Опубликовано в Научно-техническом вестнике СПбГУ ИТМО. 2008. Выпуск 53. Автоматное программирование, с. 251-258.

УДК 004.4.'232

ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО ДЛЯ ПОДДЕРЖКИ АВТОМАТНОГО ПРОГРАММИРОВАНИЯ *UNIMOD 2*. ПРОЕКТИРОВАНИЕ. ВАЛИДАЦИЯ. ВЕРИФИКАЦИЯ. РЕАЛИЗАЦИЯ

Д.Ю. Кочелаев, И.А. Лагунов, Б.З. Хасянзянов, Б.Р. Яминов
(Санкт-Петербургский государственный университет информационных технологий, механики и оптики)

В данной статье дается краткий обзор инструментального средства *UniMod 2* для поддержки автоматного программирования. Данное инструментальное средство предназначено для разработки автоматных программ и предоставляет средства для визуального проектирования, отладки, валидации и верификации автоматных программ.
Ключевые слова: автоматное программирование, инструментальное средство, валидация, верификация

Введение

В 2005 г. было разработано инструментальное средство *UniMod* [1], позволяющее визуально конструировать автоматные диаграммы и исполнять программы, созданные на их основе. Это средство базируется на следующих основных концепциях:

- *UML* [2, 3];
- автоматное программирование [4];
- платформа *Eclipse* [5];
- язык программирования *Java;*
- открытый программный код (исходные коды доступны для скачивания на сайте проекта [1]).

Инструментальное средство *UniMod* является одной из реализаций «исполняемого *UML*» [6]. При использовании автоматного подхода программа в целом проектируется с помощью двух типов *UML*-диаграмм: диаграмм классов, которые представляются в виде схем связей автоматизированных объектов [7], и диаграмм состояний, которые реализуют автоматы, указанные на диаграмме классов. Эти диаграммы исполняются автоматически, а фрагменты программ, соответствующие входным и выходным воздействиям, пишутся вручную. Таким образом, это инструментальное средство позволяет при создании программ совмещать разные уровни абстракции (диаграммы и текст на языке *Java*) и разные стили программирования (графический и текстовый).

Перечислим достоинства инструментального средства *UniMod*.

1. Возможность визуального построения диаграмм классов и состояний. Средство обеспечивает тесную интеграцию между визуальным представлением и реализацией представленных объектов в коде.
2. Возможность не только интерпретировать и компилировать построенную программу, но и осуществлять визуальную отладку, добавляя точки останова на состояния и переходы. Данная возможность совместно с автоматным представлением программы существенно облегчает поиск ошибок.
3. Возможность валидации построенных автоматных моделей. Это дает дополнительную информацию об ошибках, которые были допущены при построении диаграмм состояний.

Однако инструментальное средство *UniMod* имеет и некоторые недостатки:

- валидация осуществляется с применением неоптимального алгоритма, и ее реализация «зашита» в код инструментального средства;
- отсутствует возможность верификации программ;
- отсутствует возможность текстового ввода автоматных программ.

Разработка инструментального средства *UniMod 2*

В 2007 г. была начата работа над второй версией инструментального средства *UniMod*, названной *UniMod 2*. Цель этого проекта – устранить имеющиеся в средстве *UniMod* недостатки, усовершенствовать некоторые из его компонент, а также ввести новую функциональность, такую как верификация программ и обеспечение текстового ввода автоматных программ.

Модель данных

Одно из основных отличий инструментального средства *UniMod 2* от прототипа состоит в широком использовании ресурсов, предоставляемых платформой *Eclipse*. Это позволяет улучшить все компоненты системы. В частности, для описания автоматной метамодели применен фреймворк *EMF* (*Eclipse Modeling Framework*) [8]. Этот фреймворк предназначен для построения приложений, основанных на модельных структурах данных. Он позволяет генерировать исходный код на языке программирования *Java* для составляющих модели, а также адаптеров для просмотра и редактирования модели. Использование этого фреймворка позволяет избежать возможных при реализации функциональность вручную ошибок. Редактирование модели (например, внесение новых атрибутов у составляющих модели) выполняется визуально и не требует ручной модификации кода, так как он будет сгенерирован в дальнейшем автоматически.

