Генерация тестового набора на основе модели потока управления

П. Д. Дробинцев,
Санкт-Петербургский
государственный
политехнический университет

В. П. Котляров
Санкт-Петербургский
государственный
политехнический университет
vpk@spbstu.ru

И. В. Никифоров
Санкт-Петербургский
государственный
политехнический университет
i.nikiforov@ics2.ecd.spbstu.ru

Аннотация— **Статья посвящена подходу к генерации тестового набора в соответствии с широко известными структурными критериями покрытия на основе использования модели потока управления. Подход основан на автоматизированной генерации тестов использующей символьную верификацию. Отличительной особенностью подхода является сокращение количества генерируемых тестов за счет учета данных о потоке управления и уменьшение пространства состояний подлежащих обходу при генерации тестового набора инструментом верификации. В статье представлены основные идеи подхода, описана формальная модель потока управления и инструментальные средства для работы с ней, а также приведены результаты применения подхода в нескольких проектах по разработке промышленного программного обеспечения.**

Ключевые слова — автоматизация тестирования, формальная модель, критерии покрытия

I. Введение

Сложность разработки и тестирования современного программного обеспечения (ПО) приводит к необходимости создания новых подходов позволяющих полностью или частично автоматизировать данный процесс. Одним из наиболее активно развивающихся направлений является использование технологий верификации для обеспечения качества ПО. При этом инструмент верификации часто используется для проверки свойств модели программы, заданной на каком либо формальном языке. В силу того, что при данном подходе проверяется не реализация приложения, а только его модель, становится очевидным, что одной лишь проверки на модели недостаточно, а необходимо также проведение тестирования на основе результатов верификации. Такой подход носит название тестирования на основе моделей. Однако подобное тестирование сопряжено со следующими сложностями:

- Мощность инструментов верификации позволяет получить огромное количество сценариев (трасс), описывающих поведение системы. Полученные трассы используются для генерации тестов. Однако использование этого множества ограничено возможностями по прогону тестов на тестируемом приложении, и если в случае использования инструмента верификации получение 100 000 трасс не является проблемой, то для тестирования использование такого количества тестов невозможно в силу временных ограничений, связанных с ограничением времени, отведенного на цикл тестирования.
- При использовании инструмента верификации для генерации тестов программных систем возникает взрыв количества состояний, подлежащих обходу при получении тестового сценария. Это обусловлено тем, что состояние формируется как совокупность значений всех переменных, используемых в тестируемом приложении (их количество может быть огромным), поэтому, обеспечивая перебор значений, инструмент верификации сталкивается с необходимостью рассмотрения огромного количества состояний.

Данная работа посвящена рассмотрению подхода, позволяющего решить вышеуказанные проблемы за счет использования модели потока управления описанной на высокоуровневом графическом языке моделирования. Предлагаемый подход позволяет выделить подмножество тестов в соответствии с различными критериями тестирования, а также существенно сократить пространство состояний при работе верификатора.

II. Формальные модели

Индустрия промышленного производства программных продуктов в настоящее время достигла таких масштабов, что для обеспечения соответствующего качества, традиционных способов ручного тестирования недостаточно, индустрия требует как высокого качества тестирования, так и достаточного уровня его автоматизации. Тестирование на основе моделей способно предоставить приемлемый уровень качества и автоматизации.

Создание любой системы начинается с построения модели и изучения её свойств. Модели, которые описывают структуру системы, часто называют

архитектурными, а модели, которые описывают поведения системы – поведенческими. Модель – формализованное описание структуры и поведения системы [1]. Модель может описываться в терминах состояния системы, входных воздействий на нее, конечных состояний, потоков данных и потоков управления, возвращаемых системой результатов и т.д.

Большое количество ошибок при составлении требований возникают из-за неоднозначной интерпретации их заказчиком и разработчиком. Чтобы избежать таких ошибок используют формальные спецификации. Формальная спецификация представляет собой законченное описание модели системы и требований к ее поведению в терминах той или иной формальной нотации. Для описания характеристик системы возможно использовать несколько моделей в рамках нескольких формализмов. Обычно, чем более абстрактной является нотация моделирования, тем больше трудностей возникает при автоматизации тестирования программы на основе модели/спецификации, описанной в этой нотации. Одни нотации и языки больше ориентированы на доступность и прозрачность описания, другие – на последующий анализ и трансляцию, в частности, трансляцию спецификации в тест.

