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

eVelopers Corporation (http://www.evelopers.com), Санкт-Петербург
Санкт-Петербургский государственный университет информационных технологий, механики и оптики
vgurov@evelopers.com, shalyto@mail.ifmo.ru

В промышленном программировании обычно выделяют следующие стадии течения проекта: сбор требований, технический дизайн, написание кода, тестирование. Технический дизайн заключается в создании на основе собранных требований технического задания, описывающего модель будущей системы с помощью диаграмм, псевдокода и поясняющего текста. На стадии написания кода система реализуется для целевой программно-аппаратной платформы в соответствии с техническим заданием. На стадии тестирования проверяется соответствие реализации и требований.

Семантический разрыв передачи знаний между стадиями технического дизайна и написания кода заключается в том, что разработчик реализует систему в соответствии со своим пониманием технического задания.

Признание многими ведущими фирмами в области разработки программного обеспечения этого факта привело к появлению такого направления в программной инженерии как «проектирование на базе моделей» (*Model-Driven Architecture*) [1,2]. Основной идеей этого подхода является независимое рассмотрение моделей, создаваемых при проектировании системы, от деталей их реализации на конкретной программно-аппаратной платформе и последующий автоматизированный переход к реализации. Проектирование на базе моделей должно привести к появлению следующего поколения высокоуровневых универсальных языков программирования, в том числе графических.

Одним из возможных графических языков программирования, ориентированным на «проектирование на базе моделей», является *Unified Modeling Language* (*UML)* [3] и поэтому в последнее время идея *запускаемого UML* [4] приобретает все большую популярность. Отметим, что практическое использование *UML*, в большинстве случаев, ограничивается моделированием статической части программы с помощью диаграммы классов. Моделирование динамических аспектов программы на языке *UML* затруднено в связи с отсутствием в стандарте формального однозначного описания правил интерпретации поведенческих диаграмм, что вызвало появление большого числа проектов устраняющих указанный пробел [5-7].

Проект *UniMod* (http://unimod.sf.net) предлагает свой вариант наделения *UML* моделей операционной семантикой, предлагая собственный метод проектирования реактивных объектно-ориентированных программ с явным выделением состояний. Также в рамках проекта разрабатывается инструментальное средство для поддержки стадии проектирования, выполняемой в соответствии с упомянутым методом.

Предлагаемый метод основан на подходе к проектированию событийных объектно-ориентированных программ с явным выделением состояний, предложенным в [8] и названным «*SWITCH*-технологией» или «автоматно-ориентированным программированием». Особенность этого подхода состоит в том, что поведение в таких программах описывается с помощью графов переходов структурных конечных автоматов с нотацией, предложенной в работе [9].

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, а графы переходов строятся с помощью нотации *UML* диаграммы состояний.

Предлагаемый процесс моделирования системы состоит в следующем:

- на основе анализа предметной области разрабатывается концептуальная модель системы в виде диаграммы классов;
- в отличие от традиционных для объектно-ориентированного программирования подходов [10], из числа классов 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 связываются с объектами, которыми они управляют;
- 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.

Для построенной модели системы задается следующая операционная семантика:

- 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, 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 у состояния, в которое вложено текущее состояние;
- при переходе в конечное состояние все источники событий останавливаются. После этого работа системы завершается.

Инструментальное средство для поддержки описанного процесса является встраиваемым модулем (*plug-in*) для платформы *Eclipse* (http://www.eclipse.org) и обладает следующими возможностями:

- интерактивная проверка корректности создаваемой модели.
- подсветка семантических и синтаксических ошибок, автоматическое исправление ошибок;
- автоматическое завершение ввода и автоматическое исправление ошибок;
- интерпретационный и компиляционный подходы для запуска модели;
- запуск, локальная и удаленная отладка модели внутри среды разработки;

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 всегда.

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. 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.

При компиляционном подходе *XML*-описание модели преобразуется в код на целевом языке программирования, который компилируется и запускается уже без помощи интерпретатора.

Использование проекта *UniMod* позволяет:

- локализовать логические ошибки на стадии технического дизайна;
- 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;
- формально и наглядно описывать поведение программ и модифицировать его, изменяя, в большинстве случаев, только графы переходов;
- упростить сопровождение проектов вследствие повышения централизации логики программ.

Описанные метод и инструментальное средство *UniMod* были успешно применены при разработке ряда проектов, включая Интернет-приложения и приложения для мобильных устройств.

Инструментальное средство *UniMod* является свободно распространяемым продуктом с открытым исходным кодом. Исходный код, документация и примеры использования представлены на сайте http://unimod.sourceforge.net.

Литература

1. OMG Model Driven Architecture. http://www.omg.org/mda/
2. Frankel D. Model Driven Architecture: Applying MDA to Enterprise Computing. Willey, 2003. – P. 352.
3. Буч Г., Рамбо Г., Якобсон И. UML. Руководство пользователя. М.: ДМК, 2000. — 358 с.
4. Mellor S. et al. Executable UML: A Foundation for Model Driven Architecture. MA: Addison-Wesley, 2002. – P. 258.
5. Kennedy Carter eXecutable UML (xUML). http://www.kc.com/MDA/xuml.html
6. Nucleus UML Suite. http://www.projtech.com/embedded/nuc_modeling.html
7. I-Logix Rhapsody. http://ilogix.com/rhapsody/rhapsody.cfm
8. Шалыто А.А., Туккель Н.И. Танки и автоматы //BYTE/Россия. 2003. №2, с. 69-73. http://is.ifmo.ru/ (раздел «Статьи»).
9. Шалыто А.А., Туккель Н.И. SWITCH-технология — автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. №5, с. 45-62. http://is.ifmo.ru/ (раздел «Статьи»).
10. Грэхем И. Объектно-ориентированные методы. Принципы и практика. М.: Вильямс, 2004. — 768 с.