МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

УДК 681.3.06: 62—507
ВГК ОКП
№ регистрационный 01.20.00 13546
Инв. №

«УТВЕРЖДАЮ»
Ректор СПбГУ ИТМО
докт. техн. наук, профессор

Васильев В.Н.

ОТЧЕТ

по научно-исследовательской работе № 10038
«Разработка технологии создания программного обеспечения систем управления на основе автоматного подхода»

Этап 5
«Применение методологии создания программного обеспечения систем управления на основе автоматного подхода»
Заключительный отчет

Руководитель НИР Шалыто А.А.
докт. техн. наук,
профессор,
заведующий кафедры
«Технологии программирования»

Санкт-Петербург
2004

Список основных исполнителей

Руководитель НИР

Доктор технических наук, профессор, заведующий кафедры «Технологии программирования»	Шалыто А.А.

Исполнители

Магистрант	Наумов Л.А.
Инженер-программист	Туккель Н.И.
Аспирант	Шамгунов Н.Н.
Аспирант	Шопырин Д.Г.
Аспирант	Корнеев Г.А.
Аспирант	Гуров В.С.
Аспирант	Казаков М.А.
Аспирант	Станкевич А.С.
Аспирант	Бабаев А.А.
Аспирант	Кузнецов Д.В.
Студент	Ярцев Б.М.
Студент	Мазин М.А.
Студент	Коротков М.А.
Студент	Наумов А.С.

Реферат

Отчет содержит 92 страницы, 11 рисунков, 108 источников литературы.

Система управления, вычислительный алгоритм, схема алгоритма, состояние, автомат, граф переходов, протокол, алгоритмизация, автоматное программирование, проектирование программ, объектно-ориентированное программирование, объектно-ориентированное программирование с явным выделением состояний, паттерны проектирования.

Целью настоящего этапа является краткое изложение результатов работы, выполненной на первых четырех этапах, а также описание новых результатов, относящихся к объектно-ориентированному программированию с явным выделением состояний, полученных в 2004 году.

Приведем наименование этапов, основные результаты которых приведены в настоящей работе.

1. Этап 1. Разработка основных положений технологии создания программного обеспечения систем логического управления.
2. Этап 2. Разработка основных положений создания программного обеспечения "реактивных" систем.
3. Этап 3. Применение автоматного подхода для программной реализации вычислительных алгоритмов.
4. Этап 4. Применение автоматного подхода в объектно-ориентированном программировании.
5. Этап 5. Применение методологии создания программного обеспечения систем управления на основе автоматного подхода.

Кроме того, в отчете приведены названия статей и тезисов докладов на конференциях, опубликованных в ходе выполнения отдельных этапов проекта.

В ходе выполнения работы был создан сайт http://is.ifmo.ru, на котором, в частности, размещены отчеты по этапам темы и все опубликованные статьи по данной тематике.

Кроме того, на сайте опубликованы более 50 проектов, выполненных с применением автоматного программирования. Перечень этих проектов приведен в разд. 5.4.

Эти проекты легли в основу «Новой инициативы в программировании - движения за открытую проектную документацию», которая была предложена руководителем настоящей работы в Санкт-Петербурге в ноябре 2002 года на открытии полуфинальных соревнований командного студенческого чемпионата мира по программированию ACM. Содержание этой инициативы изложено в одноименной статье, опубликованной в ряде отечественных изданий и сайтов (в том числе указанном выше), и докладывалось в феврале 2004 г. на Linux Summit (Финляндия)- http://www.linuxsummit.org.

Содержание

Введение 7
Глава 1. Разработка основных положений технологии создания программного обеспечения систем логического управления (этап 1) .. 9
Глава 2. Разработка основных положений создания программного обеспечения «реактивных» систем (этап 2) 19
2.1. Классификация задач, решаемых с использованием автоматного подхода 23
2.1.1. Системы логического управления 24
Реализация на программируемых логических контроллерах 24
Моделирование контроллера в однозадачных операционных системах 25
Моделирование контроллера в многозадачных операционных системах 26
2.1.2. «Реактивные» (событийные) системы 27
2.2. Заключение 30
Глава 3. Применение автоматного подхода для программной реализации вычислительных алгоритмов (этап 3) 31
3.1. Преобразование итеративных алгоритмов в автоматные 32
Глава 4. Применение автоматного подхода в объектно-ориентированном программировании (этап 4) 38
Глава 5. Применение методологии создания программного обеспечения систем управления на основе автоматного подхода (этап 5). 43
5.1. Автоматный подход к разработке управляющих программ для реактивных мультиагентных систем 43
5.1.1. Введение в тему 44
5.1.2. Автоматный подход 44
5.1.3. Пример. Постановка задачи 46
5.1.4. Пример. Результаты 46
5.1.5. Выводы по разделу 48
5.2. UML. SWITCH-технология. Eclipse 48
5.2.1. Введение 49
5.2.2. Исполняемый графический язык на основе Switch-технологии и UML-нотации 52
5.2.3. *UniMod* – пакет для автоматно-ориентированного программирования 57
5.2.4. Реализация редактора диаграмм на платформе *Eclipse* .. 58

Проверка синтаксиса и семантики 59
Автоматическое завершение ввода и автоматическое исправление ошибок 60
Форматирование 62
Запуск программы 62
Отладка 62
Библиотеки 63
5.2.5. Заключение раздела 63
5.3. Другие примеры применения методологии 64
5.4. Проекты, разработанные студентами кафедры «Компьютерные технологии» СПбГУ ИТМО в рамках «Движения за открытую проектную документацию» 65
Глава 6. Публикации по результатам этапов 70
6.1. Разработка основных положений технологии создания программного обеспечения систем логического управления (этап 1) 70
6.1.1. Статьи 70
6.1.2. Тезисы 70
6.2. Разработка основных положений создания программного обеспечения "реактивных" систем (этап 2) 71
6.2.1. Статьи 71
6.2.2. Тезисы 72
6.3. Применение автоматного подхода для программной реализации вычислительных алгоритмов (этап 3) 73
6.3.1. Статьи 73
6.3.2. Тезисы 74
6.4. Применение автоматного подхода в объектно-ориентированном программировании (этап 4) 74
6.4.1. Статьи 74
6.4.2. Тезисы 75
6.5. Объектно-ориентированное программирование с явным выделением состояний (этап 5) 77
6.5.1. Статьи 77
6.5.2. Тезисы 79
Заключение 80
Список литературы 82

Введение

Целью настоящего этапа является подведение итогов работы, проделанной на протяжении первых четырех этапов, а также описание новых результатов, относящихся к объектно-ориентированному программированию с явным выделением состояний, полученных в 2004 году.

Приведем наименование этапов, основные результаты которых приведены в настоящей работе:

1. Этап 1. Разработка основных положений технологии создания программного обеспечения систем логического управления.
2. Этап 2. Разработка основных положений создания программного обеспечения "реактивных" систем.
3. Этап 3. Применение автоматного подхода для программной реализации вычислительных алгоритмов.
4. Этап 4. Применение автоматного подхода в объектно-ориентированном программировании.
5. Этап 5. Заключительный отчет. Объектно-ориентированное программирование с явным выделением состояний.

Кроме этого, в отчете приведены названия статей и тезисов докладов на различных конференциях, опубликованных исполнителями при выполнения проекта.

В ходе выполнения работы был разработан ресурс http://is.ifmo.ru, на котором, в частности, размещены отчеты по этапам темы и все опубликованные статьи по данной тематике.

Кроме того, на сайте опубликованы более 50 проектов, выполненных с применением автоматного программирования.

Эти проекты легли в основу «Новой инициативы в программировании - движения за открытую проектную документацию», которая была предложена руководителем настоящей работы в Санкт-Петербурге в ноябре 2002 года на открытии полуфинальных соревнований командного студенческого чемпионата

мира по программированию ACM. Содержание этой инициативы изложено в одноименной статье, опубликованной в ряде отечественных изданий и сайтов (в том числе указанном выше), и докладывалось в феврале 2004 года Linux Summit (Финляндия)- http://www.linuxsummit.org.

Глава 1. Разработка основных положений технологии создания программного обеспечения систем логического управления (этап 1)

В отчете по первому этапу темы были предложены и описаны основные положения технологии алгоритмизации и программирования задач логического (основанного на истинности и ложности) управления на основе теории автоматов.

Центральным понятием разработанного подхода является понятие «внутреннее состояние» (в дальнейшем «состояние»), которое совместно с понятием «входная переменная» порождает понятие «автомат без выхода», а после введения понятия «выходная переменная» - понятие «автомат».

Выполнено сравнение языков описания алгоритмов логического управления. Описана структура систем логического управления. Исследованы различные структурные модели автоматов.

Обоснован выбор графов переходов в качестве языка спецификаций.

Рассмотрены вопросы кодирования состояний автоматов и обоснован выбор многозначного кодирования состояний и конструкций, аналогичных конструкции `switch` языка *C*, для формального и изоморфного перехода от спецификаций (графов переходов) к текстам программ.

На примере разработки алгоритма управления трехпозиционным клапаном была изложена методика построения одиночного графа переходов и рассмотрен вопрос о его программной реализации. В качестве другого примера на основе этой методики построен и реализован граф переходов для управления системой воздуха среднего давления.

Кроме того, были рассмотрены вопросы организации взаимодействия в системе графов переходов и декомпозиции графов переходов и диаграмм «Графсет». Приведен пример реализации

алгоритма управления двумя клапанами на основе централизованной, децентрализованной и иерархической автоматных структур.

Предложены методы программной реализации автоматов с памятью схемами из функциональных блоков, которые строятся непосредственно по графам переходов.

Выполнено сопоставление систем с состояниями, теории управления, цепей Маркова и автоматного программирования.

Показаны недостатки современных технологий автоматизации технологических процессов, которые могут быть устранены на основе предлагаемого подхода.

Изложены основные положения технологии создания программного обеспечения систем логического управления на основе автоматного подхода.

Использование при проектировании и реализации понятий «состояние», «автомат», «независимость от «глубокой» предыстории» («будущее» зависит от настоящего и не зависит от «прошлого»), «многозначное кодирование состояний», «система взаимосвязанных графов переходов», «формальное и изоморфное программирование», «конструкция switch», обеспечивает наглядность, структурность, вызываемость, вложенность, иерархичность, управляемость и наблюдаемость программ, а также их изоморфизм (изобразительную эквивалентность) со спецификацией, по которой они формально строятся. Это позволяет Заказчику, Технологу (Проектанту), Разработчику, Программисту, Оператору и Контролеру однозначно понимать друг друга и точно знать, *что должно быть сделано, что делается и что сделано* в программно реализуемом проекте.

Это также позволяет разделять между указанными специалистами работу, а самое главное ответственность, легко и корректно вносить изменения в алгоритмы и программы, проводить сертификации построенных на основе предлагаемого подхода программ.

Предлагаемый подход назван Switch-технологией. Он может быть назван также State-технологией, или более точно Automaton-технологией.

Введены понятия «автоматное программирование» и «автоматное проектирование программ».

Излагаемая методология может рассматриваться как новое научное направление в теории автоматов, а также в теории и практике алгоритмизации и программировании задач логического управления.

Предлагаемая технология может применяться и применяется в настоящее время при разработке и программной реализации алгоритмов логического управления объектами, используемыми в различных отраслях промышленности.

Центральным понятием теории конечных автоматов является внутреннее состояние автомата («состояние»).

Состояние несет в себе всю информацию о прошлом автомата, необходимую для определения его реакции на любое входное событие.

Л. Заде перенес это понятие на теорию линейных систем, создав метод пространства состояний.

Предлагается при использовании средств вычислительной техники для построения систем логического управления ввести понятие «состояние» в теорию и практику алгоритмизации и программирования этого класса систем.

Несомненно, что значения всех внутренних переменных, в качестве которых могут выступать и выходные переменные, в произвольно построенной программе в определенный момент являются неявным заданием ее состояния. Однако, ввиду того, что в программах обычно состояние, как понятие не определяется, а поэтому и не применяется, то при разработке и отладке программ обычно используют только отдельные (обычно битовые) неупорядоченно расположенные внутренние переменные, значения которых и определяют состояния. Это приводит

к большим трудностям при создании качественного программного обеспечения.

Такая же ситуация возникает и в тех случаях, когда состояния определяются значениями некоторых входных, выходных и внутренних переменных.

Предлагаемая технология алгоритмизации и программирования базируется на понятии «состояние» и позволяет типизировать структуру алгоритмов и программ, задавая порядок введения, расположение и кодирование переменных, определяющих их состояния. В работе [1] рассмотрен вопрос о взаимном соответствии схем алгоритмов и графов переходов, последние из которых используются в предлагаемой технологии в качестве языка спецификаций.

Показано, какие структуры должны иметь схемы алгоритмов для обеспечения их изоморфизма с графами переходов и операторами switch.

Каждое состояние позволяет декомпозировать множество входных переменных на подмножества (возможно, пересекающиеся), существенно влияющие на переходы в смежные состояния, что обеспечивает возможность описания с помощью графов переходов автоматов с большим числом входов.

В программирование предлагается ввести понятие наблюдаемость, позволяющее рассматривать программу в качестве «белого ящика», в котором все внутренние переменные, число которых минимально, доступны для наблюдения.

Показаны преимущества в части читаемости, понимаемости и возможности отражения динамики графов переходов по сравнению с другими языками спецификаций, используемыми в практике проектирования рассматриваемого класса систем.

Показано, что при применении многозначного кодирования состояний для каждого графа переходов вне зависимости от числа его вершин (состояний автомата) достаточно использовать только одну внутреннюю переменную, их кодирующую.

Программирование графов переходов с помощью операций switch или их аналогов, кроме обеспечения изоморфизма со спецификацией, обеспечивает также доступность каждого значения переменной, кодирующей состояния, для всех остальных графов переходов, входящих в систему, что позволяет осуществлять взаимодействие графов без введения дополнительных внутренних переменных.

В работе [1] рассмотрены вопросы иерархического (в том числе вложенного) представления системы взаимосвязанных графов переходов. Предложен механизм описания с помощью системы взаимосвязанных графов переходов процессов параллельных по состояниям.

Разработаны методы формального и изоморфного перехода от спецификации к текстам программ на языках промышленных компьютеров (например, ассемблер и C) и программируемых логических контроллеров (языки инструкций, лестничных и функциональных схем). Формальность перехода позволяет для одного графа переходов использовать его в качестве полного теста для сертификации построенной программы, а изоморфность с графом переходов, позволяет проверять программу сверкой ее текста со спецификацией. В рамках предложенной технологии определены требования к представлению алгоритмов логического управления и программ в технической документации, их верификации и тестированию, например, в части создания методик проверки функционирования.

Предлагаемый подход позволяет Заказчику, Проектанту (Технологу), Разработчику, Программисту, Пользователю (Оператору) и Контролеру однозначно понимать (с точностью до бита, состояния и перехода), что должно быть сделано, что делается и что сделано в проекте, и тем самым решить для рассматриваемого класса задач проблему их взаимопонимания [2].

Это позволяет разделять работу и ответственность между специалистами разных областей знаний, а также между организациями, что особенно важно при работе с иностранными фирмами, в частности, из-за наличия языкового барьера и неоднозначности понимания естественных языков.