Одной из серьезных проблем при создании формальной модели системы на основе неформализованных исходных требований является их неоднозначная интерпретация заказчиком и разработчиком. На начальном этапе разработки ПО как у заказчика, так и исполнителей может отсутствовать четкое представление об архитектуре будущей системы, есть лишь общее представление о её поведении. Но уже на начальном этапе дизайна необходимо находить возможные ошибки и несоответствия требованиям. Большинство инструментов тестирования на основе моделей используют формальные нотации достаточно низкого уровня. Такие нотации как SDL [36], UML [37], конечные автоматы и др. вынуждают пользователя сразу разрабатывать поведение системы на достаточно детальном уровне. Как показывает практика, использование подобных нотаций для контроля заказчиком этапа дизайна проектируемой системы является неудобным, а детализация слишком низкоуровневой (избыточной).

Решением данной проблемы является использование более высокоуровневой формальной нотации, которая бы стала связующим звеном между исходными требованиями и формальной моделью, с которой непосредственно работают методы верификации и тестирования на основе моделей [2].

III. Методы создания тестовых сценариев на основе модели

В настоящее время методы тестирования на основе моделей используют множество типов моделей и техник, которые можно условно разделить на [3]:

A. Методы проверки согласованности автоматов и систем переходов.

Такие методы относятся к одному из трех типов, в зависимости от используемых моделей.

- Обычные и расширенные автоматы [4,5]. Методы построения тестов на основе конечных автоматов наиболее глубоко разработаны, известны их точные ограничения и гарантии полноты выполняемых проверок. Методы, использующие расширенные автоматы, сводят их к обычным, но применяют более детальные критерии покрытия, основанные на использовании данных в расширенных автоматах. Наиболее известными инструментами создания тестов на основе таких моделей являются BZ-TT [6,5], использующий модели, описанные на языке B [7], и GOTCHA-TC Beans [8,9], или UML State charts в рамках набора инструментов Telelogic [10].

- Системы переходов [19]. Такие методы чаще используются при тестировании распределенных систем, поскольку моделирование таких систем с помощью конечных автоматов очень трудоемко. Большинство этих методов не определяют практически применимых критериев полноты и не дают конечных тестовых наборов для реальных систем, поэтому использующие их инструменты опираются на те или иные эвристики для обеспечения конечности набора тестов. Первые методы построения тестов по LTS-моделям были разработаны в работах Бринксмы [11] и Тритманса [12]. Наиболее известны из инструментов, созданных на основе результатов Тритсманса, TorX [13] и TGV [14]. Помимо обычных систем переходов используются и временные (расширенные с помощью таймеров), построение тестов на их основе возможно с использованием инструмента UPAAL [15]. Примерами инструментальной поддержки таких методов являются инструменты технологии UniTESK [16,17], созданной в ИСП РАН, и инструмент SpecExplorer [18,19], разработанный в Microsoft Research.

B. Методы построения тестов на основе формального анализа свойств ПО

Данные методы используют формальный анализ для классификации тестовых ситуаций и нацеленной генерации тестов.

- Методы на основе классов эквивалентности [20,21,22]. В таких методах тестовые ситуации выбираются как представители классов эквивалентности, задаваемых критерием покрытия. Часто за основу разбиения выбирается используемый критерий покрытия, а ситуации, которые соответствуют одному покрываемому элементу в рамках этого критерия, считают эквивалентными. Далее тесты строятся так, чтобы в каждом классе эквивалентности был хотя бы

один тест. К данному классу методов относится, например, метод функциональных диаграмм Майерса [23].

- Методы на основе дедуктивного анализа (например, [24]). В этих методах выбираемые тестовые ситуации соответствуют особым случаям в дедуктивном анализе свойств тестируемой системы. Метод дедуктивного анализа (theorem proving) – проверка того, что набор утверждений, представляющий спецификацию, формально выводится из реализации и, быть может, каких-то гипотез о поведении окружения системы, сформулированных в том же формализме, что и реализация [3].

C. Методы построения тестов с помощью символьного выполнения (symbolic execution)

Обычно подобные методы используют символьное описание пути, содержащее наборы предикатов, данный путь проходит во время выполнения теста по коду программы (или пути по проверяемым формальным спецификациям). Такое описание позволяет выбирать новые тестовые ситуации так, чтобы они покрывали другие пути и строить тесты с помощью техник разрешения ограничений (constraint solving) [25,26]. Символьное выполнение в комбинации с конкретизацией тестовых данных используется и для построения тестов, нацеленных на типичные дефекты, такие, как использование неинициализированных объектов, тупики или гонки параллельных потоков [27].