Использование фреймворка *EMF* позволяет использовать уже существующие фреймворки и библиотеки для реализации других частей приложения, таких как визуализация или валидация. При этом число строк кода, который написан вручную, значительно сокращается, что ведет к снижению вероятности появления ошибки в конечном программном продукте.

Отметим также, что использование *EMF* не ограничивает в реализации модельных классов, так как сгенерированный код может быть доработан вручную. При этом изменения, внесенные в реализацию вручную, будут сохранены и при автоматической перегенерации классов из измененной модели.

Валидация

В инструментальном средстве *UniMod 2* используется новый алгоритм валидации автоматов [9]. Этот алгоритм (проверка непротиворечивости построенной модели и автоматной метамодели) является более быстродействующим, чем алгоритм, используемый в первой версии инструментального средства. Повышение производительности достигается за счет использования предварительных вычислений, а также контекстнозависимой проверки правил – правила проверяются только в контексте измененного элемента. Такой подход существенно сокращает объем вычислений, которые необходимо выполнить для проверки непротиворечивости модели, а, следовательно, увеличивает скорость этой проверки.

Рассмотрим подробнее два этапа алгоритма проверки непротиворечивости модели.

1. *Этап предварительных вычислений.* Перед началом работы с моделью собираются данные, необходимые для быстрого определения правил, которые могли нарушиться в результате изменения модели. Таким образом, для каждого правила определяется

набор атрибутов, изменение которых может повлечь за собой изменение состояния правила – определяется, выполняется правило или нет.

2. *Этап динамической проверки.* При каждом изменении модели, на основе изменного атрибута и данных, которые были получены на этапе предварительных вычислений, составляются множества правил, которые необходимо перепроверить, и выполняется их проверка. Отметим, что проверка выполняется в контексте того элемента, атрибут которого был изменен, что существенно сокращает число проверяемых правил, по сравнению с оригинальным алгоритмом.

Для записи правил, описывающих ограничения на модель, используется *Object Constraint Language* (*OCL*) [10]. Этот язык разработан компанией *IBM* специально для записи ограничений на *UML*-модели [2, 3]. Он используется при решении задачи валидации. Для иллюстрации такого способа записи ограничений рассмотрим ограничение на отсутствие переходов в начальное состояние, записанное на языке *OCL*:

```
context coremodel::State inv:
     self.substates->select(x | x.initial).
          incomingTransitions->size() = 0
```

Любое правило начинается с задания контекста, в котором необходимо осуществлять проверку. Далее следует условие, которое необходимо проверить. В данном случае у автомата выбираются все вложенные в него состояния, а для каждого начального состояния осуществляется проверка на отсутствие переходов в это состояние. Такая запись позволяет легко определить контекст, в котором необходимо проверять правило, а также атрибуты, изменение которых может повлечь за собой изменение состояния правила.

Для оценки производительности применяемого алгоритма выполнено теоретическое и практическое сравнение этого алгоритма и алгоритма, который был использован в инструментальном средстве *UniMod*. В результате показано, что предложенный алгоритм обладает существенно более высокой производительностью [9].

Верификация

Нововведением в инструментальном средстве *UniMod 2* стала возможность верификации программ [11]. Для этого указанное средство было интегрировано с верификатором *Bogor* [12] за счет создания ряда дополнительных модулей. В результате появилась возможность верифицировать автоматные программы на основе метода *Model Checking* [13].

В известных подходах для верификации автоматной программы с помощью метода *Model Checking* необходимо построить специальную формальную модель, которая будет представлять программу. Утверждения, которые необходимо верифицировать, также следует перевести из терминов программы в термины построенной модели. Поэтому существовало два схожих способа верификации программы. Первый [14] – преобразовать автоматную программу в модель *Крипке* [13], а затем верифицировать ее вручную с помощью метода *Model Checking*. Второй способ [15] – преобразовать автоматную программу во входной язык верификатора, а затем запустить верификатор для проверки корректности модели. Если автоматная программа некорректна, то верификатор возвращает отчет об ошибке. Однако отчет об ошибке возвращается в терминах преобразованной формальной модели, а не исходной автоматной программы. Таким образом, пользователю приходится преобразовывать отчет об ошибке обратно в термины исходной автоматной программы, что в некоторых случаях может быть сложной задачей.

