На правах рукописи

Мазин Максим Александрович

Автоматное программирование для среды языково-ориентированного программирования

Специальность 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург
2010

Работа выполнена в Санкт-Петербургском государственном университете информационных технологий, механики и оптики (СПбГУ ИТМО)

Научный руководитель	доктор технических наук, профессор Шалыто Анатолий Абрамович
Официальные оппоненты	доктор технических наук, профессор Воробьев Владимир Иванович
	кандидат физ.-мат. наук, доцент Новиков Федор Александрович
Ведущая организация	Санкт-Петербургский государственный университет аэрокосмического приборостроения

Защита диссертации состоится 27 декабря 2010 года в 12 часов 30 минут на заседании диссертационного совета Д 212.227.06 в Санкт-Петербургском государственном университете информационных технологий, механики и оптики по адресу: 197101, Санкт-Петербург, Кронверкский пр. 49.

С диссертацией можно ознакомиться в библиотеке СПбГУ ИТМО.

Автореферат разослан 26 ноября 2010 г.

Ученый секретарь диссертационного совета,
доктор технических наук, профессор Л.С. Лисицына

Общая характеристика работы

Актуальность проблемы. С момента рождения программирования одной из основных проблем является написания программных систем со сложным поведением. В последнее время для создания таких систем все чаще используется автоматное программирование. Существует несколько подходов адаптации автоматного программирования к различным техникам и парадигмам программирования, среди которых доминирующей является объектно-ориентированное программирование. Однако эта парадигма существует в практически неизменном виде уже более тридцати лет и требует дальнейшего развития, что может быть выполнено на основе языково-ориентированного программирования. Это должно повысить уровень абстракции программного кода, упростить его реализацию и сопровождение. Поэтому адаптация автоматного программирования применительно к языково-ориентированной парадигме является актуальной.

Цель диссертационной работы – адаптация автоматного программирования для среды языково-ориентированного программирования.

Основные задачи исследования. Для достижения указанной цели диссертации решены следующие задачи, решения которых выносятся на защиту:

1. Разработка текстового языка автоматного программирования (абстрактный и конкретный синтаксисы, операционная семантика, системы типов, кодогенератор и т. д.).
2. Разработка средств валидации автоматов в среде языково-ориентированного программирования.
3. Создание языка для многопоточного автоматного программирования, основанного на акторах.
4. Внедрение результатов работы в практику программирования в среде языково-ориентированного программирования *MPS* (*Metaprogramming System*).

Научная новизна. Научная новизна предлагаемых подходов состоит в том, что автоматное программирование адаптировано для языко-ориентированного программирования в части обеспечения языковой поддержки, валидации и многопоточности.

Методы исследования. В работе использованы методы объектно-ориентированного проектирования, метапрограммирования, теории автоматов, теории формальных грамматик, теории графов, теории алгоритмов, исчисление секвенций, теории акторных моделей.

Достоверность научных положений и практических рекомендаций, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, компьютерным моделированием, а также результатами внедрения предложенной технологии.

Практическое значение полученных результатов состоит в том, что они успешно используются и будут использоваться в дальнейшем при разработке промышленных и учебных программных проектов на основе автоматного подхода и языково-ориентированной парадигмы.

Внедрение результатов работы. Результаты диссертации использованы на практике в компании *JetBrains* (Санкт-Петербург) при разработке коммерческой системы учета ошибок *YouTrack*, сданной в эксплуатацию. Кроме того автор был одним из разработчиков инструментального средства с открытым кодом для поддержки

автоматного программирования *UniMod*. Это средство опубликовано на сайте *sourceforge.net* и скачано более 60 тысяч раз.

Полученные результаты используются также в учебном процессе на кафедре «Компьютерные технологии» СПбГУ ИТМО при выполнении курсовых и бакалаврских работ по курсу «Теория автоматов в программировании».

Апробация диссертации. Основные положения диссертационной работы докладывались на конференциях и семинарах: II конференции молодых ученых СПбГУ ИТМО (2005 г.); XXXV, XXXVI научных учебно-методических конференциях СПбГУ ИТМО «Достижения ученых, аспирантов и студентов СПбГУ ИТМО в науке и образовании» (2005, 2006 гг.); «Телематика-2003», «Телематика-2004», «Телематика-2005», «Телематика-2006», «Телематика-2007» (СПбГУ ИТМО); на семинаре «Автоматное программирование» в рамках международной конференции «International Computer Symposium in Russia (CSR-2006)» (ПОМИ им. В. А. Стеклова, 2006 г.); Второй Всероссийской научной конференции «Методы и средства обработки информации» (МГУ, 2005 г.); Четвертой Всероссийской межвузовской конференция молодых ученых (СПбГУ ИТМО, 2009 г.); международной конференции «Software Engineering Conference (Russia)» (М., 2005 г.); форуме по открытому коду «Open Source Forum» (М., 2005 г.); второй международной научной конференции «Компьютерные науки и информационные технологии» (Саратов, 2007 г.); международной конференции «110 Anniversary of Radio Invention» (СПбГЭТУ «ЛЭТИ», 2005 г.).