Более детальная информация о методах тестирования на основе моделей содержится в [28,29].

Следует отметить, что наиболее развиты методы тестирования на основе моделей первой из перечисленных выше групп. Поддерживающие их инструменты все шире используются в промышленных проектах. Методы построения тестов на основе проверки моделей и на основе символьного выполнения активно развиваются и реализуются в инструментальных средствах.

IV. Предлагаемый подход

Рассмотрим более детально этапы работы методов тестирования на основе модели с применением методов верификации. При использовании верификации на первом этапе проверяются свойства разработанной формальной модели ПО и на базе полученных результатов вносятся корректировки в требования и программный код. По сути, происходит формальный анализ заданных свойств ПО, описанный в предыдущем разделе. В процессе анализа, большинство средств верификации предоставляет возможность получения, так называемого контр-примера, который описывает поведение системы, приводящее к нарушению заданного пользователем свойства. Данный контр-пример полезно использовать при тестировании, для проверки того факта, что внесенные в код системы, на основе результатов верификации изменения, привели к исправлению ее поведения и заданное свойство больше не нарушается при функционировании ПО. Тестирование с помощью контр примеров позволяет получить тесты, проверяющие достаточно узкий класс поведений и не обеспечивающие полноты покрытия, таким образом, данные тесты рациональнее всего использовать как дополнение к регрессионному тестированию, проводимому после внесения в код системы исправлений, вызванных анализом нарушения её свойств.

Альтернативным подходом к получению тестов является их генерация в процессе обхода дерева поведения системы, при этом для сокращения количества генерируемых тестов часто используются методы символьной верификации, позволяющие получать в параметрах теста не конкретные значения переменных, а их допустимые диапазоны. Одной из основных задач при использовании подобной генерации является задача определения критериев, которым должны соответствовать тесты для обеспечения необходимого покрытия.

В рассматриваемом в рамках работы подходе используется генерация тестов посредством обхода дерева поведения системы на основе модели потока управления с использованием методов символьной верификации и ограничением количества тестов структурными критериями,

На втором этапе проводится тестирование кода с помощью тестов, полученных на предыдущем этапе. Для данного этапа характерными задачами является конкретизация параметров теста, в случае использования символьной верификации, обеспечение взаимодействия теста с приложением и анализ результатов выполнения тестов.

Предлагаемый подход, условно можно разделить на следующие составные части:

- Ручная формализация модели в высокоуровневой графической нотации. Предлагается использовать язык UCM, позволяющий формально определить потоки управления в модели, и в тоже время являющийся достаточно высокоуровневым языком доступным для понимания специалистами из предметных областей разрабатываемого ПО.
- Верификация модели и ее корректировка. На данном этапе используется система верификации VRS [38], позволяющая проводить верификацию с использованием метода проверки на модели и символьной верификации.
- Генерация тестовых сценариев по модели. На основе обхода дерева поведения проводится генерация символьных тестов, в соответствии с определенным пользователем структурным критерием. Тесты генерируются на языке MSC [35], представляющем собой диаграммы обмена последовательностями сообщений.
- Генерация исполняемых тестов. Разделяется на два этапа: на первом подготовительном происходит конкретизация тестов соответствующими значениями переменных, на

втором происходит генерация тестов с использованием системы тестирования TAT[38] на целевой язык.

- Исполнение тестов и оценка результатов. Данный этап связан с исполнением тестов и анализом полученных логов, представленных в формате MSC.