Рис. 1 описывает известные методы верификации автоматных программ.

Преимущество метода верификации, который реализован в инструментальном средстве *Unimod 2*, состоит в том, что верификатор *Bogor* работает непосредственно с автоматной программой, как если бы она уже была формальной моделью, готовой к ве-

рификации. В результате отсутствует необходимость в сложном преобразовании автоматной программы в формальную модель и обратно. Поэтому процесс верификации становится значительно проще, быстрее и надежнее. Схема этого метода верификации приведена на рис. 2.

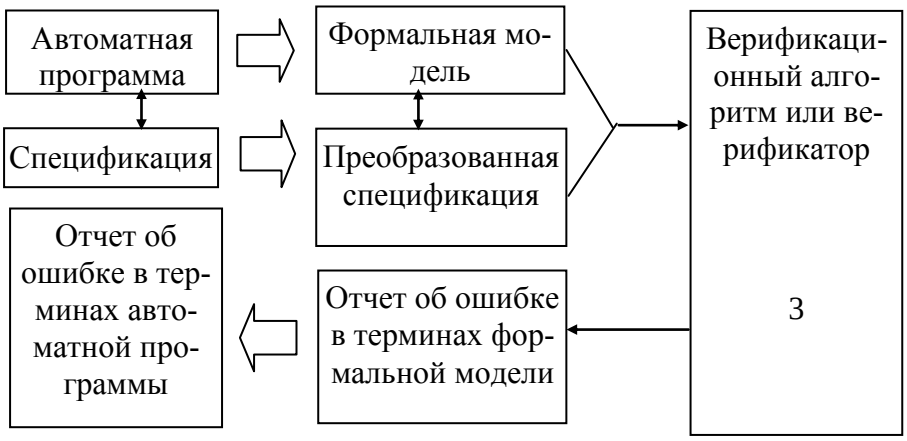

Рис. 1. Существующие методы верификации. Необходимы четыре шага: (1) преобразовать программу в формальную модель, (2) преобразовать спецификацию в спецификацию в терминах формальной модели, (3) верифицировать модель и (4) преобразовать отчет об ошибке обратно в термины автоматной программы

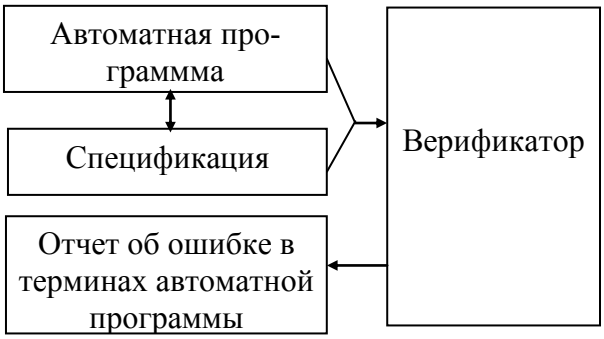

Рис. 2. Новый метод верификации — без преобразования программы

Таким образом, *UniMod 2* – это первое инструментальное средство, которое позволяет создавать автоматные программы, запускать и *автоматически* верифицировать их. Для проверки корректности программы пользователю достаточно ввести список утверждений, которые должны быть верны для программы, и запустить процесс верификации. По окончании процесса верификации пользователю сообщается, какие утверждения верны, а какие нет. Более того, если какое-то из утверждений нарушается, пользователь получает детальную историю шагов, выполненных программой, которые привели к нарушению утверждения. Это позволяет пользователю легко найти причину ошибки. Выделяются два основных преимущества верификации программ перед тестированием.

1. Верификация позволяет выявить причину ошибки, а не только саму ошибку.
2. При верификации проверяются все варианты работы программы, в то время как при тестировании проверяются только некоторые из них.

Отладка

В инструментальном средстве *UniMod 2* построение отладчиков для предметно-ориентированных языков программирования выполняется на основе технологии *EMF*

[16]. Предлагаемый метод позволяет реализовать универсальную систему отладки для произвольного предметно-ориентированного языка, которая реализует стандартные средства отладки, предусмотренные средой разработки *Eclipse*. Она включает:

- точки останова (*breakpoints*), позволяющие приостановить выполнение приложения в указанном месте;
- пошаговое выполнение;
- просмотр значений переменных в конкретный момент исполнения приложения (*context variables*).

