Вариант этой статьи опубликован в еженедельнике “PC Week/RE”. 2005. № 26, с. 18, 19.

Исполняемый *UML* из России

Вадим Гуров, Андрей Нарвский, Анатолий Шалыто

В настоящее время унифицированный язык моделирования *UML* [1] применяется в индустрии создания программного обеспечения, в основном, как язык спецификации моделей систем. Существующие *UML*-средства, как правило, предоставляют пользователю только графический редактор для «рисования» диаграмм.

UML-диаграммы делятся на два класса: для описания статики (например, диаграмма классов) и динамики (например, диаграмма состояний). Динамические свойства программ также называются поведением.

Многие из *UML*-средств позволяют автоматически создавать по диаграмме классов «скелет» кода на целевом языке программирования (*Java, C++, C#* и т.д.).

При переходе к моделированию динамики, может быть поставлен вопрос: «правомерно ли утверждать, что инструмент поддерживает моделирование поведения с помощью *UML*, если он позволяет только нарисовать диаграмму состояний и не более того?»

Отвечая на этот вопрос, отметим, что, например, такой известный инструмент как *IBM Rational Rose*, ориентированный на интеграцию с другими программными средствами для обеспечения всего жизненного цикла создания программ, декларируя поддержку *UML* различных версий ничего кроме рисования не умеет делать с упомянутым базовым средством описания поведения — с диаграммой состояний.

При этом отметим, что менее универсальные средства, такие как, например, *Gentleware Poseidon for UML,* обеспечивают автоматическую генерацию кода программы по диаграммам состояний, и только одно известное авторам средство *Telelogic TAU*, кроме генерации кода, также позволяет частично проверять корректность модели и отлаживать ее в терминах диаграмм состояний.

Однако даже эти средства обладают существенным недостатком, который является идеологическим: они, в первую очередь, ориентированы на построение кода, в то время как, по нашему мнению, основное внимание должно уделяться собственно модели, так как при ее интерпретации **код может и вовсе не строиться**. При таком подходе модель, описывающая поведение системы, сама является программой на языке *UML*-диаграмм, содержащей ссылки на методы, которые практически не содержат структур управления и реализуются на целевом языке программирования. Этот подход назван одним из создателей языка *UML* И. Якобсоном [2, 3] ***исполняемый UML*** и реализован в России в инструментальном средстве *UniMod* (http://unimod.sf.net), которому посвящена настоящая статья.

Автоматное программирование

UniMod базируется на трех «китах» — *UML,* описанном выше, автоматном программировании и среде разработки *Eclipse*. Опишем суть автоматного подхода, предложенного в работе [4].

В соответствии с парадигмой автоматного программирования, программы предлагается строить так, как строятся системы управления технологическими процессами, в которых выделяются источники информации, управляющее устройство и объекты управления. При этом в качестве управляющего устройства используется детерминированный конечный автомат. Автомат реагирует на входные воздействия и формирует выходные воздействия, «указывающие» объектам управления, что они должны «делать».

Если автомат управляет одним объектом управления, то число состояний в автомате первоначально может быть выбрано равным числу состояний объекта, а в дальнейшем, возможно, может быть и уменьшено.

Если число состояний в объекте управления огромно, то предлагается выполнить декомпозицию, аналогичную применяемой в машине Тьюринга, когда управляющее устройство с небольшим числом состояний управляет бесконечной лентой [5].

Для каждого автомата предлагается строить четыре типа документов (первые три из которых создаются на этапе автоматного проектирования): словесное описание (декларация о намерениях); диаграмма связей автомата с источниками информации и объектами управления, описывающая его интерфейс; диаграмма состояний автомата, некоторые свойства которой могут быть проверены формально; фрагмент программы, **формально и изоморфно** реализующий диаграмму состояний.

Если в системе присутствует более одного автомата, то дополнительно строится диаграмма взаимодействия автоматов. Одним из способов их взаимодействия является вложенность автоматов. Правильность взаимодействия автоматов проверяется в результате анализа протоколов их работы. Протоколирование выполняется автоматически.

Особенность предлагаемого подхода, позволяющая компактно описывать сложное поведение, связана с использованием символьных обозначений на диаграммах состояний. При этом имена на этих диаграммах выбираются в соответствии с очень жесткой дисциплиной их формирования. Она кажется непривычной и даже архаичной, но у нее есть важные для визуального конструирования достоинства.