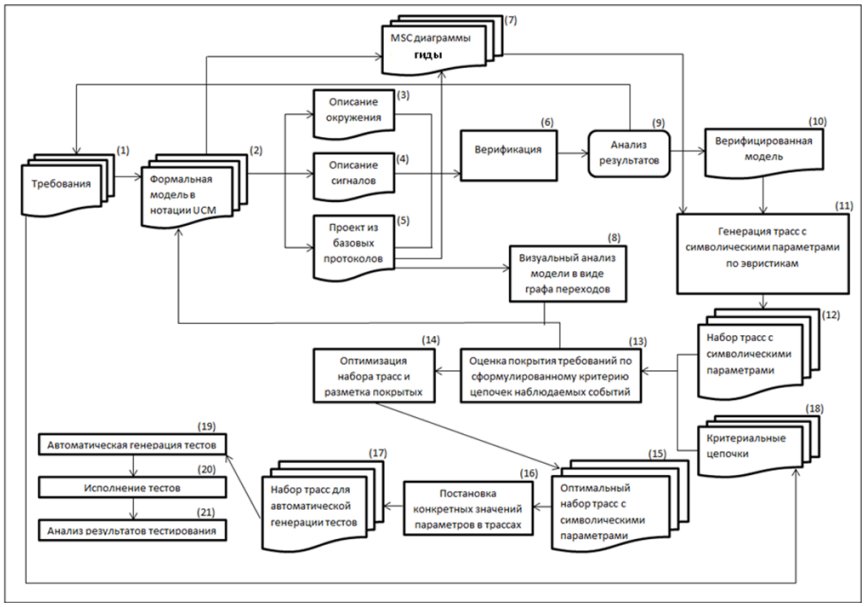

Рис 1. Технологическая цепочка предлагаемого подхода.

Реализация описанного подхода приведена на рис. 1 в виде технологической цепочки, описывающей совместное использование инструментов VRS и TAT. В рамках цепочки можно выделить следующие основные этапы:

- По исходным требованиям (1) вручную создается формальная модель системы в нотации UCM(2).
- С помощью инструмента UCM2BP [2] производится трансляция UCM проекта в набор базовых протоколов [38] (элементарных MSC-диаграмм, каждая из которых описывает законченное действие в пространстве поведения модели) (5), файлы описания сигналов (4) и окружения (3).
- Средствами VRS осуществляется верификация модели (6).
- В случае нахождения ошибок или неточностей в модели (например, недетерминированное поведение, тупиковые или недостижимые состояния и т.п.) требования исправляют, что приводит к изменениям в формальной модели.
- С помощью инструмента UCM2MSC производится генерация MSC диаграмм (7), которые транслируются в гиды [32], пригодные для инструмента VRS.
- По верифицированной модели(10) и гидам (7) генерируются трассы (12).
- Если трассы содержали символические параметры, осуществляется подстановка конкретных значений (16).
- Средствами TAT выполняется генерация тестов (19) по набору трасс (17) и исполнение тестов (20) с последующим анализом результатов тестирования (21).

Для сокращения пространства состояний анализируемой модели будущего ПО предлагается использование гидов. Гиды определяет те точки поведения системы, которые наиболее интересны пользователю и таким образом система верификации получает подсказку дающую информацию о том, через какие точки поведения должна пройти символьная трасса. Одним из наиболее эффективных методов автоматического определения гидов является их генерация на основе структурных критериев покрытия, например, критерия ветвей [40]. При этом UCM модель системы может быть использована как модель потока управления. Данные методы, позволяющие осуществлять автоматическую генерацию гидов, реализуются на 5 шаге описанной технологической цепочки, а именно, в инструменте UCM2MSC.

Как было указано выше, генерация гидов производится по формальной модели системы в нотации UCM, но так как генерация тестовых трасс производится в верификаторе VRS, то гиды реализуются в терминах модели базовых протоколов. Для этого используются инструмент UCM2BP, который отвечает за трансляцию UCM нотации в нотацию базовых протоколов. Рассмотрим особенности использования нотации UCM для реализации модели потока управления.

V. Модель потока управления в нотации UCM

Использование UCM для создания формальной модели поведения системы подразумевает описание последовательности действий, которые может производить система в ответ на внешние воздействия пользователей или систем окружения. Варианты использования отражают не только функциональность системы с точки зрения описания ее архитектуры, но и определяют особенности потока управления, лежащего в основе будущего приложения. Явное представление вариантов (режимов функционирования) поведения приложения привносит важную информацию в процесс разработки программных систем [30]:

- заполняет промежуток между словесным описанием требований и детальным дизайном системы;
- позволяет разработать архитектуру системы на высоком уровне абстракции, а так же задать поведение системы на полученной архитектуре;
- помогает разработчику предсказывать поведение сложных систем;
- предоставляет удобную нотацию, позволяющую изображать на диаграмме параллельные процессы, таймеры, точки прерывания, аспекты.

Дизайн системы в нотации UCM представляет собой набор взаимодействующих между собой диаграмм. В свою очередь каждая из диаграмм сосредоточена на описании взаимодействия компонентов (агентов, процессов системы), объектов, наблюдателей и подсистем. Каждый компонент или подсистема содержит элементы ответственности (Responsibilities), соответствующие событиям в системе, а так же упорядоченную последовательность их возникновения, которая, по сути, является описанием потока управления. Таким образом, совокупность компонентов и диаграмм дает пользователю наглядное представление поведения системы и взаимодействий между ее компонентами. Разработка UCM диаграмм осуществляется с использованием графического редактора UCMNavigator [31].

