На правах рукописи

Шопырин Данил Геннадьевич

Методы объектно-ориентированного проектирования и реализации программного обеспечения реактивных систем

Специальность 05.13.13 – Телекоммуникационные системы
и компьютерные сети

АВТОРЕФЕРАТ

диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург
2005

Работа выполнена в Санкт-Петербургском государственном университете информационных технологий, механики и оптики

Научный руководитель: доктор технических наук, профессор
Шалыто Анатолий Абрамович

Официальные оппоненты: доктор технических наук
Марлей Владимир Евгеньевич

кандидат физ.-мат. наук
Новиков Федор Александрович

Ведущая организация: Ленинградский отраслевой научно-исследовательский институт связи (Санкт Петербург)

Защита диссертации состоится «___» октября 2005 года в ___ часов на заседании диссертационного совета Д.212.227.05 в Санкт-Петербургском государственном университете информационных технологий, механики и оптики, 197101, Санкт-Петербург, ул. Саблинская 14, СПбГУ ИТМО.

С диссертацией можно ознакомиться в библиотеке СПбГУ ИТМО.

Автореферат разослан «___» сентября 2005 г.

Ученый секретарь
совета Д.212.227.05,
кандидат технических наук,
доцент Поляков Владимир Иванович

Общая характеристика работы

Актуальность проблемы. При разработке объектно-ориентированного программного обеспечения телекоммуникационных систем актуальна задача проектирования, описания и реализации их поведения. Программные системы можно разделить на три класса: *преобразующие*, *интерактивные* и *реактивные* системы. Реактивные системы взаимодействуют с окружающей средой посредством обмена сообщениями в темпе, задаваемом средой.

Время отклика *реактивной* системы задается ее окружением. Для *реактивных* систем характерны детерминированность и параллелизм. Сбои в работе многих *реактивных* систем крайне нежелательны.

Все вышеуказанные свойства характерны и для многих телекоммуникационных систем, компоненты которых взаимодействуют между собой посредством обмена сообщениями. В качестве каналов связи используются радиосвязь, спутниковая связь, ***Internet***-соединения и т.д. Скорость обработки сообщений обычно строго регламентируется используемыми протоколами и/или конфигурацией системы. Телекоммуникационным системам присущи детерминированное поведение и параллелизм. Сбои в работе *ответственных* телекоммуникационных систем могут привести к катастрофическим последствиям.

Наиболее часто для решения задач проектирования, описания и реализации телекоммуникационных систем используются такие средства как язык описания и спецификации ***SDL*** (Specification and Description Language), универсальный язык моделирования ***UML*** (Unified Modeling Language), а также язык синхронного программирования ***SyncCharts***.

Во всех вышеперечисленных языках в качестве средства описания поведения объектов и процессов используются графические языки, основанные на теории автоматов. Однако диаграммы описания поведения объектов в этих языках обладают рядом недостатков:

- громоздкость;
- отсутствие удобных и непротиворечивых способов перехода от спецификации к реализации.

Для устранения этих недостатков с 1991 года в России разрабатывается ***SWITCH***-технология, также известная как «автоматное программирование», предназначенная для спецификации и реализации систем со сложным поведением. Графы переходов, используемые в *автоматном программировании*, весьма компактны, так как используются совместно со схемами связи автоматов, подробно описывающими их интерфейс. Однако в автоматном программировании остаются недостаточно проработанными следующие вопросы:

- совместное использование объектно-ориентированного и автоматного программирования;
- объектно-ориентированное проектирование программного обеспечения *реактивных* систем;

- объектно-ориентированная реализация программного обеспечения *реактивных* систем.

Поэтому исследования, направленные на решение проблем проектирования и реализации реактивных систем и совместного использования объектно-ориентированного и автоматного программирования, весьма актуальны.

Целью диссертационной работы является разработка методов объектно-ориентированного проектирования и реализации программного обеспечения *реактивных* систем, в основном телекоммуникационных, на основе конечных автоматов.

Основные задачи исследования состоят в следующем:

1. Совершенствование методов реализации автоматных систем на основе ***SWITCH***-технологии.
2. Разработка графической нотации для проектирования автоматных объектов, которая позволяет описывать декомпозицию и структурирование их логики с помощью наследования.
3. Разработка методов реализации автоматных объектов на универсальных языках программирования, обеспечивающих декомпозицию и структурирование логики с помощью наследования.

Методы исследования. В работе использованы методы теории автоматов, объектно-ориентированного проектирования и программирования.

Научная новизна. В работе получены следующие научные результаты, которые выносятся на защиту:

1. Предложен метод реализации автоматных систем на основе библиотеки ***STOOL*** (SWITCH-Technology Object Oriented Library), устраняющий ряд недостатков ***SWITCH***-технологии.
2. Разработана графическая нотация для проектирования автоматных объектов, которая позволяет описывать декомпозицию и структурирование их логики с помощью наследования.
3. Разработан метод реализации автоматных объектов на основе виртуальных методов, обеспечивающий декомпозицию и структурирование логики с помощью наследования.
4. Разработан метод реализации автоматных объектов на основе виртуальных вложенных классов, обеспечивающий декомпозицию и структурирование логики с помощью наследования.

Результаты диссертации были получены в ходе выполнения научно-исследовательских работ «Разработка технологии программного обеспечения систем управления на основе автоматного подхода», выполненной по заказу Министерства образования РФ в 2001 – 2005 гг., и «Разработка технологии автоматного программирования», выполненной по гранту Российского фонда фундаментальных исследований № 02-07-90114 в 2002 – 2003 гг. (http://is.ifmo.ru, раздел «Наука»).

Достоверность научных положений, выводов и практических рекомендаций, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, а также результатами внедрения методов, предложенных в диссертации, на практике.

