УДК 519.686.2

А.А. ЛЕТИЧЕВСКИЙ (мл.)

ПАРАДИГМЫ МОДЕЛЬНОГО И СИМВОЛЬНОГО ТЕСТИРОВАНИЯ ПРОГРАММНЫХ СИСТЕМ

Аннотация. Исследовано модельное тестирование программных систем — направление, известное в индустрии как Model-Based Testing (MBT). Проведен обзор основных методов и систем MBT, которые позволяют автоматизировать труд инженера-тестировщика и значительно повысить качество тестирования программной системы. Рассмотрена технология MBT, ее компоненты и основные инструменты, а также описаны проблемы, возникающие при применении данного метода для генерации тестовых сценариев. Представлены символьный подход, позволяющий решить ряд рассмотренных проблем генерации, а также система Универсальный Генератор Сценариев, основанная на символьном подходе, разработанном в Институте кибернетики им. В.М. Глушкова НАН Украины. Впервые символьный подход расширен — добавлен этап исполнения тестов, что позволяет значительно усовершенствовать MBT-подход для потребностей современной промышленности.

Ключевые слова: модельное тестирование, символьное моделирование, базовые протоколы, верификация, исполнение тестов, пользовательские сценарии.

ВВЕДЕНИЕ

Повышение требований к качеству систем, в частности к системам, критическим с точки зрения безопасности, привело к необходимости использования формальных методов в области программной инженерии. При тестировании программного продукта формальные методы играют значительную роль в автоматизации этого процесса. Развитие методов проверки моделей (Model Checking), а также дедуктивных методов обусловило появление такой области, как модельное тестирование (Model Based Testing, MBT). В последнее десятилетие появилось множество инструментов, которые интенсивно используются в тестировании сложных индустриальных систем. Тем не менее, не все традиционные средства модельного тестирования способны эффективно решать проблемы, связанные с тестированием. Такие специфические проблемы, как достижение покрытия тестами, явление экспонециального взрыва, тестирование недетерминированных и распределенных систем до сих пор не решены. Одним из средств их решения является символьное моделирование. В Институте кибернетики имени В.М. Глушкова НАН Украины разработан Универсальный Трассовый Генератор (Универсальный Генератор Сценариев), основанный на символьном подходе, позволяющий решать такие проблемы. В данной статье рассмотрены основные понятия и компоненты технологии, а также представлен обзор существующих инструментов модельного тестирования, которые дают возможность оценить значение символьного подхода в сравнении с существующими методами.

МЕТОД МОДЕЛЬНОГО ТЕСТИРОВАНИЯ

Рассматриваемый MBT-метод заключается в автоматизации создания тестовых наборов в соответствии с такими определенными пользователем критериями, как покрытие, метод тестирования, набор тестируемых свойств, включая свойство безопасности. При этом используются модели, описывающие поведение тестируемой системы, в отличие от традиционного способа, когда инженер-тестировщик создает тесты вручную.

Актуальность проблемы модельного тестирования обусловлена следующими факторами:

— в процессе разработки программного обеспечения 40 % трудозатрат приходится на тестирование;

© А.А. Летичевский (мл.), 2015

— тестирование является одним из самых трудоемких процессов, в котором существенно используется создание тестов вручную;

— при возрастающей сложности промышленных систем и увеличении количества тестируемого кода прежние ручные методы тестирования становятся неэффективными.

Традиционное тестирование как часть процесса разработки программного обеспечения является стандартным действием, состоящим из этапов планирования, документирования, создания и исполнения тестов. В зависимости от задач тестирования рассматривают различные его виды — общесистемное, интеграционное и регрессивное тестирование, а также многие другие с выбором соответствующих технологий, например технологии «черного ящика», при которой тестируемый код недоступен и используются только интерфейс кода с его окружением, и технологии «белого ящика», когда код является открытым и его можно применить при тестировании. Существует достаточно широкий спектр инструментов поддержки исполнения тестов в зависимости от тестируемых приложений, например тестирование интерфейсных программ или реактивных систем. Создание тестов базируется на требованиях к системе, которые детализируются в процессе разработки для инженеров-тестировщиков в виде так называемых функциональных спецификаций.

Далее исследуется модельное тестирование в рамках традиционного процесса разработки программного обеспечения для моделей CMM (Capability Maturity Model) [1]. Данная технология не рассматривается в рамках процесса разработки модели Agile [2], хотя последняя совместно с технологией MBT также заслуживает внимания. Модель CMM четко определяет такие этапы разработки, как сбор требований, дизайн, реализация и тестирование, а также алгоритм процесса для достижения необходимого качества продукта.

Общая схема MBT-метода приведена на рис. 1. Начальным этапом MBT-метода является создание тестовой модели на основании набора требований, обычно представленного в виде технического задания для разработки. Из множества требований выделяют поведенческие, содержащие информацию о возможных сценариях работы системы, в которых записаны запросы к ней и ее ожидаемые ответы, основанные на реализуемых в системе алгоритмах. Как правило, поведенческие требования вначале описываются в виде текстов на естественном языке и формализуются в виде схем взаимодействий, автоматных моделей и других видов формальных спецификаций.

Рассмотрим интерактивные программы, взаимодействующие с внешней средой в реальном времени, а также вычислительные программы, которые по исходным данным получают результат и выводят его. Основным режимом работы интерактивных программ является запрос–ответ или стимул–реакция. В отличие от вычислительных программ интерактивные имеют нетривиальные поведенческие модели. Поведение вычислительной программы по отношению к внешней среде вырождено: ввод исходных данных и вывод результата без взаимодействия с внешней средой. Поэтому MBT-тестирование в основном применяется для разработки интерактивных программ. Для того чтобы наблюдать внутреннее поведение вычислительной программы (ветвления и циклы), превра-

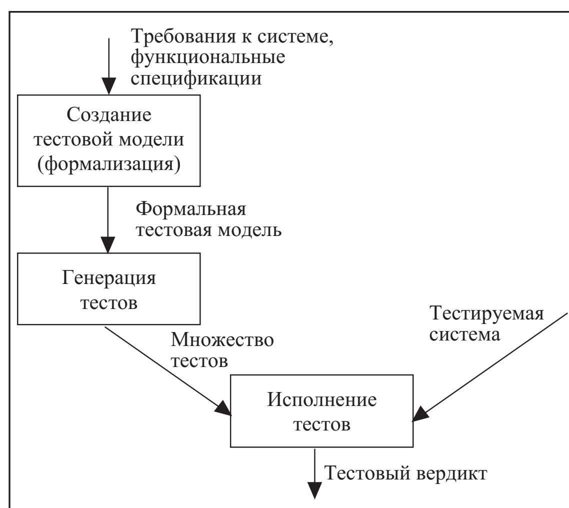

Рис. 1. Схема тестирования с помощью модели

щаем ее в интерактивную путем включения дополнительных операторов взаимодействия с внешней средой.