Публикации. По теме диссертации опубликована 31 научная работа, из которых 24 – печатные. Результаты диссертации опубликованы в следующих журналах из списка ВАК: «Программирование», «Информационно-управляющие системы», «Научно-технический вестник СПбГУ ИТМО» и «Компьютерные инструменты в образовании».

Свидетельства об официальной регистрации программ для ЭВМ. На инструментальное средство для поддержки автоматного программирования, разработанное в рамках диссертации, получены следующие свидетельства: «Ядро автоматного программирования» №2006 613249 от 14.09.2006, «Встраиваемый модуль автоматного программирования для среды разработки *Eclipse*» №2006 613817 от 7.11.2006.

Участие в научно-исследовательских работах. Основные результаты по теме диссертации получены в ходе научно-исследовательских и опытно-конструкторской работ, проводимых на кафедре «Компьютерных технологий» СПбГУ ИТМО в:

- 2002, 2003 гг. по теме «Разработка технологии автоматного программирования» (грант Российского фонда фундаментальных исследований по проекту 02–07–90114);
- 2002 – 2004 гг. по теме «Разработка технологии создания программного обеспечения систем управления на основе автоматного подхода» (Министерство образования и науки РФ);
- 2005, 2006 гг. по теме «Автоматное программирование» (Федеральная целевая научно-техническая программа «Исследования и разработки по приоритетным направлениям науки и техники» на 2002 – 2006 гг.);
- 2008 г. по теме «Разработка основных положений создания программных систем управления со сложным поведением на основе объектно-ориентированного и автоматного подходов» (Министерство образования и науки РФ);

- 2009 г. по теме «Методы повышения качества при разработке автоматных программ с использованием функциональных и объектно-ориентированных языков программирования» (Федеральная целевая программа «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 гг.).

Структура диссертации. Диссертация изложена на 136 страницах и состоит из введения, пяти глав и заключения. Список литературы содержит 119 наименований. Работа иллюстрирована 65 рисунками.

Содержание работы

Во введении обосновывается актуальность темы диссертационной работы. Ставятся цель и задачи исследования, решения которых выносятся на защиту.

В первой главе проведен обзор особенностей автоматного программирования в зависимости от использования различных парадигм программирования.

Автоматное программирование, иначе называемое «программирование от состояний» или «программирование с явным выделением состояний» – это подход к разработке программного обеспечения (ПО), основанный на расширенной модели конечных автоматов и ориентированный на создание широкого класса приложений. При этом речь идет не только и не столько об использовании конечных автоматов в программировании, сколько о методе создания программ в целом, поведение которых описывается автоматами. Первоначально автоматное программирование называлось *SWITCH*-технологией.

Парадигма автоматного программирования состоит в представлении сущностей со сложным поведением в виде *автоматизированных объектов управления*. Для этого сущность со сложным поведением разделяется на две части:

- управляющую часть, ответственную за поведение – выбор выполняемых действий, зависящий от текущего состояния и входного воздействия, а также за переход в новое состояние;
- управляемую часть, ответственную за выполнение действий, выбранных управляющей частью, и, возможно, за формирование некоторых компонентов входных воздействий для управляющей части – *обратных связей*.

В соответствии с теорией управления, управляемая часть называется *объектом управления*, а управляющая часть – *системой управления*. В рамках рассматриваемой парадигмы для реализации управляющей части используется система взаимосвязанных автоматов, каждый из которых называется *управляющим автоматом* или просто *автоматом*.

Автоматное программирование до последнего времени поддерживалось следующими программными платформами:

- специализированные языки **программируемых логических контроллеров**, например, лестничные схемы, функциональные блоки. Существуют методы изоморфного преобразования автоматов в конструкции этих языков: предлагается рисовать диаграммы состояний вручную, проводить формальный перевод в лестничные схемы или в функциональные блоки, а по построенной схеме генерировать программу;
- **процедурные** языки программирования, например, язык *C*. Для этих языков существуют методы автоматного проектирования: *проектирование сверху вниз* и *проектирование от объектов управления и событий*. Также существуют

инструментальные средства, поддерживающие автоматное программирование для этих языков: *Visio2Switch*, *MetaAuto*, *Visio2Auto*;

- **объектно-ориентированные** языки программирования, например, языки *C++*и *Java*. При участии автора был разработан метод создания объектно-ориентированных автоматных программ. При использовании этого метода при построении диаграмм в рамках *SWITCH*-технологии сохраняется автоматный подход, но применяется стандартная *UML*-нотации. При этом предлагается, используя нотацию *UML*-диаграмм классов, строить схемы связей автоматов, а графы переходов – используя нотацию *UML*-диаграмм состояний. Для поддержки этого метода автором и В. Гуровым было создано инструментальное средство *UniMod*, на основе которого В. Гуров защитил диссертацию на тему «Технология проектирования и разработки объектно-ориентированных программ с явным выделением состояний (метод, инструментальное средство, верификация)»;
- **функциональные** языки программирования, например, языки *Erlang* и *Haskel*. Существуют методы преобразования автоматов в конструкции таких языков;
- **динамические** языки программирования, например, язык *Ruby*. Существуют библиотеки, реализующие внутренние проблемно-ориентированные языки автоматного программирования.