Практическое значение работы заключается в том, что все полученные результаты могут быть использованы, а некоторые уже используются, в практической деятельности. Предложенные методы проектирования и реализации автоматных объектов упрощают поддержку и сопровождение программ за счет сокращения дублирования логики автоматов. Предложенные методы реализации автоматных объектов основаны только на *стандартных* возможностях используемых языков программирования, что снижает *риски* при их практическом применении. Практическая ценность подтверждается внедрением результатов работы.

Внедрение результатов работы. Результаты, полученные в диссертации, используются на практике:

1. В компании *Транзас* (Санкт-Петербург) при разработке системы мониторинга мобильных объектов ***Navi-Manager***, и в частности, ее коммуникационного сервера.
2. В компании *Транзас* (Санкт-Петербург) при разработке каркаса редактора пространственных данных ***Iris***.
3. В учебном процессе на кафедре «Компьютерные технологии» СПбГУ ИТМО при чтении лекций по курсу «Теория автоматов в программировании».

Апробация диссертации. Основные положения диссертационной работы докладывались на научно-методических конференциях «Телематика-2003», «Телематика-2004», «Телематика-2005» (Санкт-Петербург) и XXXIII конференции профессорско-преподавательского состава СПбГУ ИТМО (Санкт-Петербург, 2004).

Публикации. По теме диссертации опубликовано 7 печатных работ, в том числе в журналах «Информационно-управляющие системы» и «Информационные технологии моделирования и управления».

Структура диссертации. Диссертация изложена на 187 страницах и состоит из введения, пяти глав и заключения. Список литературы содержит 88 наименований. Работа содержит 56 рисунков и две таблицы.

Содержание работы

Во введении описывается предмет исследования, ставятся цель и задачи исследования, обосновывается актуальность темы диссертационной работы. Дается оценка новизны полученных результатов, формулируются положения, выносимые на защиту.

Первая глава содержит обзор современного состояния проблемы проектирования и реализации реактивных систем на основе конечных автоматов.

Для проектирования и реализации реактивных систем на базе конечных автоматов широко применяются такие графические языки, как ***SDL***, ***UML***, а также ***SyncCharts***. В языке ***SDL*** конечные автоматы используются для описания поведения системы. Диаграммы ***Statecharts*** языка ***UML*** добавляют в традиционную модель конечных автоматов такие понятия как *иерархия* и *параллелизм*. Язык ***SyncCharts*** основан на *синхронной гипотезе* или *гипотезе атомарных реакций*.

Одним из недостатков описанных языков проектирования является отсутствие нотации для декомпозиции и структурирования логики конечных автоматов с помощью наследования. В настоящей работе предлагается графическая нотация, устраняющая этот недостаток.

Для устранения некоторых недостатков упомянутых языков, таких, например, как громоздкость, в России разрабатывается ***SWITCH***-технология, также известная как «автоматное программирование». Семантика используемых в ***SWITCH***-технологии графов переходов близка к семантике языка ***Statecharts***, но не эквивалентна ей. В ***SWITCH***-технологии вводятся понятия *автомат* и *вложение автоматов*, и отсутствуют понятия *вложенных* и *ортогональных* состояний. Вложение и ортогонализация состояний в ***SWITCH***-технологии реализуется посредством вложения автоматов и введения понятия «система автоматов».

Основными недостатками реализации программ на основе ***SWITCH***-технологии являются использование двух последовательно выполняющихся операторов `switch`, отсутствие автоматического контроля изменения состояний автоматов и механизмов автоматического протоколирования. В настоящей работе предлагается метод реализации программ, являющийся развитием ***SWITCH***-технологии и устраняющий некоторые ее недостатки.

После перехода на объектно-ориентированную методологию программирования, использование конечных автоматов сократилось по причине отсутствия полностью объектно-ориентированных методов их реализации. Многие номинально объектно-ориентированные методы реализации конечных автоматов, на самом деле, больше относятся к *структурному* программированию.

Наиболее распространенные *не* объектно-ориентированные методы реализации конечных автоматов основаны на вложенных условных операторах `if`, вложенных операторах `switch`, операторах `goto` и таблицах переходов.

Наиболее известным объектно-ориентированным методом реализации автоматных объектов является паттерн проектирования ***State***. На данный момент известно множество альтернативных вариантов паттерна ***State***.

Известен метод реализации автоматных объектов, при использовании которого состояние автоматного объекта хранится в виде набора указателей на методы, реализующие поведение объекта в данном состоянии. Известны также методы реализации, позволяющие расширять логику автоматных объектов с помощью наследования.

Одним из недостатков упомянутых методов реализации автоматных объектов является отсутствие способов декомпозиции и структурирования логики конечных автоматов с помощью наследования. В настоящей работе предлагаются методы, устраняющие этот недостаток.

Вторая глава содержит описание метода реализации автоматных систем на основе упомянутой выше библиотеки ***STOOL***, которая устраняет ряд недостатков реализации систем на основе ***SWITCH***-технологии. Библиотека ***STOOL*** базируется на следующих принципах:

- автоматы, входные и выходные воздействия являются объектами;
- явно вводятся понятия класс автоматов и экземпляр класса автоматов;
- каждый автомат не располагает никакой информацией о других автоматах системы;
- автоматные объекты хранят только свое текущее управляющее состояние. Вычислительные состояния вынесены за пределы системы автоматов и хранятся в специальном объекте – *окружении*.

Библиотека ***STOOL*** предоставляет абстрактные базовые классы для реализации автоматов, входных и выходных воздействий, а также «инфраструктуру» для организации системы в целом. Диаграмма основных классов библиотеки STOOL приведена на рис. 1.