Формализация поведения системы дополняется использованием метаданных [2], которые могут быть добавлены практически ко всем элементам UCM диаграммы. Метаданные представляют собой набор данных формата: «Маркер» - «Текстовая информация заданного типа», в рамках которого происходит определение операций с переменными модели, а также прием и передача сигналов для обмена с окружением.

VI. Инструментальная реализация

В технологической цепочке VRS/TAT за генерацию тестовых трасс отвечает инструмент VRS. Технология VRS использует нотацию базовых протоколов, что не обеспечивает UCMNavigator .

Предлагаемый подход использует инструмент UCM2BP, отвечающий за автоматическую трансляцию UCM проекта в нотацию базовых протоколов. UCM2BP позволяет получить набор базовых протоколов, которые используются в качестве шагов MSC сценария.

UCM диаграмма представляет собой структурированный граф. Задача обхода UCM диаграммы и построения MSC сценариев сводится к обходу графа. При увеличении размера UCM проекта, количество возможных сценариев растет экспоненциально, что приводит к так называемой проблеме «взрыва состояний». Для решения данной проблемы в разработанном подходе используются:

- структурный критерий покрытия: критерий ветвей
- гиды с глубиной

A. *Критерий покрытия*

Современные методы тестирования обычно оценивают полноту покрытия тестовым набором по таким критериям, как число выполненных операторов программы, ветвей, путей, проверенных значений данных, покрытие граничных значений функции, переходов между состояниями и т.д. В разработанном подходе для автоматической генерации набора MSC сценариев используется структурный критерий ветвей.

Структурные критерии покрытия обеспечивают оценку полноты тестирования и основаны на структуре тестируемой системы. В основе структурных критериев полноты лежит простая идея: если ошибка находится в какой-то конструкции кода или компоненте тестируемой системы, то тест, инициировавший выполнение этой конструкции или компоненты, скорее всего, сможет её обнаружить.

Условие критерия тестирования ветвей (критерий P2 [40]) – набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза. Это достаточно сильный и при этом экономичный критерий, поскольку множество ветвей тестируемой системы конечной не так уж велико. UCM нотация позволяет осмысленно выделить ветви на диаграмме - участки пути между операторами ветвлений (OrFork, OrJoin, AndFork, AndJoin, Waiting place, Timer), а также стартовыми (StartPoint) и конечными (EndPoint) UCM элементами.

Выбор критерия покрытия по ветвям позволяет производить генерацию конечного числа тестовых сценариев, причем количество полученных сценариев меньше или равно общему количеству ветвей всей системы. Такой подход позволяет добиться адекватности набора тестов и хорошей скорости генерации тестовых сценариев.

B. *Использование гидов*

Использование гидов позволяет пользователю задавать направление поиска трасс для тестовых сценариев при обходе пространства поведения модели. Метод подразумевает наложение ограничений на размер тестового сценария, что дает возможность проверить его допустимость. Критерии покрытия формулируют дополнительные ограничения на поиск, отсекая ветви поведения модели, не удовлетворяющие тестовому сценарию. Подробное описание концепции и реализации метода гидов приведено в [32].

Основная идея представлена на рис. 2, где изображено дерево поведения модели, построенное на основании множества базовых протоколов, описывающих

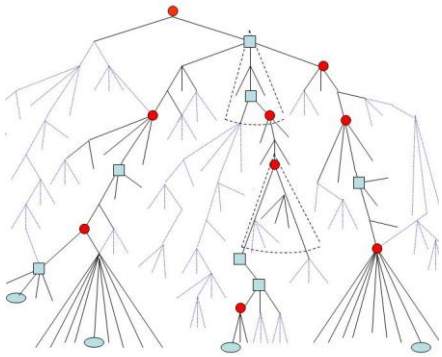

Рис 2. Применение гидов при обходе дерева поведения.

модель (вершины – состояния модели, переходы – базовые протоколы).

Пользователь, зная, какие именно элементы дерева поведения модели (сигналы, изменения атрибутов и состояний и т.п.) должны быть включены в тестовый сценарий, чтобы соответствующее требование было покрыто, может указать определенные состояния и переходы, т.е. задать промежуточные точки (отмечены кружками), через которые осуществляется трассовая генерация. Таким образом, существенно ограничивается пространство поиска от одного события (промежуточной цели) в тестовом сценарии до другого. При необходимости путь от начального события до конечного может быть уточнен дополнительными событиями (отмечены квадратами на рис. 2) конкретизирующими путь в дереве поведений. Разработанные методы позволяют создавать гиды по UCM диаграммам. С помощью UCM2BP UCM диаграммы транслируются в набор базовых протоколов, а имена, соответствующих UCM элементам, протоколов используются для построения гидов.