Далее в первой главе приведен обзор различных взглядов на область программирования, связанную с применением проблемно-ориентированных языков:

- **модельно-ориентированный поход** (*model driven architecutre*) состоит в описании функциональности систем в виде платформо-независимых моделей с помощью специальных проблемно-ориентированных языков программирования;
- **порождающее программирование** (генеративное программирование), при применении которого предлагается настраивать и связывать компоненты с помощью высокоуровневых средств, а низкоуровневый код порождать автоматически;
- **метапрограммирование** – создание программ (например, кодогенераторов), которые используют код других программ или свой собственный в качестве входных данных;
- **ментальное программирование** или программирование намерений (*intentional programming*) состоит в приведении программного кода к виду как можно более точно отражающему ход мысли программиста;
- **языково-ориентированное программирование** – подход, при котором вместо создания программ на языках общего назначения, сначала разрабатываются проблемно-ориентированный язык для данной предметной области, а затем на этом языке создается программа.

Все эти названия отражают *различные аспекты* общего подхода к разработке ПО, и, так или иначе, связаны с разработкой проблемно-ориентированных или предметно-ориентированных языков (*domain specific language*).

Также в первой главе рассмотрены особенности среды языков-ориентированного программирования *MPS* (*Metaprogramming System*), приводящие к необходимости создания новых подходов к автоматному программированию:

1. Специфика создания языков, без разработки компиляторов.
2. Возможность встраивать сообщения об ошибках на лету в процессе программирования позволяет сообщать об ошибках валидации. Позволяет

выполнять валидацию автоматов «на лету», непосредственно в процессе написания их кода.

3. Возможность интеграции в среде *MPS* языков друг с другом. Открывает возможность построения языковых средств многопоточного автоматного программирования. Традиционная реализация многопоточного автоматного взаимодействия предполагает использование общей памяти, что приводит к необходимости синхронизации потоков непосредственно в программном коде.

Первая глава завершается формулировкой задач, решаемых в диссертационной работе.

Анализ особенностей среды языково-ориентированного программирования показал целесообразность исследования в области автоматного программирования для этой среды.

Вторая глава посвящена созданию языка и инструментального средства для автоматного программирования, при использовании которых код автомата пишется в виде текста, а по нему, при необходимости, генерируется диаграмма переходов.

При создании языка в среде *MPS* требуется разработать:

- **структуру** абстрактного синтаксического дерева (АСД) для создаваемого языка. Узлам АСД могут соответствовать такие понятия как «объявление класса», «вызов метода», «операция сложения» и т. п.;
- **модель проекционного редактора** для каждого типа узла АСД. Задание редактора для узла АСД равноценно заданию конкретного синтаксиса для этого узла. При этом если для традиционных текстовых языков программирования создание удобного редактора – отдельная сложная задача, то для языков, созданных с помощью среды *MPS*, *редакторы являются частью языка*. Эти редакторы поддерживают автоматическое завершение ввода текста, навигацию по коду, выделение цветом ошибок в коде программы и т. д.;
- **модель системы типов** для языка;
- **модель трансформации программы** на задаваемом языке в исполняемый код.

Таким образом, создание языка в среде *MPS* сопровождается созданием инструментального средства разработки программ на этом языке.

Отметим, что среда *MPS* позволяет, как создавать новые языки, так и расширять языки уже созданные с помощью этой системы.

В отличие от традиционных языков, языки, созданные с помощью среды *MPS*, не являются текстовыми в традиционном понимании, так как при программировании на них пользователь пишет не текст программы, а вводит ее в виде АСД с помощью специальных проекционных редакторов. Структура и внешний вид этих редакторов таковы, что работа с моделью программы для пользователя выглядит, как традиционная работа с текстом программы. Отказ от традиционного текстового ввода программ значительно упрощает создание новых языков – исчезает необходимость в разработке лексических и синтаксических анализаторов, и, как следствие, перестают действовать ограничения на класс грамматик языков. Недостатком такого подхода является зависимость языков от среды *MPS* – невозможно разрабатывать программы без этой среды. Однако подобное ограничение присуще и традиционным, чисто текстовым языкам, которые зависят от компиляторов. Впрочем, после трансляции программы, написанной на языке, созданном в среде *MPS*, исполняемый код перестает зависеть от этой среды.

Ядро среды *MPS* написано на кросс-платформенном языке *Java*. В связи с этим в среде *MPS* существуют развитые средства для взаимодействия с *Java*-платформой. Среда *MPS* позволяет писать код только на тех языках, которые созданы в этой среде, поэтому для написания *Java*-кода в среде *MPS* разработан язык *baseLanguage*. Этот язык является почти полной реализацией спецификации *Java* 5. В нем определены такие конструкции, как «класс», «интерфейс», «метод», «предложение», «выражение» и т. д.