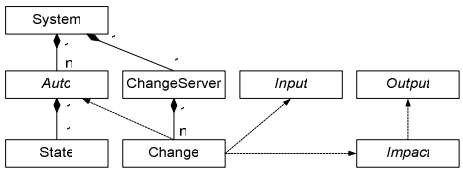

Рис. 1. Диаграмма основных классов библиотеки ***STOOL***

`Auto` – базовый класс для реализации классов автоматов. `State` – класс, который хранит состояние автомата. Каждый экземпляр класса `Auto` содержит экземпляр класса `State`. Класс `State` обеспечивает протоколирование любого (даже ошибочного) изменения состояния автомата. `Input` – базовый класс для реализации входных воздействий. `Output` – базовый класс для реализации выходных воздействий. `Impact` – класс, управляющий процессом выполнения выходного воздействия, с помощью методов «Выполнить», «Откатить» и «Подтвердить». `System` – класс, управляющий системой автоматов. `Change` – класс, управляющий системным переходом. `ChangeServer` – класс, управляющий созданием и уничтожением системных переходов.

По сравнению с уже существующими методами реализации программных систем на основе ***SWITCH***-технологии, библиотека ***STOOL*** обладает следующими достоинствами:

- для реализации графа переходов используется только один оператор `switch`;
- различаются действия и деятельности в состояниях;

- обеспечивается автоматическое протоколирование всех изменений, происходящих в системе автоматов;
- выделено состояние системы в целом;
- обеспечивается возможность повторного использования автоматов;
- реализован механизм обработки исключительных ситуаций;
- обеспечивается поддержка параллельных вычислений.

В работе на примере продемонстрирована целесообразность применения предложенной библиотеки.

Третья глава содержит описание предлагаемой графической нотации для проектирования автоматных объектов. Ее основным достоинством является возможность декомпозиции и структурирования логики поведения автоматных объектов с помощью наследования.

Граф переходов автоматного объекта изображается внутри рамки. Заголовок рамки содержит строку в формате

<Автоматный класс> (<Интерфейс>)[: <Базовый класс> [,<Базовый класс>]].

Состояния автоматного объекта изображаются в виде прямоугольников с закругленными углами. Начальное состояние выделяется жирным прямоугольником. Переходы изображаются в виде направленных дуг. Рядом с каждой дугой располагается спецификация перехода в формате

<Причина> [(<Условие>)] [/ <Действие>].

Базовый класс автомата, если он есть, указывается в заголовке рамки после двоеточия. Все состояния и переходы базового класса неявно переходят в производный автоматный класс. Состояния и группы состояний базового класса, упоминаемые на диаграмме производного класса, помечаются «жирной точкой». Разрешается перегрузка (замещение) переходов базового класса. Дуга перегруженного перехода имеет в качестве своего начала символическое изображение базового класса в виде закрашенной жирной точки.

Внутри рамки автоматного объекта изображается отсортированная по алфавиту таблица сокращений.

Структурирование логики позволяет обобщать поведение, общее для нескольких состояний. На диаграмме поведения состояния, входящие в одну группу, обводятся пунктиром. Состояния, входящие в одну группу, являются потомками одного базового класса состояния. Группы состояний могут вкладываться друг в друга, образуя иерархии.

Рассмотрим, например, семейство классов, предоставляющих доступ к файлу: доступ на чтение (`ReadFile`), доступ на запись (`WriteFile`) и доступ на чтение/запись (`ReadWriteFile`). Диаграммы поведения этих классов приведены на рис. 2.

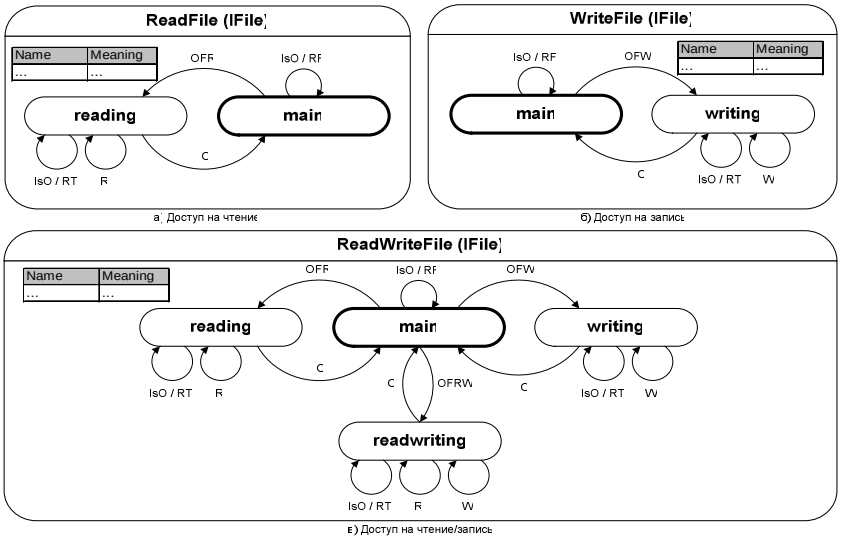

Рис. 2. Диаграммы поведения без использования наследования

Поведение описанных классов может быть обобщено (выделены одинаковые компоненты) и структурировано с помощью наследования. Диаграмма поведения этих классов, с использованием наследования, приведена на рис. 3.