К интерактивным программам относятся также системы взаимодействующих программ. В их моделях имеются средства организации параллельных вычислений. Количество сценариев поведения параллельных процессов может быть сколь угодно велико и МВТ-метод в данном случае незаменим.

Создание тестовой модели — это формализация информации, представленной в поведенческих требованиях в терминах входного языка выбранной системы генерации тестов. Ряд систем используют поведенческую информацию, формализованную с помощью диаграмм языка UML (Universal Modelling Language) или таких формальных языков-спецификаций, как *Z*-нотация.

С помощью тестовой модели, совместимой с входным языком некоторого инструмента, для генерации тестов можно получить набор сценариев поведения системы, удовлетворяющих исходным требованиям. Такие сценарии в дальнейшем используются для окончательной генерации тестов, вид которых обусловлен уже следующим инструментом. Последний в свою очередь исполняет тестирование готовой системы или ее детализированной модели. Сгенерированные сценарии содержат информацию о данных, которые воспринимаются на входе тестируемой системы, и ее соответствующие ответы. Эти данные содержатся в тесте и используются системой исполнения тестов для анализа ответов системы и их сравнения с ожидаемыми ответами. Обнаруженные несовпадения фиксируются в вердиктах инструментов исполнения тестов и выдаются пользователю.

Приведенную схему МВТ-метода можно детализировать в зависимости от методов и инструментов генерации, а также способов исполнения тестов. Так, возможна генерация абстрактных сценариев, которые покрывают не конкретные данные, а множества их значений. Исполнение тестового набора представляет собой, как правило, исполнение тестов с конкретными данными. Тогда возникает необходимость конкретизации сценариев из абстрактных тестов. При этом используются различные методики рассмотрения конкретных значений, например граничных либо произвольных значений из некоторого множества, и конкретизируются как воздействия на тестируемую систему, так и ожидаемые ответы.

Рассмотрим подробнее компоненты процесса МВТ и критерии генерации тестов.

МОДЕЛИ ДЛЯ ГЕНЕРАЦИИ ТЕСТОВЫХ СЦЕНАРИЕВ

Модель для генерации тестов можно подготовить из поведенческих требований к строящейся системе и детализировать ее в процессе разработки. Дизайн-модель системы, которая создана по набору имеющихся требований, также можно использовать в качестве модели для генерации тестовых сценариев.

Модель генерации тестов чаще всего описывается на языке UML [3], представляющем ее с помощью соответствующих диаграмм, в частности диаграмы автоматов. Последняя является конечным автоматом с простыми и композитными состояниями вместе с переходами. Каждый переход осуществляется под воздействием некоторого сигнала извне от сущностей, называемых актерами. Переход может происходить при некотором условии (guard) и изменении среды системы. На рис. 2 приведена простая диаграмма автоматов.

Другие диаграммы языка UML представляют собой композицию диаграмм автоматов, в том числе и как параллельных процессов, а также структур, позволяющих объединять диаграммы в блоки. Диаграммы классов определяют классы и методы систем, используемых в диаграммах автоматов.

Язык SysML [4] является расширением языка UML, в котором не имеется программно-ориентированных ограничений, однако добавлены два дополнительных вида диаграмм: требований и параметрические, в результате чего значительно расширилось множество представленыых систем, а язык стал легче для понимания и изучения. Для систем AGEDIS, Conformiq, ParTeG и CertifIt язык UML и его расширения используются как входные.

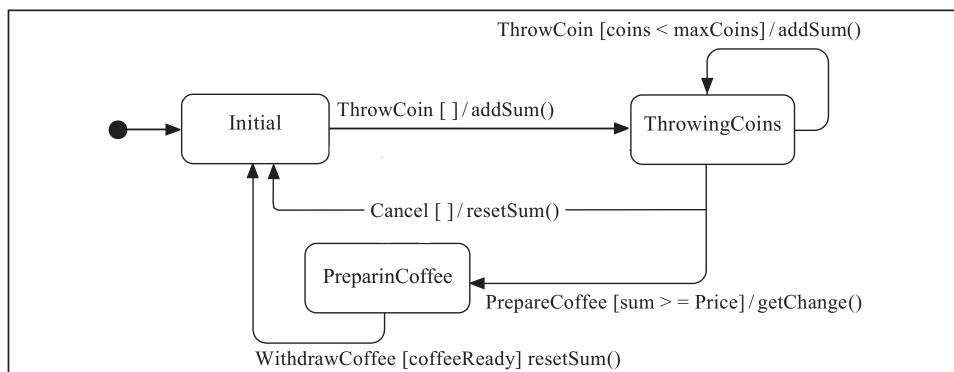

Рис. 2. Простая диаграмма автоматов

Таблица 1. Инструменты МВТ, использующие автоматные спецификации, представляющие кофейный автомат

Модель	Система
FSM	GOTCHA [6], GraphWalker [7], NModel [8], PyModel [9], SpecExplorer, TestOptimal [10]
ASM	SpecExplorer
EFSM	GraphWalker, TestCast [11], TestOptimal

Язык BPMN (Business Process Model and Notation) состоит из набора интуитивно понятных элементов, позволяющих определять более сложные конструкции. Элементы с потоком управления объединяются в диаграммы, которые моделируются, транслируются в исполняемые спецификации и конвертируются в другие языки моделирования. Язык BPMN менее известен в области МВТ, однако ряд инструментов используют его в качестве языка входных спецификаций; он внедрен для создания моделей тестирования для потребностей промышленности.