Язык автоматного программирования в среде *MPS* назван *stateMachine*. Он представляет собой автоматное расширение языка *baseLanguage.* Основная цель, которая преследовалась при разработке языка *stateMachine*, – создание языка, позволяющего описывать поведение классов в виде автоматов, не накладывая на сами классы никаких дополнительных ограничений. Более того, автоматное описание поведения класса должно быть инкапсулировано – код, использующий класс не должен «знать» о том, каким образом задано поведение класса. Этот подход отличается от предлагаемого системой *UniMod* тем, что позволяет использовать автоматы не только в программах, написанных в соответствии со *SWITCH*-технологией, но и в традиционных объектно-ориентированных программах.

Каждый автомат в языке *stateMachine* связан с некоторым классом и описывает его поведение. Для того чтобы задать поведение класса с помощью автомата необходимо в этом классе определить события, на которые будет реагировать автомат. Особенностью языка *stateMachine* является то, что события в нем – это методы специального вида. В языке *Java* декларации методов различаются по способу реализации:

- обычные методы, реализация которых следует сразу за объявлением метода;
- нативные методы, реализованные на платформо-зависимых языках программирования;
- абстрактные методы, которые вообще не имеют реализации, непосредственно связанной с декларацией.

В диссертации предлагается еще один способ реализации – декларация метода, реализация которого находится в автомате и зависит от текущего его состояния. Для этого язык *stateMachine* расширяет язык *baseLanguage* конструкцией *событие* (*EventMethodDeclaration*), соответствующего указанной декларации метода. На рис. 1 приведен пример объявления события *e1*.

```
public class A1 extends <none> implements <none> {
  /**
    Событие
   */
  public event e1(int p1, int p2);

  /**
    Обычный метод
   */
  public void z1() { }
}
```

Рис. 1. Пример объявления события

Рассмотрим операционную семантику и функциональные возможности языка *stateMachine*.

Реакция автомата на событие зависит от состояния, в котором автомат находится. В каждом состоянии для каждого события может быть определен набор переходов. Для перехода может быть задано *условие на переходе*, *действие на переходе* и *целевое*

состояние. В процессе обработки события для текущего состояния перебираются все переходы, помеченные данным событием, до тех пор, пока не будет найден первый переход, условие на котором выполняется. Если такой переход найден, то выполняется действие на переходе и осуществляется переход в целевое состояние. Отсутствие условия на переходе интерпретируется, как тождественно истинное условие.

События в языке *stateMachine* могут иметь параметры. На переходе с каждым параметром события может быть связано *фильтрующее значение*. Если такое значение задано, то переход выполняется только, если значение параметра обрабатываемого события совпадает с фильтрующим значением.

Состояния автомата бывают двух типов: обычные и конечные. Конечные состояния отличаются тем, что не могут иметь исходящих переходов. Следовательно, когда автомат оказывается в конечном состоянии, он перестает обрабатывать события.

Для состояния могут быть определены действия при входе в состояние и действия при выходе из состояния.

Первое по порядку состояние, объявленное в автомате, считается начальным. При создании экземпляра класса, для которого определен автомат, после выполнения конструктора осуществляется переход в начальное состояние.

Обычные состояния могут быть вложены друг в друга. При этом если состояние содержит другие состояния, то оно называется составным. При переходе в составное состояние выполняется переход в первое по порядку вложенное в него состояние. Поэтому автомат после обработки события не может оказаться в составном состоянии. Каждый исходящий из составного состояния переход работает так, как если бы такой переход был добавлен к каждому вложенному состоянию.

Каждый автомат в языке *stateMachine* связан с некоторым классом, имеет доступ к его полям и может вызывать методы этого класса. Поэтому в языке *stateMachine* нет необходимости в специальных конструкциях для взаимодействия между автоматами, так как вместо вложения одного автомата в другой можно использовать агрегацию одного *автоматного класса* другим автоматным классом, а посылка события из одного автомата в другой не что иное, как вызов метода.

С каждым автоматом может быть связано несколько объектов-слушателей (*listener*), которые оповещаются о стадиях обработки события. Например, таким слушателем может быть объект, выводящий сообщения о действиях автомата в специальный файл протокола работы автомата.

После написания программы на языке *stateMachine*, она сначала транслируется в *Java*-код, а затем компилируется стандартным *Java*-компилятором. Преимуществом этого языка является простота его использования в объектно-ориентированных приложениях, написанных на языке *Java*. При применении этого языка проверка корректности программы осуществляется на стадии ее написания, а не в процессе компиляции.

Многие программисты предпочитают набирать код программы в текстовом виде. Поэтому разработанный автором язык *stateMachine* является текстовым языком, реализованным в среде *MPS*. Однако диаграммы переходов позволяют представлять автоматы более наглядно. Поэтому предлагается по текстовому описанию автомата автоматически строить диаграмму переходов. Для этого реализованы средства интеграции языка *stateMachine* со средой *MPS*, которые выполняют построение диаграммы автоматически.