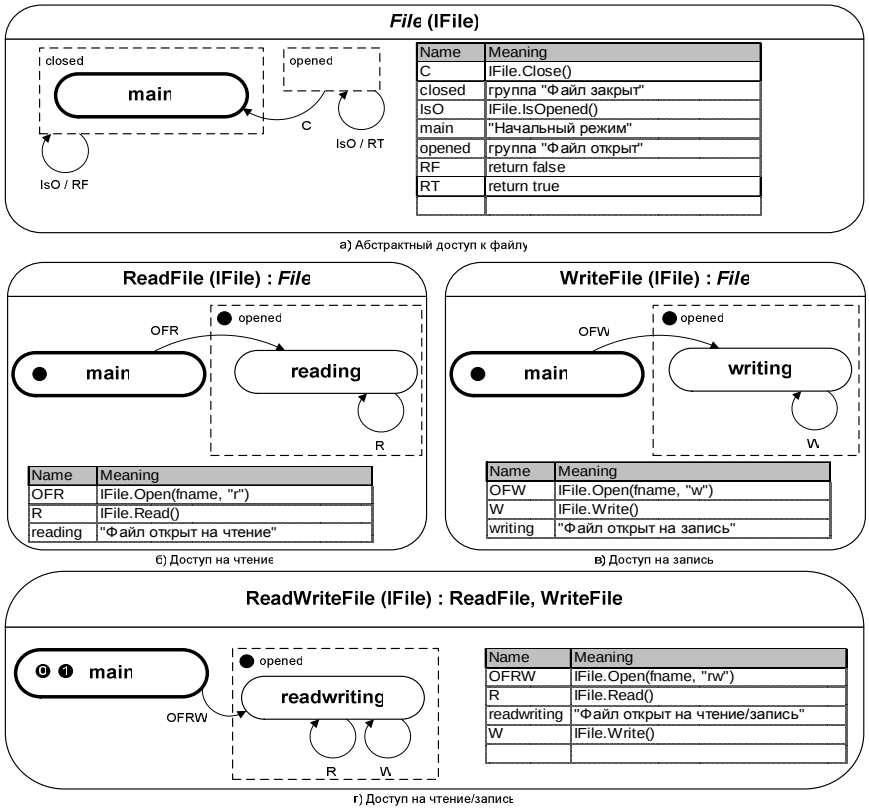

Рис. 3. Диаграммы поведения с использованием наследования

Автоматные классы `ReadFile` и `WriteFile` являются потомками класса `File`. Абстрактный автоматный класс `File` структурирует логику, вводя группы состояний `closed` и `opened`. Автоматный класс `ReadWriteFile` является потомком классов `ReadFile` и `WriteFile`. Все состояния и перехо-

ды, определенные в базовых классах, неявно переходят в класс `ReadWriteFile`.

Поведение любого из автоматных классов может быть расширено с помощью наследования. На рис. 4 приведена диаграмма поведения автоматного класса `AppendFile`, расширяющего логику автоматного класса `ReadWriteFile`, добавляя в него еще одно состояние – `appending`. Расширение происходит *инкрементально*, без изменения уже существующих автоматных классов.

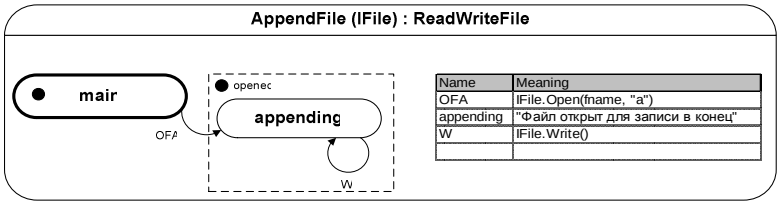

Рис. 4. Диаграмма поведения класса `AppendFile` с использованием наследования

Диаграмма поведения класса `AppendFile`, приведенная на рис. 4, эквивалентна диаграмме поведения класса `AppendFile`, приведенной на рис. 5, которая построена без использования наследования. Отметим, что использование наследования *значительно* сокращает дублирование состояний и переходов.

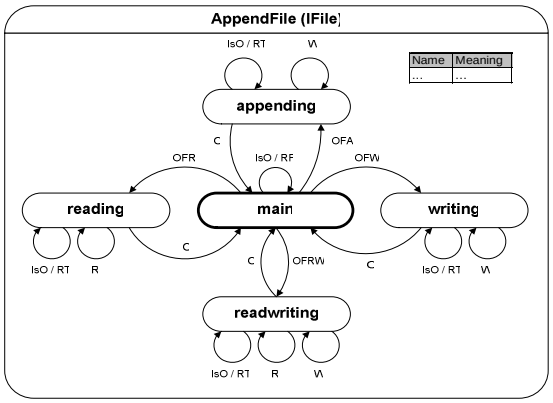

Рис. 5. Диаграмма поведения класса `AppendFile` без использования наследования

Таким образом, предложенная графическая нотация для проектирования автоматных объектов позволяет обобщать, декомпозировать, структурировать и расширять логику автоматных объектов с помощью наследования. При этом отметим, что по предложенным диаграммам поведения с наследованием, текст программы может быть построен формально и изоморфно с помощью методов, описанных в следующей главе.

Четвертая глава содержит описание методов реализации автоматных объектов, обеспечивающих возможность декомпозиции и структурирования логики автоматных объектов с помощью наследования.

Оба метода позволяют реализовать автоматный объект в соответствии с принципами объектно-ориентированного программирования.

Инкапсуляция. Факт наличия автоматной логики скрывается от клиентов объекта. Экземпляры классов состояний могут иметь вычислительные состояния.

Полиморфизм. Клиент может взаимодействовать с разными классами автоматных объектов универсальным образом.

Наследование. Поведение автоматного объекта может быть расширено с помощью существующего в языках программирования механизма наследования. Похожее поведение в разных состояниях и в разных автоматах может быть, например, обобщено и повторно использовано, как показано в предыдущей главе.

В методе реализации автоматных объектов на основе виртуальных методов (***VM***-метод) автоматный объект реализуется посредством трехуровневой структуры, состоящей из *интерфейса*, *посредника* и *контекста* (рис. 6). Реализация контекста основана на стандартном механизме виртуальных функций. Пространство состояний автомата отображается на множество виртуальных методов, так что каждому состоянию соответствует один и только один метод класса, называемый виртуальным *методом состояния*.