Чтобы получить желаемые трассы в инструменте VRS, используются созданные гиды. Гиды задают направление, в котором VRS обходит дерево поведения системы. VRS позволяет использовать глубины в гидах, т.е. задавать участки трассы, в которых переход от одного базового протокола к другому возможен с использованием нескольких путей содержащих между искомыми базовыми протоколами несколько промежуточных протоколов, количество которых ограничивается глубиной.

Использование глубин возможно в нескольких ситуациях:

- для сокрытия альтернативного выбора,
- для сокрытия цикла,
- для сокрытия параллелизма.

Пример использования гидов приведен на рис. 3. В примере, если существует необходимость сгенерировать сценарий, который направлен на покрытие ветви содержащей элемент R6, то без анализа метаданных неизвестно какой элемент будет использован R1 или R2, а также будет ли задействован цикл R5. В таком случае, чтобы учесть все варианты путей, которые могут привести к применению R6 необходимо построить 4 гида:

- R0, R1, R3, R4, R6;
- R0, R1, R3, R4, R5, R4, R6;
- R0, R2, R3, R4, R6;
- R0, R2, R3, R4, R5, R4, R6.

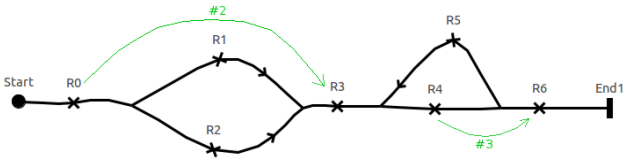

Рис 3. Пример использования глубины в гидах.

Однако, если указать глубины в гиде, то можно построить лишь один гид: R0, R3#2, R6#3 (# - разделитель имени элемента и величины глубины). Данный гид указывает направление в дереве поведения и интерпретируется следующим образом: применить R0, затем применить не более 1-го протокола, прежде чем применить R3, а затем применить R6, применив при этом не более 3-х протоколов, выбор которых осуществляет система верификации.

Таким образом, использование критерия ветвей и гидов с глубиной позволяет получить набор сценариев, где каждой ветви соответствует один гид, по которому будет сгенерирована одна трасса. Такой подход решает проблему «взрыва состояний», а также существенно сокращает время генерации сценариев.

C. Автоматическая генерация на основе критерия ветвей

- Генерация базовых протоколов по UCM проекту.
- Сопоставление базовых протоколов и элементов UCM диаграммы.
- Выделение ветвей в структуре UCM.
- Генерация гида для каждой ветви.
- Произведение оптимизации - удаление избыточных диаграмм.

Использование программных объектов базовых протоколов при генерации с помощью UCM2BP позволяет сохранить структуру UCM. Программные объекты базовых протоколов содержат ссылки на предыдущие и последующие протоколы, сгенерированные в соответствие со структурой UCM диаграммы. Также они содержат ссылки на элементы UCM. Все это позволяет однозначно интерпретировать структуру UCM проекта при генерации MSC диаграмм в терминах модели базовых протоколов.

UCM диаграмма (рис.4) состоит из следующих ветвей: R1,R2,R3,R4,R5,R6,R7.

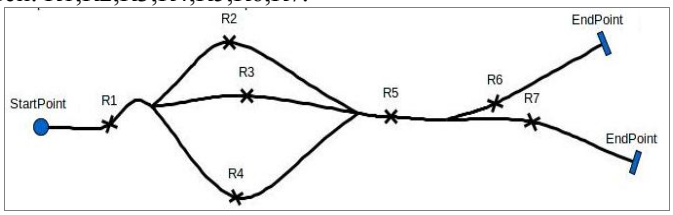

Рис 4. Пример UCM диаграммы.

Для каждой ветви генерируется гид, который содержит в себе путь от стартового состояния системы до данной ветки. Для примера на рис.4 будут сгенерированы следующие гиды:

R1
R1 R2
R1 R3
R1 R4
R1 R5(#2)
R1 R5(#2) R6
R1 R5(#2) R7

Отметим, что получившийся набор гидов является избыточным, так как главной задачей при использовании критерия ветвей является покрытие каждой ветви не менее одного раза. Так как некоторые ветки покрываются в нескольких гидах, то следует провести оптимизацию, и удалить гиды с повторяющимися ветками. Таким образом, после оптимизации, состоящей из удаления избыточных гидов, будет получен следующий набор:

R1 R2
R1 R3
R1 R4
R1 R5(#2) R6
R1 R5(#2) R7

Данного набора гидов достаточно для генерации тестовых трасс удовлетворяющих структурному критерию покрытия по ветвям. Следует отметить, что использование верифицированной модели базовых протоколов для генерации трасс дает уверенность в корректности результата.

