
Автоматическая генерация кода программ с явным выделением состояний

Канжелев С.Ю. *магистрант СПбГУ ИТМО*
Шалыто А.А. *доктор технических наук*
профессор СПбГУ ИТМО

О чем доклад?

- Как описать сложную логику работы приложения.
 - Как преобразовать это описание в код максимально удобным способом.
 - Инструментальное средство *MetaAuto*.
-

Мотивация

- Существует разрыв между фазами проектирования и реализации.
 - В большинстве случаев моделируют статическую часть программы с помощью диаграммы классов.
 - Сложную логику невозможно описать.
 - Диаграммы взаимодействия и последовательности бесполезны.
 - Диаграммы состояний использовать сложно.
-

Что нам нужно?

- Необходимо научиться описывать сложную логику.
 - Необходимо инструментальное средство для преобразования этого описания в исходный код программы.
-

Как описывать сложную логику работы программы

Программирование с явным выделением состояний.

Программа с явным выделением состояний

- Явное выделение состояний:
 - Вместо набора флагов – выделенное состояние.
 - Непредвиденные переходы исключаются.
 - Ускорение тестирования.
-

Описание программы с явным выделением состояний

- Диаграммы состояний UML или аналогичные (графы переходов автоматов).
 - Требуется автоматическая генерация кода или исполнение программы по графам переходов.
-

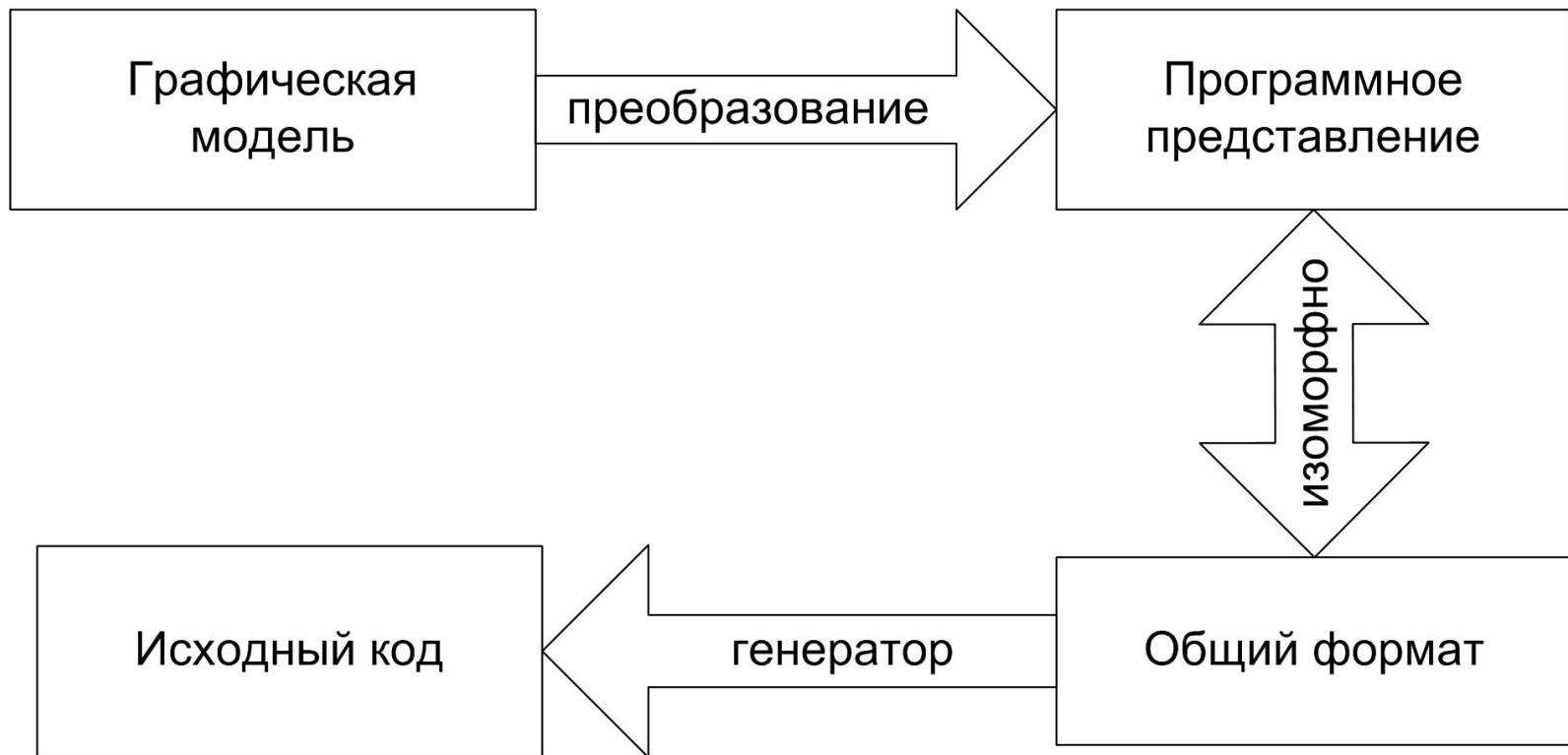
Преимущества программ с явным выделением состояний.