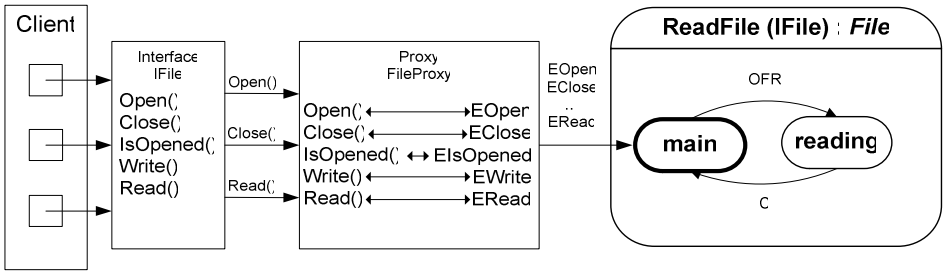

Рис. 6. Реализация автоматных объектов на основе виртуальных методов

Метод реализации автоматных объектов на основе **виртуальных методов** состоит в следующем:

1. Автоматный объект разделяется на три части: *интерфейс*, *посредник* и *контекст*.
2. Посредник реализует интерфейс, преобразуя вызовы его методов в сообщения, которые затем передаются для обработки контексту.
3. Контекст реализуется следующим образом:
   3.1. Поведение автоматного объекта в каждой из его групп состояний реализуется с помощью защищенного метода группы состояний.
   3.2. Поведение автоматного объекта в каждом из его состояний реализуется с помощью защищенного метода состояния.
   3.3. Текущее состояние хранится в виде приватного указателя на соответствующий метод состояния.
   3.4. Доступ к контексту предоставляется в виде открытого метода, принимающего сообщение и переадресовывающего его текущему методу состояния.
4. Методы состояния и группы состояний реализуются следующим образом:
   4.1. Методы состояния и группы состояний реализуются в виде защищенных методов контекста. Все методы имеют одинаковую сигнатуру.

4.2. В качестве единственного параметра методы принимают сообщение, соответствующее вызову метода интерфейса.
4.3. Метод состояния или группы состояний последовательно, для каждого перехода, возможного в данном состоянии или группе состояний, проверяет выполнение разрешительной части спецификации перехода и в случае успеха совершает этот переход, возвращая значение `true`.
4.4. В случаях, когда переход не был выполнен:
4.4.1. Если состояние или группа состояний вложены в некоторую группу, то управление передается методу этой группы.
4.4.2. В противном случае возвращается значение `false`.

Недостатками ***VM***-метода являются высокая трудоемкость: хранения *вычислительных состояний*, обеспечения константности данных и преобразования методов в сообщения и обратно.

Метод реализации автоматных объектов на основе виртуальных вложенных классов (***VIC***-метод) устраняет перечисленные недостатки, сохраняя возможность декомпозиции и структурирования логики с помощью наследования (рис. 7). Механизм *виртуальных вложенных классов*, доступный в некоторых языках программирования (например, ***Python***), позволяет перегружать вложенный класс базового класса в его потомках. В таких языках программирования, как ***C++*** и ***C#***, существует возможность эмулировать данный механизм.

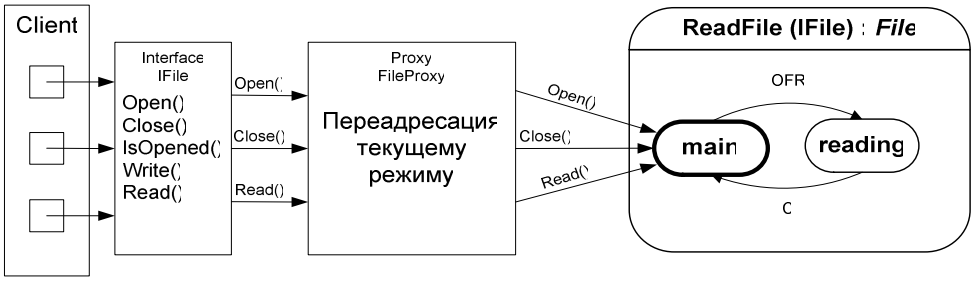

Рис. 7. Реализация автоматных объектов на основе виртуальных вложенных классов

Метод реализации автоматных объектов на основе **виртуальных вложенных классов** состоит в следующем:

1. Автоматный объект разделяется на три части: *интерфейс*, *посредник* и *контекст*.
2. Посредник реализует интерфейс, переадресовывая вызовы его методов контексту.
3. Контекст реализуется следующим образом:
3.1. Поведение автоматного объекта в каждой из его групп состояний реализуется с помощью вложенного защищенного класса группы состояний, реализующего интерфейс автоматного объекта.

3.2. Поведение автоматного объекта в каждом из его состояний реализуется с помощью вложенного защищенного класса состояния, реализующего интерфейс автоматного объекта.
3.3. Текущее состояние хранится в виде приватного указателя на экземпляр соответствующего класса состояния.
3.4. Переключение состояний осуществляется с помощью фабрик состояний.
3.5. Фабрики состояний реализуются в виде защищенных методов контекста, возвращающих экземпляры классов состояний.
3.6. Доступ к контексту предоставляется в виде открытого метода, возвращающего текущий экземпляр класса состояния.

4. Классы состояний и групп состояний реализуются в виде вложенных защищенных классов следующим образом:
4.1. Классы состояний и групп состояний реализуют интерфейс автоматного объекта.
4.2. Каждый метод класса состояния или группы состояний реализует переходы, возможные в данном состоянии или группе состояний и имеющие в качестве своей причины вызов соответствующего метода интерфейса.
4.3. Метод класса группы состояний может быть вынесен в контекст, для обеспечения возможности дальнейшего расширения поведения в этой группе.
4.4. Каждый метод класса состояния или группы состояний последовательно, для каждого из возможных состояний, проверяет выполнение условия перехода и в случае успеха совершает этот переход, завершая свое выполнение.
4.5. В случаях, когда переход не был выполнен:
4.5.1. Если состояние или группа состояний вложены в некоторую группу, то управление передается базовому классу;
4.5.2. Если тип возвращаемого значения позволяет вернуть значение δ (например: `false`, `null`), идентифицирующее *неуспех*, то возвращается значение δ.
4.5.3. Если тип возвращаемого значения (например, `void`) метода m не позволяет вернуть значение δ, идентифицирующее *неуспех*, то в случае необходимости метод m заменяется методом m', тип возвращаемого значения которого позволяет вернуть δ (без изменения интерфейса автоматного объекта).