В третьей главе описан метод валидации автоматов. Рассмотрена валидация следующих свойств автоматных моделей:

- полнота и непротиворечивость условий на переходах, содержащих булевы формулы с предикатами сравнения входных переменных с константой и входных переменных друг с другом.
- достижимость любого состояния из начального состояния;
- достижимость из любого состояния конечного состояния.

Задача автоматической валидации автоматных моделей с предикатами сравнения входных переменных с константами в условиях на переходах была решена автором ранее. Для дальнейшего увеличения выразительной мощности автоматных моделей необходимо расширить класс формул для выражения условий на переходах двуместными предикатами сравнения входных переменных друг с другом.

Для описания предложенных методов и обоснования их корректности в работе выполнено построение формальной модели валидируемого автомата. Показан метод сведения задачи проверки полноты и непротиворечивости переходов к задаче проверки тавтологичности булевой формулы, содержащей в качестве пропозициональных переменных булевы константы, входные переменные, операции сравнения входных переменных с константами и другими входными переменными.

Предлагаемый метод проверки тавтологичности булевых формул состоит из трех этапов:

1. Приведения исходной формулы методами секвенциального исчисления к набору элементарных секвенций – контрпримеров, состоящих из пропозициональных переменных.
2. Исключения из этого набора секвенций, содержащих булеву константу «ложь» в левой части, булеву константу «истина» в правой части или одинаковые переменные одновременно и в левой и в правой части.
3. Исключения из оставшегося набора секвенций, задающих некорректное отношение порядка на множестве входных переменных и констант.

Пустота полученного в результате набора контрпримеров эквивалентна тавтологичности исходной формулы. Если контрпримеры существуют, то они используются для *автоматизированного исправления ошибок* неполноты и противоречивости переходов в среде *MPS* (рис. 2, рис. 3).

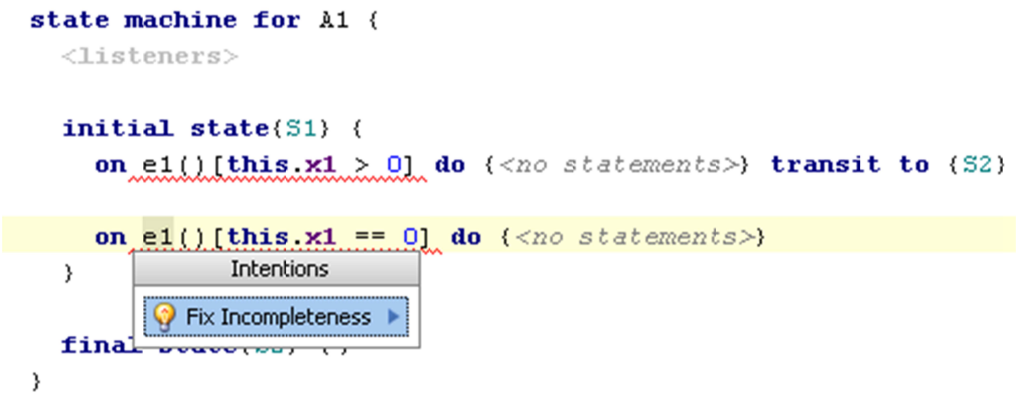

Рис. 2. Диалог исправления неполноты переходов

```
state machine for A1 {
  <listeners>

  initial state(S1) {
    on e1()[this.x1 > 0] do {<no statements>} transit to (S2)

    on e1()[this.x1 == 0 || (this.x1 < 0)] do {<no statements>}
  }

  final state(S2) { }
}
```

Рис. 3. Код автомата после применения автоматизированного исправления ошибки неполноты переходов

Достижимость любого состояния из начального и достижимость конечного состояния из любого состояния предлагается проверять модификацией алгоритма поиска в глубину, учитывающей вложение состояний друг в друга.

В четвертой главе описаны акторный язык и инструментальное средство для многопоточного автоматного программирования в среде *MPS*.

В настоящее время растет актуальность разработки параллельных программ. Это связано с тем, что дальнейшее повышение производительности вычислительных систем более не может достигаться увеличением тактовой частоты процессоров, и главной тенденцией становится рост числа ядер в системах. Однако для того, чтобы программа выполнялась эффективно на нескольких ядрах, она должна быть написана с применением техник параллельного программирования.

Универсальные императивные языки программирования, наиболее распространенные на сегодняшний день, хорошо подходят для реализации последовательных алгоритмов, но плохо приспособлены для написания параллельных алгоритмов. Автоматическому распараллеливанию хорошо поддаются программы, написанные на функциональных языках программирования.

В связи с этим естественно адаптировать абстракции параллельных функциональных программ для императивных языков программирования. К таким абстракциям, в частности, относится *акторная модель*, реализованная, например, в языке программирования *Erlang*.

Эта модель базируется на понятии *актора* – примитива параллельных вычислений, обладающего потоком выполнения и способного обмениваться сообщениями с другими акторами.