- Облегчение проектирования
 - Облегчение документирования
 - Ускорение процесса тестирования.
-

Аналоги

- Для многих языков программирования не созданы соответствующие инструментальные средства.
 - Существующие инструментальные средства не позволяют настраивать получаемый исходный код.
-

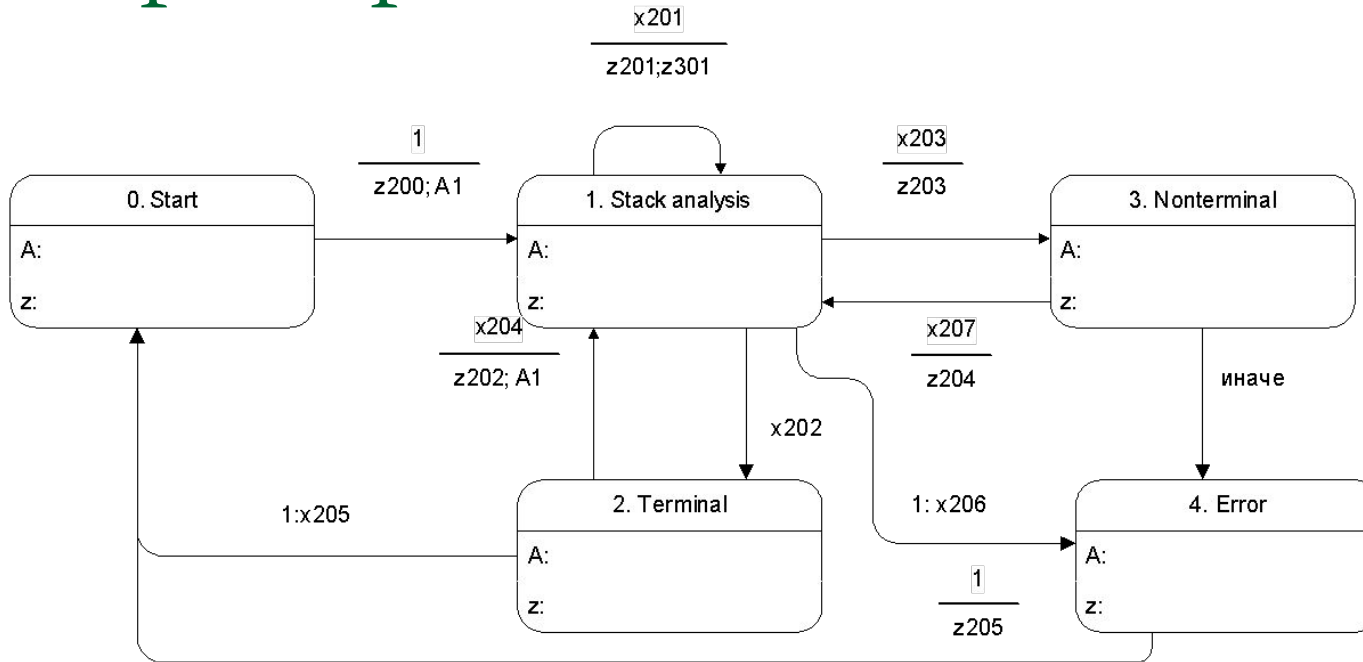
Процесс генерации исходного кода



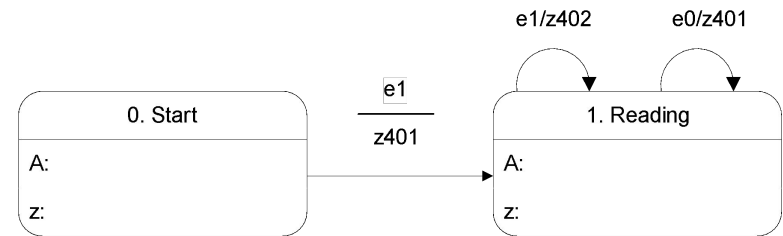
Генерация кода

- подстановки (templates C++);
 - подстановки с исполнением кода (ASP);
 - **обработчики данных регулярной структуры (XSLT).**
 - Наличие групп состояний.
 - Наличие групповых переходов.
 - Наличие логических выражений.
-

Пример. Шаг 1



Visio file



Пример. Шаг 2

XML file

```
<?xml version="1.0" encoding="Windows-1251"?>
```

```
...
<state name="Top" description="">
  <state name="s1" description="Stack analysis" />
  <state name="s0" description="Start" />
  <state name="s3" description="Nonterminal" />
  <state name="s2" description="Terminal" />
  <state name="s4" description="Error" />
</state>
<transition sourceRef="s1" targetRef="s4" priority="1">
  <condition>
    <conditionNode name="206" type="INPUT_VARIABLE" />
  </condition>
</transition>
<transition sourceRef="s1" targetRef="s2">
  <condition>
    <conditionNode name="202" type="INPUT_VARIABLE" />
  </condition>
</transition>
<transition sourceRef="s1" targetRef="s3">
  <condition>
    <conditionNode name="203" type="INPUT_VARIABLE" />
  </condition>
...
...
```

```
<!--=====-->
<!--MODEL-->
<ELEMENT model (stateMachine*)>
<!ATTLIST model
  name ID #IMPLIED
  description CDATA #IMPLIED>
<!--=====-->

<!-- STATE MACHINE -->
<ELEMENT stateMachine (state?)>
<!ATTLIST stateMachine
  name ID #REQUIRED
  description CDATA #IMPLIED>
<!--=====-->

<!-- STATE -->
<ELEMENT state (state*, stateMachineRef?, outputAction?)>
<!ATTLIST state