Оба предложенных метода реализации автоматных объектов обеспечивают возможность декомпозиции и структурирования логики с помощью наследования.

Преимуществами ***VIC***-метода по сравнению с ***VM***-методом являются отсутствие трудоемкого преобразования методов интерфейса в события и возмож-

ность хранить вычислительные состояния в классах управляющих состояний. С другой стороны ***VM***-метод основан на стандартном механизме виртуальных методов и не требует создания экземпляров классов состояний.

Пятая глава содержит описание внедрения результатов настоящей работы на практике. Результаты, полученные в диссертации, используются в компании *Транзас* (Санкт-Петербург) при разработке:

- системы мониторинга мобильных объектов ***Navi-Manager***, и в частности, ее коммуникационного сервера;
- каркаса для построения редакторов пространственных данных ***Iris***.

Система ***Navi-Manager*** применяется для мониторинга различных типов морских, речных и наземных мобильных объектов, сопровождения технологических процессов на береговых объектах ГМССБ, охраны исключительной экономической зоны и т.д.

Эта система предоставляет возможность работы с глобальной системами спутниковой связи ***Inmarsat-C*** и ***Inmarsat-D***$^{+}$ (рис. 8, а). Одной из основных проблем, возникающих при работе с системой ***Inmarsat***, является сложность тестирования таких приложений. По ряду причин тестирование на реальной системе осуществляется исключительно редко (в основном на этапе окончательного тестирования). Для тестирования взаимодействия с системой ***Inmarsat*** применяются эмуляторы (рис. 8, б). Система глобальной спутниковой связи является типичным представителем реактивных систем. Поведение системы связи на всех уровнях определяется посредством обмена сообщениями между участниками системы. Эмулятор системы также является реактивной системой и может быть спроектирован и реализован на основе совместного использования объектно-ориентированного и автоматного программирования с помощью методов, предложенных в настоящей работе.

Эмулятор должен обеспечивать возможность: имитации серверов доступа систем ***Inmarsat-C*** и ***Inmarsat-D+*** с интерфейсами идентичными интерфейсам реальных серверов (`http`, `e-mail` и т. д.), использования упрощенного интерфейса для модульного тестирования, имитации поведения мобильных объектов, расширения логики поведения серверов и мобильных объектов, изменения масштаба времени.

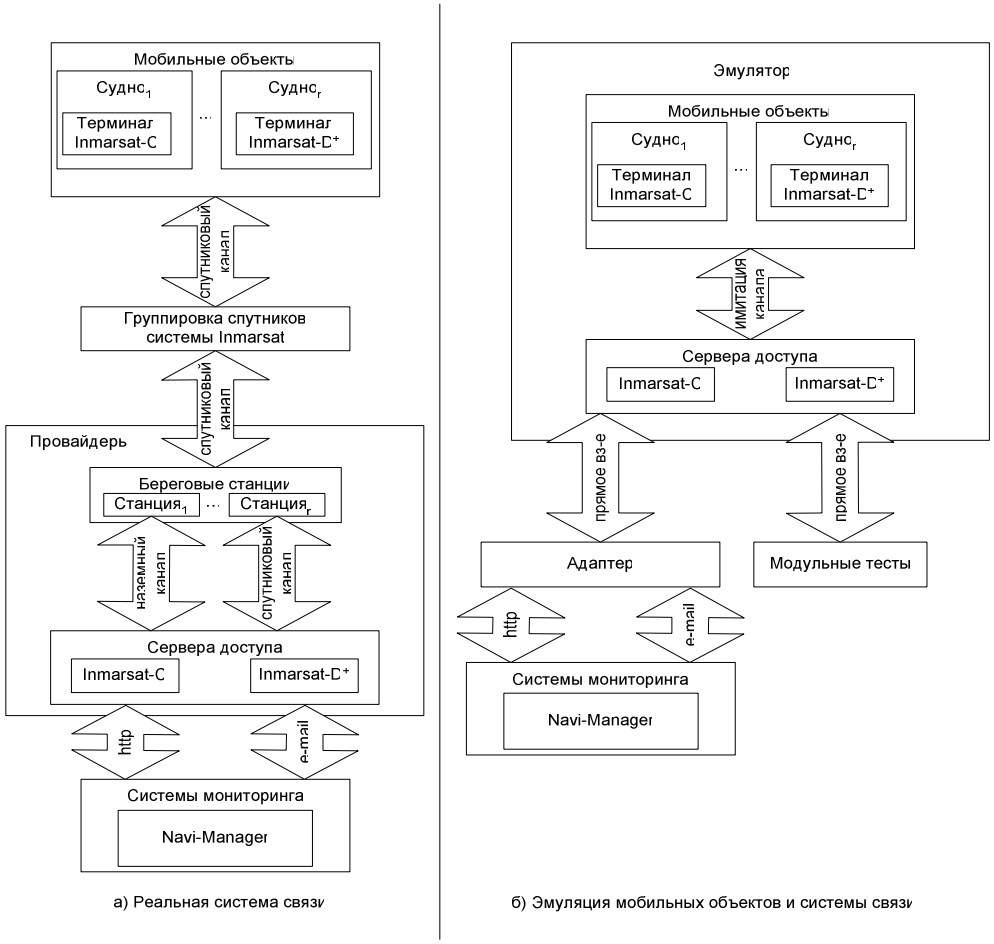

Рис. 8. Структурная схема системы мониторинга

Упрощенные диаграммы поведения эмуляторов терминалов систем ***Inmarsat-C*** и ***Inmarsat-D+*** приведены на рис. 9.

