного кода

Для запуска файла *Makefile* воспользуемся следующей командной строкой:

```
Utils\nmake.exe
```

Получим файл *tooltip.js*, содержащий исходный код на языке *JavaScript*, листинг которого приведен ниже:

```
//--- this file is machine generated ---
//Model: Visio project
/// Автомат для отображения всплывающих подсказок
function A_ToolTip(t, e)
{
 switch (t.y)
 {
  case 0:
   if      (e == 2)  { t.y = 0;}
   else if ((e == 1) || (e == 3))  {t.z3(); t.z1(); t.z4();  t.y = 1;}
  break;

  case 1:
   if      (e == 2)  {t.z3();  t.y = 0;}
   else if (e == 3)  {t.z3(); t.z1(); t.z4();  t.y = 1;}
   else if ((e == 4) && (t.x3()))  {t.z3(); t.z5(); t.z6();  t.y = 2;}
   else if ((e == 4) && (!(t.x3())))  {t.z3(); t.z5(); t.z13(); t.z2();
            t.y = 3;}
  break;

  case 2:
   if      (e == 3)  {t.z7();  t.y = 2;}
   else if ((e == 5) && (t.x1()))  {t.z8();  t.y = 2;}
   else if ((e == 5) && (!(t.x1())))  {t.z8(); t.z9(); t.z2();  t.y = 3;}
   else if (e == 2)  { t.y = 4;}
  break;

  case 3:
   if      (((e == 4) || (e == 2)) && (!(t.x4()))) {t.z3(); t.z11();
             t.y = 0;}
   else if (e == 3)  {t.z10();  t.y = 3;}
   else if (((e == 4) || (e == 2)) && (t.x4()))  {t.z3(); t.z6();  t.y = 4;}
  break;

  case 4:
   if      ((e == 5) && (!(t.x2())))  {t.z12(); t.z9(); t.z11();  t.y = 0;}
   else if (e == 1)  {t.z10();  t.y = 2;}
   else if (e == 3)  {t.z7();  t.y = 4;}
   else if ((e == 5) && (t.x2()))  {t.z12();  t.y = 4;}
   else if (e == 2)  { t.y = 4;}
  break;
 }
}
```

## 5.4. Шаг 4. Реализация входных и выходных переменных

Описание входных и выходных переменных реализовано в файле *FadingToolTip.js*. Более подробно этот вопрос рассмотрен в разд. 2.5.1.

# Заключение

В работе рассмотрена методика построения элемента управления пользовательского интерфейса *ToolTip* на языке *JavaScript* с применением конечных автоматов.

Также на примере реализации этого элемента показаны преимущества такого подхода и автоматической генерации с помощью инструментального средства *MetaAuto* [4] по сравнению с классическими методами программирования и ручной реализацией конечных автоматов.

В ходе работы, для генерации кода на языке *JavaScript* по графу переходов, описан шаблон *XSLT* для языка *JavaScript*. Этот шаблон разработан на основе предложенного в работе [4] шаблона для языка *C++*.

Работа демонстрирует, что программирование с явным выделением состояний хорошо подходит для реализации программ на языке *JavaScript*.

## Источники


1. *Принг Э.* Конечные автоматы в *JavaScript*. Часть 1: Разработаем виджет. http://www.ibm.com/developerworks/ru/library/wa-finitemach1/index.html

2. *Принг Э.* Конечные автоматы в *JavaScript.* Часть 2: Реализация виджета. http://www.ibm.com/developerworks/ru/library/wa-finitemach2/wa-finitemac_ru.html

3. *Pring E. J.* Finite state machines in *JavaScript*. Part 3: Test the widget. http://www.ibm.com/developerworks/java/library/wa-finitemach3/index.html.

4. *Канжелев С. Ю., Шалыто А. А.* Преобразование графов переходов, представленных в формате *MS Visio*, в исходные коды программ для различных языков программирования (инструментальное средство *MetaAuto*). СПбГУ ИТМО. 2005. http://is.ifmo.ru/projects/metaauto/.

5. *Казаков М. А., Шалыто А. А.* Реализация анимации при построении визуализаторов алгоритмов на основе автоматного подхода //Информационно-управляющие системы. 2005. № 4, с. 51–60. http://is.ifmo.ru/works/visanim/