

УДК 004.8

*А.А. Шалыто, Л.А. Наумов*

Федеральное государственное унитарное предприятие «НПО «Аврора»  
Санкт-Петербургский государственный университет информационных технологий,  
механики и оптики, г. Санкт-Петербург, Россия, shalyto@mail.ifmo.ru

## Методы объектно-ориентированной реализации реактивных агентов на основе конечных автоматов

В работе рассматриваются методы реализации автоматов в рамках объектно-ориентированного подхода к программированию реактивных агентов. Поддержка различных вариантов взаимодействия автоматов позволяет использовать эти методы для построения многоагентных систем. Это обеспечивает возможность решения одной из важнейших проблем реализации объектно-ориентированных систем – связи между статическими и динамическими свойствами объектов.\*

### Введение

При построении многоагентных систем все шире используются реактивные агенты [1]. В качестве наиболее распространенной математической модели для построения агентов этого класса применяются конечные автоматы. В работе [2] была предложена технология автоматного программирования для построения таких агентов. Однако для их реализации использовался процедурный подход.

Цель настоящей работы состоит в рассмотрении различных методов реализации автоматов в рамках объектно-ориентированного подхода к программированию реактивных агентов. Поддержка различных вариантов взаимодействия автоматов позволяет использовать эти методы для построения многоагентных систем.

### Методы объектно-ориентированной реализации конечных автоматов

В работе [3] было предложено проектировать программы для систем логического управления на основе использования конечных автоматов. Такой способ построения программ был назван «*Switch*-технология», или «автоматное программирование».

В работе [4] этот подход был развит для проектирования программного обеспечения событийных систем. Он является процедурным и поэтому был назван «процедурное программирование с явным выделением состояний».

---

\*Предложенные методы использованы при разработке проектов в рамках «Движения за открытую проектную документацию», размещенных на сайте <http://is.ifmo.ru>.

В работе [5] рассматриваемый подход был расширен на объектно-ориентированные программы и получил название «объектно-ориентированное программирование с явным выделением состояний». При этом автоматы реализовывались в качестве методов классов.

В ходе педагогического эксперимента, описанного в работе [6], в настоящее время выполнено более пятидесяти проектов с применением объектно-ориентированного программирования с явным выделением состояний. При выполнении проектов был создан ряд методов реализации автоматов, отличающихся от метода, предложенного в работе [5].

Выполним классификацию этих методов и кратко опишем их.

1. Автоматы как методы классов [5]. Этот подход близок к процедурному стилю программирования и может быть назван «обертывание автоматов в классы».

2. Автоматы как классы. Базовый класс, реализующий типовые функции автоматов, не применяется [7].

3. Автоматы как классы с использованием базового класса. Этот подход основан на совместном применении преимуществ как объектного, так и автоматного стилей программирования. При этом автоматы разрабатываются как наследники класса, реализующего базовую функциональность. Базовый класс и другие классы образуют библиотеку, предоставляемую разработчику.

3.1. В работе [8] приводится простейшая библиотека классов для разработки программного обеспечения в рамках объектно-ориентированного программирования с явным выделением состояний.

При использовании этой библиотеки проектирование каждого автомата состоит в создании по словесному описанию (декларация о намерениях) схемы связей, описывающей его интерфейс, и графа переходов, определяющего его поведение. По этим двум документам формально и изоморфно может быть построен модуль программы, соответствующий автомату.

Применяя объектную парадигму, автоматы разрабатываются как наследники базового класса `Automat`. Этот класс реализует типовые функции автоматов (основные и вспомогательные). В наследниках определяются только функции, специфические для автоматов.

Перечислим основные функции автоматов, реализованные в базовом классе:

- 1) организация выполнения действий в вершинах графа переходов (для автоматов Мура), на его дугах и петлях (для автоматов Мили), а также в вершинах, на дугах и петлях (для автоматов Мура – Мили);
- 2) организация взаимодействия автоматов:
  - вызов автоматов с определенными событиями;
  - реализация вложенных автоматов;
  - обмен номерами состояний между автоматами.

Отметим, что если взаимодействие по вложенности возможно только «сверху вниз» в иерархии автоматов, то остальные два способа могут осуществляться в обе стороны, как «сверху вниз», так и «снизу вверх».

Из вспомогательных функций автоматов в классе `Automat` реализована поддержка протоколирования. При этом возможно:

- 1) автоматическое протоколирование:
  - при начале работы автомата в определенном состоянии с определенным событием;

- при переходах из состояния в состояние;
- при завершении работы автомата в определенном состоянии;
- 2) добавление описаний входных и выходных воздействий автомата.

В классах-наследниках переопределяется ряд функций базового класса и добавляются входные воздействия (события и переменные), внутренние переменные, выходные воздействия, объекты управления, а также вложенные и вызываемые автоматы.