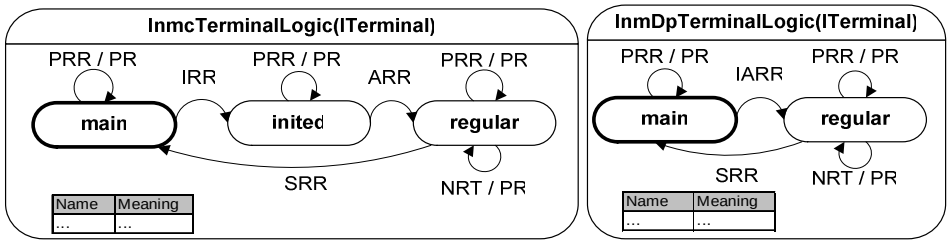

Рис. 9. Упрощенные диаграммы поведения эмуляторов терминала без использования наследования

Эквивалентные диаграммы поведения с использованием наследования автоматных объектов приведены на рис. 10. Как следует из этих диаграмм, использование декомпозиции и структурирования логики позволяет в значительной мере сократить дублирование. В работе приведен исходный код на языке ***C#***, реализующий эмуляторы терминалов системы ***Inmarsat***.

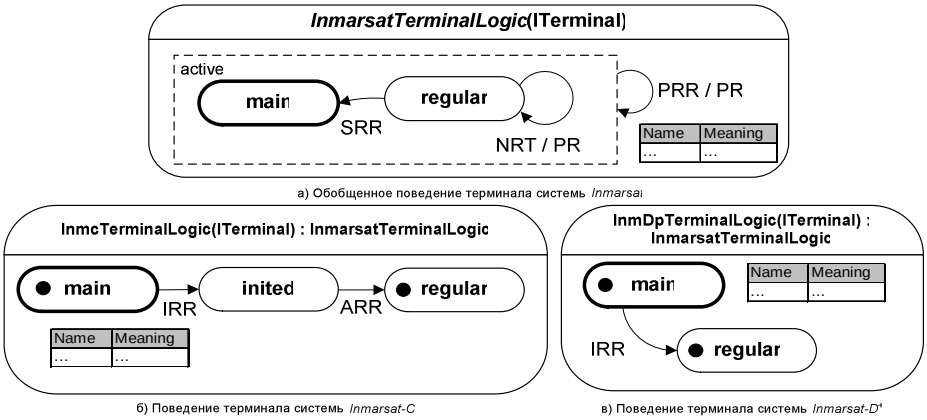

Рис. 10. Диаграммы поведения эмуляторов терминала с использованием наследования

Переходя к рассмотрению второй практической задачи, отметим, что визуальное редактирование пространственных данных является важной частью многих приложений в сфере навигационных, береговых и тренажерных систем. Большинство современных программных систем, разрабатываемых в этих областях, являются сложными, распределенными программно-аппаратными комплексами. Актуальной задачей является создание повторно используемого каркаса, облегчающего реализацию визуального редактирования в подобных системах.

Каркас ***Iris***, разрабатываемый в компании *Транзас*, предназначен для построения редакторов пространственных (*spatial*) данных. Архитектура каркаса ***Iris*** основана на принципах и идеях, предложенных Дж. Влиссидесом и М. Линтоном в каркасе для построения прикладных графических редакторов ***Unidraw***. При этом учтена специфика работы с пространственными данными в условиях распределенной программной системы.

Механизм интерактивного редактирования в каркасе ***Iris*** состоит из пятерки объектов: *инструмент*, *манипулятор*, *метка*, *призрак и команда*. Манипуляторы являются типичными представителями реактивных системам и для их разработки применяются методы проектирования и реализации автоматных объектов, предложенные в настоящей работе.

Вся логика, связанная с осуществлением манипуляции, *полностью* сконцентрирована внутри манипуляторов. Часть иерархии манипуляторов, определяемых каркасом ***Iris***, приведена на рис. 11.

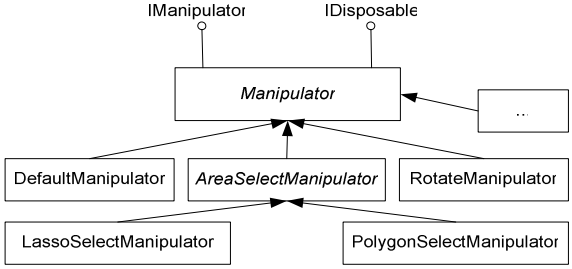

Рис. 11. Иерархия манипуляторов, предопределенных в каркасе ***Iris***

В работе подробно рассмотрены вопросы декомпозиции и структурирования логики автоматных объектов с помощью наследования на примере площадных манипуляторов выделения, определяемых каркасом ***Iris***. Диаграммы поведения манипуляторов площадного выделения, построенные без использования наследования, приведены на рис. 12.

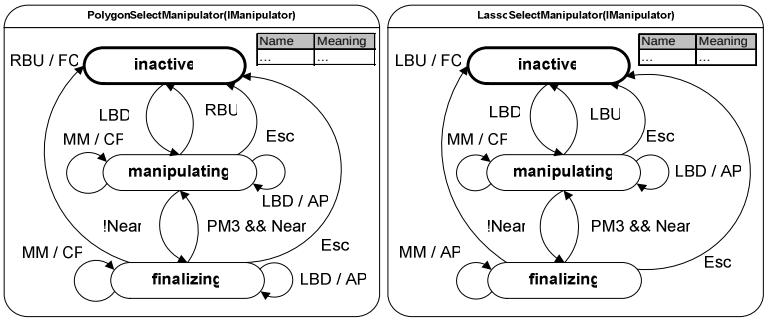

Рис. 12. Диаграммы поведения манипуляторов выделения

Поведение данных манипуляторов может быть обобщено и структурировано с помощью наследования так, как показано на рис. 13.