  name ID #REQUIRED
  type CDATA #IMPLIED
  description CDATA #IMPLIED>

<ELEMENT stateMachineRef (actionNode*)>
<!--=====-->

<!-- TRANSITION -->
<ELEMENT transition (state*, condition, outputAction)>
<!ATTLIST transition
```

Пример. Шаг 3

```
<?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="distinctConditions" match="//conditionNode"
  use="@name"/>
<xsl:key name="distinctActions" match="//actionNode" use="@name"/>

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

  namespace Automatas
  {
    public class BaseAutomata
    {
    }
    <xsl:apply-templates select="stateMachine"></xsl:apply-templates>
  }
</xsl:template>
```

XSLT-шаблон

```
<!--Automata processing-->
<xsl:template match="stateMachine">
  // &lt;summary&gt;
  // <xsl:value-of select="@description" />
  // &lt;/summary&gt;
  public abstract class <xsl:value-of select="@name"/> : BaseAutomata
  {
    protected string y = "s0";

    public void A(int e)
    {
      switch (y)
      {
        <xsl:apply-templates select="state//state[count(state) = 0]"
          mode="SWITCH_BLOCK">
          <xsl:sort select="@name" data-type="text" />
        </xsl:apply-templates>
      }
    }
  }

  <xsl:variable name="stateMachineName" select="@name"/>

  <xsl:apply-templates select="//actionNode
    [generate-id(.) = generate-id(key('distinctActions', @name)
    [ancestor::stateMachine/@name=$stateMachineName])]"
    mode="FUNCTION_DEFINITIONS">
  <xsl:sort select="@type"/>
  <xsl:sort select="@name"/>
  </xsl:apply-templates>

  <xsl:apply-templates select="//conditionNode
    [generate-id(.) = generate-id(key('distinctConditions', @name)
    [ancestor::stateMachine/@name=$stateMachineName])]"
    mode="FUNCTION_DEFINITIONS">
  <xsl:sort select="@type"/>
  <xsl:sort select="@name"/>
  </xsl:apply-templates>
  }
</xsl:template>
<!--End Automata processing-->
```