Нотация Z (Z-notation) [5] основана на языке аксиоматической теории множеств Цермелло–Френкеля и представляет собой язык формальных спецификаций для описания и моделирования систем.

Примерами автоматных моделей являются FSM (Finite State Machine), ASM [16] (Abstract State Machine), EFSM (Extended State Machines), причем FSM/ASM/EFSM-спецификации используются для описания электронных устройств и программных систем и распространены в области МВТ (табл. 1).

Широко известные модели VDM [12], LOTOS [13], Petri net [14] и другие используются в основном в исследовательских работах многих университетов, но не занимают должного места в промышленном применении МВТ.

Язык TASM (Timed Abstract State Machines) [15] разработан в Массачусетском технологическом институте как спецификация временных автоматов для описания систем, в которых время является атрибутом. Язык TASM — расширение языка ASM (Abstract State Machine) [16], в нем спецификации представлены с помощью правил. Каждое правило содержит условия его исполнения и изменения атрибутов системы, а также триггерное событие. В язык TASM вставлены временные спецификации, определяющие дополнительные ограничения на выполнения этих правил. Приведем пример правил языка TASM:

```
R1: Turn On
Time = [4, 10]
if light = OFF and switch = UP then
light := ON
```

Построение тестовых моделей по трудоемкости сопоставимо с кодированием, однако на более высоком уровне абстракции. Создание модели основано на требованиях, которые можно детализировать в процессе разработки системы.

Требованиями могут являться блок-схемы (use-cases), представляющие обобщенные сценарии либо локальные реакции системы на внешнее воздействие, записанные на естественном языке.

Вопрос достаточности информации в требованиях важен при разработке модели для тестирования. Иногда имеющегося уровня абстракции недостаточно для того, чтобы написать тест или тестовую модель по требованиям. Возможна детализация и уточнение модели в процессе разработки, что может применяться в модели генерации тестов.

Модели, использующие информацию о состояниях системы (state-based) и возможных ее сценариях (scenario-based) либо другие формальные подходы (например, logic-based), обусловливают методы генерации тестов и выбор исходной модели. Например, требования к реактивным системам определяются упорядоченными событиями, что объединяет сценарный и автоматный подходы.

В системе VRS (Verification Requirements Specifications) [17] входным языком являются спецификации UCM (Use Case Maps) — стандарты в области специфицирования телекоммуникаций [18]. Язык прост в изучении и понимании, а также представляет сценарный подход графически. Объединенная с языком базовых протоколов [19], разработанным в Институте кибернетики им. В.М. Глушкова, UCM-нотация используется для генерации тестов в системе инсерционного моделирования. Базовые протоколы на этом языке представляют локальные описания поведения системы с помощью пред- и постусловий, а также действий, которые определены посредством MSC-диаграмм (Message Sequence Chart) [20]. На рис. 3 приведены диаграмма и элементы языка базовых протоколов.

Выбор модели определяется потребностями промышленности, профессиональными навыками разработчиков, наличием средств моделирования и многими другими факторами. Например, в моделях, разработанных для автомобильной промыш-

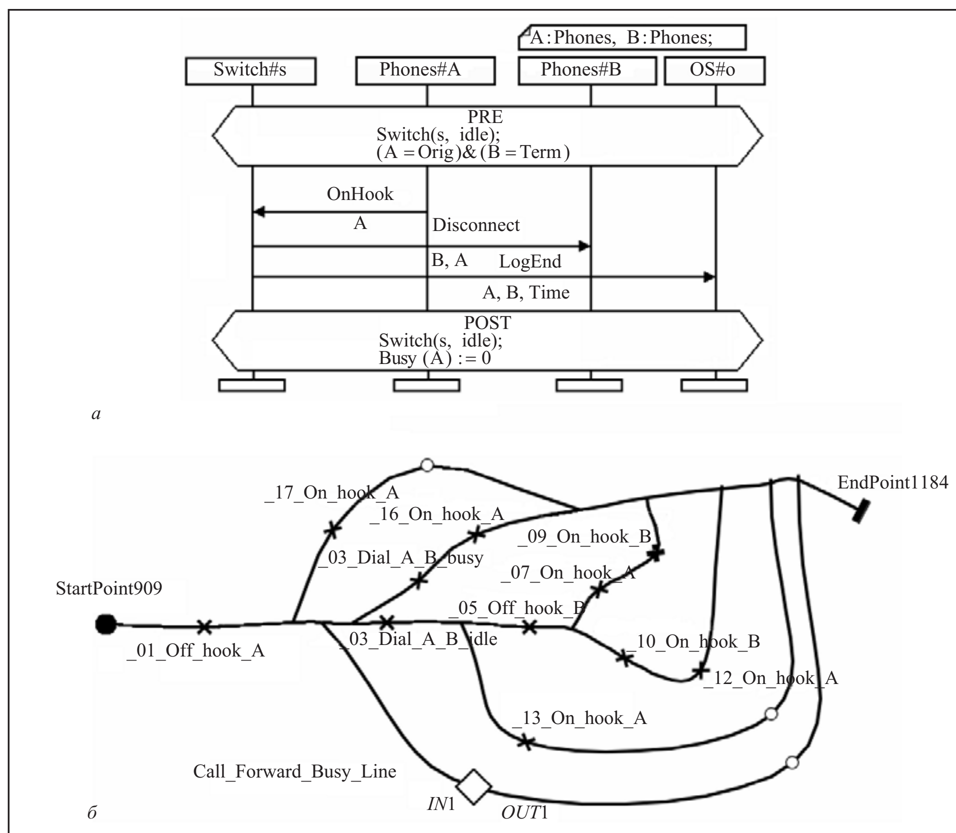

Рис. 3. Спецификация базовых протоколов (*а*) и UCM-диаграмма (*б*), представляющая фрагмент протокола телефонии

ленности, предпочтительно использование языка SysML, отвечающего требованиям заказчиков и традиционному дизайну в языке UML. Спецификации, основанные на сценарных диаграммах, отображающих поток состояний (state flow charts), ближе к динамическим моделям, связанным с временными спецификациями.