Разработанный язык *actors* для среды *MPS* позволяет использовать акторы наряду с обычными объектами при программировании на языке *Java*. Декларация типа актора, поддерживаемая языком, синтаксически совпадает с декларацией *Java*-класса с той лишь разницей, что вместо ключевого слова *class* используется ключевое слово *actor*. Для типа актора могут быть определены конструкторы, поля и объявления типов обрабатываемых сообщений. Синтаксис последних полностью совпадает с синтаксисом объявлений обычных методов в языке *Java*. Аналогично, синтаксис отправки сообщений актору, не отличается от синтаксиса вызова методов у объекта. Однако, в отличие от вызова методов, отправка сообщений происходит асинхронно: сообщения помещаются в индивидуальную очередь актора и последовательно актором обрабатываются.

При этом гарантируется, что сообщения, направленные одному и тому же актору, не могут обрабатываться параллельно двумя разными потоками. Таким образом, эмулируется обладание актором потоком управления.

Язык *actors* позволяет при посылке сообщения указать минимальную задержку, с которой оно должно быть обработано. Для этого в этот язык введена конструкция *defer*, которая может быть применена к оператору посылки сообщения. В качестве параметра конструкция *defer* принимает величину минимальной задержки, для задания которой используется встроенное в среду *MPS* языковое расширение *Dates*.

В традиционном объектно-ориентированном программировании некоторые методы в результате своего выполнения возвращают вычисленные значения. Эти значения становятся доступны вызывающему коду сразу по окончанию синхронного выполнения таких методов. Так может быть организованно взаимодействие между вызываемым и вызывающим кодом.

При программировании с использованием акторов нет возможности использовать возвращаемые значения, так как к тому моменту, когда операция асинхронной посылки сообщения оказывается выполнена, само сообщение может быть еще не обработано, а возвращаемое значение не вычислено.

Для того чтобы вызываемый код мог оповестить вызывающий код об окончании обработки события и передать вычисленное значение, можно передавать вызывающий актор в качестве параметра сообщения. По окончании обработки сообщения вызываемый актор должен послать вызывающему актору ответное сообщение.

Однако при таком подходе код обрабатывающего сообщение актора должен «знать» о структуре посылающего сообщение актора, то есть возникает нежелательная сильная связанность кода.

Для того чтобы обойти эту проблему и позволить использовать возвращаемые обработчиками сообщений значения, в языке *actors* существует поддержка механизма отложенных результатов. Для любого сообщения актора, так же как и для любого метода в языке *Java*, могут быть заданы тип возвращаемого значения и набор генерируемых исключений. Код обработчика сообщения должен либо вычислить значение соответствующего типа, либо сгенерировать исключение.

Разработанные языки *stateMachine* и *actors* являются расширениями языка *baseLanguage*. В совокупности они образуют язык многопоточного автоматного программирования.

Концепт *ActorConcept*, использующийся в языке *actors* для представления актора, унаследован от концепта *ClassConcept* языка *baseLanguage*. Каждый автомат в языке *stateMachine* связан с классом, автоматное поведение которого он описывает. В частности, таким классом может быть экземпляр концепта *ActorConcept* (рис. 4).

Таким образом, комбинация языков *stateMachine* и *actors* позволяет описывать автоматы, обладающие собственным потоком управление и обрабатывающие события асинхронно. Такие автоматы сочетают в себе достоинства автоматного программирования и акторной модели. Следовательно, они могут быть использованы для многопоточного автоматного программирования.

```
public actor A  {
  public A() { }

  public event e1();
}

state machine for A {
  initial state(S1) {
    on e1 do {} transit to (S2)
  }
  state(S2) {
    on e1 do {} transit to (S1)
  }
}
```

Рис. 4. Код на языках *stateMachine* и *actors*

Пятая глава содержит описание результатов внедрения разработанных средств автоматного программирования в промышленную разработку программного обеспечения.

Разработанный язык автоматного программирования был применен при разработке нескольких подсистем коммерческой системы учета программных дефектов *YouTrack*. В качестве примера в диссертации описывается реализация подсистемы управления и слежения за парком серверов с развернутыми на них экземплярами системы *YouTrack*.

Эта подсистема реализована в виде отдельного *WEB*-приложения с базой данных и позволяет выполнять следующие задачи:

- запускать и останавливать виртуальные хосты (компьютеры), работающие под управлением *Amazon Elastic Cloud*;
- создавать виртуальное устройство хранения данных (жесткий диск) и подключать его к запущенному хосту;
- загружать новые или обновлять существующие версии программного обеспечения на запущенном виртуальном хосте;
- запускать и останавливать *Java* процессы на указанных виртуальных хостах;
- запускать и останавливать экземпляры системы *YouTrack* внутри *Java*-процессов;
- следить за состоянием всех запущенных процессов;
- внешним пользователям регистрировать заявки на создание и запуск собственного экземпляра *YouTrack*.