Предлагаемый подход позволяет также:

- использовать теорию автоматов при алгоритмизации и программировании процессов управления;
- первоначально описывать желаемое поведение управляющего «устройства», а не его структуру, которая является вторичной и поэтому труднее читаемой и понимаемой;
- ввести в алгоритмизацию и программирование в качестве основного понятия «состояние»;
- ввести понятия «автоматное программирование» и «автоматное проектирование программ»;
- начинать построение алгоритмов и программ с формирования дешифратора состояний, а не событий;
- применять основные структурные модели теории автоматов и ввести новые;
- использовать в качестве языка алгоритмизации графы переходов и системы взаимосвязанных графов переходов;
- при построении графов переходов исключить зависимость от «глубокой» предыстории (как это имеет место, например, в теории марковских процессов);
- применять многозначное кодирование состояний для каждого графа переходов, и вне зависимости от числа его вершин использовать только одну внутреннюю переменную, их кодирующую;
- применять в качестве основной алгоритмической модели графы переходов автомата Мура;

- обеспечить реализацию таких свойств алгоритмов управления как композиция, декомпозиция, иерархичность, параллелизм, вызываемость и вложенность;
- иметь один язык спецификаций при различных языках программирования, в том числе и специализированных, которые применяются в программируемых логических контроллерах;
- проводить алгоритмизацию в результате взаимного общения Заказчика, Проектанта (Технолога) и Разработчика. При этом выдача Технического Задания превращается из однократного события с «бесконечными» последующими дополнениями в однократный процесс общения, завершающийся созданием графа переходов или системы взаимосвязанных графов переходов, в которых учтены все детали с точностью до каждого состояния, перехода и бита;
- применять графы переходов без флагов и умолчаний в качестве проверяющего теста и строить граф проверки или граф достижимых маркировок для других классов графов переходов или систем взаимосвязанных графов переходов с целью проверки их поведения;
- использовать методы формального и изоморфного перехода от спецификации к программам логического управления на различных языках программирования;
- при применении алгоритмических языков программирования высокого уровня проводить программирование с помощью операторов `switch` или их аналогов. Это кроме изоморфизма со спецификацией обеспечивает доступность каждого значения переменной, кодирующей состояния каждого графа переходов, для всех остальных графов, входящих в систему, и поэтому не требует введения дополнительных внутренних переменных для реализации взаимодействия графов в системе;

- ввести в программирование понятие «наблюдаемость», что обеспечивает возможность рассмотрения программы в качестве «белого ящика», в котором все внутренние переменные, число которых минимально, доступны для наблюдения;
- на ранних стадиях проектирования учесть все детали Технического Задания и продемонстрировать Заказчику, как оно понято;
- участникам разработки общаться не традиционным путем в терминах технологического процесса (например, не «идет» режим экстренного пуска), а на промежуточном полностью формализованном языке (своего рода техническом эсперанто), на котором объясняться можно, например, следующим образом: «в третьем графе переходов, в пятой вершине, на четвертой позиции – изменить значение **0** на **1**». Это не вызовет разночтений, возникающих из-за неоднозначности понимания даже для одного естественного языка, а тем более для нескольких таких языков, в случае, когда Участники разработки представляют разные страны, и не требует привлечения Специалистов, знающих технологический процесс, для корректного внесения изменений;
- снять с Программиста необходимость знания особенностей технологического процесса, а с Разработчика – тонкостей программирования;
- программисту функциональных задач ничего не додумывать за Заказчика, Проектанта (Технолога) и Разработчика, а только однозначно и формально реализовывать систему взаимосвязанных графов переходов в виде программ, что позволяет резко снизить требования к его квалификации, а, в конечном счете, и вовсе отказаться от его услуг и автоматизировать процесс программирования или перейти к «автопрограммированию» Разработчиком. Последнее, од-

нако, возможно только в случае, когда программирование является «открытым», что не всегда имеет место в особенности при работе с инофирмами или их подразделениями, занимающимися разработкой систем управления, а не только аппаратуры;

- оставлять понятные «следы» после завершения разработки. Это позволяет проводить модификацию программ новым людям, что при традиционном подходе чрезвычайно трудоемко («проще построить программу заново, чем разобраться в чужой программе»). При этом необходимо отметить, что структурирование и комментарии указанную проблему решают лишь частично;
- упростить внесение изменений в спецификацию и программу и повысить их «надежность»;
- сделать алгоритм управления инвариантным к используемым языкам программирования, что открывает возможность формирования и поддержания библиотек алгоритмов, записанных формально;
- Заказчику, Проектанту (Технологу) и Разработчику контролировать тексты функциональных программ, а не только результаты их выполнения, как это имеет место в большинстве случаев в настоящее время;
- устранить не равную “прочность” приемки аппаратуры и программ Заказчиком. В первом случае Заказчик кроме функционирования проверяет еще много других характеристик (например, качество печатных плат и их покрытий, качество пайки, номиналы и обозначения элементов), а во втором – все внимание уделяет только проверке функционирования и не исследует внутренняя организацию программ и технологию их построения.

Разработанный подход существенно дополняет международный стандарт IEC 1131-3 [3], распространяющийся на описания языков программирования программируемых логических контрол-

леров, позволяя при разных языках программирования иметь один язык спецификаций.

Появление в 1996 году программного продукта «S7-HiGraph Technology Software» для программируемых логических контроллеров фирмы Siemens [4], в котором в качестве языка программирования используются диаграммы состояний (другое название графов переходов), еще более укрепило уверенность автора в правильности предложенной в настоящей работе методологии.

Предложенная технология впервые была опубликована в работе [5] в 1991 году и успешно использована в НПО «Аврора» при разработке систем управления техническими средствами судов, которые построены, в частности, на базе средств вычислительной техники таких фирм как «ABB Stromberg» (Финляндия) [5], «Norcontrol» (Норвегия) [6], «FF-Automation» (Финляндия) [7].

В заключение отметим, что без использования технологии алгоритмизации и программирования, направленной на создание «качественных» программ, невозможно решить проблему полноты их тестирования, также как нельзя обеспечить контролепригодность аппаратуры, если это свойство не закладывать при ее проектировании.

Предлагаемая технология может быть основой для повышения безопасности программного обеспечения [8] для систем логического управления.

Разработанная технология не исключая возможности применения других методов построения программного обеспечения “без ошибок” [9], существенно более конструктивна, так как позволяет начинать «борьбу с ошибками» еще на стадии алгоритмизации.

Глава 2. Разработка основных положений создания программного обеспечения «реактивных» систем (этап 2)

Целью этого этапа работы являлась разработка основных положений технологии создания программного обеспечения «реактивных» (событийных) систем.

Эта технология должна поддерживать все этапы создания программного обеспечения для рассматриваемого класса систем. К ним относятся: изучение предметной области, анализ, проектирование, реализация, отладка, сертификация и документирование.

Расширено по отношению к системам логического управления понятие «входное воздействие» за счет использования «событий», которые в отличие от «входных переменных», не опрашиваются программой, а вызывают соответствующие им обработчики. Расширено также понятие «выходное воздействие» за счет перехода от двоичных переменных к произвольным подпрограммам (функциям). Дополнена нотация, применяемая при построении графов переходов, например, за счет перечисления вложенных автоматов.

При использовании предлагаемой технологии в отличие от объектно-ориентированного проектирования программ построение всех основных моделей основано на применении только автоматной терминологии, а для описания динамики используется модель только одного типа — система взаимосвязанных графов переходов. Применение графов переходов в качестве языка спецификации делает обозримым даже весьма сложное поведение программы и позволяет легко вносить изменения, как в спецификацию, так и в ее реализацию.

В работе рассмотрен вопрос о программной реализации иерархии графов переходов для произвольного их количества и произвольного уровня вложенности. Каждый граф переходов

формально и изоморфно реализуется по шаблону в виде подпрограммы на выбранном языке программирования.

Автоматическое ведение протокола в терминах спецификации (автоматов) обеспечивает возможность сертификации программы. Для сертификации в терминах, понятных Заказчику, могут применяться "короткие" протоколы, в которых указываются только формируемые выходные воздействия.

Подробное документирование проекта создания программного обеспечения упрощает внесение изменений в него даже через длительный срок после выпуска, в том числе и специалистами, не участвовавшими в проектировании.

Выполненная работа является продолжением исследований, проведенных по первому этапу темы, в рамках которого была разработана Switch-технология, предназначенная для алгоритмизации и программирования задач логического управления.

"Реактивные" системы обладают спецификой, требующей разработки варианта Switch-технологии. Перечислим основные отличия рассматриваемых классов систем.

Если в системах логического управления в качестве входных воздействий используются опрашиваемые программой двоичные входные переменные и предикаты, соответствующие определенным состояниям автоматов, взаимодействующих с рассматриваемым автоматом, то для «реактивных» систем это понятие расширено. Во-первых, в качестве входных переменных используются любые подпрограммы (функции), возвращающие двоичные значения, а во-вторых, введены события, не только обеспечивающие возможность выполнения переходов в автоматах, но инициирующие их запуск. События могут также инициировать реализацию выходных воздействий в случае, когда состояние автомата не изменяется.

Другое отличие «реактивных» систем от систем логического управления состоит в том, что в них в качестве выход-

ных воздействий применяются не двоичные переменные, а произвольные подпрограммы.

Еще одно отличие систем рассматриваемого класса по сравнению с системами логического управления состоит в используемой операционной системе. Если в системах логического управления при их реализации на программируемых логических контроллерах применяются «встроенные» операционные системы, работающие обычно циклически, то для «реактивных» систем характерно применение операционных систем реального времени (ОС РВ), таких как, например, ОС РВ **QNX**, свойства которых влияют на программную реализацию алгоритмов. В рассматриваемых системах прерывания обрабатываются операционной системой и передаются программе в виде сообщений, которые, в свои очередь, обрабатываются как события с помощью соответствующих обработчиков. При этом после обработки очередного события автомат сохраняет свое состояние и «засыпает» до появления следующего события.

Также как и в системах логического управления, в «реактивных» системах алгоритмы представляются в виде системы взаимосвязанных автоматов. При этом если в системах первого типа взаимодействие автоматов в основном осуществляется за счет обмена номерами состояний, а вложенность присутствует в «зачаточном» состоянии, то в «реактивных» системах число способов взаимодействия увеличилось.

Кроме того, если в системах логического управления наиболее целесообразно применять такую структурную модель как автомат Мура, то в «реактивных» (событийных) системах более рационально использовать другую модель — смешанный автомат, совмещающий в себе свойства автоматов Мура и Мили.

Указанные особенности событийных систем сделали целесообразным:

– их разработку в виде системонезависимой и системозависимой частей;

– разработку новой структурной схемы систем этого класса, в которой повышен уровень централизации управления за счет выноса логики из обработчиков событий с целью формирования системы взаимосвязанных автоматов, которые запускаются из обработчиков событий;

– реализацию функций входных и выходных воздействий отдельно от автоматов и вызов этих функций из функций, соответствующих автоматам;

– разработку схемы взаимодействия автоматов;

– изменение нотации, используемой при построении графов переходов, особенно в части отображения вложенных автоматов;

– разработку шаблона для формальной и изоморфной реализации графа переходов, в вершины которого могут быть вложены автоматы;

– осуществление взаимодействия автоматов в системе за счет обмена номерами состояний, вложенности (в состояния) и вызываемости (из выходных воздействий);

– автоматическое ведение протокола, в котором при фиксированных значениях входных переменных, указываются значения этих переменных, события, запуск автоматов, их состояния в момент запуска, переходы в новые состояния, завершение работы автоматов, выходные воздействия и время начала выполнения каждого из них;

– автоматическое построение «короткого» протокола в дополнение к «полному», в котором фиксируются только события и инициируемые ими выходные воздействия, интересующие Заказчика.

Разрабатываемая технология поддерживает все этапы создания программного обеспечения: изучение предметной области, анализ, проектирование, реализация, отладка, сертификация и документирование.

Эта технология применялась авторами при создании программного обеспечения для ряда «реактивных» систем, что отражено в последующих разделах отчета.

2.1. Классификация задач, решаемых с использованием автоматного подхода

Опыт разработки и применения автоматного программирования (Switch-технология) позволил выделить три класса задач, в которых может быть применен предлагаемый подход:

– системы логического управления, характерной особенностью которых является двоичная форма представления входных и выходных воздействий и ввод входных воздействий путем опроса;

– «реактивные» системы, характерной особенностью которых является наличие среди входных воздействий событий;

– вычислительные алгоритмы, такие как, например, алгоритмы сортировки, поиска и т.п.

Специфика каждого из перечисленных классов задач является причиной различных особенностей в реализации (табл. 1).

Таблица 1. Особенности реализации различных классов задач, решаемых с использованием автоматного подхода

Особенности реализации	Классы решаемых задач		
	Системы логического управления	"Реактивные" (событийные) системы	Вычислительные алгоритмы
Типы входных воздействий автомата (без учета обмена номерами состояний)	• двоичные переменные	• двоичные переменные • события	• двоичные переменные
Типы выходных воздействий автомата	• действия • деятельности	• действия	• действия
Способы реализации входных и выходных воздействий	• переменные • функции	• переменные • функции	• функции

Способы запуска автоматов	• бесконечный цикл • периодический	• по событиям	• конечный цикл

Различные варианты реализации перечисленных классов задач порождают различные структуры получаемого в результате программного обеспечения.

Важнейшая особенность структур программного обеспечения, разрабатываемого с использованием автоматного подхода, заключается в том, что в центре находится управляющий автомат, а в общем случае — система взаимосвязанных автоматов [1-3]. При этом программное обеспечение можно классифицировать по способу реализации входных и выходных воздействий и способу запуска автоматов.

Отметим, что каждый из указанных способов запуска выполняется циклически, а их основные отличия состоят в особенностях реализации тела цикла.

2.1.1. Системы логического управления

Реализация на программируемых логических контроллерах

В системах логического управления (при их реализации на программируемых логических контроллерах) автоматы запускаются в бесконечном цикле в соответствии с принятой в контроллерах организацией исполнения управляющих программ. При этом тело цикла состоит из трех частей.

Сначала контроллер осуществляет опрос входных переменных и помещает их текущие значения в оперативную память (выполняется фотография входов). После этого все автоматы выполняются последовательно в заданном порядке. В завершении цикла контроллер записывает текущие значения выходных переменных в выходные регистры. После этого цикл повторяется. Особенность данного способа заключается в том, что шаги

цикла выполняются с частотой, определяемой производительностью контроллера. Благодаря этому задержка реакции управляющей программы на изменение значений входных переменных не будет превышать длительности одного программного цикла.

Моделирование контроллера в однозадачных операционных системах

Используемый в контроллерах способ выполнения управляющих программ может быть промоделирован на компьютере. Для реализации бесконечного цикла в случае применения языка C может использоваться оператор for(;;), а выходной преобразователь в общем случае представляет собой набор операторов условного перехода, выбирающих необходимые выходные воздействия в зависимости от значений выходных переменных.

При реализации систем управления на компьютерах, входные и выходные воздействия могут быть реализованы в виде функций. Благодаря этому появляется возможность создания с использованием SWITCH-технологии систем управления более широкого класса, чем системы логического управления.

Реализация входных и выходных воздействий в виде функций имеет существенные отличия по сравнению с их реализацией в виде переменных. Из структуры программного обеспечения исчезают входной и выходной преобразователи, так как функции, реализующие входные переменные, вызываются только тогда, когда эти переменные встречаются в условиях переходов, а функции, реализующие выходные воздействия, вызываются однократно при выдаче соответствующего воздействия. При этом количество выходных функций удваивается по сравнению с аналогичным количеством выходных переменных. Например, для управления световым индикатором, который находится в выключенном или включенном состоянии, может использоваться одна выходная переменная, нулевое значение которой выключает индикатор, а значение равное единице — включает. При реализа-

ции выходных воздействий в виде функций, для управления таким индикатором потребуются две выходные функции, одна из которых выключает индикатор, а другая включает. При желании две эти функции могут быть заменены одной функцией с параметром и условным переходом внутри нее.

Моделирование контроллера в многозадачных операционных системах

Рассмотренный выше способ запуска автоматов в непрерывном цикле является естественным для контроллеров, может быть использован в программном обеспечении, функционирующем под управлением однозадачных операционных систем, но является нежелательным в случае использования многозадачных операционных систем.

При разработке программ для многозадачных операционных систем, непрерывный цикл может быть заменен циклом, тело которого выполняется только при приходе какого-либо сообщения, например о срабатывании таймера, период которого равен требуемому времени задержки реакции системы управления на изменение значений входных переменных.