Во-первых, имена получаются короткими, и поэтому даже сложные выражения легко размещаются, например, на дугах, по которым выполняются переходы в диаграммах состояний.

Во-вторых, имена типизированы на лексическом уровне, что очень важно для читаемости диаграмм.

Такой стиль формирования имен противоречит общепринятому правилу обязательного использования длинных содержательных идентификаторов (построение самодокументирующихся диаграмм). При применении предлагаемого подхода смысл каждого символа изложен на диаграмме связей в виде текстового описания на естественном языке **без сокращений и аббревиатур**. Текстовые описания пишутся **на горизонтальных линиях** диаграмм связей, а **не на дугах** (кривых линиях) диаграмм состояний, как это делается обычно.

Это позволяет реализовывать диаграммы состояний со сложной логикой на переходах и большим числом действий.

Есть основания предполагать, что автоматный подход, как стиль программирования [6], получит широкое распространение не только при процедурной, но и при объектно-ориентированной реализациях.

Это предположение основано на том, что если объектно-ориентированное программирование получило широкое распространение, так как базируется на таких понятных человеку абстракциях как «класс» и «объект», то автоматное программирование основано на такой характерной для человека абстракции как «состояние» (например, «состояние души»). Эта абстракция при совместном использовании с такими абстракциями как «входное воздействие» и «выходное воздействие», образует понятие «автомат». При этом отметим, что под входным воздействием понимается либо «входная переменная», либо также интуитивно понятное человеку понятие — «событие».

Совместное применение понятий «класс», «объект» и «автомат» (последнее объединяет понятия «состояние» и «событие») приводят к стилю программирования, названному в работе [4] «объектно-ориентированным программированием с явным выделением состояний».

Большое число студенческих проектов, выполненных на основе автоматного программирования, приведено на сайте http://is.ifmo.ru. Каждый из этих проектов включает проектную документацию, содержащую указанные выше артефакты [7].

Из изложенного следует, что автоматное программирование реализует популярную сегодня концепцию разработки программ на базе моделей (Model Driven Architecture – MDA) [8].

Платформа *Eclipse*

Относительно третьего «кита» отметим, что сегодня многие ведущие компании мира начинают переводить свои программные продукты на платформу *Eclipse* (http://www.eclipse.org), основы которой были заложены корпорацией *IBM*. Платформа *Eclipse* является «средством разработки средств разработки». Она поставляется бесплатно с открытым исходным кодом.

Более подробно с платформой *Eclipse* и плагинами (*plug-in*) к ней можно ознакомиться в работе [9].

Проект *UniMod*

Для поддержки автоматного программирования создан проект *UniMod* (http://unimod.sf.net). Он реализован на языке *Java* в виде плагина к среде разработки *Eclipse* и использует нотацию *UML*.

Первоначальное представление о проекте *UniMod* можно получить по динамическим демонстрационным материалам, доступным по ссылкам http://unimod.sf.net/viewlet/animated-demo-rus.html и http://unimod.sf.net/viewlet/debugger-demo-rus.html.

Концептуально модель приложения при использовании *UniMod* состоит всего из двух типов диаграмм:

- диаграммы связей — упрощенной *UML*-диаграммы классов, представляемой в форме структурной схемы системы управления, содержащей источники

событий, управляющее устройство (автомат) и объекты управления. Отличие этой диаграммы классов от традиционной не только в форме, но и в содержании: в ней наряду со «статическими» классами обязательно присутствует явно выделенный класс, определяющий поведение системы;

- *UML*-диаграммы состояний, описывающей поведение автоматов. Формальные свойства диаграмм этого типа, такие, как полнота и непротиворечивость, проверяются автоматически в интерактивном режиме. В этом же режиме может выполняться графическая отладка диаграмм. *UniMod* позволяет как интерпретировать диаграммы состояний, так и строить по ним код, например, для мобильных устройств.

Эти диаграммы строятся как при создании приложения «с нуля», так и при рефакторинге.

Во втором случае при помощи автоматизированных средств или вручную строится традиционная диаграмма классов приложения. Для каждого класса на этой диаграмме решается вопрос, является ли он источником событий или объектом управления или и тем и другим вместе. После этого строится диаграмма связей, которая в качестве объектов содержит выделенные из указанной диаграммы классов источники событий и объекты управления, соединяемые через дополнительно вводимый объект — автомат.