Несложно заметить, что некоторые из полученных гидов не ведут процесс генерации трасс до конечных состояний системы. В то же время получение «длинного» гида (до конечного состояния) дает возможность формирования теста, приводящего систему в заранее известное состояние, определяющее окончание работы системы. Подобная генерация также может быть автоматизирована. Для рассматриваемого примера, набор «длинных» гидов будет следующим:

R1 R2 R5 R6
R1 R2 R5 R7
R1 R3 R5 R6
R1 R3 R5 R7
R1 R4 R5 R6
R1 R4 R5 R7
R1 R5(#2) R6
R1 R5(#2) R7

Количество гидов увеличилось, это объясняется тем, что на уровне генерации гиды без анализа метаданных не дают возможность определить в какое из альтернативных конечных состояний придет система. Поэтому для сохранения покрытия по критерию ветвей, «короткие» гиды продлеваются до всех возможных конечных состояний. Данная особенность приводит к ситуации, когда количество гидов становится избыточным, причем не все из них могут быть покрыты соответствующими трассами. Это происходит из-за несоответствия потока управления описанного на диаграмме метаданным, указанным в элементах responsibility. В результате избыточные гиды, по которым не сгенерируется трасса, должны быть исключены из тестового набора.

В результате автоматической генерации на основе анализа модели потока управления получается набор тестов, которые направлены на обнаружение ошибок и удовлетворяют структурному критерию покрытия ветвей.

VII. Результаты применения подхода

Результаты применения описанного подхода в четырех проектах среднего размера по разработке модулей телекоммуникационных приложений приведены в таблице 1. В качестве документации на проект были использованы UCM диаграммы, а выходом являлся готовый тестовый набор, для прогона на реальном коде приложения.

Все результаты по проектам были получены с помощью автоматического режима для получения набора трасс покрывающих модель по критерию ветвей с использованием «длинных» гидов, что позволило ускорить разработку тестовых сценариев в 250 раз по сравнению с традиционным ручным подходом.

Таблица 1. Результаты применения

Название проекта	Кол-во базовых протоколов	Кол-во ветвей	Кол-во гидов	Кол-во сгенерированных символьных трасс
Проект 1	358	149	437	372
Проект 2	163	240	139	131
Проект 3	191	111	87	72
Проект 4	214	200	118	104

В тоже время результаты, полученные в рамках применения описанного подхода, свидетельствуют о более чем 70%-ном сокращении временных затрат по сравнению с подходом основанным на простом обходе дерева поведения системы без использования гидов.

Список литературы

[1] IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Standard 610.12-1990, IEEE Standard, IEEE, NY (1990).

[2] Никифоров И.В. Система автоматического проектирования требований, заданных в нотации UCM // Диссерт. на соискание уч. ст. магистра. СПб.: СПбГПУ, 2011.

[3] Кулямин В.В. Методы верификации программного обеспечения // Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению “Информационно-телекоммуникационные системы”. 2008. 117 с. http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf

[4] Broy M., Jonsson B., Katoen J.-P., Leucker M., Pretschner A. (eds.). Model Based Testing of Reactive Systems. LNCS 3472, Springer, 2005. 659 p.

[5] Utting M., Legeard B. Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann, 2007. 456 p.

[6] Ambert F., Bouquet F., Chemin S., Guenaud S., Legeard B., Peureux F., Vacelet N., Utting M. BZ-TT: A tool-set for test generation from Z and B using constraint logic programming. Proc. of FATES’2002 – August 2002. P. 105-119.

[7] Abrial J.-R. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996. 813 p.

[8] Farchi E., Hartman A., Pinter S. S. Using a model-based test generator to test for standard conformance. IBM Systems Journal, 41(1), 2002. P. 89-110.

[9] GOTCHA-TCBeans // www.haifa.ibm.com/projects/verification/gtcb/index.html

[10] http://www-03.ibm.com/software/products/en/ratitau

[11] Brinksma E. A theory for the derivation of tests. Proc. of 8-th International Conference on Protocol Specification, Testing and Verification, North Holland, 1988. P. 63-74.