Сообщения от таймера принадлежат к разновидности входных воздействий, называемых событиями. Отметим, что в большинстве контроллеров отсутствует возможность порождения и обработки программных событий.

При запуске автоматов от таймера-генератора синхроимпульсов в структуру программы добавляется обработчик события «Срабатывание таймера».

При запуске автоматов от таймера из структуры программы исчезает в явном виде бесконечный цикл. Однако запуск автоматов остается циклическим, учитывая циклический способ обработки событий в многозадачных операционных системах.

Для уменьшения загрузки процессора в многозадачной системе период срабатывания таймера, запускающего автоматы,

может изменяться в зависимости от состояния, в котором находится система управления.

Пусть, например, к разрабатываемой системе управления, функционирующей в режимах «Останов», «Работа» и «Авария» предъявлено требование, состоящее в том, что задержка реакции на изменение входных воздействий не может превышать 0.1 с. Для выполнения этого требования необходимо запрограммировать таймер, используемый в качестве генератора синхроимпульсов, на период, не превышающий указанный. Однако далее это требование может быть уточнено и выяснится, например, что оно предъявлялось только для режима «Работа», а в режиме «Авария» задержка реакции ограничивается значением 1 с, в режиме «Останов» — 10 с.

2.1.2. «Реактивные» (событийные) системы

В отличие от систем логического управления, в событийных системах входные воздействия разделяются на два типа: входные переменные и события. Возможность применения в качестве входных воздействий событий позволяет использовать Switch-технологию при создании программного обеспечения систем управления, принадлежащих к «реактивным» системам (системам, реагирующим на события). К таким системам относятся, например, все системы, реализующие пользовательский интерфейс в современных графических оболочках.

При этом, развивая структуру программ, в которой автоматы запускаются из обработчика события срабатывания таймера, будем запускать автоматы из обработчиков всех событий, присутствующих среди входных воздействий.

Отметим, что один и тот же источник входных воздействий может одновременно являться источником и входных переменных и событий. Поясним это на примере кнопки, которая

может являться источником входных воздействий, перечисленных ниже:

– событие «Нажатие кнопки»;

– событие «Отпускание кнопки»;

– входная переменная «Кнопка нажата» (ее инверсия – «Кнопка отпущена»).

Таким образом, входные переменные указывают на текущее значение входного воздействия, а события — на изменение этого значения. Поэтому входные переменные на временной диаграмме отмечаются на вертикальной оси, а события — на оси времени. Такое время в работе [4] названо «событийным».

Из изложенного следует, что в событийных системах время присутствует в явном виде среди входных воздействий, в отличие от систем логического управления, в которых оно может присутствовать только в выходных воздействиях (функциональные элементы задержки). В вычислительных алгоритмах время в явном виде не используется.

В качестве названий двоичных входных переменных используются предложения в повествовательной форме (например, "Кнопка нажата"). Названия событий содержат отглагольные существительные (например, "Нажатие кнопки").

В случае, когда необходимо решить, в каком виде рассматривать входное воздействие, возможны варианты, перечисленные ниже.

1. Входная переменная. Этот вариант является наиболее предпочтительным, не считая ситуаций, описанных в следующих пунктах.

2. Событие. Использование событий целесообразно, в первую очередь, для входных воздействий, имеющих кратковременный характер. К источникам таких воздействий можно отнести, например, кнопки без памяти и таймеры. Отметим, однако, что факт возникновения подобных входных воздействий мо-

жет быть запомнен. После этого они могут рассматриваться как входные переменные.

3. Входная переменная и событие одновременно. Для ускорения реакции на изменение входной переменной, оно может рассматриваться как событие. В качестве примера входного воздействия, которое необходимо рассматривать одновременно и как входную переменную, и как событие, можно привести сигнализатор открытого положения двери микроволновой печи. Входное воздействие от этого сигнализатора должно рассматриваться как входная переменная, так как его необходимо опрашивать, проверяя допустимость включения печи. В то же время, при открытии двери печи во время ее работы, она должна быть отключена немедленно, а не тогда, когда входная переменная «Дверь открыта», будет опрошена в результате прихода какого-либо другого события. Отметим, что в данном примере рассматривается модель микроволновой печи, так как в реальности открывание двери непосредственно размыкает ее питающую цепь.

Запуск автоматов в событийных системах происходит из обработчиков событий. При этом в функцию, реализующую автомат, номер произошедшего события передается в виде параметра. Структура программ в общем случае содержит:

– обработчики событий;

– систему взаимосвязанных автоматов;

– функции, реализующие входные переменные;

– функции, реализующие выходные воздействия;

– функции протоколирования;

– вспомогательные модули.

2.2. Заключение

В заключение раздела отметим, что в предыдущие десятилетия большинство программистов имели либо инженерное, либо математическое образование с соответствующей, устоявшейся веками, культурой. Современные программисты воспитываются иначе [10], а дисциплине программирования должного внимания не уделяется.

Предлагаемая технология является новой попыткой введения такой дисциплины и основана на априорном задании требуемых состояний и их визуализации.

Опыт применения предлагаемой технологии подтвердил следующее высказывание: «…то, что не специфицировано формально, не может быть проверено, а то, что не может быть проверено, не может быть безошибочным" [11]. Поэтому авторы надеются (особенно учитывая мнение о работе [1], высказанное в [12]), что предложенный подход, по крайней мере, для систем логического управления и «реактивных» систем является приближением к «серебряной пуле» [13] в части создания качественных программ, тем более, что Ф. Брукс отозвался благосклонно только о подходе Д. Харела [14], также основанном на применении автоматов, преимущество по сравнению с которым показано в [15].

Целесообразность использования автоматного подхода подтверждается и тем, что создатель операционной системы UNIX К. Томпсон на вопрос о текущей работе ответил: «Мы создали язык генерации машин с конечным числом состояний, так как реальный селекторный телефонный разговор — это группа взаимодействующих машин с конечным числом состояний. Этот язык применяется в «Bell Labs» по прямому назначению — для создания указанных машин, кроме того с его помощью стали разрабатывать драйверы» [16].

Применение в предлагаемой парадигме понятия «автомат» в качестве центрального, соответствует его месту в теории

управления, что принципиально отличает этот подход от других парадигм программирования.

Предложенный подход для рассматриваемых классов систем, весьма распространенных на практике, в соответствии с принципом Оккама "не размножает сущности без необходимости" (как, например, UML) и обладает «минимализмом» [17], достаточным для обеспечения понимаемости программ специалистами, не являющимися разработчиками этих программ.

Отметим также, что предложенная парадигма существенно отличается от изложенной в работе [18]. Основное отличие заключается в том, что предлагаемая технология, не зависящая от сред разработки и реализации, а в работе [18] рассмотрены программные продукты для моделирования.

Рассматриваемая парадигма автоматного программирования, которая может быть названа также «программирование с явным выделением состояний», начинает все шире использоваться [19–21]. Для большей ее популяризации создан сайт http://is.ifmo.ru.

Глава 3. Применение автоматного подхода для программной реализации вычислительных алгоритмов (этап 3)

Целью настоящего этапа является описание вопросов применения автоматного программирования применительно к вычислительным алгоритмам (алгоритмам дискретной математике).

При этом в частности, рассмотрено использование вычислительной модели, основанной на изменении машины Тьюринга за счет применения структурного автомата с произвольными множествами входных и выходных воздействий вместо абстрактного. Эта модель может быть использована при реальном программировании, что показано на примерах.

Предлагался метод преобразования итеративных программ в автоматные программы, который может быть применен также для преобразования неструктурированных схем алгоритмов в автоматные программы. Показано, что рассматриваемый метод обладает рядом достоинств по сравнению с методом Ашкрофта-Манны. Предлагаемый метод иллюстрируется примерами.

Кроме того, предложен метод преобразования программ с явной рекурсией в итеративные автоматные программы. Приведены примеры классических задач, решаемых с использованием предлагаемого подхода.

3.1. Преобразование итеративных алгоритмов в автоматные

Для систем логического управления и событийных систем выделение состояний является естественным, так как они в значительной мере определяются физическими состояниями объектов управления.

Для вычислительных алгоритмов выделение состояний не столь естественно, однако и для этого класса задач автоматный подход может быть применен.

Такой подход позволяет унифицировать процесс построения структурированных программ с визуализацией их состояний, что имеет большое значение особенно для целей обучения. Это не удается обеспечить при традиционном процедурном подходе, так как в нем в качестве базовых используются понятия «условие» и «действие», а не «состояние.

Программы, построенные на основе явного выделения состояний, назовем *автоматными*.

Психологические трудности [22], возникающие при непосредственном построении алгоритмов для вычислительных задач с применением автоматов, связаны, в частности, с непривычной реализацией циклов. Эти трудности привели к тому, что в работе [23] автоматный подход был использован только для

построения одного из алгоритмов поиска подстрок, в то время как для десятков других алгоритмов, описанных в этой книге, автоматы не применялись ни при алгоритмизации, ни при реализации.

Разделим рассматриваемый класс алгоритмов на два подкласса: без использования событий (вычислительные алгоритмы) и с их использованием (событийно-вычислительные алгоритмы).

Настоящая работа посвящена первому из указанных подклассов. В этом подклассе будем рассматривать схемы итеративных алгоритмов с одним входом и одним выходом, в которых через каждую вершину проходит путь от входа схемы к ее выходу (отсутствуют недостижимые вершины и бесконечные циклы).

Покажем, что схему произвольного итеративного алгоритма (программы), принадлежащую этому подклассу, можно реализовать одним оператором `do-while`, телом которого является один оператор `switch`. Последний изоморфно реализует граф переходов конечного автомата, формально построенный по исходной схеме. Отметим, что в отличие от событийных систем, в рассматриваемом подклассе алгоритмов при реализации каждого автомата используется один оператор `switch`.

Эта реализация проще получаемой при применении метода Ашкрофта-Манны [24, 25], который требует введения состояния для каждой вершины неструктурированной схемы алгоритма и приводит к построению его структурированной схемы с числом вершин, определяемым соотношением:

$$B = 4Y + 3O + 2,$$

где Y и O — количество условных и операторных вершин в неструктурированной схеме алгоритма, соответственно.

Подход к структурированию программ с использованием булевых признаков [25] в настоящей работе не рассматривается, так как он является эвристическим.

Ниже излагается метод построения по схеме итеративного алгоритма (программы) графа переходов, по которому, в свою очередь, строится структурированная программа, являющаяся автоматной. Эта схема может быть как структурированной, так и неструктурированной.

Метод состоит из этапов, перечисленных ниже.

1. Используя подход, предложенный в работе [26] для аппаратных реализаций схем алгоритмов, в заданную схему в зависимости от выбранного для ее замены типа автомата (Мура или Мили) вводятся пометки, соответствующие номерам его состояний. При этом для автоматов обоих типов начальной и конечной вершинам схемы присваивается номер 0, что обеспечивает построение "замкнутого" графа переходов. При построении автомата Мура k-й группе соединенных последовательно операторных вершин (она может состоять также и из одной вершины) присваивается номер k. При построении автомата Мили соответствующий номер присваивается точке, следующей за последней из последовательно соединенных операторных вершин группы, причем указанные точки для различных групп могут совпадать. Это приводит к тому, что автомат Мили имеет число состояний, не превышающее их число в эквивалентном автомате Мура. Номера, применяемые для построения автомата Мили, будем указывать на схеме алгоритма в скобках, а номера для построения автомата Мура — без скобок. Используемая нумерация начальной и конечной вершин отличается от принятой в работах [24, 25], где этим вершинам присваиваются различные номера, а получающийся при этом граф переходов разомкнут.

2. Обеспечение «замкнутости» графа переходов и выбранная стандартная программная реализация могут привести к необходимости введения дополнительного состояния (как для автоматов Мили, так и для автоматов Мура), если схема алгоритма содержит контур без операторных вершин и пометок других состояний.
3. В случае построения автомата Мили в схеме алгоритма за счет дублирования некоторых операторных вершин уменьшается число точек, следующих за ними, что обеспечивает сокращение числа состояний автомата этого типа.
4. В схеме программы выделяются пути между смежными точками (пометками), включая пути, начинающиеся и оканчивающиеся в одной и той же точке. Для каждого пути выписываются условия перехода и выполняемые при этом действия. На основе выделенных состояний и переходов (включая петли) строится граф переходов автомата выбранного типа.
5. Построенный граф переходов изоморфно реализуется с помощью оператора `switch` языка *C* или его аналога из других языков программирования.
6. Полученный фрагмент программы используется в качестве тела оператора `do-while`, условием выхода из которой является нахождение автомата в состоянии `0`.

При необходимости после получения графа переходов по нему может быть построена структурированная схема программы, соответствующая оператору `do-while`, тело которого изоморфно графу переходов и содержит дешифратор состояний. Схемы с такой структурой назовем *автоматными*.

В отдельных случаях в построенной схеме может быть уменьшено число вершин за счет потери указанного изоморфизма (но не структурированности), по аналогии с тем, как это выполняется в работе [27] для схем, построенных методом Аш-

крофта-Манны, с помощью перехода к рекурсивным структурированным схемам.

Завершив изложение метода, отметим, что подход, описанный в работе [26], состоит только из первого и четвертого этапов.

С помощью предлагаемого метода решаются две задачи: преобразование итеративных программ (обычно структурированных) в автоматные и преобразование неструктурированных схем алгоритмов в автоматные.

Сопоставление рассмотренных в настоящей работе традиционных схем алгоритмов и графов переходов показывает, что последние более компактны и понятней специфицируют алгоритм. Это связано с тем, что построение графа переходов сводится к исследованию всех путей между соседними пометками в традиционной схеме алгоритма и компактному их представлению в виде помеченных дуг и петель графа.

Преимущество автоматного подхода по сравнению с традиционным достигается за счет добавления к понятиям «входное и выходное воздействия» понятия «состояние», а по сравнению с методом Ашкрофта-Манны — за счет сокращения числа состояний.

Предложенное для вычислительных алгоритмов явное выделение состояний позволяет рассматривать такие алгоритмы не в терминах условных переходов (и, соответственно, флагов), а в терминах состояний, что более наглядно отражает их суть. Кроме того, задание вычислительных алгоритмов в виде графов переходов позволяет формализовать задачу их визуализации, что существенно упрощает их отладку и особенно важно с точки зрения обучения [28].

В работе предложен также метод преобразования программ с явной рекурсией в автоматные программы.

Программа с явной рекурсией преобразуется в итеративную программу, построенную с применением, например, автомата Мили. При этом работа рекурсивной программы моделируется итеративной автоматной программой, в которой явно используется стек.

Явное выделение стека по сравнению со «скрытым» его применением в рекурсии, позволяет программно задавать его размер, следить за его содержимым и добавлять отладочный код в функции, реализующие операции над стеком.

Предлагаемый метод иллюстрируется примерами преобразований классических рекурсивных программ, которые приведены в порядке их усложнения (факториал, числа Фибоначчи, задача о ранце, ханойские башни).

На примере задачи о ханойских башнях рассматривается подход, в котором при реализации автоматной программы применяются классы. Это позволяет ограничить область видимости используемых переменных пределами класса и эффективно реализовать сам автомат и его входные и выходные воздействия в виде методов.

Показано, что метод, предложенный для построения автоматных программ по итеративным программам, применим также и к рекурсивным программам.

В работе [29] отмечено, что итеративные алгоритмы по вычислительной мощности эквивалентны рекурсивным. Однако, в известной авторам литературе, формальный метод преобразования рекурсивных алгоритмов в итеративные отсутствует. В работах [30,31] приведены примеры такого преобразования (для концевых рекурсий), выполненного эвристически. Предложенный в ходе выполнения этого этапа работ метод устраняют указанный пробел.