В процессе создания тестовой модели возникает проблема ее соответствия требованиям заказчика, а также вопрос о ее корректности. Тестирование такой модели тестами на более высоком уровне абстракции, например, если модель представлена в виде SDL-спецификаций, описано в [21]. Однако корректность модели можно проверить и формальными методами. В ней могут иметься тупики (deadlocks) и недетерминизмы. В частности, автор настоящей статьи является одним из разработчиков системы VRS, которая верифицирует формальные требования для таких неисправностей.

КРИТЕРИИ ГЕНЕРАЦИИ ТЕСТОВЫХ НАБОРОВ

Во всех имеющихся инструментах генерация тестовых наборов определяется некоторыми критериями. С одной стороны, с помощью метода генерации выбирается набор необходимых тестов, с другой — задается критерий завершения их генерации. Для этого ряд параметров регулируется непосредственно в инструментах. Одним из таких параметров является покрытие. Причем покрытие модели измеряется степенью покрытия всех возможных сценариев поведения объекта, описанного в модели, а покрытие кода тестируемой системы измеряется при исполнении тестов и определяется количеством покрытых операторов языка программирования, на котором написана система. Покрытие кода системы при тестировании методом «черного ящика» показывает, насколько качественно и удачно подобраны набор тестов и параметры их генерации.

На абстрактном уровне все модели для генерации тестов представляют собой транзиционные системы. Используя терминологию транзиционных систем, выделим следующие виды покрытия модельных сценариев:

— покрытие всех состояний транзиционной системы (в случае использования автоматной UML-диаграммы или UCM-карты необходимо получить все сценарии, покрывающие вершины этой диаграммы или карты, а в случае применения языка TASM требуется покрыть каждое правило описания модели);

— покрытие всех переходов транзиционной системы (получение всех сценариев, покрывающих все дуги, соединяющие вершины автоматной UML-диаграммы либо все ветви в правилах на языке TASM, либо каждую дугу между вершинами UCM-карты);

— покрытие всех путей между указанными состояниями (простых или с определенным числом повторений).

Рассматриваются также локальные покрытия (test stubs), а именно все сценарии, покрывающие один из компонентов системы с определенным ранее видом покрытия. Примером такого локального покрытия может быть сценарий, заданный пользователем в виде пути между состояниями системы. Множество сценариев, покрывающих пользовательский путь, должно покрываться по всем заданным переходам.

Описанные покрытия являются критерием «потока управления» системы. Другие виды покрытия можно получить согласно степени покрытия данных, например тестов с конкретными значениями атрибутов модели системы.

В параметризованные или абстрактные тесты можно подставлять различные значения в качестве параметров. Эти подстановки определяют уровень покрытия тестов данными: одно значение для одного параметра, все значения параметров, граничные значения параметров, все пары значений параметров.

Покрытие кода (Script-flow-covering) определяет степень покрытия выполненных инструкций или операторов тестируемого кода и запускает тесты на автоматизированной системе исполнения тестов. Вид такого покрытия зависит от прохождения всех ветвей, операторов, функций, условий и других элементов тестируемой программы.

Таблица 2. Виды покрытий в системах тестирования

Система тестирования	Входная модель	Покрытие			
		потока управления	потока данных	требований	кода
TestMaster	EFSM	Все переходы, все пути	—	Да	Функции
GOTCHA (IBM)	FSM	Все вершины, все переходы	Одно значение, все значения	—	—
GraphWalker	FSM/EFSM	Все вершины, все переходы	—	Да	Операторы
TestCast	EFSM	Все переходы, пары переходов, Scenario mode	—	—	—
NModel	FSM	Все переходы, все пути	Одно значение, все значения	Да	Функции
PyModel	FSM	Все вершины, все пути	Одно значение, все значения	—	Функции
SpecExplorer (Microsoft)	FSM/ASM	Все переходы, все пути	Одно значение, все значения, граничные значения	Да	Функции
TestOptimal	FSM/EFSM	Все вершины, все переходы, Scenario mode	—	Да	—
AGEDIS	UML	Все вершины, все переходы	—	—	Функции
Conformiq	UML/SysML	Все вершины, все переходы, все пары переходов, все пути	граничные значения	Да	Функции, операторы, ветви, условия
ParTeG	UML	Все вершины, все переходы	—	—	Операторы, ветви, условия
CertifyIt (SmartTesting)	UML	Все вершины, все переходы, все пары переходов, Scenario mode	Одно значение, все значения, граничные значения	Да	Функции, операторы, ветви, условия
BPM-Xchange	UML/SysML/ BPMN	Все пути	—	—	Операторы, ветви, условия

Рассматривают также степень покрытия исходных требований к системе при соответствии (трассировке) модели требованиям.

Среди существующих инструментов тестирования с помощью моделей имеются такие, которые обеспечивают различные виды покрытия Они приведены в табл. 2; здесь использованы данные из [22], а также данные, основанные на изучении инструментов непосредственно автором настоящей статьи.

МЕТОДЫ ГЕНЕРАЦИИ ТЕСТОВ

Для генерации сценариев поведения системы, соответствующих заданному покрытию, используются различные алгоритмы, одним из которых является решение задачи «китайского почтальона» [23]. С помощью этого алгоритма проводится поиск кратчайшего замкнутого пути по каждому ребру связного ориентированного графа как минимум один раз. Этот граф связывают с потоком управления модели и решение данной задачи реализуют в терминах языка спецификаций модели.