Если автомат весьма сложен, то его целесообразно декомпозировать. При этом диаграмма связей является также и диаграммой взаимодействия автоматов, для каждого из которых строится своя диаграмма состояний. В вершины диаграмм состояний могут быть вложены другие автоматы, что расширяет *UML*, в котором допустимы лишь вложенные состояния.

Модель приложения в общем случае может содержать произвольное количество диаграмм названных типов.

В диаграммах используются символические имена для обозначения событий (переменные с параметрами) и методов — входных и выходных воздействий. При этом события формируются источниками событий, а входные и выходные воздействия выполняются в объектах управления. Источники событий и объекты управления **реализуются вручную на языке *Java*** программистом, а диаграммы создаются архитектором и выполняются автоматически, что соответствует известному принципу «разделяй и властвуй».

Авторами выполнен ряд проектов с применением предлагаемого средства, которое показало свою эффективность. Проект является бесплатным и открытым и размещен на сайте *Sourceforge* (http://sf.net), что способствует его распространению и внедрению [9-12].

Из изложенного следует, что для реализации идеи исполняемого *UML* не обязательно поддерживать весь или почти весь [13] спектр его диаграмм. Например, *UniMod* поддерживает всего два типа диаграмм, причем диаграмма связей в *UniMod* является весьма узким, и к тому же специализированным подмножеством диаграммы классов *UML*.

UniMod является средством для визуального конструирования программ, которое по мнению, приведенному в одном из отзывов на это средство [10], «может стать новой технологией программирования», а исполняемый *UML* – новым языком программирования.

На конференции «Russian Outsourcing & Software Summit» (St. Petersburg, 2005) представитель корпорации *Borland* Сергей Орлик привел результаты опроса бизнес-аналитиков, архитекторов и программистов по вопросу о том, какие типы *UML*-диаграмм ими используются. Было установлено, что архитекторы применяют значительно большую

номенклатуру диаграмм по сравнению с программистами. При этом, в частности, если первые в своей работе используют диаграммы состояний, то вторые их даже не упоминают.

При таком подходе о качественной реализации поведения программ говорить не приходится, так как программисты в лучшем случае «подсматривают» в диаграммы состояний. При внесении изменений это резко затрудняет синхронизацию указанных диаграмм и кода, реализующего поведение, что, в конце концов, приводит к тому, что диаграммы не соответствуют коду и их нецелесообразно включать в документацию.

Для инструментального средства *UniMod* **применение** диаграмм состояний на всех этапах процесса разработки **неизбежно**, так как они являются и спецификацией, и программой, и проектной документацией одновременно.

UniMod достиг той степени зрелости, что позволяет проводить разработку его новых версий с помощью самого себя.

UniMod воплощает утверждение Б. Страуструпа о том, что «генерирующая система должна быть спроектирована так, чтобы сочетать достоинства спецификаций высокого уровня и языка программирования высокого уровня» [14]. Это удается осуществить в рассматриваемом инструментальном средстве за счет использования автоматов для описания поведения, имеющих «крепкий теоретический фундамент», что позволяет «сгенерировать код по спецификации высокого уровня». Таким образом, «прием, хорошо работающий в **специфической** области конечных автоматов», удается перенести в область проектирования поведения **универсальных** программ.

В интервью с П. Кузаном, отвечающим за продукты линейки *Together* корпорации *Borland* [15], сказано, что бесплатная версия продукта *Together Designer Community Edition* позволяет только рисовать модели. Бесплатный *UniMod* является полнофункциональным.

У описываемого проекта есть не только сторонники [10]. Так, некоторые специалисты утверждают, что предлагаемое средство не обеспечивает масштабируемости программ.

Во-первых, это не так, а во-вторых, это звучит весьма странно из уст людей, которые не могут противопоставить какое-либо другое открытое инструментальное средство, реализующее исполняемый *UML*.

Даже если наши оппоненты были бы и правы, то, все равно, как говорил В.И. Ленин, «не следует браться за решение больших задач, не научившись решать малые». Тот, для кого Ленин теперь не авторитет, должен помнить, что «когда человек хочет передвинуть гору, он начинает с того, что убирает мелкие камни», а не наоборот.