Можно считать, что предложенные методы преобразования итеративных и рекурсивных программ в автоматные обеспечивают преобразование процедурных знаний в декларативные [32].

Глава 4. Применение автоматного подхода в объектно-ориентированном программировании (этап 4)

На четвертом этапе формировались основы объектно-ориентированного программирования с явным выделением состояний.

Подробно рассматривались два подхода:

- реализация автоматов, как методов классов;
- реализация автоматов, как классов.

Показано, что первый из этих подходов позволяет локализовать в методах классов их поведение, проводить автоматическое протоколирование, описывающее работу всех автоматов в терминах состояний, переходов, входных и выходных воздействий с указанием соответствующих объектов, а также упрощает понимание проектной документации.

Второй подход позволяет в полной мере совместить гибкость объектно-ориентированного программирования с наглядностью и ясностью автоматного подхода.

Предложена методика преобразования объектной программы, написанной в рамках второго подхода, в процедурную программу для выполнения ее, например, на микроконтроллере.

Кроме указанных подходов был приведен также ряд других методов совместного использования автоматного и объектно-ориентированного стилей программирования. Эти методы подробно рассмотрены в рамках соответствующих проектов, опубликованных на сайте http://is.ifmo.ru (раздел «Проекты»). Созданные подходы могут быть классифицированы следующим образом.

1. Автоматы, как методы классов. Этот подход основан на процедурном стиле программирования [33] и может быть назван «обертыванием автоматов в классы».
2. Автоматы, как классы без использования класса базового автомата [34].
3. Автоматы, как классы с использованием класса базового автомата. Этот подход основан на совместном применении всех преимуществ, как объектного, так и автоматного стилей программирования. При этом автоматы разрабатываются, как наследники класса, реализующего базовую функциональность. Базовый класс и другие необходимые классы помещаются в библиотеку, предоставляемую разработчику.

   3.1. В работе [35] приводится пример простейшей библиотеки классов для разработки программного обеспечения в рамках объектно-ориентированного программирования с явным выделением состояний. В нее входят два класса: класс Automat для дальнейшего наследования от него создаваемых автоматов и вспомогательный класс AutoService. Первый из них обеспечивает, в частности, выполнение шага и пост-шага автомата, автоматическое протоколирование и т.п. Класс AutoService используется при протоколировании.

   3.2. В работе [36] 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.

3.3. В работе [37] предложена еще одна библиотека для объектно-ориентированной реализации автоматов, названная Auto-Lib.

3.4. В работе [38] предложена библиотека, позволяющая «собирать» простые автоматы из наследников базовых классов «состояние автомата» и «переход между состояниями». Эта библиотека позволяет обеспечить изоморфизм между текстом программы и графом переходов при наличии в нем групповых переходов.

4. Использование паттернов проектирования [39]. Наряду с применением библиотек при объектно-ориентированной реализации автоматов могут использоваться и разрабатываться паттерны проектирования.

   4.1. Описанный в работе [40] паттерн Automat позволяет проектировать и реализовывать программное обеспечение, пользуясь классами, реализующими 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 [39], реализует абстракцию «состояние». Для реализации конкретного состояния необходимо разработать наследника базового класса State и переопределить в нем функцию переходов.

   Похожий подход рассмотрен в работе [41]. В ней для каждого автомата создан базовый класс «Состояние», от которого наследуются конкретные классы, реализующие состояния данного автомата. Переходы между состояниями обеспечиваются базовыми классами состояний, а непосредственно осуществляются в классах наследниках.

5. Интерпретируемое описание автоматов.

5.1. В работах [42, 43] предложен подход, позволяющий автоматически преобразовывать графы переходов в текстовое описание в формате XML. На языке Java разработана среда исполнения полученного XML-описания. При этом сначала указанное описание однократно и целиком преобразуется в соответствующее внутреннее объектное представление программы. В результате образуется система, состоящая из среды исполнения и объектного представления программы. 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.

5.2. В работе [44] предложено использовать *XML* для автоматного описания динамики изменения внешнего вида виртуального устройства – видеопроигрывателя Crystal Player (http://www.crystalplayer.com).

6. Механизм обмена сообщениями и автоматы.

6.1. В ходе выполнения работ [45-47] по реализации классического параллельного алгоритма синхронизации цепи стрелков выяснилось, что автоматы, построенные по предложенному в работе [33] шаблону, реализующему событийно-управляемые автоматы, который состоит из двух операторов switch, не позволяет реализовать взаимодействующие параллельные процессы. Для решения этой задачи в работе [45] было предложено использовать механизм обмена сообщениями. Для этого была разработана библиотека SWMEM (Switch Message Exchange Mechanism). При этом в шаблон для реализации автоматов были внесены сле-

дующие 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.

6.2. В работе [48] механизм обмена сообщениями между параллельно «расположенными» автоматами реализуется за счет введения такой сущности, как «общая шина». Этот подход позволяет однотипно реализовать разнотипные по своей природе алгоритмы, которые могут быть иерархическими, вложенными или параллельными. Для реализации параллельно работающих автоматов было предложено изменить шаблоны, рассмотренные в работах [33, 45], строя автомат с помощью двух функций: функции перехода-действия и функции обновления. Первая из них сначала осуществляет выходные 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. В общем цикле работы автоматов для их синхронизации сначала должны вызываться все функции перехода-действия, а затем – все функции обновления. После этого обновляется массив сообщений, посылаемых в «общую шину».

Существуют также и другие подходы к совместному использованию объектного и автоматного подходов.

Глава 5. Применение методологии создания программного обеспечения систем управления на основе автоматного подхода (этап 5).

Методологии создания программного обеспечения на основе автоматного подхода, разработанная в ходе выполнения предыдущих этапов, была использована при решении широкого круга задач.

При этом ниже в качестве примеров описывается применение этой методологии при создании:

- мультиагентной системы, построенной с использованием конструктора «Lego Mindstorms»;
- инструментального средства UniMod, предназначенного для поддержки автоматного подхода с использованием нотации языка UML для платформы Eclipse.

5.1. Автоматный подход к разработке управляющих программ для реактивных мультиагентных систем

Автоматный подход, предложенный авторами для разработки управляющей программы виртуального, автономного реактивного агента, применяемого в известной игре «Robocode», в настоящем разделе распространяется на создание управляющих программ для реактивных мультиагентных систем, работающих в реальных условиях.

Эффективность предлагаемого подхода демонстрируется на примере создания транспортных систем, состоящих из двух взаимодействующих роботов, собранных из конструктора «Lego Mindstorms». Преимущества подхода состоят в формализации процесса реализации программ и в упрощении их тестирования и внесения изменений. Все проектные решения предлагается документировать.

5.1.1. Введение в тему

В последнее время все большее внимание уделяется программированию мультиагентных систем [73-76]. Одним из важнейших классов агентов [77-80] являются реактивные агенты [78-80].

Для описания поведения реактивных агентов целесообразно применение конечных автоматов [81-83], которые используются также для описания "психологии поведения" в работе [84] и соответствуют концепции интеллекта как системы управления поведением [85].

Несмотря на теоретические исследования по применению автоматов при программировании реактивных мультиагентных систем [86] и использование автоматов в отдельных проектах, они обычно не применяются при создании программного обеспечения агентов-роботов, которые могут быть созданы, например, с помощью всемирно известного конструктора Lego Mindstorms. В частности, в работах [16,17] программировать эти роботы Lego Mindstorms предлагается традиционным путем.

Цель настоящей работы - описать автоматный подход к разработке управляющих программ реактивных мультиагентных систем. Подход демонстрируется на примере создания управляющих программ транспортной системы, состоящей из двух взаимодействующих роботов Lego Mindstorms.

5.1.2. Автоматный подход

Основные положения этого подхода были разработаны при создании управляющей программы для виртуального, автономного реактивного агента, применяемого в известной игре «Robocode» [87-89].

Для систем реактивных агентов, работающих в реальных условиях, этот подход может быть сформулирован следующим образом.

1. Строится схема взаимодействия агентов.

2. Для каждого агента разрабатывается диаграмма классов.
3. Для каждого класса создается его структурная схема.
4. Если поведение класса описывается системой автоматов, то строится схема их взаимодействий. Автоматы могут взаимодействовать тремя способами:
   - ü по вложенности;
   - ü по вызываемости;
   - ü за счет обмена номерами состояний.
5. Для каждого автомата создается четыре документа:
   - ü словесное (вербальное) описание поведения автомата;
   - ü схема связей автомата, задающая его интерфейс, в которой указываются символьные обозначения входных воздействий и выходных переменных и их полные названия, а также названия источников и приемников информации;
   - ü граф переходов, в котором состояния имеют названия, а все остальные составляющие помечены символами. Это позволяет компактно и формально описывать даже весьма сложное поведение агентов;
   - ü по графу переходов формально и изоморфно строится текст программы.

   Отметим, что выходные переменные могут являться функциями, в том числе весьма сложными.
6. Для каждого класса пишется программа, изоморфная его структурной схеме.
7. Управляющие программы загружаются в соответствующие агенты.
8. Выполняется отладка системы, которая упрощается за счет возможности наблюдения за состояниями каждого автомата.
9. Осуществляется разработка и выпуск открытой проектной документации [90].

5.1.3. Пример. Постановка задачи

Разработать на основе конструктора Lego Mindstorms транспортную систему, обеспечивающую доставку заданного количество предметов небольшого размера из одного места в другое. Доставка предметов ограничивается размерами небольшой комнаты или стола.

5.1.4. Пример. Результаты

Используя свободно распространяемую программу MLCAD, роботы были построены виртуально, а затем по полученным чертежам - реально.

Разработанная система состоит из транспортного робота и робота-поставщика, а их взаимодействие осуществляется путем обмена сообщениями через инфракрасные порты.

Транспортный робот должен перемещаться от места доставки до места выдачи предметов и обратно по определенному пути - черной линии на белом поле. Место доставки и место выдачи предметов отмечены соответствующими кусками фольги. Препятствий на пути нет. Цвета черной линии и поля предполагаются равномерными.

Робот-поставщик стоит на месте выдачи предметов. Он может получать сообщения, передаваемые с помощью инфракрасного порта транспортного робота о выдаче необходимого количества предметов (одного, двух или трех), который, в свою очередь, получает аналогичные сообщения с пульта оператора.

Каждый робот содержит микроконтроллер, в память которого через указанный инфракрасный порт могут быть загружены операционная система (ОС) и разработанная управляющая программа. Память микроконтроллера содержит всего 32 Кб, а самая продвинутая на сегодняшний день ОС lejOS занимает 17 Кб.

Для каждого из роботов была создана управляющая программа на языке *Java* под ОС lejOS. Из изложенного следует,

что в каждом роботе для управляющих программ остается мало памяти. Поэтому несмотря на использование языка *Java*, программирование было выполнено практически процедурно – автоматы «обернуты» классами.

Программа управления транспортным роботом содержит два класса, первый из которых обеспечивает посылку сообщений роботу-поставщику и поворот для движения в обратном направлении, а второй – движение робота по линии.

Каждый класс включает по одному автомату с восемью состояниями. У первого автомата состояния называются «Начало», «Инициализация», «Движение», «Поворот», «Отъезд», «Ожидание», «Авария», «Сообщение с пульта», а у второго - «Начало», «Конец», «Авария», «Поиск линии», «Поворот», «Корректировка», «Движение вперед», «Подтверждение цвета».

Взаимодействие автоматов выполняется за счет вызова второго автомата в состоянии “Движение по линии” первого автомата.

Программа управления роботом-поставщиком реализуется одним классом, содержащим автомат с шестью состояниями («Начало», «Выдача предметов», «Ожидание сообщения», «Посылка сообщения», «Корректировка», «Вращение»).

Важнейшее достоинство предлагаемого подхода проявилось на этапе отладки системы. Во-первых, она почти сразу стала работать, а ее доводка выполнялась сравнительно просто, так как имелась возможность на отладочный экран для каждого робота выводить номер состояния, в котором он находится. В используемых роботах на каждый экран можно вывести только одно слово из четырех символов. При таких ограниченных ресурсах при другом подходе эффективная отладка весьма затруднительна.

5.1.5. Выводы по разделу

Из рассмотренного примера следует, что в рамках предлагаемого подхода состояния в программе, как и в машине Тьюринга, разделены на управляющие, которых сравнительно немного, и вычислительные, число которых практически неограниченно [91]. При этом, находясь в одном из управляющих состояний, автомат может проходить большое число вычислительных состояний, что позволяет (как и в гибридных автоматах [92]) при необходимости реализовывать сложное поведение каждого агента. Эта идея использована при разработке программного обеспечения для виртуального, автономного реактивного агента в проекте [93] и реальных взаимодействующих реактивных агентов в проекте [94], в котором подробно изложено решение рассматриваемого примера.

Из рассмотрения документации на эти проекты, которая разрабатывалась в рамках "Движения за открытую проектную документацию" [90], следует, что предлагаемый подход отделяет проектирование агентов от их программирования [95, 96] и позволяет создавать программное обеспечение высокого качества, в которое весьма легко вносить изменения.

В заключение отметим, что в отличие от известных работ, в настоящей работе используются два типа взаимодействий: между агентами и между автоматами в одном агенте. Причем в обоих случаях может быть использована вложенность.

Проект был выполнен при поддержке ЗАО «Аркадия» (Россия, Санкт-Петербург, http://www.arcadia.spb.ru/), за что авторы благодарны ее генеральному директору Аркадию Григорьевичу Хотину.

5.2. UML. SWITCH-технология. Eclipse

В настоящем разделе описываются метод и процесс моделирования поведения программы с явным выделением состояний,

основанный на Switch-технологии и UML-нотации. Также описано создание для платформы Eclipse инструмента, поддерживающего этот метод.

5.2.1. Введение

В последнее время идея «запускаемого *UML*» [95] приобретает все большую популярность. Это связано с тем, что практическое использование Unified Modeling Language (UML) [96], в большинстве случаев ограничивается применением при моделировании статической части программы с помощью диаграмм классов. Моделирование динамических аспектов программы на языке *UML* затруднено в связи с отсутствием в стандарте формального однозначного описания правил интерпретации поведенческих диаграмм. Также следует отметить, что связь поведенческих диаграмм с кодом на целевом языке программирования в современных широко известных средствах моделирования, например *IBM Rational Rose*, реализована слабо, либо вообще не реализована.

Ивар Якобсон, один из создателей языка моделирования *UML*, в докладе «Четыре основные тенденции в разработке программного обеспечения (ПО)» [96] перечислил важнейшие, по его мнению, направления развития процесса разработки ПО.

Он отметил, что технологическая база разработки объектно-ориентированного ПО, состоящая из языка *UML* и стандартного процесса разработки Rational Unified Process (RUP) [98], приобрела устойчивое состояние. По его мнению, следующим шагом должно стать их широкое внедрение. Уже сегодня можно говорить что «процесс пошел»: И. Якобсон утверждает, что язык *UML* преподается более чем в 1000 университетах мира, а, в присутствии авторов настоящего отчета, представители фирмы IBM в одном из своих докладов в течение четырех часов демонстрировали на примере использование *UML* и *RUP*.

И. Якобсон обозначил следующие тенденции разработки ПО.

1. Аспектно-ориентированное программирование [99]. Аспекты упрощают добавление в систему сквозной функциональности, такой, например, как протоколирование или проверка прав доступа. Отметим, что И. Якобсон отождествляет понятие аспекта с вариантом использования, что позволяет моделировать аспекты с помощью языка *UML*.
2. Исполняемый *UML*. В настоящее время *UML* применяется, в основном, как язык спецификации моделей систем. Существующие *UML*-средства позволяют строить различные диаграммы и автоматически создавать по диаграмме классов «скелет» кода на целевом языке программирования (языки *Java* и *C#*). Некоторые их этих средств также предоставляют возможность автоматически генерировать код логики программы по диаграммам состояний.