Обход переходов транзиционной системы, в которой каждый переход может отождествляться с дугой ориентированного графа, определяется предусловиями

Таблица 3. Используемые методы генерации сценариев различными системами

Методы генерации	Системы
Обход всех состояний	Conformiq [24], GraphWalker
Вероятностные методы	Conformiq, GraphWalker
Программируемые стратегии	PyModel, CertifyIt [25], SpecExplorer
Символьные методы	RT_Tester [26]
Алгоритмы покрытия с использованием эвристик	TestCast, TestOptimal

и поэтому алгоритмы, применяемые к потоку управления, не годятся. Тем не менее их используют для генерации абстрактных тестов или пользовательских сценариев (user-guides), которые в дальнейшем инстанциируются конкретными значениями. Применение алгоритма для решения задачи китайского почтальона в целях обхода переходов транзиционной системы может привести к генерации недостижимых путей, для которых построить конкретные тесты невозможно.

Значительный вклад в технологию обхода состояния транзиционных систем сделан в области Проверки Моделей (Model Checking), основной целью которой является попытка обойти все состояния системы в целях нахождения ошибки или достижимости какого-либо свойства. Поскольку невозможно решить проблему достижимости для систем с бесконечным количеством состояний или эффектом экспоненциального взрыва, используются такие эвристики и символьные представления, как предикатная абстракция, Binary Decision Diagrams, абстрактная интерпретация и метод инвариантов. Эти методы не применяются в должной мере в MBT, поскольку задача заключается не в полном обходе состояний, а в достижении определенного покрытия.

При выборе сценария в точках разветвления, а также при подборе конкретных значений в атрибутах системы используются вероятностные методы.

Программы генераторов трасс написаны на таких языках программирования, как Python, C# и Java. Основа методов генерации в целом идентична, но отличны свойства, позволяющие добиться более точного покрытия или обойти большее количество состояний за определенное время.

Некоторые инструменты используют в качестве методов генерации специально разработанные эвристики, ориентированные на определенные критерии тестирования.

Свойства алгоритмов генерации сценариев определяются возможностями достижения того или иного покрытия или их комбинирования. Различные инструменты генерируют сценарии, которые преобразуются в тесты посредством конкретизации, выбора тестирующей и тестируемой системы или ее компонентов. При генерации абстрактных тестов во избежание генерации недостижимых тестов генератор должен работать не только с потоком управления модели, но и с ее атрибутами. В таких случаях используются символьные методы, основанные на существующих технологиях SMT-проверки моделей.

В табл. 3 приведены системы, соответствующие используемым методам генерации трасс.

ПРОБЛЕМЫ MBT-ТЕСТИРОВАНИЯ

В последнее десятилетие интенсивно развивался MBT-метод и многие системы стали коммерческими, однако ряд проблем ограничивает его использование, прежде всего проблема экспоненциального взрыва, возникающего при генерации тестовых наборов.

В процессе достижения некоторого покрытия часть компонентов модели остается непокрытой, поскольку отыскать к ним путь невозможно по причине

большого количества состояний, которые нужно пройти. Кроме того, неизвестно, достижимы ли эти компоненты вследствие экспоненциального взрыва или это действительно недостижимые состояния ввиду наличия тупиков в модели или зацикливания. Сложность заключается в том, что задача отыскания тупиков в модели сопоставима с задачей генерации тестов и может также не решаться вследствие экспоненциального взрыва.

В больших проектах для описанных систем MBT риск столкнуться с такой проблемой велик. Практически все системы работают с конкретными значениями в тестах, которые могут подбираться случайным образом. Многообразие комбинаций конкретных значений, интерливинг событий в параллельных процессах, циклы в моделях — все это причины экспоненциального взрыва.

Другой проблемой MBT-метода является корректность исходной модели, которую также необходимо тестировать. Обычно такие системы ревьюируют вручную либо проводят верификацию различных свойств безопасности.

Так называемая проблема отображения (mapping problem) возникает в процессе создания конкретных тестов, а именно при сопоставлении уровней абстракции тестируемой системы с имеющимся набором тестов. Если тестируемая система является исполняемой программой, написанной на языке программирования и воспринимающей тестовые данные извне, то необходимо учитывать следующие моменты:

— соответствие наборов входных данных, т.е. соответствие описанных интерфейсов как на уровне программы, так и на уровне набора тестов (данные, не описанные в модели, которые система может воспринимать дополнительно, должны намеренно игнорироваться или тестирующая система должна уметь от них абстрагироваться, не провоцируя нежелательных сценариев);

— при тестировании конкретными тестами можно пропустить некоторые сценарии поведения, ведущие к ошибке или не соответствующие поведению модели, что невозможно выявить традиционным MBT-методом «черного ящика» ввиду детализации модели в терминах программы.

Описанные проблемы частично решают системы, использующие символьные методы.

СИМВОЛЬНЫЕ МЕТОДЫ В ТЕСТИРОВАНИИ

Методы символьного моделирования позволяют работать с формулами, покрывающими множества наборов значений атрибутов систем, что обеспечивает более полное покрытие состояний. Символьное моделирование для генерации абстрактных тестов рассматривают в целях запуска конкретных тестов на тестируемом продукте, например инструмент RT-tester, работающий с UML/SysML-моделями, трансформирует их во внутреннее представление как структуры Крипке. Требования к модели выражаются в терминах LTL-формул. Тесты генерируются с помощью SMT-решателя для различных критериев покрытия.

Другие виды тестирования используют исходный текст программы системы, написанный на некотором языке программирования, и осуществляют ее символьное исполнение. При этом генерируются сценарии, содержащие в качестве входных и выходных сигналов формулы, из которых можно получить конкретные значения. В процессе символьного исполнения проверяются нарушения следующих свойств безопасности: обращение к нулевой памяти, деление на нуль, выход за пределы структур данных. Кроме того, проверяются свойства или аннотации, заданные пользователем в различных точках программы. К таким инструментам относится разработка Microsoft SAGA [27], интенсивно используемая в тестировании свойств безопасности, с помощью которой были выявлены многие серьезные ошибки.

Инструмент PathFinder [28] проводит символьное моделирование Java-кода, проверяя безопасность и пользовательские свойства; инструменты KLOVER [29]

и Barad [30] тестируют графический интерфейс, используя символьное моделирование строк.