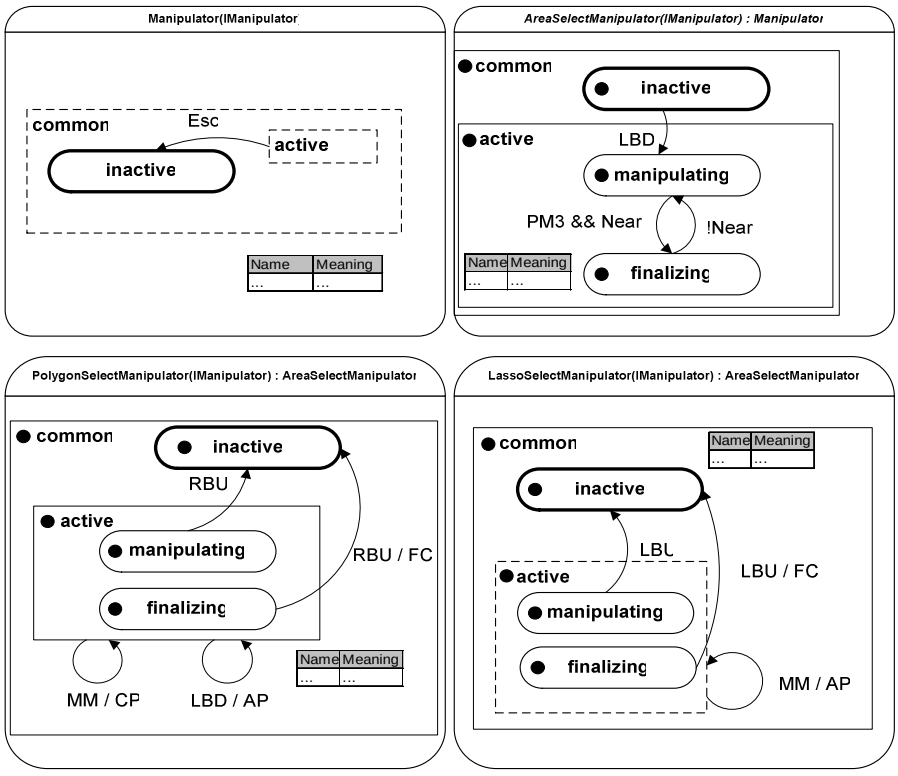

Рис. 13. Диаграммы поведения манипуляторов выделения с использованием наследования

В работе приведены фрагменты исходного кода, реализующего поведение манипуляторов площадного выделения и других манипуляторов каркаса ***Iris***.

Результаты внедрения подтверждают практическую ценность предложенных методов. Выполненный в работе *метрический* анализ различных вариантов реализации манипуляторов, демонстрирует целесообразность использования предложенных методов проектирования и реализации реактивных систем.

Заключение

В диссертации получены следующие научные результаты:

1. Предложен метод реализации автоматных систем на основе библиотеки ***STOOL***, устраняющий ряд недостатков существующих методов реализации систем на основе ***SWITCH***-технологии.
2. Разработана графическая нотация для проектирования автоматных объектов, позволяющая описывать декомпозицию и структурирование их логики с помощью наследования.
3. Разработан метод реализации автоматных объектов на основе виртуальных методов, обеспечивающий декомпозицию и структурирование логики автоматных объектов с помощью наследования.
4. Разработан метод реализации автоматных объектов на основе виртуальных вложенных классов, обеспечивающий декомпозицию и структурирование логики автоматных объектов с помощью наследования.
5. Результаты работы внедрены при разработке системы мониторинга мобильных объектов ***Navi-Manager*** и каркаса для построения редакторов пространственных данных ***Iris*** в компании *Transas*, а также в учебном процессе на кафедре «Компьютерные технологии» СПбГУ ИТМО при чтении лекций по курсу «Теория автоматов в программировании».

Список публикаций

1. **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.
2. **Шалыто А. А., Шопырин Д. Г.** Объектно-ориентированный подход к автоматному программированию //Информационно-управляющие системы. 2003, № 5, с. 29–39.
3. **Шалыто А.А., Шопырин Д.Г.** Синхронное программирование //Информационно-управляющие системы. 2004, № 3, с. 35–42.
4. **Шопырин Д.Г.** Разработка промежуточного языка представления конечных автоматов //Труды XI Всероссийской научно-методической конференции "Телематика-2004". СПб.: СПбГИТМО (ТУ). 2004. Т. 1, с. 195–197.
5. **Шопырин Д.Г.** Метод проектирования и реализации конечных автоматов на основе виртуальных вложенных классов //Информационные технологии моделирования и управления. 2005, № 1(19), с. 87–97.
6. **Шопырин Д.Г.** Объектно-ориентированная реализация конечных автоматов на основе виртуальных методов // Информационно-управляющие системы. 2005, № 3, с. 36–40.
7. **Шопырин Д.Г.** Программирование с явным выделением состояний на платформе .Net //Труды XII Всероссийской научно-методической конференции "Телематика-2005". СПб.: СПбГИТМО (ТУ). 2005. Т. 1, с. 86–87.

Кроме того, по теме диссертации имеется пять отчетов по выполненным научно-исследовательским работам (http://is.ifmo.ru, раздел «Наука»).

Тиражирование и брошюровка выполнены в
Центре «Университетские телекоммуникации»
Санкт-Петербург, Саблинская ул. 14. Тел (812)233-46-69
Тираж 100 экз.