   Однако можно считать, что в настоящее время указанная функциональность существует в «зачаточном состоянии», так как известные инструменты не позволяют в полной мере эффективно связывать модель поведения, которую можно описывать с помощью четырех типов диаграмм (состояний, деятельностей, кооперации или последовательностей), с генерируемым кодом. Это во многом определяется отсутствием в языке *UML* формального однозначного описания операционной семантики для поведенческих диаграмм. Отметим, что в новой редакции языка *UML* (*UML 2*) такая семантика должна появиться.

   Отсутствие однозначной указанной операционной семантики при традиционном написании программ приводит к дублированию описания поведения, как в модели, так и на

целевом языке, а также к произвольной интерпретации поведенческих диаграмм программистом. Более того, описание поведения в модели часто носит неформальный характер. Появление операционной семантики зафиксирует однозначность понимания диаграмм и приведет к появлению исполняемого *UML*, для которого код, в привычном смысле этого слова, может не генерироваться вообще.

3. Процесс разработки ПО должен быть активным. Существующие средства разработки требуют длительного времени для их изучения. И. Якобсон считает, что средства разработки должны предсказывать действия разработчика и предлагать варианты решения возникших проблем в зависимости от текущего контекста. Отметим, что подобный подход реализован во многих современных средах разработки (например, *Borland JBuilder*, *Eclipse*, *IntelliJ IDEA*) для текстовых языков программирования, но не для языка *UML.*
4. Разработанное ПО также должно быть активным, однако, не для разработчика, а для конечного пользователя.

По мнению авторов, наиболее интересными и востребованными на сегодняшний день тенденциями являются вторая и третья.

Признание многими ведущими в области разработки ПО фирмами того факта, что программы необходимо не писать «на авось» (как сказал по-русски на конференции «Microsoft Research Academic Days in St.-Petersburg, April 21-23, 2004» создатель языка *Eiffel* Бертран Мейер), а проектировать, привело к появлению такого направления в программной инженерии как «проектирование на базе моделей» (Model-Driven Design) [100-102].

Основной идеей такого подхода является независимое рассмотрение моделей, создаваемых при проектировании системы, от деталей их реализации на конкретной программно-аппаратной платформе. Проектирование на базе моделей должно привести к появлению универсальных графических языков программирования.

Далее (на примере разработанного авторами проекта с открытым исходным кодом *UniMod* (http://unimod.sourceforge.net)) описаны применение UML-нотации для создания графического языка описания систем со сложным поведением и инструмент моделирования для этого языка, который поддерживает активный процесс разработки ПО.

При этом отметим, что идея применения *UML*, как запускаемого языка для таких систем, не нова и описана, например, в работах [103,104]. Однако подход, предлагаемый авторами, как будет показано ниже, более универсален и лучше формализован.

5.2.2. Исполняемый графический язык на основе Switch-технологии и UML-нотации

Для создания графического языка диаграммы необходимо наделить операционной семантикой.

В работе [105] предложен метод проектирования событийных объектно-ориентированных программ с явным выделением состояний, названный «Switch-технологией». Особенность этого подхода состоит в том, что поведение в таких программах описывается с помощью графов переходов структурных конечных автоматов с нотацией, предложенной в работе [33].

Switch-технология определяет для каждого автомата два типа диаграмм (схему связей и граф переходов) и их операционную семантику. При наличии нескольких автоматов также строится схема их взаимодействия. Switch-технология задает

свою нотацию диаграмм. Предлагается, сохранив автоматный подход, использовать UML-нотацию при построении диаграмм в рамках Switch-технологии. При этом, используя нотацию диаграмм классов языка *UML*, строятся схемы связей автоматов, определяющих их интерфейс, а графы переходов создаются с помощью нотации диаграммы состояний *UML*.

Предлагаемый процесс моделирования системы состоит в следующем:

- на основе анализа предметной области разрабатывается концептуальная модель системы, определяющая сущности и отношения между ними;
- в отличие от традиционных для объектно-ориентированного программирования подходов [106], из числа сущностей выделяются источники событий, объекты управления и автоматы. Источники событий активны — они по собственной инициативе воздействуют на автомат. Объекты управления пассивны — они выполняют действия по команде от автомата. Объекты управления также формируют значения входных переменных для автомата. Автомат активируется источниками событий и на основании значений входных переменных и текущего состояния воздействует на объекты управления, переходя в новое состояние;
- используя нотацию диаграммы классов, строится схема связей автомата, задающая его интерфейс. На этой схеме слева отображаются источники событий, в центре — автоматы, а справа — объекты управления. Источники событий с помощью UML-ассоциаций связываются с автоматами, события которым они поставляют. Автоматы соединяются с объектами, которыми они управляют;

- схема связей, кроме задания интерфейса автомата, выполняет функцию, характерную для диаграммы классов — задает объектно-ориентированную структуру программы;
- каждый объект управления содержит два типа методов, реализующих входные переменные (xj) и выходные воздействия (zk);
- для каждого автомата с помощью нотации диаграммы состояний строится граф переходов типа Мура-Мили, в котором дуги могут быть помечены событием (ei), булевой формулой из входных переменных и формируемыми на переходах выходными воздействиями. В вершинах могут указываться выходные воздействия и имена вложенных автоматов. Каждый автомат имеет одно начальное и произвольное количество конечных состояний;
- состояния на графе переходов могут быть простыми и сложными. Если в состояние вложено другое состояние, то оно называется сложным. В противном случае состояние простое. Основной особенностью сложных состояний является то, что наличие дуги, исходящей из такого состояния, заменяет однотипные дуги из каждого вложенного состояния;
- все сложные состояния неустойчивы, а все простые, за исключением начального — устойчивы. При наличии сложных состояний в автомате появление события может привести к выполнению более одного перехода. Это происходит в связи с тем, что сложное состояние является неустойчивым и автомат осуществляет переходы до тех пор, пока не достигнет первого из простых (устойчивых) состояний. Отметим, что если в графе переходов сложные состояния отсутствуют, то, как и в Switch-

технологии, при каждом запуске автомата выполняется не более одного перехода;

- каждая входная переменная и каждое выходное воздействие являются методами соответствующего объекта управления, которые реализуются вручную на целевом языке программирования;
- использование символьных обозначений в графах переходов позволяет весьма компактно описывать сложное поведение проектируемых систем. Смысл таких символов задает схема связей. При наведении курсора на соответствующий символ на графе переходов во всплывающей подсказке отображается его текстовое описание.

На рис. 8 приведена схема связей автомата, а на рис. 9 — его граф переходов, которые построены в UML-нотации описанным выше способом.

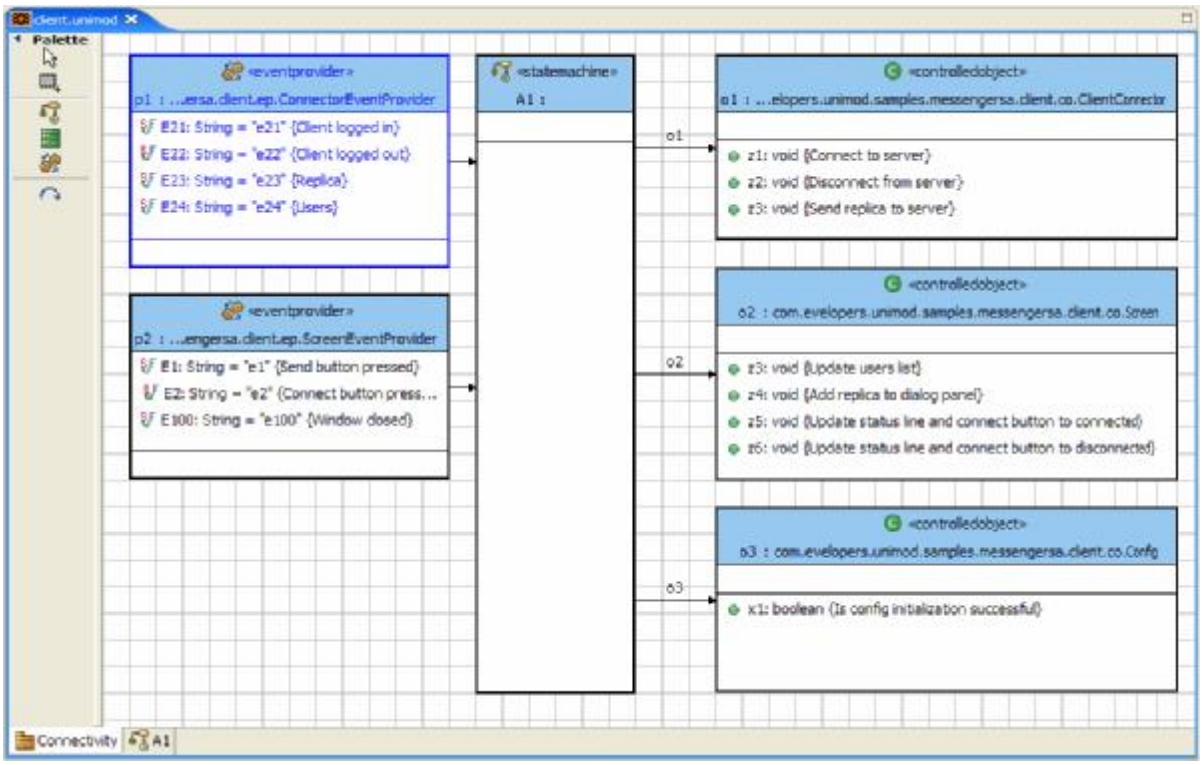

Рис. 8. Пример схемы связей автомата

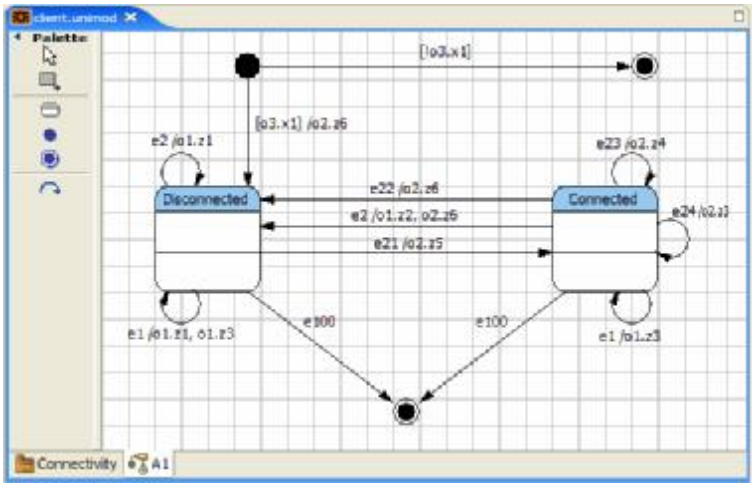

Рис. 9. Пример графа переходов автомата

Зададим операционную семантику для модели системы:

- при запуске модели инициализируются все источники событий. После этого они начинают воздействовать на связанные с ними автоматы;
- каждый автомат начинает свою работу из начального состояния, а заканчивает — в одном из конечных;
- при получении события автомат выбирает все исходящие из текущего состояния переходы, помеченные символом этого события;
- автомат перебирает выбранные переходы и вычисляет булевы формулы, записанные на них, до тех пор, пока не найдет формулу со значением true;
- если переход с такой формулой найден,то автомат выполняет выходные воздействия, записанные на дуге, и переходит в новое состояние. В нем автомат выполняет выходные воздействия, а также запускает вложенные автоматы;
- если переход не найден, то автомат продолжает поиск перехода из состояния, в которое вложено текущее состояние;

- при переходе в конечное состояние автомат останавливает все источники событий. После этого работа системы завершается.

Описав исполняемый графический язык на основе *UML*-нотации и его операционную семантику, перейдем к описанию процесса создания инструмента моделирования, который будет поддерживать активный процесс разработки программ на его основе.

5.2.3. *UniMod* — пакет для автоматно-ориентированного программирования

Пакет *UniMod* обеспечивает разработку и выполнение автоматно-ориентированных программ. Он позволяет создавать и редактировать UML-диаграммы классов и состояний, которые соответствуют схеме связей и графу переходов, и поддерживает два типа реализации — на основе интерпретации и компиляции. В первом случае имеется возможность:

- преобразовывать диаграммы в формат *XML;*
- исполнять XML-описание с помощью интерпретатора, созданного на основе набора разработанных базовых классов. Эти классы реализуют, например, такие функции как обработка событий, сохранение текущего состояния, протоколирование.

Во втором случае диаграммы непосредственно преобразуется в код на целевом языке программирования, который впоследствии компилируется и запускается.

Проектирование программ с использованием пакета *UniMod* предполагает следующий подход: логика приложения описывается структурным конечным автоматом, заданным в виде набора указанных выше диаграмм, построенных с использованием UML-

нотации. Источники событий и объекты управления задаются кодом на целевом языке программирования.

При применении языка Java источникам событий и объектам управления соответствуют классы. Для этого языка в пакете *UniMod* реализован интерпретатор XML-описаний структурных конечных автоматов, которые пакет строит на основе указанных диаграмм. При запуске программы интерпретатор загружает в оперативную память XML-описание и создает экземпляры источников событий и объектов управления. В процессе работы указанные источники формируют события и направляют интерпретатору, который обрабатывает их в соответствии с логикой, описываемой автоматом. При этом он вызывает методы объектов управления, реализующие входные переменные и выходные воздействия.

Компилируемый подход целесообразно применять для устройств с ограниченными ресурсами, например для мобильных телефонов. Указанный подход является типичным для «классической» Switch-технологии.

5.2.4. Реализация редактора диаграмм на платформе *Eclipse*

Инструмент для создания указанных диаграмм является встраиваемым модулем (plug-in) для платформы *Eclipse* (http://www.eclipse.org). Эта платформа обладает рядом преимуществ перед такими продуктами, как, например, *IntelliJ IDEA* или *Borland JBuilder*:

- является бесплатным продуктом с открытым исходным кодом;
- содержит библиотеку для разработки графических редакторов – Graphical Editing Framework;

- активно развивается фирмой *IBM* и уже сейчас обладает не меньшей функциональностью, чем упомянутые выше аналоги.

Для обеспечения процесса активной разработки программ на текстовых языках в перечисленных выше средствах разработки реализованы:

- подсветка семантических и синтаксических ошибок;
- автоматическое завершение ввода и автоматическое исправление ошибок;
- форматирование и рефакторинг [107] кода;
- запуск и отладка программы внутри среды разработки.

В английском языке эти возможности называются «code assist». При создании модуля для платформы *Eclipse* авторы перенесли указанные подходы на процесс редактирования диаграмм.

Проверка синтаксиса и семантики

Для текстовых языков программирования редакторы осуществляют проверку принадлежности программы к заданному языку и выделяют (подсвечивают) места в коде, содержащие синтаксические ошибки. К семантическим ошибкам для текстовых языков программирования относится, например, использование необъявленных переменных, вызовы несуществующих методов, некорректное приведение типов.

В стандарте на язык *UML* синтаксис и семантика диаграмм определяется набором ограничений, записанных на языке объектных ограничений (Object Constraint Language). Данный набор ограничений должен удовлетворяться для любой правильно построенной диаграммы. Именно на этих ограничениях и основана проверка синтаксиса и семантики диаграмм.

Авторами предлагается расширить множество ограничений следующим образом:

- все состояние на диаграмме состояний должны быть достижимы;
- множество исходящих переходов для любого состояния должно быть полно и непротиворечиво. Это означает, что при обработке любого события не должно быть альтернативных переходов и хотя бы один переход должен выполняться.

Проверка корректности диаграмм происходит следующим образом. В фоновом режиме запускается процесс, который при любом изменении диаграммы, проверяет ее на корректность. При нахождении ошибки некорректный элемент на диаграмме выделяется цветом. Так на рис. 10 приведен пример диаграммы с недостижимым состоянием.