Визуализация

Для построения редактора моделей, используемого для отображения и управления процессом отладки, была использована технология *Graphical Modeling Framework* (*GMF*) [17]. Эта технология позволяет по описанию метамодели и дополнительных конфигураций построить полнофункциональный графический редактор, встраивающийся как расширение в среду разработки *Eclipse*. Конфигурации задают список элементов метамодели, которые будут использованы на диаграмме, а также их графическое представление и наборы связанных с ними элементов управления редактора.

Также предусмотрена возможность выделения некоторых элементов модели в редакторе [16]. Эта возможность используется для отображения нарушенных ограничений, которые были выявлены в процессе валидации [9]. Этот же механизм будет применяться и для визуализации результатов верификации. Таким образом, разработан и внедрен единый механизм визуализации действий над моделью (отладка), изменений модели и выделения ее элементов (валидация).

Текстовый язык автоматного программирования

Визуальный подход к разработке автоматных программ обладает существенным недостатком – процесс визуального построения диаграмм более трудоемок по сравнению с использованием традиционных текстовых языков программирования. Для решения этой проблемы был разработан текстовый язык автоматного программирования *FSML* (*Finite State Machine Language*) [18], а также реализован редактор этого языка для инструментального средства *UniMod*.

Выделим особенности языка *FSML* и определяемые им особенности инструментального средства *UniMod*.

- Текстово-визуальный подход к разработке автоматных программ позволяет совместно использовать текстовый редактор языка *FSML* и графический редактор диаграмм *UniMod* для создания и модификации автоматных моделей.
- Явное задание графа переходов гарантирует его изоморфность программе. Это позволяет программисту избежать многих ошибок на этапе кодирования.
- Интеграция кода на языке *FSML* с объектно-ориентированным кодом обеспечивается автоматической генерацией автоматной модели по *FSML*-программе и использованием объектов управления. При этом отсутствует необходимость в автоматически сгенерированном коде на языке общего назначения.
- Переиспользование компонентов кода возможно благодаря поддержке вложенных автоматов и интеграции с объектно-ориентированным кодом через объекты управления. Вложенные автоматы позволяют использовать повторно элементы логики, а объекты управления – элементы поведения.
- Краткость и понятность синтаксиса языка обусловливают эффективность его использования в целом.

Редактор языка *FSML* связывает этот язык с инструментальным средством *UniMod*. При этом редактор является эффективным средством разработки автоматных программ. Рассмотрим реализованные в нем функции.

- Построение автоматной модели по коду программы является основной функцией редактора. Для этого используются лексический и синтаксический анализаторы, выполняющие разбор программы. Поскольку эти средства естественно представляются в виде событийной системы, они были реализованы на основе автоматного подхода с использованием инструментального средства *UniMod*. При этом лексический анализатор является поставщиком событий-лексем, а синтаксический анализатор – автоматом, управляющим процессом разбора *fsml*-программы.
- Валидация программы проверяет автоматную программу на наличие ошибок синтаксиса и подсвечивает их.
- Автоматическое завершение ввода пользователя позволяет значительно ускорить процесс разработки автоматных программ, что дает дополнительное преимущество текстовому подходу по сравнению с визуальным [19].

Заключение

В результате разработки инструментального средства *UniMod 2* были усовершенствованы или добавлены следующие компоненты системы:

- изменено представление автоматной модели для обеспечения удобства работы с ней;
- усовершенствован алгоритм валидации программ;
- обеспечена возможность верификации автоматных программ;
- перепроектирован редактор модели для интеграции в него удобных инструментов редактирования, предоставляемых технологией *GMF*;
- введен отладчик программ, написанных на предметно-ориентированных языках программирования, адаптированный для отладки автоматных моделей;
- создана единая система визуализация отладки и валидации;
- разработан текстовый язык автоматного программирования *FSML* и создан редактор для него, интегрируемый с инструментальным средством *UniMod* [18].