Заметим, что описанные инструменты фактически яляются верификационными, поскольку не существует тестовой модели системы в качестве исходных данных. Аннотации пользователя можно считать некоторой частью модели исходной системы, но они не всегда полностью отражают поведенческие требования к системе.

Универсальный Генератор Сценариев (Generic Trace Generator, GTG) разработан в Институте кибернетики им. В.М. Глушкова [31]. Исходной моделью для генерации сценариев является UCM-нотация с языком базовых протоколов.

Множество начальных состояний S_0 моделируемой системы представляется формулой в базовом логическом языке. Эту формулу назовем символьным состоянием среды. Она покрывает множество наборов значений атрибутов системы, которые могут быть в начальном состоянии. В UCM-диаграмме начальное состояние представлено стартовой точкой. Двигаясь по диаграмме через точки, с которыми связаны переходы системы, можно применять базовые протоколы B_i, получая символьные состояния среды S_i: $S_0 \xrightarrow{B_1} S_1 \xrightarrow{B_2} \ldots$ Полученная последовательность базовых протоколов $B_1, B_2, \ldots$ является сценарием (трассой) и ее можно развернуть в MSC-диаграмму подстановкой процессных действий базового протокола. Процесс генерации таких последовательностей называется трассовой генерацией. Каждый шаг движения выполняется следующим образом:

— проверяется применимость базового протокола с помощью определения выполнимости конъюнкции текущего символьного состояния среды и предусловия протокола;

— изменяется символьное состояние среды в случае выполнимости с помощью функции предикатного трансформера.

Свойства предикатных трансформеров рассмотрены в [32] и формально определяются как функция $pt(S_i \wedge \mathrm{Pre}(B_i), \mathrm{Post}(B_i)) \to S_{i+1}$, где $\mathrm{Pre}(B_i)$ и $\mathrm{Post}(B_i)$ являются пред- и постусловиями базового протокола B_i; S_i и S_{i+1} — соответственно старое и новое символьное состояние среды.

Поскольку UCM-диаграмма представляет недетерминированную транзиционную систему, соответственно на каждом шаге генерации последовательности возможно некоторое множество продолжений, определенное применяемыми базовыми протоколами.

Стратегия выбора очередного базового протокола задается критерием генерации тестов с помощью запроса на тестирование. Последний содержит требуемое покрытие, которое в терминах UCM-диаграмм определяет покрытие всех базовых протоколов, всех пар базовых протоколов, соединенных дугами диаграммы, покрытие всех состояний, всех путей или всех последовательностей базовых протоколов, ведущих из заданной одной или нескольких стартовых точек в соответствующее заданное множество конечных точек. В запросе могут содержаться различные ограничения на длину трассы, на количество циклов. Конструкции «стабы» (stubs) определяются своим критерием покрытия. Стратегия генерации может регулироваться такими параметрами, как генерация кратчайших или длиннейших трасс.

После окончания генерации трасс в соответствии с запросом могут остаться непокрытые состояния вследствие наличия тупиковых состояний либо недостаточно хорошо подобранных параметров запроса. Экспоненциальный взрыв множества состояний также может являться причиной недостижимости состояний.

Для решения этой проблемы в Универсальном Генераторе Сценариев создан сервис для проверки достижимости состояния. Поскольку проблема достижимости неразрешима, инструмент включает множество эвристик, с помощью которых возможно достижение состояния или будет доказано, что оно недостижимо.

Одной из эвристик является обратное символьное моделирование. Генерация трассы в этом случае происходит от недостигнутой точки к достигнутым состояниям с помощью обратного предикатного трансформера.

В поиске достижимости используются также инварианты, которые генерируются в процессе генерации трасс. Недостижимость непокрытого состояния определяется как отсутствие пересечения между предусловием базового протокола и постинварианта. Возможно также использование комбинации обратного моделирования и инвариантов.

Множество сгенерированных трасс формирует тестовый набор для дальнейшего его исполнения соответствующими инструментами. Каждая трасса содержит символьное состояние среды в виде формулы.

Тест создается как MSC-диаграмма, в которой выбраны тестируемая и тестирующая системы. Сообщения в диаграмме определяют точки взаимодействия этих систем. Формула содержит множество значений атрибутов, входных для тестируемой системы, и множество ее прогнозируемых значений–ответов.

Одним из способов тестирования является получение тестов, содержащих вместо формул конкретные значения входных и выходных атрибутов систем. Универсальный Генератор Сценариев генерирует конкретные значения по выбранной методике с помощью SMT-решателя. В дальнейшем тесты можно конвертировать в стандартный формат тестов, например TTCN, в целях их исполнения на существующих стандартных средствах.

Метод конкретизации тестов применяется для определенного класса систем. Тем не менее, если система задана на довольно высоком уровне абстракции, возможны ситуации, когда нельзя корректно определить конкретные значения. Другим недостатком этого метода является пропуск некоторых сценариев поведения тестируемой системы вследствие детализации модели и перехода на другой уровень абстракции.

Кроме конкретизации и исполнения конкретных тестов, можно рассматривать метод символьного исполнения тестов или символьного тестирования.

СИМВОЛЬНОЕ ТЕСТИРОВАНИЕ

Символьное тестирование базируется на символьном исполнении программы в соответствии со сгенерированным множеством тестов. Схема символьного тестирования приведена на рис. 4.

Входом процедуры символьного тестирования является множество сгенерированных символьных трасс в виде MSC-диаграмм, размеченных формулами над параметрами сообщений, и исходный код тестируемой системы. Этот код начинает исполняться посредством символьных методов следующим образом. Каждый оператор исходного кода на некотором языке программирования представляется в виде пред- и постусловия, зафиксированных в базовом протоколе. Вначале исходный код конвертируется в UCM-диаграмму, которая представляет поток управления программы, а в точках “responsibility” содержатся базовые протоколы. Некоторые операторы, например оператор цикла, могут конвертироваться в несколько базовых протоколов.