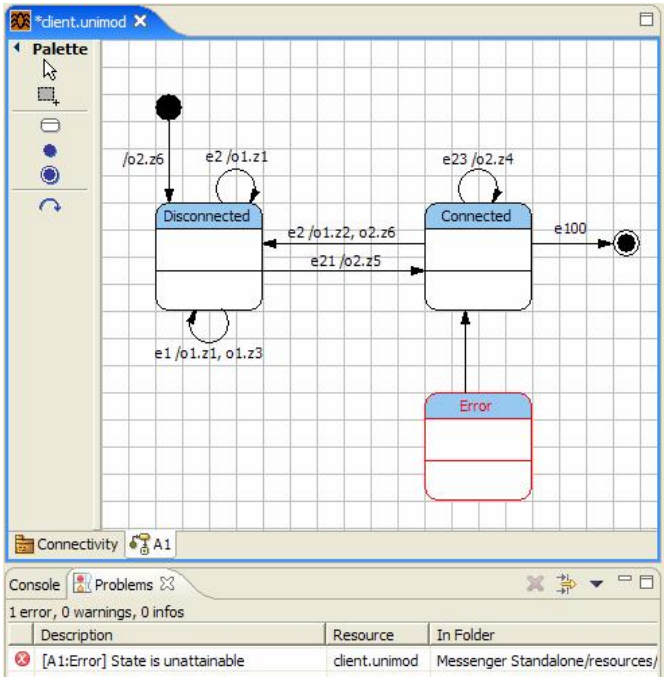

Рис. 10. Недостижимое состояние на графе переходов

Автоматическое завершение ввода и автоматическое исправление ошибок

Традиционно, автоматическим завершением ввода называется подход, благодаря которому среда по заданному началу

лексемы определяет набор допустимых конструкций, префиксом которых данное начало является, и предлагает пользователю выбрать одну из них. Автоматическое исправление ошибок предполагает, что редактор для каждой найденной ошибки указывает пользователю варианты ее исправления.

В случае текстового редактора оба подхода основываются на знании грамматики языка и наборе семантических правил.

В предлагаемом графическом редакторе диаграмм в UML-нотации эти подходы реализованы авторами на базе ограничений, определенных стандартом языка *UML* и описанных выше дополнительных ограничений. Так, для недостижимого состояния, представленного на рис. 10, пользователю будет предложено добавить переход в это состояние из любого достижимого (рис. 11).

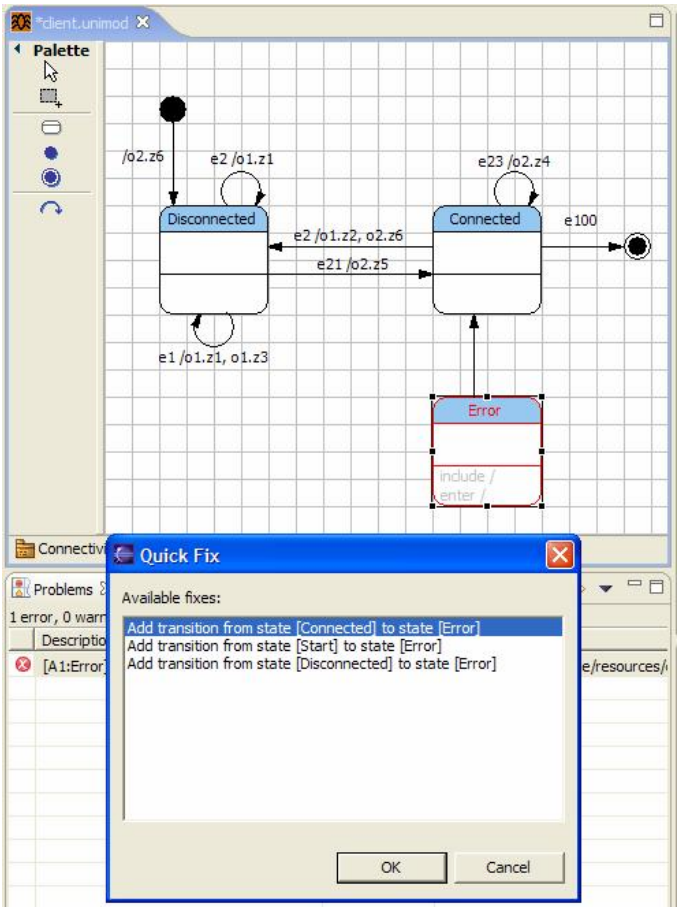

Рис. 11. Предлагаемые варианты исправления ошибки на диаграмме

Форматирование

Форматирование кода облегчает его чтение. Многие текстовые редакторы позволяют автоматически форматировать код.

Аналогом форматирования кода применительно к диаграммам, по мнению авторов, является их укладка (layout). Задача укладки диаграмм является существенно более сложной, чем форматирование кода, так как общепринятые эстетические критерии качества укладки отсутствуют. В проекте ***UniMod*** раскладка диаграмм осуществляется методом отжига [108].

Запуск программы

Для запуска программы, написанной на текстовом языке программирования, ее текст либо компилируется в код, исполняемый операционной системой (*C++*, ***Pascal***) или виртуальной машиной (***Java***, ***C#***), либо непосредственно исполняется интерпретатором (***JavaScript***, ***Basic***).

Подобные решения доступны и для графического языка программирования. Например, для интерпретационного подхода при запуске диаграммы, ее содержимое преобразуется в XML-описание, которое передается интерпретатору. Интерпретатор, в соответствии с операционной семантикой, изложенной выше, «выполняет» XML-описание. Это описание является изоморфным представлением содержимого диаграмм, и поэтому можно говорить о «запуске» диаграмм, как программ.

Отладка

Обычно после локализации ошибки, отладка программ представляет собой трассировку программного кода оператор за оператором с одновременным анализом значений переменных.

Для графической автоматной модели отладка — это трассировка графа переходов, с анализом текущего состояния, событий и значений входных переменных. При необходимости возможна текстовая отладка кода выходных воздействий.

Библиотеки

Большинство существующих языков программирования поддерживают идеологию библиотек и каркасов (frameworks). Библиотека — программный модуль, реализующий функциональность в рамках некоторой предметной области. Каркас — это набор программных базовых сущностей из некоторой предметной области, уточняя и дополняя которые следует строить программу. Для текстовых языков библиотеки и каркасы, как правило, представляют собой скомпилированный код, подключаемый к программе в процессе ее компиляции (статические библиотеки) или во время исполнения (динамические библиотеки).

В рамках автоматного подхода в библиотеки и каркасы следует включать заранее скомпилированные источники событий и объекты управления. Опыт разработки показывает, что приложения, работающие в одной предметной области, различаются не столько выполняемыми атомарными действиями, сколько логикой выполнения этих действий. Этот факт позволяет надеяться, что, создав для некоторой предметной области библиотеку источников событий и объектов управления, при программировании приложений из этой предметной области можно будет полностью отказаться от написания кода на текстовом языке программирования.

5.2.5. Заключение раздела

В настоящем раздела излагается подход к созданию графического языка для автоматно-ориентированного программирования.

Этот подход позволяет:

- сокращать объем кода на текстовом языке программирования;

- отказаться от текстового программирования для некоторой предметной области при наличии библиотеки источников событий и объектов управления для нее;
- строить предложенные в *SWITCH*-технологии схемы связей и графы переходов в *UML*-нотации диаграмм классов и диаграмм состояний соответственно, и включать их в проектную документацию [90];
- формально и наглядно описывать поведение программ и модифицировать их, изменяя в большинстве случаев только графы переходов;
- упростить сопровождение проектов вследствие повышения централизации логики программ.

Исходные тексты, документация и примеры использования программного пакета *UniMod* представлены на сайте http://unimod.sourceforge.net.

5.3. Другие примеры применения методологии

Кроме приведенных выше задач, эта методология была использована также при реализации:

- визуализаторов алгоритмов, используемых при обучении программированию и дискретной математики. Подробно с решением этой задачи можно ознакомиться в статьях, ссылки на которые приведены в разд. 6.5 – публикации с номерами 2 и 5 (http://is.ifmo.ru, раздел «Статьи»), а также в материалах конференции, ссылка на которую приведена в разд. 6.5 – тезисы с номером 3 (http://tm.ifmo.ru). Несколько визуализаторов, разработанных на основе автоматного подхода, а также проектная документация к ним, приведены на сайте http://is.ifmo.ru, раздел «Визуализаторы»;

- алгоритма обхода деревьев. Подробно с решением этой задачи можно ознакомиться в статье, ссылка на которую приведена в разд. 6.5 – публикация с номером 8 (http://is.ifmo.ru, раздел «Статьи»), а также в материалах конференции, ссылка на которую приведена в разд. 6.5 – тезисы с номером 5 (http://tm.ifmo.ru).

5.4. Проекты, разработанные студентами кафедры «Компьютерные технологии» СПбГУ ИТМО в рамках «Движения за открытую проектную документацию»

1. Туккель Н.И., Шалыто А.А. Система управления танком для игры "Robocode". Вариант 1. Объектно-ориентированное программирование с явным выделением состояний.
2. Кузнецов Д.В., Шалыто А.А. Система управления танком для игры "Robocode". Вариант 2.
3. Добрицкий И.О., Куликов А.А., Шалыто А.А. Игра "Lines".
4. Хокканен А.В., Шалыто А.А. Имитатор игрового автомата класса "Однорукий бандит".
5. Марков С.М., Шалыто А.А. Система управления травоядным существом для игры "Terrarium".
6. Пенев В.П., Степаненков В.В., Сучкоусов Е.А., Шалыто А.А. Компьютерная игра "Automatic Bomber".
7. Подтопельный М.А., Чеботарева А.А., Шалыто А.А. Робот, ищущий выход из лабиринта.
8. Лопатухина А.Д. Модель искусственного интеллекта игрового "бота".
9. Южаков Е.М. Построение автономного виртуального робота на основе автоматного подхода (на примере игры «CodeRally», предложенной на Java Challenge туре чем-

пионата мира по программированию по версии ACM 2003 г.).

10. Веденеев В.В., Соловьев П.С. Система управления текстовой игрой "Завалинка".
11. Шалыто А.А., Туккель Н.И., Верба М.Т. Пример программной документации на подсистему управления печатью к статье SWITCH-технология: автоматный подход к созданию программного обеспечения "РЕАКТИВНЫХ" систем (в формате pdf).
12. Туккель Н.И., Шалыто А.А. Система управления дизель-генератором (фрагмент). Программирование с явным выделением состояний.
13. Туккель Н.И., Шалыто А.А., Ваганов С.А. Использование SWITCH-технологии при разработке программ в среде "FLORA/C+" (модель технологического процесса в цехе холодной прокатки).
14. Корниенко А.А., Курочкин Ю.В., Шалыто А.А. Эмуляция пользовательского оконного интерфейса.
15. Степанов О.Г., Шалыто А.А. Система эмуляции поведения "умной" мухи.
16. Агафонов К.А., Порох Д.С., Шалыто А.А. Реализация протокола "SMTP" на основе SWITCH-технологии.
17. Штучкин А.А., Шалыто А.А. Совместное использование теории построения компиляторов и SWITCH-технологии (на примере построения калькулятора).
18. Лазута В.А. Графический пользовательский интерфейс для проекта "Immerssion".
19. Бондаренко К.А., Шалыто А.А. Разработка XML - формата для описания внешнего вида видеопроигрывателя с использованием конечных автоматов.
20. Дистель А.А., Кобак Д.А., Шалыто А.А. Система управления дорожным светофором.
21. Мясников А.И. Кнопочный телефон.

22. Василенко Д.В., Закиров Р.З., Шалыто А.А. Система безопасности банковского комплекса.
23. Заякин Е.А., Шалыто А.А. Метод устранения повторных фрагментов кода при реализации конечных автоматов.
24. Мазин М.М., Парфенов В.Г., Шалыто А.А. Разработка интерактивных приложений Macromedia Flash на базе автоматной технологии.
25. Бабаев А.А., Чижова Г.А., Шалыто А.А. Создание скелетной анимации на основе автоматного программирования.
26. Аничкин И.М. Использование автоматного программирования при построении редактора графа переходов.
27. Первушин Е.В., Шалыто А.А. Моделирование банкомата.
28. Алексеев В.А., Ларионов А.В. Сравнение программ управления кофеваркой "Mark 4 Special Coffee Maker", реализованных на основе нотации Буча и SWITCH-технологии.
29. Кессель С.В. Разработка системы управления кофеваркой на основе автоматного подхода.
30. Пестов А.А., Шалыто А.А. Преобразование недетерминированного конечного автомата в детерминированный.
31. Альшевский Ю.А., Раер М.Г., Шалыто А.А. Механизм обмена сообщениями для параллельно работающих автоматов (на примере системы управления турникетом).
32. Пак С.В., Шалыто А.А. Задача об обедающих философах.
33. Шопырин Д.Г., Шалыто А.А. Объектно-ориентированный подход к автоматному программированию.
34. Гуисов М.И., Шалыто А.А. Задача Д. Майхилла «Синхронизация цепи стрелков». Вариант 1.
35. Гуисов М.И., Кузнецов А.Б., Шалыто А.А. Интеграция механизма обмена сообщениями в Switch-технологию.
36. Гуисов М.И., Кузнецов А.Б., Шалыто А.А. Задача Д. Майхилла «Синхронизация цепи стрелков». Вариант 2.
37. Наумов А.С., Шалыто А.А. Система управления лифтом.

38. Наумов Л.А., Шалыто А.А. Автоматное решении задачи Д. Кнута о лифте.
39. Билык В.С., Писарьков В.С., Шалыто А.А. Автоматная реализация иерархического меню.
40. Астафуров А.А., Шалыто А.А. Разработка и применение паттерна “Automata”.
41. Наумов Л.А. Бакалаврская работа на тему 'Разработка среды и библиотеки CAME&L для решения задач с использованием клеточных автоматов'.
42. Фельдман П.И., Шалыто А.А. Объектно-ориентированная модификация автоматного подхода (на примере системы анимации моделей).
43. Добровольский В.А., Степук А.В. Простой аудиопроигрыватель.
44. Гуров В.С., Шалыто А.А. Построение простого клиент-серверного приложения на основе автоматного подхода (ICQ и автоматы).
45. Лысенко Е.А., Скаков П.С. Транслитерация между различными системами записи японских слов.
46. Коротков М.А., Лукьянова А.П., Шалыто А.А. Система управления взаимодействием скриптов в Web-программировании.
47. Юдаев П.С., Суворов К.Д. Система эмуляции биржи.
48. Князев Е.Е., Сытник С.А., Шалыто А.А. Моделирование процесса управления ядерным реактором.
49. Гаврилов М.И., Медвинский М.Д., Шалыто А.А. Система управления объектами в играх типа «Стратегия».
50. Беляев А.В., Суясов Д.И., Шалыто А.А. Игра «Космонавт».
51. Дмитриченко М.И., Шалыто А.А. Имитатор микроволновой печи.

52. Карпец А.А. Разработка утилиты «Kmake» для управления компиляцией и сборкой проектов на основе автоматного подхода.
53. Ковалев А.С., Лукьянова А.П., Шалыто А.А. Новый метод вычисления булевых формул.
54. Дронь В.О., Плодовитова А.А. Система управления моделью фотоаппарата.
55. Сапунков В.С., Шалыто А.А. Система управления игрой «Змейка».

Некоторые из приведенных выше проектов, помимо сайта http://is.ifmo.ru, опубликованы также в электронном журнале «Мир ПК – Диск»:

- проекты 3 и 6 на диске 2003 № 8;
- проекты 24 и 38 на диске 2004 № 2;
- проект 15 на диске 2004 № 3;
- проект 19 на диске 2004 № 4;
- проект 17 на диске 2004 № 6;
- проект 25 на диске 2004 № 10.