В результате анализа предметной области были выделены следующие классы:

- *YouTrackInstance* – соответствует экземпляру системы *YouTrack*, ассоциирован с *Java*-процессом, в котором запущен;
- *YouTrackServer* – соответствует *Java*-процессу, ассоциирован с множеством объектов класса *YouTrackInstance*.
- *EC2Host* – отвечает за представление виртуального хоста из *Amazon Elastic Cloud*, агрегирует множество экземпляров класса *YouTrackServer*, которые запущены на хосте.

На рис. 5 представлена диаграмма классов описанной предметной области.

Состояние перечисленных классов должно сохраняться при перезагрузках *WEB*-приложения. Для этого целесообразно использовать базу данных. В рамках проекта *YouTrack* для хранения объектов в базе данных при участии автора был разработан проблемно-ориентированный язык *DNQ* (*Data Navigation and Query*). В этом языке определен концепт *хранимый класс* (*PersistentClassConcept*), который расширяет понятие

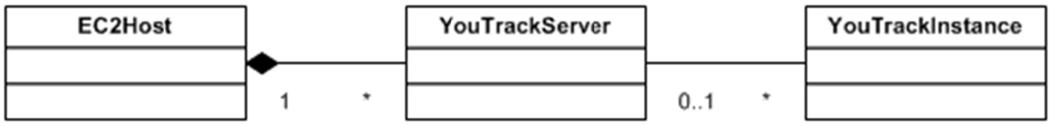

Рис. 5. Диаграмма классов предметной области

Java-класса языка *baseLanguage*. Таким образом, язык *DNQ* позволяет проектировать схему базы данных в виде набора хранимых классов. Запросы к данным записываются в виде выражений языка для работы с коллекциями (*collections*). Эти выражения при генерации преобразуются в оптимизированные запросы к базе данных.

Хранимые классы *DNQ* – это расширенные классы языка *baseLanguage*. Поэтому к ним применимы все расширения последнего, в том числе, и автоматное расширение, описанное в данной работе.

При применении языка *stateMachine* для описания поведения обычных *Java*-классов языка *baseLanguage* в процессе генерации кода автоматная часть редуцируется до конструкций языка *baseLanguage*. После этого генерируется текст программы. При использовании языка *stateMachine* для описания поведения хранимых классов языка *DNQ* генерация кода выполняется в несколько этапов:

1. Программа на языках *DNQ*, *stateMachine* и *baseLanguage* редуцируются до программы, содержащей только конструкции языков *DNQ* и *baseLanguage*. При этом для хранения состояния создается хранимое перечисление, содержащее все состояния класса, а в соответствующий хранимый класс добавляется ассоциация с созданным перечислением для хранения текущего состояния.
2. Код на языке *DNQ* редуцируется до кода на языке *baseLanguage*. При этом конструкции вида *switch(state)*, где *state* – имя ассоциации с хранимым перечислением, редуцируются в последовательность операторов *if*, *else-if*. Это связано с тем, что целевой код, генерируемый из хранимых перечислений, не является *Java*-перечислением и поэтому не может быть использован в качестве аргумента конструкции *switch*.
3. Из модели на языке *baseLanguage* выполняется генерация текста программы.

Из изложенного следует, что предложенный язык автоматного программирования *stateMachine* можно использовать для описания поведения хранимых классов, состояние которых сохраняется в базе данных. При применении автоматного языка для описания поведения хранимых классов за счет механизма расширения *MPS*-языков удается полностью переиспользовать все его синтаксические конструкции. Предлагаемый подход позволяет унифицировать описание поведения всех классов в программе, что повышает читаемость кода и упрощает его поддержку.

Заключение

В диссертационной работе решены следующие задачи:

1. В среде языково-ориентированного программирования разработан текстовый язык и инструментальное средство для автоматного программирования. Для автоматов, написанных на этом языке, при необходимости обеспечивается автоматическое построение диаграммы состояний, а не наоборот, что важно для профессиональных программистов, привыкших к текстовому представлению программ.
2. Разработаны средства валидации автоматов для среды языково-ориентированного программирования.

3. Разработан акторный язык и инструментальное средство для многопоточного программирования, и выполнена их интеграция с языком и инструментальным средством для автоматного программирования. Таким образом, обеспечена поддержка многопоточного автоматного программирования.
4. Результаты работы внедрены при создании в среде языково-ориентированного программирования *MPS* коммерческой системы учета программных дефектов *YouTrack* в компании *JetBrains* (Санкт-Петербург).

Все полученные результаты обладают новизной, так как среда языково-ориентированного программирования *MPS* построена на новых принципах создания языков программирования и является первой в мире промышленной системой этого класса.