Открытая проектная документация

В заключении отметим, что разработка на базе моделей, и, в частности, применение инструментального средства *UniMod*, изменяют взгляд на существующую практику сокрытия исходного кода и разработку проектной документации.

По мнению известного специалиста в области дискретной математики Ф.А. Новикова [16] «защита интеллектуальной собственности разработчиков путем сокрытия исходных кодов программ — это укоренившийся, но общественно вредный анахронизм. Практика сокрытия исходного кода невыгодна самим разработчикам.

Действительно, сокрытие исходного кода затрудняет его повторное использование, и упущенная выгода намного перевешивает ущерб от злонамеренного нарушения авторских прав. При разработке, ориентированной на модели, сокрытие моделей — нонсенс, противоречащий самой сути подхода.

Общественные движения, направленные на расширение публичного доступа к информации, экономически обоснованы, и поэтому победа будет за ними.

В этой связи, по нашему мнению, особого внимания заслуживает „Движение за открытую проектную документацию“, поскольку оно в наибольшей степени соответствует духу модельно-ориентированной разработки программ [7]».

При использовании предлагаемого инструментального средства вопрос о качественной проектной документации, по которому была дискуссия в еженедельнике *PC Week/RE* [17], решается сам собой.

Источники

1. *Буч Г., Рамбо Г., Якобсон И.* UML. Руководство пользователя. М.: ДМК, 2000, 358 с.
2. *Mellor S.*, *Balcer M.* Executable UML. A Foundation for Model-Driven Architecture. MA: Addison-Wesley. 2002. 368 p.
3. *Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А.* UML. SWITCH-технология. Eclipse //Информационно-управляющие системы. 2004. № 6, с. 12-17, http://is.ifmo.ru/works/uml-switch-eclipse/
4. *Шалыто А.А.* Технология автоматного программирования //Мир ПК. 2003. № 10, с. 74-78, http://is.ifmo.ru/works/tech_aut_prog/
5. *Шалыто А.А., Туккель Н.И.* От тьюрингова программирования к автоматному //Мир ПК. 2002. № 2, с. 144-149, http://is.ifmo.ru/works/turing/
6. *Непейвода Н.Н.* Стили и методы программирования. М.: Интернет-Университет Информационных технологий, 2005, 316 с.
7. *Шалыто А.А.* Новая инициатива в программировании. Движение за открытую проектную документацию //PC Week/RE. 2003. № 40, с. 38, 39, 42, http://is.ifmo.ru/works/open_doc/
8. *Кузнецов И.* Архитектура, управляемая моделью //IT News. 2005. № 11, с. 28-29.
9. *Платформа Eclipse* //Мир ПК — Диск. 2005. № 6.
10. *UniMod User Reports*. http://is.ifmo.ru/unimod_en/_unimoduser.pdf
11. *O'Reilly CodeZoo. UniMod*. http://www.codezoo.com/pub/component/260?category=89
12. *Wikipedia. Finite state machine. Tools.* http://en.wikipedia.org/wiki/Finite_automaton
13. *Гома Х.* UML. Проектирование систем реального времени, параллельных и распределенных приложений. М.: ДМК, 2002, 435 с.
14. *Страуструп Б.* Язык программирования C++. М.: Бином, СПб.: Невский диалект, 2001, 1098 с.
15. *Елманова Н.* Borland Together 2005 for Visual Studio .NET //Компьютер пресс. 2005. № 6, с.159-162.
16. *Новиков Ф.А.* Дискретная математика для программистов. СПб.: Питер, 2003. 460 с.
17. *Колесов А.* Открытая проектная документация? А может быть, лучше качественная! //PC Week. 2003. № 44, с. 56.

Об авторах: ***Гуров Вадим Сергеевич****, магистр математики, ведущий разработчик в компании eVelopers Corp., адрес: vgurov@evelopers.com;* ***Нарвский Андрей Сергеевич****,*

канд. техн. наук, директор компании eVelopers Corp., адрес: anarvsky@evelopers.com;
***Шалыто Анатолий Абрамович**, докт. техн. наук, профессор, заведующий кафедрой «Технологии программирования» СПбГУ ИТМО, адрес: shalyto@mail.ifmo.ru*