В работе [8] предлагаемый подход иллюстрируется примером моделирования лифта (программа *Lift*).

Эта программа является объектно-ориентированной. Такую программу удобно разрабатывать на персональном компьютере и легко переносить на PC-подобные контроллеры. Однако, кроме таких контроллеров, в системах управления используются также микроконтроллеры, для которых отсутствуют компиляторы с объектно-ориентированных языков. Поэтому для микроконтроллеров применяется процедурное программирование.

В работе [8] предлагается методика преобразования ядра объектно-ориентированной программы с явным выделением состояний на языке C++ в процедурную программу с явным выделением состояний на языке C.

В данном случае под ядром программы понимается ее фрагмент, в котором отсутствует интерфейсная часть и не реализованы функции входных и внутренних переменных, а также выходных воздействий.

Методика иллюстрируется примером переноса ядра программы *Lift* на микроконтроллер *Siemens SAB 80C515*. При этом использовалась среда *Keil  $\mu$ Vision 2*. Полученная в результате программа также размещена на сайте <http://is.ifmo.ru> в разделе «Проекты».

Подход, близкий к описанному выше, предлагается в работе [9].

3.2 В работе [10] предложена библиотека *STOOL (Switch-Technology Object Oriented Library)*, в которой не только автомат, но и его логические составные части имеют соответствующие базовые классы. Кроме того, библиотека предоставляет возможность разработки многопоточного программного обеспечения.

Особенность предлагаемого подхода состоит в том, что автоматы предлагается использовать не как методы классов, а как объекты, являющиеся потомками класса *Auto*. Это более общий подход, так как автоматы-методы можно легко свести к автоматам-объектам, но не наоборот.

Класс *State* представляет собой состояние автомата, а класс *Info* – описание автомата. Этот класс требуется для организации автоматического протоколирования.

Каждый автомат при запуске делает не более одного перехода.

Рассматриваются два варианта реализации алгоритмов с применением автоматов:

- автомат реализуется внутри цикла типа `while`;
- автомат используется непосредственно, без применения цикла.

Автоматы первого типа удобны при реализации вычислительных алгоритмов, а второго типа – для реактивных агентов.

Перегруженные операторы `operator int()` и `operator=(int)` класса *State* позволяют пользоваться экземпляром класса *State* так, как будто он является целочисленной переменной.

Использование объекта вместо скалярной переменной позволяет вынести из оператора `switch` все функции, отличные от основных – функций переходов, входных переменных и выходных воздействий. Это обеспечивает также возможность выделения глобального состояния системы и реализации одинаковым образом действий и деятельности.

Предлагаемый подход в рамках объектно-ориентированного программирования обеспечивает сохранение в программах оператора `switch`, позволяющего целостно, формально и изоморфно реализовывать графы переходов автоматов.

3.3. В работе [11] предложена еще одна библиотека для объектно-ориентированной реализации автоматов, названная *Auto-Lib*, и приведен пример ее использования.

3.4. В работе [12] предложена библиотека, позволяющая «собирать» простые автоматы из наследников базовых классов «состояние автомата» и «переход между состояниями». Эта библиотека обеспечивает изоморфизм между текстом программы и графом переходов даже при наличии в нем групповых переходов (переходов из гиперсостояний).

4. Применение паттернов проектирования [13]. Наряду с использованием библиотек при объектно-ориентированной реализации автоматов могут разрабатываться и применяться паттерны проектирования.

4.1. Описанный в работе [14] паттерн *Automat* позволяет проектировать программное обеспечение, используя классы, реализующие следующие понятия: «состояние», «условие перехода», «действие», «переход», «дуга перехода», «автомат». При этом класс, реализующий последнее понятие, является базовым для разрабатываемых автоматов и содержит в себе их основную логику.

4.2. Использование паттерна *State*. Данный паттерн, описанный в работе [13], представляет собой абстракцию «состояние». Для реализации конкретного состояния необходимо разработать наследника базового класса *State* и переопределить в нем функцию переходов.

Похожий подход рассмотрен в работе [15]. В ней для каждого автомата предложено создать базовый класс «состояние», от которого наследуются конкретные классы, реализующие состояния данного автомата. Переходы между состояниями обеспечиваются базовыми классами состояний, но непосредственно осуществляются в классах-наследниках.

5. Динамическое построение автоматов. В предыдущих методах использовалась статическая реализация автоматов, при этом автомат описывался некоторым кодом до выполнения. Затем этот код выполнялся. В работах [16], [17] предложен метод динамического построения автоматов. Это позволяет, в частности, реализовывать автоматы, число состояний которых заранее не известно. Создание и модификация автоматов, их состояний и переходов осуществляется с использованием разработанных библиотек.

6. Реализация автоматов на основе интерпретации.