Пример. Шаг 4

```
///--- This file is machine generated ---  
//Model: ModelName  
namespace Automatas  
{  
    public class BaseAutomata  
    {  
        /// <summary>  
        /// Lexical analyzer  
        /// </summary>  
        public abstract class A2 : BaseAutomata  
        {  
            protected string y = "s0";  
            public void A(int e)  
            {  
                switch (y)  
                {  
                    case "s0":  
                        if (e == 0)        {z401(); y = "s1";}   
                        break;  
                    case "s1":  
                        if (e == 1)        {z200(); y = "s1";}   
                        else if (e == 0) {z401(); y = "s1";}   
                        break;  
                }  
            }  
            /// <summary>  
            ///   
            /// </summary>  
            protected abstract void z200();  
            /// <summary>  
            /// Initialize and return the first match  
            /// </summary>  
            protected abstract void z401();  
        }  
    }  
}
```

```
/// <summary>  
/// Syntactical analyzer  
/// </summary>  
public abstract class A1 : BaseAutomata  
{  
    protected string y = "s0";  
    public void A(int e)  
    {  
        switch (y)  
        {  
            case "s0":  
                if (true)  {z200(); Call_A2(0); y = "s1";}   
                break;  
            case "s1":  
                if (x206())          { y = "s4";}   
                else if (x201()){z201(); z301(); y = "s1";}   
                else if (x202()){ y = "s2";}   
                else if (x203()){z203(); y = "s3";}   
                break;  
            case "s2":  
                if (x205())          { y = "s0";}   
                else if (x204())  
                    {z202(); Call_A2(1); y = "s1";}   
                else if (true)      { y = "s4";}   
                break;  
            case "s3":  
                if (x207()) {z204(); y = "s1";}   
                else if (true) { y = "s4";}   
                break;  
            case "s4":  
                if (true) {z205(); y = "s0";}   
                break;  
        }  
    }  
    /// <summary>  
    /// Command in the top of the stack  
    /// </summary>  
    /// <returns>Is condition correct</returns>  
    protected abstract bool x201();  
    /*Часть входных переменных и действий пропущено*/  
}
```

Код программы

Интеграция с MS Visual Studio 2003

The image displays the MS Visual Studio 2003 environment with the following components:

- Solution Explorer (MetaAuto):**
 - References: Interop.Visio.dll, MetaAuto.dll
 - Config
 - MetaAuto
 - Visio2Xml: Interop.Visio.dll, MetaAuto.dll, Visio2Xml.exe, Visio2Xml.exe.config
 - XSLTransform: XSLTransform.exe, XSLTransform.exe.config
 - automatas.cs.xslt, makefile, nmake.exe
 - Parsers
 - ActionParser.cs
 - Analyzers.vsd
 - Automatas.cs
 - ConditionParser.cs
 - LexicalAnalyzer.cs
 - RegexpParser.cs
 - SyntacticalAnalyzer.cs
 - Resources
 - ActionNode.cs
 - AssemblyInfo.cs
 - Automata.cs
 - AutomataCollection.cs
- Code Editor (Automatas.cs):**

```
file is machine generated ---
io project

tomatas

class BaseAutomata

mary>
ический анализатор
mary>
bstract class A2 : BaseAutomata

ected string y = "0";

ic void A(int e)

switch (y)

case "0":
```
- Diagram Editor:** Shows a state transition diagram with two states: "1. Анализ стека" and "2. Терминал". Transitions are labeled with "A:" and "z:". A transition from state 1 to state 2 is labeled "x202", and a transition from state 2 to state 1 is labeled "x204".

Применения

- При создании самого инструментального средства
 - Созданы шаблоны для языков C#, C++, *Assembler*
 - Предполагается использовать для встроенных систем
 - настраиваемость
 - простота использования
-

Вопросы?

Спасибо за внимание

Дополнительная информация:

<http://is.ifmo.ru>

[kanzhsers@rain.ifmo.ru](mailto:kanzhser@rain.ifmo.ru)