Статьи в журналах из перечня ВАК

1. *Гуров В. С., Мазин М. А., Шалыто А. А.* UniMod – инструментальное средство для автоматного программирования // Научно-технический вестник СПбГУ ИТМО. 2006. Вып. 30, с. 32 – 44.
2. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* Инструментальное средство для поддержки автоматного программирования // Программирование. 2007. № 6, с. 65 – 80.
3. *Гуров В. С., Мазин М. А., Шалыто А. А.* Автоматическое завершение ввода условий в диаграммах состояний //Информационно-управляющие системы, 2008. № 1, с. 24 – 33.
4. *Гуров В. С., Мазин М. А., Шалыто А. А.* Текстовый язык автоматного программирования //Научно-технический вестник СПбГУ ИТМО. 2008. Вып. 53, с. 258 – 263.
5. *Мазин М. А., Жукова А. Р.* Акторное расширение языка Java в среде MPS //Научно-технический вестник СПбГУ ИТМО. 2010. Вып. 66, с. 72 – 77.

Другие публикации

6. *Мазин М. А., Шалыто А. А.* Анимация. Flash-технология. Автоматы // Компьютерные инструменты в образовании. 2003. № 4, с. 39 – 47.
7. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse //Информационно-управляющие системы. 2004. № 6, с. 12 – 17.
8. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* Разработка средств автоматизации построения объектно-ориентированных программ с явным выделением состояний //Научно-технический вестник СПбГУ ИТМО. 2004. Вып. 16. с. 88 – 100.
9. *Мазин М. А., Шалыто А. А.* Macromedia Flash и автоматы //Мир ПК. Диск. 2004. № 2.
10. *Мазин М. А., Шалыто А. А.* Преступники и автоматы //Мир ПК. 2004. № 9, с. 82 – 84.
11. *Мазин М. А., Шалыто. А. А.* Анимация. Flash-технология. Автоматы //Информатика. 2006. № 11, с. 36 – 47.
12. *Мазин М. А., Парфенов В. Г., Шалыто А. А.* Автоматная реализация интерактивных сценариев образовательной анимации /Труды X Всероссийской научно-методической конференции «Телематика-2003». СПбГИТМО (ТУ). 2003, с. 177, 178.
13. *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». СПбГУ ИТМО. 2004, с. 189 – 191.
14. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* Исполняемый UML. Проект UniMod /Software Engineering Conference (Russia). М.: Руссофт. 2005.
15. *Gurov V. S., Mazin M. A., Narvsky A. S, Shalyto A. A.* UniMod: Method and Development of Reactive Object-Oriented Programs with Explicit States Emphasis /Proceedings 2005 of St. Petersburg IEEE Chapters. International Conference «110 Anniversary of Radio Invention». SPb ETU «LETI». 2005, pp. 106 – 110.
16. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* UniMod: Метод и средство разработки реактивных объектно-ориентированных программ с явным выделением состояний /Труды Второй Всероссийской научной конференции «Методы и средства обработки информации». МГУ. 2005, с. 361 – 366.
17. *Гуров В. С., Мазин М. А.* Создание системы автоматического завершения ввода с использованием пакета UniMod /Вестник II межвузовской конференции молодых ученых. Т.1. СПбГУ ИТМО. 2005, с. 73 – 87.
18. *Гуров В. С., Мазин М. А., Шалыто А. А.* Операционная семантика UML-диаграмм состояний в программном пакете UniMod /Труды XII Всероссийской научно-методической конференции «Телематика-2005». СПбГУ ИТМО. 2005, с. 74 – 76.
19. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* Проект с открытым кодом UniMod – инструментальное средство для автоматного программирования на платформе Eclipse /Open Source Forum. М.: Руссофт. 2005.
20. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.*Инструментальное средство для поддержки автоматного программирования /Перспективы систем информатики. Рабочий семинар «Наукоемкое программное обеспечение». Новосибирск: Институт систем информатики им. А. П. Ершова. 2006, с. 52 – 54.
21. *Гуров В. С., Мазин М. А., Шалыто А. А.* Текстовый язык автоматного программирования //Научно-технический вестник СПбГУ ИТМО. 2007. Вып. 42, с. 29 – 32.
22. *Гуров В. С., Мазин М. А., Зубок Д. А., Парфенов В. Г., Шалыто А. А.* Два подхода к созданию программ с использованием инструментального средства UniMod /Труды XIV Всероссийской научно-методической конференции «Телематика-2007». Т.2. СПбГУ ИТМО. 2007, с. 428.
23. *Гуров В. С., Мазин М. А., Шалыто А. А.* Текстовый язык для автоматного программирования / Труды XIV Всероссийской научно-методической конференции «Телематика-2007». Т.2. СПбГУ ИТМО. 2007, с. 424, 425.
24. *Гуров В. С., Мазин М. А., Шалыто А. А.* Текстовый язык автоматного программирования /Труды второй международной научной конференции «Компьютерные науки и информационные технологии». Саратов: СГУ. 2007, с. 33 – 35.

Личный вклад. В работах, выполненных в соавторстве, личный вклад автора в равных долях с соавторами.

Тиражирование и брошюровка выполнены
в центре «Университетские телекоммуникации»
Санкт-Петербург, Саблинская ул. 14, тел. (812)233-46-69.
Объем 1,0 у.п.л. Тираж 100 экз.