6.1. В работе [18] предложен подход, позволяющий автоматически преобразовывать графы переходов в текстовое описание в формате *XML*. На языке *Java* разработана среда исполнения полученного XML-описания.

Сначала указанное описание однократно и целиком преобразуется в соответствующее внутреннее объектное представление программы. В результате образуется система, состоящая из среды исполнения и объектного представления прог-

раммы. При этом каждое входное и выходное воздействие реализуется вручную, в соответствии с его функциональностью. Упомянутая система при появлении события анализирует его и входные переменные и выполняет выходные воздействия, а также запускает вложенные автоматы.

6.2. В работе [19] описан программный пакет *UniMod* (<http://unimod.sourceforge.net>), который реализует подход, изложенный в предыдущем пункте. Он позволяет решать задачи автоматизации построения событийных объектно-ориентированных программ с явным выделением состояний. При этом для проектирования конечных автоматов совместно применяются *Switch*-технология и унифицированный язык моделирования *UML*. В этом случае схема связей автомата изображается с помощью диаграммы классов, а граф переходов – с помощью диаграммы *Statechart*. Этот пакет состоит из следующих частей:

- ядро, содержащее объектную метамодель конечного автомата, алгоритмы разбора и интерпретации булевых формул, проверки корректности конечного автомата и среду исполнения *XML*-описания модели конечного автомата;
- встраиваемый модуль для среды разработки диаграмм на языке *UML*, который помогает создавать схемы связей автомата, графы переходов в виде *UML*-диаграмм, а также выполняет генерацию *XML*-описаний.

6.3. В работе [20] предложено использовать язык *XML* для автоматного описания динамики изменения внешнего вида виртуального устройства – видеопроигрывателя *Crystal Player* \*

## 7 Механизм обмена сообщениями и автоматы.

7.1 В ходе выполнения работ [21], [22] по реализации классического параллельного алгоритма синхронизации цепи стрелков выяснилось, что автоматы, построенные по предложенному в работе [4] шаблону, реализующему событийно-управляемые автоматы, который состоит из двух операторов *switch*, не позволяют реализовать взаимодействующие параллельные процессы.

Для решения этой задачи в работе [21] было предложено использовать механизм обмена сообщениями, для поддержки которого разработана библиотека *SWMEM* (*Switch Message Exchange Mechanism*). При этом в шаблон для реализации автоматов были внесены следующие изменения: шаг работы автомата разделен на три этапа (выбор перехода, совершение действий на переходе и обновление переменной состояния); введены переменная для учета приоритетов условий на дугах графа переходов и переменная для хранения выбранного действия и последующего его выполнения.

7.2 В работе [23] механизм обмена сообщениями между параллельно «расположенными» автоматами реализуется за счет введения такой сущности, как «общая шина», позволяющей реализовывать децентрализованные реактивные многоагентные системы.

Этот подход обеспечивает однотипную реализацию разнотипных по своей природе алгоритмов, которые могут быть иерархическими, вложенными или параллельными. Для реализации параллельно работающих автоматов предложено изменить шаблоны, описанные в работах [4], [21], строя автомат с помощью двух функций – функции перехода-действия и функции обновления.

\* <http://www.crystalplayer.com>

Первая из этих функций сначала реализует входные воздействия как в состоянии, так и на переходе, а потом определяет номер нового состояния и осуществляет выходные воздействия в нем. Вторая функция обеспечивает выполнение одинаковых действий: обновляет состояние автомата и массив поступающих ему сообщений. Для синхронизации автоматов сначала должны вызываться все функции перехода-действия, а затем все функции обновления.

8 Язык автоматного программирования *State*. Широкое применение автоматов при разработке программного обеспечения ограничивается отсутствием непосредственной поддержки автоматов в языках программирования. Для устранения этого недостатка в работе [24] на базе языка *C#* за счет добавления базовой абстракции «состояние» был предложен язык автоматного программирования *State*.

## Заключение

В настоящей работе предложены различные методы решения важнейшей задачи реализации объектно-ориентированных систем [25] – обеспечение связи между статическими и динамическими свойствами объектов. Это открывает возможность использования различных подходов к объектно-ориентированной реализации поведения реактивных многоагентных систем.

Все предложенные подходы апробированы в проектах, опубликованных на сайте <http://is.ifmo.ru>. Проекты выполнены в рамках новой инициативы в программировании – «Движение за открытую проектную документацию» [6].