По некоторым из этих работ были опубликованы статьи:

- реализация интерактивных сценариев образовательной анимации на языке «ActionScript» пакета «Macromedia Flash» (проект 24). Подробно с решением этой задачи можно ознакомиться в статье, ссылка на которую приведена в разд. 6.5 – публикация с номером 1 (http://is.ifmo.ru, раздел «Статьи»);
- реализация системы управления лифтом, описанной в книге Д. Кнута «Искусство программирования» (проект 37). Подробно с решением этой задачи можно ознакомиться в статье, ссылка на которую приведена в разд. 6.5 – публикация с номером 4 (http://is.ifmo.ru, раздел «Статьи»);

Глава 6. Публикации по результатам этапов

По результатам выполненных этапов опубликованы следующие работы.

6.1. Разработка основных положений технологии создания программного обеспечения систем логического управления (этап 1)

6.1.1. Статьи

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управление. СПб.: Наука. 1998, 628 с.
2. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управление // Промышленные АСУ и контроллеры. 1999. № 9, с. 33-37.
3. Шалыто А.А. Реализация алгоритмов логического управления программами на языке функциональных блоков // Промышленные АСУ и контроллеры. 2000. № 4, с. 45-50.
4. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука. 2000, 780 с.
5. Шалыто А.А. Автоматное проектирование программ. Алгоритмизация программирования задач логического управления // Известия РАН. Теория и системы управления. 2000. № 6, с. 63-81.

6.1.2. Тезисы

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управление // Тезисы докладов Всероссийской научно-методической конференции "Телематика-99". СПб.: СПбГИТМО (ТУ), 1999. С. 62-63.

2. Туккель Н.И., Шалыто А.А. Сравнение событийного и автоматного подходов к программированию задач логического управления // Тезисы докладов Всероссийской научно-методической конференции "Телематика-99". СПб.: СПбГИТМО (ТУ), 1999, с. 63-64.
3. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управление // Тезисы докладов международной конференции по проблемам управления. М.: Институт проблем управления. Том 3, 1999, с. 337-339.
4. Туккель Н.И., Шалыто А.А. Применение SWITCH-технологии для программирования в событийных системах // Труды международной научно-технической конференции "Пятьдесят лет развития кибернетики". СПб.: СПбГТУ, 1999, с. 124-125.

6.2. Разработка основных положений создания программного обеспечения "реактивных" систем (этап 2)

6.2.1. Статьи

1. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем //Промышленные АСУ и контроллеры. 2000. № 10, с. 44-48.
2. Шалыто А.А., Туккель Н.И. Автоматный подход к созданию программного обеспечения для систем логического управления и "реактивных" систем //Системы управления и обработки. ФГУП "НПО "Аврора"". 2000. Вып. 2, с. 165-173.
3. Шалыто А.А. Алгоритмизация и программирование для систем логического управления и "реактивных" систем (обзор) //Автоматика и телемеханика. 2001. № 1, с. 3-39.

4. Шалыто А.А., Туккель Н.И. Программирование с явным выделением состояний //Мир ПК. 2001. № 8, с. 116-121, № 9, с. 132-138.
5. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем //Известия ВУЗов. Приборостроение. 2001. №9, с. 28-35.
6. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. № 5, с. 45-62.

6.2.2. Тезисы

1. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем /Тезисы докладов Международной научно-методической конференции "Телематика-2000". СПб.: СПбГИТМО (ТУ), 2000, с. 88-91.
2. Шалыто А.А., Туккель Н.И. SWITCH-технология - автоматный подход к созданию программного обеспечения "реактивных" систем /Материалы международной научно-технической конференции "Кибернетика и технологии XXI века". Воронеж: ВГТУ, 2000, с. 308-316.
3. Шалыто А.А., Туккель Н.И., Ваганов С.А. Повышение централизации управления при программировании "реактивных" систем /Труды международной научно-методической конференции "Телематика-2001". СПб.: СПбГИТО (ТУ), 2001, с. 174-176.

4. Туккель Н.И., Шалыто А.А. Применение автоматного программирования для создания "реактивных" систем управления /Материалы международной научно-практической конференции "Математическое моделирование в образовании, науке и производстве". Тирасполь: Приднепровский гос. университет им. Т.Г. Шевченко, 2001, с. 469-471.

6.3. Применение автоматного подхода для программной реализации вычислительных алгоритмов (этап 3)

6.3.1. Статьи

1. Шалыто А.А., Туккель Н.И. Реализация вычислительных алгоритмов на основе автоматного подхода // Телекоммуникации и информатизация образования. 2001. № 6, с.35–53.
2. Шалыто А.А., Туккель Н.И. От тьюрингова программирования к автоматному // Мир ПК. 2002. № 2, с.144–149.
3. Шалыто А.А., Туккель Н.И., Шамгунов Н.Н. Ханойские башни и автоматы // Программист. 2002. № 8, с. 82–90.
4. Шалыто А.А., Туккель Н.И. Преобразование итеративных алгоритмов в автоматные // Программирование. 2002. № 5, с. 12–26.
5. Шалыто А.А., Туккель Н.И., Шамгунов Н.Н. Реализация рекурсивных алгоритмов на основе автоматного подхода // Телекоммуникации и информатизация образования. 2002. № 5, с. 72-99.
6. Шалыто А.А., Туккель Н.И., Шамгунов Н.Н. Задача о ходе коня // Мир ПК. 2003, № 1, с. 152-155.

6.3.2. Тезисы

1. Казаков М.А., Шалыто А.А., Туккель Н.И. Использование автоматного подхода для реализации вычислительных алгоритмов /Труды международной научно-методической конференции "Телематика'2001". СПб.: СПбГИТМО (ТУ), 2001, с. 174–176.
2. Шалыто А.А., Туккель Н.И. Применение SWITCH-технологии для решения классических задач распознавания цепочек символов /Труды международной научно-методической конференции "Телематика'2001". СПб.: СПбГИТМО (ТУ), 2001, с. 177–179.
3. Шалыто А.А., Туккель Н.И. Автоматное программирование как практическое развитие тьюрингова программирования /Тезисы докладов международной научной конференции "Интеллектуальные и многопроцессорные системы - 2001". Таганрог: ТРТУ, 2001, с. 123–126.
4. Шалыто А.А., Туккель Н.И. Реализация рекурсивных алгоритмов автоматными программами /Труды международной научно-методической конференции "Телематика'2002". СПб.: СПбГИТМО (ТУ), 2002, с.181–182.

6.4. Применение автоматного подхода в объектно-ориентированном программировании (этап 4)

6.4.1. Статьи

1. Туккель Н.И., Шалыто А.А. Танки и автоматы // BYTE/Россия. 2003. №2, с. 69-73.
2. Шалыто А.А. Новая инициатива в программировании. Движение за открытую проектную документацию // Мир ПК-ДИСК. 2003. № 8; Мир ПК. 2003. № 9, с. 52-56; PC Week. 2003. № 40, с. 38-39, 42; Информатика. 2003. № 44, с. 22-24, 31.

3. Шалыто А.А. Технология автоматного программирования // Мир ПК. 2003. № 10, с. 74-78.
4. Туккель Н.И., Шалыто А.А. Автоматное и синхронное программирование // Искусственный интеллект. 2003. № 4, с. 82-88.
5. Шалыто А.А. Технология автоматного программирования // Современные технологии. СПбГУИТМО, с. 18-26.

6.4.2. Тезисы

1. Гуров В.С., Нарвский А.С., Шалыто А.А. Автоматизация проектирования событийных объектно-ориентированных программ с явным выделением состояний // Труды X Всероссийской научно-методической конференции "Телематика-2003". 2003. Т.1, с. 282-283.
2. Шопырин Д.Г., Шалыто А.А. Применение класса "STATE" в объектно-ориентированном программировании с явным выделением состояний // Труды X Всероссийской научно-методической конференции "Телематика-2003". СПб.: СПбГИТМО (ТУ). 2003. Т.1, с.284-285.
3. Корнеев Г.А., Шалыто А.А. Реализация конечных автоматов с использованием объектно-ориентированного программирования // Труды X Всероссийской научно-методической конференции "Телематика-2003". СПб.: СПбГИТМО (ТУ). 2003. Т.2, с.377-378.
4. Корнеев Г.А., Казаков М.А., Шалыто А.А. Построение логики работы визуализаторов алгоритмов на основе автоматного подхода // Труды X Всероссийской научно-методической конференции "Телематика-2003". СПб.: СПбГИТМО (ТУ). 2003. Т.2, с.378-379.

5. Мазин М.А., Парфёнов В.Г., Шалыто А.А. Автоматная реализация интерактивных сценариев образовательной анимации // Труды X Всероссийской научно-методической конференции "Телематика-2003". СПб.: СПбГИТМО (ТУ). 2003. Т.2, с.379-380.
6. Шалыто А.А. Технология автоматного программирования // Труды Первой Всероссийской научной конференции "Методы и средства обработки информации". М.: МГУ. 2003, с.528-535.
7. Туккель Н.И., Шалыто А.А. Автоматное и синхронное программирование // Материалы международной научно-технической конференции "ИМС-2003" "Интеллектуальные и многопроцессорные системы - 2003". Таганрог-Донецк, ТРТУ. 2003.т.2, с. 15-17.
8. Штучкин А.А., Шалыто А.А. Совместное использование теории построения компиляторов и Switch-технологии // Труды X Всероссийской научно-методической конференции "Телематика-2003". СПб.: СПбГИТМО (ТУ). 2003. Т.1, с.286-287.
9. Бабаев А.А., Чижова Г.А., Шалыто А.А. Метод создания скелетной анимации на основе автоматного программирования // Труды X Всероссийской научно-методической конференции "Телематика-2003". СПб.: СПбГИТМО (ТУ). 2003. Т.2, с.375-376.
10. Shalyto A.A., Naumov L.A. Automata Programming as a Sort of Synchronous Programming // Proceedings of "East-West Design & Test Conference. (EWDTC-03). Yalta: Kharkov National University of Radio-electronics, 2003, p.140-143.

11. Shalyto A.A., Naumov L.A. Automata Theory for Multi-Agent Systems Implementation // Proceedings of International Conference Integration of Knowledge Intensive Multi-Systems: Modeling, Exploration and Engineering. KIMAS-03. IEEE Boston. 2003, p.65-70.

6.5. Объектно-ориентированное программирование с явным выделением состояний (этап 5)

6.5.1. Статьи

1. Мазин М.А., Парфенов В.Г., Шалыто А.А. Анимация. Flash-технология. Автоматы // Компьютерные инструменты в образования. 2003. № 4, с. 39-47.
2. Казаков М.А., Корнеев Г.А., Шалыто А.А. Метод построения логики работы визуализаторов алгоритмов на основе конечных автоматов // Телекоммуникации и информатизация образования. 2003. № 6, с. 27-58.
3. Шопырин Д.Г., Шалыто А.А. Объектно-ориентированный подход к автоматному программированию // Информационно-управляющие системы. 2003. № 5, с. 29-39.
4. Наумов Л.А., Шалыто А.А. Искусство программирование лифта. Объектно-ориентированное программирование с явным выделением состояний // Информационно-управляющие системы. 2003. № 6, с. 38-49
5. Казаков М.А., Шалыто А.А. Использование автоматного программирования для реализации визуализаторов // Компьютерные инструменты в образовании. 2004. № 2. с. 19-33.
6. Шопырин Д.Г. Шалыто А.А. Синхронное программирование // Информационно-управляющие системы. 2004. № 3, с. 35-42.
7. Мазин М.А., Шалыто А.А. Преступники и автоматы // Мир ПК. 2004. №9. с. 82-84.

8. Корнеев Г.А., Шамгунов Н.Н., Шалыто А.А. Обход деревьев на основе автоматного подхода. Компьютерные инструменты в образовании. 2004. №3. с. 32-37.
9. Шалыто А.А., Наумов Л.А. Методы объектно-ориентированной реализации реактивных агентов на основе конечных автоматов // Искусственный интеллект. 2004. №4. с. 756-762.
10. Наумов Л.А., Шалыто А.А. Искусство программирование лифта. Объектно-ориентированное программирование с явным выделением состояний // Мир ПК – ДИСК. 2004. №2. С. 18
11. Мазин М.А., Шалыто А.А. Macromedia Flash и автоматы // Мир ПК – ДИСК. 2004. №2. С. 9.
12. Степанов О.Г., Шалыто А.А. Система эмуляции поведения "умной" мухи // Мир ПК – ДИСК. 2004. №3. С. 26.
13. Бондаренко К.А., Шалыто А.А. Разработка XML - формата для описания внешнего вида видеопроигрывателя с использованием конечных автоматов// Мир ПК – ДИСК. 2004. №4. С. 17.
14. Шалыто А.А., Наумов Л.А. Реализация автоматов в объектно-ориентированных программах // Мир ПК – ДИСК. 2004. №4. С. 3.
15. Коротков М.А. Доказательства в математике и проектная документация // Мир ПК – ДИСК. 2004. №4. С. 3.
16. Штучкин А.А., Шалыто А.А. Совместное использование теории построения компиляторов и SWITCH-технологии (на примере построения калькулятора) // Мир ПК – ДИСК. 2004. №6. С. 29
17. Бабаев А.А., Чижова Г.А., Шалыто А.А. Создание скелетной анимации на основе автоматного программирования // Мир ПК – ДИСК. 2004. №10. С. 11.

18. Беляев А.В., Суясов Д.И., Шалыто А.А. Компьютерная игра “Космонавт”. Проектирование и реализация //Компьютерные инструменты в образовании. 2004. № 4, с.75 – 84.

6.5.2. Тезисы

1. Shalyto A.A., Naumov L.A. Automata Theory for Applied Programming // Вторая международная конференция по проблемам управления. Избранные труды в двух томах М.: Институт проблем управления. 2003. Т. 2, с. 86-92.
2. Гуров В.С., Мазин М.А., Шалыто А.А. UniMod – программный пакет для разработки объектно-ориентированных приложений на основе автоматного подхода // Труды XI Всероссийской научно-методической конференции "Телематика-2004". СПб.: СПбГУ ИТМО. 2004. Т. 1, с. 189-191.
3. Казаков М.А., Шалыто А.А. Реализация визуализаторов на основе автоматного подхода // Труды XI Всероссийской научно-методической конференции "Телематика-2004". СПб.: СПбГУ ИТМО. 2004. Т. 1, с. 191-193.
4. Шалыто А.А., Наумов Л.А. Реализация автоматов в объектно-ориентированных программах // Труды XI Всероссийской научно-методической конференции "Телематика-2004". СПб.: СПбГУ ИТМО. 2004. Т. 1, с. 332-334.
5. Корнеев Г.А., Шамгунов Н.Н., Шалыто А.А. Обход деревьев на основе автоматного подхода // Труды XI Всероссийской научно-методической конференции "Телематика-2004". СПб.: СПбГУ ИТМО. 2004. Т. 1, с. 182-183.
6. Шалыто А.А., Наумов Л.А. Методы объектно-ориентированной реализации реактивных агентов на основе конечных автоматов // Искусственный интеллект. Интеллектуальные и многопроцессорные системы. Материалы Международной научно-технической конференции. Т. 1. Таганрог-Донецк. 2004. с. 279-284.

7. Shalyto A.A., Naumov L.A. New Initiative in Programming. Foundation for Open Project Documentation // Proceedings of East-West Design & Test Workshop (EWDTW’2004). Ukraine. Yalta. 2004. pp. 64-69.

Заключение

В результате выполнения настоящей работы были разработаны основные положения создания программного обеспечения систем управления на основе автоматного похода.

При этом в ходе выполнения первого этапа были созданы основы технологии автоматного программирования для систем логического управления.

В результате выполнения второго этапа был предложен подход к использованию автоматов в рамках процедурного (императивного) программирования, названный «программирование с явным выделением состояний»

При выполнении третьего этапа были намечены пути объединения объектной и автоматной парадигм программирования. Подход, основанный на их совместном использовании, назван «объектно-ориентированное программирование с явным выделением состояний».