[12] Tretmans J. A Formal Approach to Conformance Testing. PhD thesis, University of Twente, Enschede, The Netherlands, 1992. 282 p.

[13] Tretmans J., Belinfante A. Automatic testing with formal methods. Proc. Of 7-th European Conference on Software Testing, Analysis and Review, Barcelona, Spain, November 1999. P. 8-10.

[14] Fernandez J.-C., Jard C., Jeron T., Viho C. Using On-the-Fly Verification Techniques for the Generation of Test Suites. Proc. of 8-th International Conference on Computer Aided Verification, LNCS 1102, Springer, 1996. P. 348-359.

[15] UPPAAL // http://www.uppaal.com/

[16] Кулямин В.В., Петренко А.К., Косачев А.С., Бурдонов И.Б. Подход UniTesK к разработке тестов // Программирование. – 2003. - №29(6). С. 25-43.

[17] UniTESK // http://www.unitesk.ru/

[18] Spec Explorer: Microsoft Research // http://research.microsoft.com/specexplorer

[19] Spec Explorer: MSDN//http://msdn.microsoft.com/en-us/library/ee620411.aspx

[20] Ammann P., Black P. E. Abstracting formal specifications to generate software tests via model checking. Proc. of 18-th Digital Avionics Systems Conference, IEEE, October 1999 – Vol.2. P. 10.A.6-1-10.A.6-10.

[21] Gargantini A., Heitmeyer C. Using model checking to generate tests from requirements specifications. ACM SIGSOFT Software Engineering Notes, 24(6), November 1999. P. 146-162.

[22] Visser W., Pasareanu C. S., Khurshid S. Test input generation with Java PathFinder. ACM SIGSOFT Software Engineering Notes, 29(4), July 2004. P. 97-107.

[23] Myers G. J. The Art of Software Testing. John Wiley & Sons, 1979. 192 p.

[24] Engel C., Hahnle R. Generating unit tests from formal proofs. Y. Gurevich, B. Meyer, eds. Proc. of TAP 2007. LNCS 4454, Springer-Verlag, 2007. P. 169–188.

[25] Boyapati C., Khurshid S., Marinov D. Korat: automated testing based on Java predicates. Proc. of International Symposium on Software Testing and Analysis, 2002. P. 123–133.

[26] Gotlieb A., Botella B., Rueher M. Automatic test data generation using constraint solving techniques. ACM SIGSOFT Software Engineering Notes, 23(2), 1998. P. 53-62.

[27] Sen K., Agha G. CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools. Proc. of Computer Aided Verification, Aug 2006. P. 419-423.

[28] Бурдонов И.Б., Косачев А.С., Пономаренко В.Н., Шнитман В.З. Обзор подходов к верификации распределенных систем. Препринт Института системного программирования РАН, № 16, 2006. 61 с.

[29] Utting M., Pretschner A., Legeard B. A Taxonomy of Model-Based Testing. Technical Report, Department of Computer Science, The University of Waikato, New Zealand, 2006. 17 p.

[30] Buhr R. J. A., Casselman R. S. Use Case Maps for Object-Oriented Systems, PrenticeHall, 1995. 302 p.

[31] Use Case Map Navigator -http://jucmnav.softwareengineering.caG. Eason, B. Noble, and I.N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529-551, April 1955. (references)

[32] Колчин А.В. Разработка инструментальных средств для проверки формальных моделей асинхронных систем. Диссерт. на соискание уч. ст. канд. физ.-мат. наук. Киев, 2009. 140 с.

[33] Brinksma E. A theory for the derivation of tests. Proc. of 8-th International Conference on Protocol Specification, Testing and Verification, North Holland, 1988. P. 63-74.

[34] Tretmans J. A Formal Approach to Conformance Testing. PhD thesis, University of Twente, Enschede, The Netherlands, 1992. 282 p.

[35] ITU Recommendation Z.120. Message Sequence Charts (MSC), 11/99

[36] ITU-T z.100 (08/2002)// Telecommunication standardization sector of UTI, 202 с.– http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-Z.100-200208-I

[37] http://www.uml.org/

[38] Дробинцев П.Д. «Интегрированные технология обеспечения качества программных продуктов с помощью верификации тестирования» Диссертация на соискание степени кандидата технических наук 2006

[39] Коликова Т.В., Котляров В.П. Основы тестирования программного обеспечения. ISBN: 5-9556-0027-2 2006г. 285 с.

[40] Beizer B. Software testing techniques. Second Edition. International Thomson Computer Press 1990 ISBN 1850328803 580 pages.