## Литература

1. Luger G. Artificial Intelligence. Structures and strategies for Complex Problems Solving. – Addison Wesley, 2002.
2. Naumov L., Shalyto A. Automata Theory for Multi-Agent Systems Implementation // Proc. of Integration of Knowledge Intensive Multi-Agent Systems. – Boston (MA). – 2003 // <http://is.ifmo.ru>, раздел «Наука».
3. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. – СПб.: Наука, 1998.
4. Шалыто А.А., Туккель Н.И. SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем // Программирование. – 2001. – № 5 // <http://is.ifmo.ru>, раздел «Статьи».
5. Шалыто А.А., Туккель Н.И. Танки и автоматы // ВУТЕ/Россия. – 2003. – № 2 // <http://is.ifmo.ru>, раздел «Статьи».
6. Shalyto A., Naumov L. Foundation for Open Project Documentation // Linux Summit – 2004 // <http://linuxsummit.org>.
7. Наумов А.С., Шалыто А.А. Система управления лифтом. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
8. Наумов Л.А., Шалыто А.А. Автоматное решение задачи Д. Кнута о лифте. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
9. Корнеев Г.А., Шалыто А.А. Реализация конечных автоматов с использованием объектно-ориентированного программирования // Труды X Всероссийской науч.-методич. конф. «Телематика-2003». – Т. 2. – 2003 // <http://tm.ifmo.ru>.
10. Шопырин Д.Г., Шалыто А.А. Объектно-ориентированный подход к автоматному программированию. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».

11. Фельдман П.И., Шалыто А.А. Совместное использование объектного и автоматного подходов в программировании. – СПб.: СПбГУ ИТМО, 2004 // <http://is.ifmo.ru>, раздел «Проекты».
12. Заякин Е.А., Шалыто А.А. Метод устранения повторных фрагментов кода при реализации конечных автоматов. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
13. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влассидес– СПб.: Питер, 2001.
14. Астафуров А.А., Шалыто А.А. Разработка и применение паттерна «Automata». – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
15. Кузнецов Д.В., Шалыто А.А. Система управления танком для игры «Robocode». Вариант 2. – СПб.: СПбГУ ИТМО, 2003.
16. Наумов А.С., Шалыто А.А. Объектно-ориентированное программирование с явным выделением состояний. – СПб.: СПбГУ ИТМО, 2004 // <http://is.ifmo.ru>, раздел «Проекты».
17. Фельдман П.И., Шалыто А.А. Разработка средств для отладки автоматных программ, построенных на основе предложенной библиотеки классов. – СПб.: СПбГУ ИТМО, 2004 // <http://is.ifmo.ru>, раздел «Проекты».
18. Гуров В.С., Нарвский А.С., Шалыто А.А. Автоматизация проектирования событийных объектно-ориентированных программ с явным выделением состояний // Труды X Всероссийской науч.-методич. конф. «Телематика-2003». – Т.1. – 2003 // <http://tm.ifmo.ru>.
19. Гуров В.С., Мазин М.А., Шалыто А.А. UniMod – программный пакет для разработки объектно-ориентированных приложений на основе автоматного подхода // Труды XI Всероссийской науч.-методич. конф. «Телематика-2004». – Т.1. – 2004 // <http://tm.ifmo.ru>.
20. Бондаренко К.А., Шалыто А.А. Разработка XML-формата для описания внешнего вида видеопроектировщика с использованием конечных автоматов. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
21. Гуисов М.И., Кузнецов А.Б., Шалыто А.А. Интеграция механизма обмена сообщениями в Switch-технологии. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
22. Гуисов М.И., Кузнецов А.Б., Шалыто А.А. Задача Д. Майхилла «Синхронизация цепи стрелков». Вариант 2. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
23. Альшевский Ю.А., Раер М.Г., Шалыто А.А. Система управления турникетом. – СПб.: СПбГУ ИТМО, 2003 // <http://is.ifmo.ru>, раздел «Проекты».
24. Шамгунов Н.Н., Шалыто А.А. Язык автоматного программирования с компиляцией в Microsoft CLR // Microsoft Research Academic Days in Saint-Petersburg. – 2004. – April 21-23.
25. Graham I. Object-Oriented Methods. Principles and Practice. – Addison-Wesley, 2001.

*А.А. Шалыто, Л.О. Наумов*

**Методи об'єктно-орієнтованої реалізації реактивних агентів на основі скінченних автоматів**

У роботі розглядаються методи реалізації автоматів у рамках об'єктно-орієнтованого підходу до програмування реактивних агентів. Підтримка різноманітних варіантів взаємодії автоматів дозволяє використовувати ці методи для побудови багатоагентних систем. Це забезпечує можливість рішення однієї з найважливіших проблем реалізації об'єктно-орієнтованих систем – зв'язку між статичними і динамічними властивостями об'єктів.

In this paper we examine different methods of finite automata implementation in the framework of object-oriented programming with explicit states separation. Principal concern is in the area of reactive agents development. Support of various ways of automata interactions allows using these methods for multi-agent systems implementation. The solution of the important problem's of connection between static and dynamic properties of objects for object-oriented approach is provided. Offered methods were used for projects development in the framework on "Foundation for Open project Documentation".

*Статья поступила в редакцию 18.06.2004*