Литература

1. Гуров В.С., Мазин М.А.. Нарвский А.С., Шалыто А.А. Инструментальное средство для поддержки автоматного программирования // Программирование. – 2007. – № 6. – С. 65–80. – Режим доступа: http://is.ifmo.ru/works/_2008_01_27_gurov.pdf
2. Гома Х. UML. Проектирование систем реального времени, распределенных и параллельных приложений. – М.: ДМК, 2002.
3. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. – М.: ДМК, 2000.
4. Шалыто А.А. Switch-технология. Алгоритмизация и программирование задач логического управления. – СПб: Наука, 1998. – Режим доступа: http://is.ifmo.ru/books/switch/6
5. Веб-сайт проекта Eclipse. – Режим доступа: http://www.eclipse.org/
6. Mellor S., Balcer M. Executable UML: A Foundation for Model-Driven Architecture. – Addison-Wesley, 2002.
7. Поликарпова Н.И., Шалыто А.А. Учебно-методическое пособие по дисциплине «Автоматное программирование». – СПбГУ ИТМО, 2007. – Режим доступа: http://is.ifmo.ru/books/_umk.pdf

8. Веб-сайт проекта Eclipse Modeling Framework. – Режим доступа: http://www.eclipse.org/modeling/emf/
9. Кочелаев Д.Ю. Методы динамической проверки правил непротиворечивости автоматной модели. Бакалаврская работа. – СПбГУ ИТМО, 2007. – Режим доступа: http://is.ifmo.ru/papers/_kochelaev-bachelor.pdf
10. Warmer J., Kleppe A. The Object Constraint Language: Getting Your Models Ready for MDA. – Addison-Wesley, 2003.
11. Гуров В.С., Шалыто А.А., Яминов Б.Р. Технология верификации автоматных моделей программ без их трансляции во входной язык верификатора / Материалы международной научно-технической конференции «Многопроцессорные вычислительные и управляющие системы». – Т. 1. – 2007. – С. 198–203.
12. Гуров В.С., Шалыто А.А., Яминов Б.Р. Технология верификации автоматных программ без их трансляции во входной язык верификатора. Материалы международной научно-технической конференции “Многопроцессорные вычислительные и управляющие системы ” (МВУС-2007). Т. 1, с. 198–203. – Режим доступа: http://is.ifmo.ru/verification/_jaminov.pdf
13. Robby, Dwyer M., Hatcliff J. Bogor: A Flexible Framework for Creating Software Model Checkers /IEEE Conf. of the Testing: Academic & Industrial Conference – Practice and Research Techniques (TAIC PART) 2006, pp. 3–22. – Режим доступа: http://ieeexplore.ieee.org/iel5/11139/35654/01691665.pdf?arnumber=1691665
14. Кларк Э., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002. – Режим доступа: http://is.ifmo.ru/verification/_klark_gamberg_pered_verification.djvu
15. Вельдер С.Э., Шалыто А.А. Верификация простых автоматных программ на основе метода Model Checking / Материалы XV научно-методической конференции «Высокие интеллектуальные технологии и инновации в образовании и науке». – СПбГПУ, 2008. – С. 285–288. – Режим доступа: http://is.ifmo.ru/download/2008-02-25_politech_verification.pdf
16. Лукин М.А., Шалыто А.А. Автоматизация верификации визуальных автоматных программ / Материалы XV научно-методической конференции «Высокие интеллектуальные технологии и инновации в образовании и науке». – Режим доступа: СПбГПУ, 2008. – С. 296, 297. – Режим доступа: http://is.ifmo.ru/download/2008-02-25_politech_tezis.pdf
17. Хасянзянов Б. С. Метод создания отладчиков для доменно-ориентированных языков программирования на основе технологии Eclipse Modeling. Бакалаврская работа. – СПбГУИТМО, 2007. – Режим доступа: http://is.ifmo.ru/papers/domainlanguagedebugmethod/
18. Веб-сайт проекта Eclipse GMF. – Режим доступа: http://www.eclipse.org/modeling/gmf/
19. Лагунов И.А. Разработка текстового языка автоматного программирования и его реализация для инструментального средства UniMod на основе автоматного подхода. Бакалаврская работа. – СПбГУ ИТМО, 2008. – Режим доступа: http://is.ifmo.ru/papers/fsml/fsmleditor.pdf
20. Гуров В.С., Мазин М.А., Шалыто А.А. Автоматическое завершение ввода условий на диаграмме состояний // Информационно-управляющие системы. – 2008. – №1. – С. 24–33.