Таким образом, символьное исполнение исходного кода происходит также с помощью предикатных трансформеров. Отличие состоит в том, что поток управления программы или движение по UCM-диаграмме с выбором базового протокола осуществляются в соответствии со сгенерированными тестами.

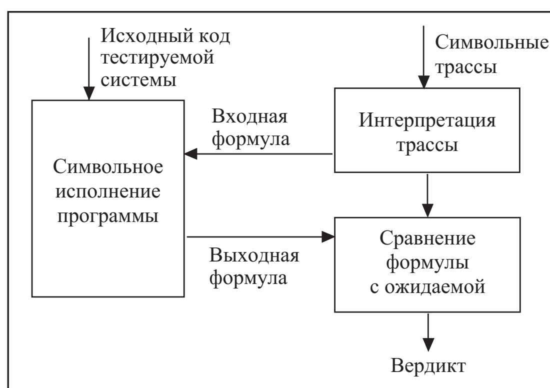

Рис. 4. Схема символьного тестирования

Тестируемая система воспринимает входные формулы, применяет соответствующий

базовый протокол и возвращает новое символьное состояние среды, которое анализируется в соответствии со следующей процедурой.

Пусть X и Y — соответственно ожидаемое и текущее символьное состояние среды. Тогда если:

— конъюнкция $Y \wedge \neg X$ невыполнима, то тест соответствует ожидаемому поведению;

— конъюнкция $Y \wedge \neg X$ выполнима, то тестирование можно продолжить, однако появится предупреждение, что существуют состояния, наличие которых в системе не предполагается, и формула $Z = Y \wedge \neg X$ покроет эти состояния;

— конъюнкция $X \wedge Y$ невыполнима, то тестирование не выполняется.

Обратным моделированием от состояния Z можно получить начальные или промежуточные состояния, при которых возникают непредвиденные состояния в тесте.

СИМВОЛЬНОЕ ТЕСТИРОВАНИЕ ОНЛАЙН

В рамках символьного тестирования можно рассматривать процесс порождения трасс с их исполнением. Это дает возможность обрабатывать недетерминированные ответы системы согласно тестовой модели, что является одной из проблем MBT.

Рассмотрим процедуру, входом которой есть модель взаимодействия тестирующей и тестируемой системы, исходный код последней, являющейся реализацией модели на некотором языке программирования, а также их параллельную композицию, а именно композицию UCM-диаграмм (рис. 5). Начинаем символьное моделирование системы и синхронизируем символьное исполнение ее исходного кода, конвертированного в UCM-нотацию, с помощью входных данных.

Новые состояния среды, которые построены как на модели, так и на ее реализации, сравниваются и анализируются согласно процедуре символьного тестирования, описанной ранее.

Для реализации данного подхода необходимо соблюдение некоторых условий. Одно из них состоит в том, что уровень абстракции модели должен обеспечить обозримость сигналов, которыми программа обменивается со своей средой. Это значит, с одной стороны, что множество сигналов должно быть одним и тем же для модели и для реализации, а с другой — обеспечиваться соответствие между переменными в программе и в модели. Так, переменные в программе, которые не содержатся в атрибутах модели, необходимо поместить под квантор существования в символьном состоянии среды для реализации.

Рассмотренный вид тестирования относится к так называемому тестированию по методу «черного ящика», когда тестирующая система управляется тестами или генерирующимися онлайн трассами. В этом случае можно управлять достижением того или иного вида покрытия модели, но не покрытием кода реализации модели. Если значительные участки кода не покрыты сгенерированным тестовым набором, то не гарантируется качественного тестирования. Повысить уровень качества можно за счет достижения покрытия кода системы с помощью символьного тестирования по методу «белого ящика». Чтобы организовать такое тестирование необходимо осуществить внешний контроль исполняемого кода реализации модели.

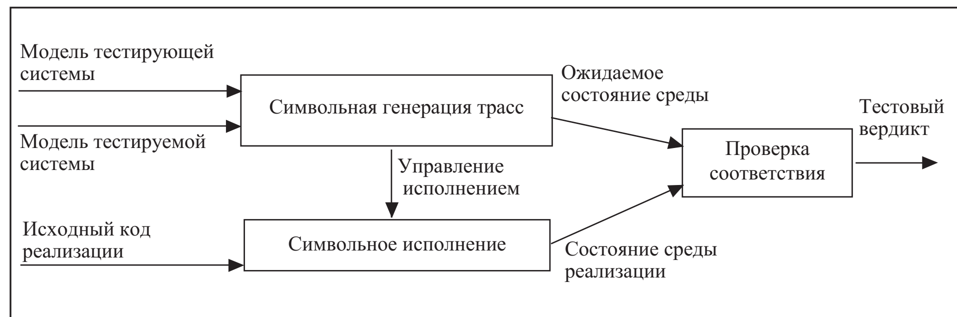

Рис. 5. Схема символьного тестирования онлайн

В отличие от тестирования по методу «черного ящика» начнем символьное исполнение реализованной системы синхронно с параллельным исполнением модели с помощью входных сигналов или данных.

Двигаясь по UCM-диаграмме, представляющей модель, получаем для нее символьные состояния среды и сравниваем с ожидаемыми состояниями, полученными при символьном исполнении модели. В этом случае анализ проходит как в описанной ранее процедуре.

Символьное тестирование по методу «белого ящика» позволяет также оценить соответствие модели ее реализации с точностью до атрибутов, объявленных в модели. Это значит, что можно определить покрытие на исходном коде для сценария поведения системы относительно интересуемых атрибутов модели. Это избавит от генерации сценариев поведения системы, не описанных в модели и являющихся детализацией.

Такое покрытие называется покрытием аспекта системы и строится с помощью анализа потоков данных. Отметим, что задача построения аспекта является NP-полной и в этом случае используются эвристики, аналогичные применяемым для определения достижимости, а также отметим, что слишком высокий уровень абстракции модели не позволяет адекватно оценить прохождения теста в символьном моделировании. В этом случае необходимо пользоваться комбинированием конкретных и символьных тестов.