В ходе выполнения четвертого этапа работы было показано, что применение автоматного подхода бывает весьма целесообразным при реализации классических вычислительных алгоритмов (алгоритмов, применяемых в дискретной математике). При этом в частности, были предложены методы преобразования итеративных и рекурсивных программ в автоматные.

На пятом этапе работы указанные выше подходы были использованы при решении ряда задач.

Направления дальнейших исследований.

На основе выполненных работ могут быть сформулированы направления дальнейших исследований. Так, в частности, при

выполнении проектов, список которых приведен в разд. 5.4, был предложен ряд методов совместного использования объектной и автоматной парадигм программирования, что требует их осмысления и классификации.

Кроме того, недостатки широко известного в объектно-ориентированном проектировании паттерна «State», предназначенного для реализации объектов, поведение которых варьируется в зависимости от их состояния, делают целесообразным разработку для этих целей нового паттерна, лишенного указанных недостатков.

Еще одно направление дальнейших исследований связано с тем, что в последнее время все шире применяются языки программирования, ориентированные на предметную область. Поэтому естественно разработать язык, поддерживающий автоматное программирование. При этом первый шаг в разработке такого языка состоит в расширении какого-либо известного языка программирования высокого уровня дополнительными конструкциями, характерными для автоматного программирования.

Список литературы

1. Шалыто А.А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. Таль А. А., Айзерман М. А., Розоноэр Л. И. и др. Логика. Автоматы. Алгоритмы. М.: Физматгиз, 1963.
3. Байцер Б. Архитектура вычислительных комплексов. Т.1. М.: Мир, 1974.
4. TSX T607. Programming Terminal. User's Manual. Book 3. Graphset language. Telemecanique. 1987.
5. Дейл Н., Уимз Ч., Хедингтон М. Программирование на C++. М.: ДМК, 2000.
6. Иодан Э. Структурное проектирование и конструирование программ. М.: Мир, 1979.
7. Лингер Р., Миллс Х., Уитт С. Теория и практика структурного программирования. М.: Мир, 1982.
8. Руднев В. В. Конечный автомат как объект управления // Автоматика и телемеханика. 1978. № 9.
9. Руднев В. В. Простые иерархические системы взаимосвязанных графов // Автоматика и телемеханика. 1979. № 1.
10. Черняк Л. Создание программ как инженерная дисциплина //ComputerWorld Россия. 2000. № 37.
11. Зайцев С.С. Описание и реализация протоколов сетей ЭВМ. М.: Наука, 1989.
12. Герр Р. Новый поворот // PC Magazine / Russian Edition. 1998. №10.
13. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. СПб.: Символ, 2000.
14. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК, 2000.

15. Шалыто А.А. Алгоритмизация и программирование для систем логического управления и "реактивных" систем // Автоматика и телемеханика. 2001. № 1.

16. Кук Д., Урбан Д., Хамилтон С. Unix и не только. Интервью с Кеном Томпсоном //Открытые системы. 1999. №4.

17. Герр Р. Отладка человечества // PC Magazine / Russian Edition. 2000. № 5.

18. Бенькович Е.С., Колесов Ю.Б., Сениченков Ю.Б. Практическое моделирование динамических систем. СПб.: БХВ-Петербург, 2002.

19. Богатырев Р. Об асинхронном и автоматном программировании // Открытые системы. 2001. № 3.

20. Любченко В.С. Мы выбираем, нас выбирают... (к проблеме выбора алгоритмической модели) // Мир ПК. 1999. №3.

21. Кузнецов Б.П. Психология автоматного программирования // BYTE/Россия. 2000. № 11.

22. Непейвода Н.Н., Скопин И.Н. Основания программирования. Москва – Ижевск: Институт компьютерных исследований, 2003.

23. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. М.: МЦНМО, 1999.

24. Aschcroft E., Manna Z. The translation of "goto" programm into "while" programm // Proceeding of 1971 IFIP Congress.

25. Йодан Э. Структурное проектирование и конструирование программ. М.: Мир, 1979.

26. Баранов С.И. Синтез микропрограммных автоматов (граф-схемы и автоматы). Л.: Энергия, 1979.

27. Лингер Р., Миллс Х., Уитт Б. Теория и практика структурного программирования. М.: Мир, 1982.

28. Казаков М.А., Шалыто А.А., Туккель Н.И. Использование автоматного подхода для реализации вычислительных алгоритмов //Труды международной научно-методической конференции "Телематика'2001". СПб.: СПбГИТМО, 2001.

29. Брукшир Дж. Введение в компьютерные науки. М.: Вильямс, 2001.

30. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Том 1. Синтаксический анализ. М.: Мир, 1978.

31. Стивенс Р. Delphi. Готовые алгоритмы. М.: ДМК, 2001.

32. Станкевич Л.А. Интеллектуальные технологии представления знаний. Интеллектуальные системы. СПб.: СПбГТУ, 2000.

33. Шалыто А.А., Туккель Н.И. SWITCH-технология — автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. № 5.

34. Наумов А.С., Шалыто А.А. Система управления лифтом. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru, раздел «Проекты».

35. Наумов Л.А., Шалыто А.А. Искусство программирования лифта. Объектно-ориентированное программирование с явным выделением состояний // Информационно-управляющие системы. 2003. № 6.

36. Шопырин Д.Г., Шалыто А.А. Объектно-ориентированный подход к автоматному программированию. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru, раздел «Проекты».

37. Фельдман П.И., Шалыто А.А. Совместное использование объектного и автоматного подходов в программировании. СПб.: СПбГУ ИТМО, 2004. http://is.ifmo.ru, раздел «Проекты».

38. 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».

39. Гамма Э., Хелм Р., Джонсон Р. и др. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001.

40. Астафуров А.А., Шалыто А.А. Разработка и применение паттерна «Automata». СПб.: СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

41. Кузнецов Д.В., Шалыто А.А. Система управления танком для игры «Robocode». Вариант 2. СПб.: СПбГУ ИТМО, 2003. http://is.ifmo.ru, раздел «Проекты».

42. Гуров В.С., Нарвский А.С., Шалыто А.А. Автоматизация проектирования событийных объектно-ориентированных программ с явным выделением состояний // Труды X Всероссийской научно-методической конференции "Телематика-2003". 2003. Т.1.

43. Гуров В.С., Шалыто А.А. XML и автоматы. СПб.: СПбГУ ИТМО. 2004. http://is.ifmo.ru, раздел «Проекты».

44. Бондаренко К.А., Шалыто А.А. Разработка XML - формата для описания внешнего вида видеопроигрывателя с использованием конечных автоматов. СПб.: СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

45. Гуисов М.И., Кузнецов А.Б., Шалыто А.А. Интеграция механизма обмена сообщениями в Switch-технологию. СПб.: СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

46. Гуисов М.И., Шалыто А.А. Задача Д. Майхилла «Синхронизация цепи стрелков». Вариант 1. СПб.: СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

47. 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».

48. Альшевский Ю.А., Раер М.Г., Шалыто А.А. Система управления турникетом. СПб.: СПбГУ ИТМО. 2003. http://is.ifmo.ru, раздел «Проекты».

49. Automata Studies //Ed. Shannon C.E., McCarthy J. Princeton Univ. Press, 1956. (Автоматы //Ред. Шеннона К.Э., МакКарти Дж. М.: Изд-во иностр. лит., 1956).

50. Rubin M., Scott D. Finite automata and their decision problem //IBM J. Research and Development. 1959. V.3. № 2. (Кибернетический сборник. Вып.4. М.: Изд-во иностр. лит., 1962).

51. Глушков В. М. Синтез цифровых автоматов. М.: Изд-во физ.-мат. лит., 1962.с.

52. Kleene S. C. Representation of Events in Nerve Nets and Finite Automata. 1956.

53. Steling S., Maassen O. Applied Java Patterns. Pearson Higher Education. 2001. (Стелтинг С., Массен О. Применение шаблонов Java. Библиотека профессионала. М.: Вильямс, 2002).

54. Grand M. Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML. Wiley, 2002. (Гранд М. Шаблоны проектирования в Java. М.: Новое знание, 2004).

55. Java Data Objects (JDO). http://java.sun.com/products/jdo/index.jsp.

56. Eliens A. Principles of Object-Oriented Software Development. MA.: Addison-Wesley, 2000. (Элиенс А. Прин-

ципы объектно-ориентированной разработки программ. М.: Вильямс, 2002).

57. Sandén B. The state-machine pattern // Proceedings of the conference on TRI-Ada '96.

58. Adamczyk P. The Anthology of the Finite State Machine Design Patterns. http://jerry.cs.uiuc.edu/~plop/plop2003/Papers/Adamczyk-State-Machine.pdf

59. Odrowski J., Sogaard P. Pattern Integration – Variations of State // Proceedings of PLoP96. http://www.cs.wustl.edu/~schmidt/PLoP-96/odrowski.ps.gz.

60. Sane A., Campbell R.. Object-Oriented State Machines: Subclassing, Composition, Delegation, and Genericity // OOPSLA ’95. http://choices.cs.uiuc.edu/sane/home.html.

61. Шалыто А. А., Туккель Н. И. От тьюрингова программирования к автоматному. // Мир ПК. 2002. № 2. http://is.ifmo.ru, раздел «Статьи».

62. Harel D. Statecharts: A visual formalism for complex systems //Sci. Comput. Program. 1987. Vol.8.

63. Шамгунов Н. Н., Корнеев Г А. Шалыто А. А. Паттерн State Machine для объектно-ориентированного проектирования автоматов // Информационно-управляющие системы. 2004. № 5.

64. Шамгунов Н. Н., Шалыто А. А. Язык автоматного программирования с компиляцией в Microsoft CLR. // Microsoft Research Academic Days in St. Petersburg, 2004.

65. Gosling J., Joy B., Steele G., Bracha G. The Java Language Specification. http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html.

66. Appel A. W. Modern Compiler Implementation in Java. NY, 1998.

67. Green A. Trail: The Reflection API. http://java.sun.com/d1ocs/books/tutorial/reflect/

68. Naur P. et al. Revised Report on the Algorithmic Language ALGOL 60 //Communications of the ACM. 1960. Vol.3. № 5.

69. Aho A., Sethi R., Ullman J. Compilers: Principles, Techniques and Tools. MA: Addison-Wesley, 1985. (Axo A., Сети Р., Ульман Дж. Компиляторы: принципы, технологии и инструменты. М.: Вильямс, 2001).

70. Object-Oriented Programming Concepts // http://java.sun.com/docs/books/tutorial/java/concepts/

71. Appel A. W. Modern Compiler Implementation in Java. NY, 1998.

72. http://www.cs.princeton.edu/~appel/modern/java/ Modern Compiler Implementation in Java.

73. Luger G. Artifical Intelligence. Structures and Strategies for Complex Problem Solving. MA: Addison Wesley, 2002.

74. Shoham Y. Agent Oriented Programming // Journal of Artifical Intelligence. 1993. Vol.60. № 1.

75. Genesereth M.R., Ketchpel S.P. Software agents // Communications of the ACM. 1994. Vol.37. №7.

76. Bradshaw J. An Introduction to Software Agents // Software Agents. MA: AAAI Press, 1997. http://agents.umbc.edu/introduction/01-Bradshaw.pdf

77. Тарасов В.Б. От многоагентных систем к интеллектуальным организациям: философия, психология, информатика. М.: Эдиториал УРСС, 2002.

78. Brooks R. Intelligence Without Representation // Artifical Intelligence. 1991. № 47.

79. Seel N. Intentional Description of Reactive Systems // Decentralized Artifical Intelligence II. Amsterdam: Elsevier North-Holland, 1991.

80. Ferber J. Les systems multi-agents vers une intelligence collective. Paris: InterEditions. 1995.

81. Shalyto A. Cognitive Properties of Hierarchical Representations of Complex Logical Structure //10th IEEE International Symposium on Intelligent Control 1995. Monterey, California.

82. Harel D., Politi M. Modeling reactive systems with statecharts. NY: McGraw-Hill, 1998.

83. Шалыто А.А. Алгоритмизация и программирование для систем логического управления и реактивных систем //Автоматика и телемеханика. 2001. № 1. http://is.ifmo.ru/works/app-aplu/1/

84. Piaget J. Structuralism. NY: Basic Books. 1970.

85. Miller G., Gallanter E., Pribram K. Plans and the structure of behaviour. NY: Holt, 1960.

86. Naumov L., Shalyto A. Automata Theory for Multi-Agent Systems Implementation //Proceedings of International Conference "Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering". KIMAS-03. Boston: IEEE Boston Section. 2003.

87. Шалыто А, Туккель Н. Танки и автоматы //Byte/Россия. 2003. № 2. http://is.ifmo.ru/download/tanks_new.pdf

88. Туккель Н., Шалыто А. Система управления танком для игры “Robocode”. Версия 1. Проектная документация. http://is.ifmo.ru/projects/tanks/

89. Кузнецов. Д., Шалыто А. Система управления танком для игры “Robocode”. Версия 2. http://is.ifmo.ru/projects/robocode2/

90. Шалыто А. Новая инициатива в программировании. Движение за открытую проектную документацию //Мир ПК. 2003. № 9. http://is.ifmo.ru/works/open_doc/

91. Шалыто А., Туккель Н. От тьюрингова программирования к автоматному //Мир ПК. 2002. № 2. http://is.ifmo.ru/works/turing/

92. The Hybrid Systems Project. http://control.ee.ethz.ch/~hybrid/

93. Беляев А.В., Суясов Д.И., Шалыто А.А. Игра “Космонавт”. Проектная документация. http://is.ifmo.ru/projects/cosmo

94. Котов В.Н., Шалыто А.А., Ярцев Б.М. Разработка программного обеспечения для роботов Lego Mindstorms на основе автоматного подхода (Проект “Изенгард”) Проектная документация. http://is.ifmo.ru/projects/lego

95. Mellor S. et al. Executable UML: A Foundation for Model Driven Architecture. MA: Addison-Wesley, 2002.

96. Буч Г., Рамбо Г., Якобсон И. UML. Руководство пользователя. М.: ДМК, 2000.

97. Jacobson I. Four Macro Trends in Software Development Y2004. http://www.ivarjacobson.com/postnuke/html/modules.php?op=modload&name=UpDownload&file=index&req=getit&lid=9

98. Якобсон И., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002.

99. Kiczales G., Lamping J., Mendhekar A. et al. Aspect-oriented programming // In ECOOP'97 – Object-Oriented Programming. 11th European Conference. 1997. LNCS 1241. http://citeseer.ist.psu.edu/kiczales97aspectoriented.html (русский перевод – http://www.javable.com/columns/aop/workshop/02/)

100. First European Conference on Model-Driven Software Engineering. Germany. 2003. http://www.agedis.de/conference/

101. International Workshop «e-Business and Model Based in System Design». IBM EE/A. SPb.: SPb ETU, 2004.

102. OMG Model Driven Architecture. http://www.omg.org/mda/

103. Harel D., Gery E. Executable Object Modeling with Statecharts. // In Proceedings of the 18th International Conference on Software Engineering, IEEE Computer Society Press, January 1996, pp. 246-257. http://citeseer.ist.psu.edu/harel97executable.html

104. XJTek AnyState. http://www.xjtek.com/anystates/

105. Шалыто А.А., Туккель Н.И. Танки и автоматы // BYTE/Россия. 2003. №2, с. 69-73. http://is.ifmo.ru/ (раздел «Статьи»).

106. Грэхем И. Объектно-ориентированные методы. Принципы и практика. М.: Вильямс, 2004.

107. Фаулер М. Рефакторинг. Улучшение существующего кода. М.: Символ-Плюс, 2003.

108. Fruchterman T. M. J., Reingold E. M. Graph Drawing by Force Directed Placement. // Software - Practice and Experience. 1991, № 21(11).