ЗАКЛЮЧЕНИЕ

В статье дан обзор существующих методов модельного тестирования. Предложен новый подход к модельному тестированию с помощью символьного моделирования, позволяющий решить ряд проблем, которые появляются при применении MBT-метода. Основой подхода является применение символьных методов, использующих предикатные трансформеры и генерацию инвариантов.

Универсальный Генератор Сценариев является системой символьного моделирования, которая позволяет снизить степень экспоненциального взрыва при генерации тестов, используя устранение интерливинга для параллельных процессов.

Технология символьного тестирования в отличие от существующих методов символьного исполнения позволяет значительно повысить качество тестирования за счет комбинирования методов «черного» и «белого» ящика. В существующих системах MBT не имеется аналогов символьного тестирования.

Предложенный подход символьного тестирования предполагает дальнейшие исследования в области использования SMT-технологии в тестировании на основе модели. Исследования прежде всего направлены на повышение эффективности символьного MBT. Для этого необходимо улучшать дедуктивные механизм системы, возможно интегрируя ее с другими известными SMT-решателями. Работы в области генерации инвариантов цикла и построения абстракций, основанных на анализе потоков данных и абстрактной интерпретации, позволят сократить перебор состояний и представить Универсальный Генератор Сценариев как одну из перспективных систем в области MBT.

СПИСОК ЛИТЕРАТУРЫ

1. Paulk M.C., Weber C.V., Curtis B., Chrissis M.B. capability maturity model for software (Version 1.1) // Technical Report (Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University). — 1993. — 65 p.
2. Beck K. et al. Manifesto for agile software development (2001) // Agile Alliance. Retrieved 14 June 2010.
3. Ларман К. Применение UML 2.0 и шаблонов проектирования. — М.: Вильямс, 2006. — 736 с.
4. Weilkiens T. Systems engineering with SysML/UML: modeling, analysis, design. — The OMG Press, 2008. — 299 p.
5. Z Formal Specification Notation — Syntax, Type System and Semantics // ISO/IEC 13568:2002 — 2002. — 196 p.

6. http://www.haifa.ibm.com/projects/verification/gtcb/.
7. http://www.graphwalker.org/.
8. http://www.codeplex.com/Nmodel.
9. http://staff.washington.edu/jon/ pymodel/www/.
10. http://www.testoptimal.com.
11. http://www.elvior.com/motes/generator.
12. Dines B., Jones C.B. The Vienna development method: the meta-language // Lecture Notes in Computer Science. — Berlin; Heidelberg; New York: Springer, 1978. — **61.** — 398 p.
13. van Eijk P.H.J., Vissers C.A., Diaz M. The formal description technique LOTOS: results of the ESPRIT/SEDOS Project. — Amsterdam: Elsevier, 1989. — 462 p.
14. Petri C.A., Reisig W. Petri net // Scholarpedia. — 2008. — 3(4):6477.
15. Ouimet M., Berteau G., Lundqvist K. Modeling an electronic throttle controller using the timed abstract state machine language and toolset / T. Kuhne (Ed.) // MoDELS 2006 Workshops. Lecture Notes in Computer Science.. — 2007. — **4364.** — P. 32–41.
16. Borger E., Stark R. Abstract state machines. — Springer-Verlag, 2003. — 420 p.
17. Letichevsky A., Letychevskyi O., Weigert T., Kapitonova J., Volkov V., Baranov S. , Kotlyarov V. Basic protocols, message sequence charts, and the verification of requirements specications // Computer Networks. 2005. — **47.** — P. 662–675.
18. ITU-T Recommendation, Z.151, User Requirements Notation (URN) — Language definition.
19. Летичевский А.Ад., Капитонова Ю.В., Волков В.А., Летичевский А.А., Баранов С.Н., Котляров В.П., Вейгерт Т. Спецификация систем с помощью базовых протоколов // Кибернетика и системный анализ. — 2005. — № 4. — С. 3–21.
20. TU-T Recommendation, Z.120, Message Sequence Charts.
21. Wong W.E., Restrepo A., Qi Y., Choi B. An EFSM-based test generation for validation of SDL specifications // Proceedings of 3 International wWorkshop on AST. — 2008. — P. 25–32.
22. Shafique M., Labiche Y. A systematic review of state-based test tools // International Journal on Software Tools and Technology Transfer . — 2013. — DOI 10.1007/s10009-013-0291-0.
23. Parampreet Kaur et al. Coverage analysis and chinese postman algorithm for efficient model-based test generation // International Journal of Computer Science and Mobile Computing. — 2013. — **2**, N 4. — P. 491–494.
24. http://www.conformiq.com.
25. http://www.smartesting.com.
26. Peleska J. Industrial-strength model-based testing — state of the art and current challenges // Proc. Eighth Workshop on Model-Based Testing. — 2013. — P. 3–28.
27. Godefroid P., Levin M.Y., Molnar D. SAGE: whitebox fuzzing for security testing // Magazine Queue-Networks. — 2012. — **10**, N 1. — 20 p.
28. Visser W., Mehlitz P. Model checking programs with Java PathFinder // Lecture Notes in Computer Science. — 2005. — **3639.** — p. 27.
29. Li G., Ghosh I., Rajan S.P. KLOVER: A symbolic execution and automatic test generation tool for C++ programs // Lecture Notes in Computer Science. — 2011. — **6806.** — P. 609–615.
30. Ganov S., Killmar C., Khurzid S., Dewayne E. Perry test generation for graphical user interfaces based on symbolic execution // Proceedings of the 3rd International Workshop on AST. — 2008. — P. 33–40.
31. Летичевский А.А., Летичевский А.Ал., Песчаненко В.С., Губа А.С. Генерация символьных трасс в системе инсерционного моделирования // Кибернетика и системный анализ. — 2015. — **51**, № 1. — С. 7–19.
32. Летичевский А.А., Годлевский А.Б., Летичевский А.А. (мл.), Потиенко С.В., Песчаненко В.С. Свойства предикатного трансформера системы VRS // Кибернетика и системный анализ. — 2010. — № 4. — С. 3–16.

Поступила 12.01.2015