Модел. и анализ информ. систем. Т. **20**, № **6** (2013) 64–77

УДК 004.415

Автоматизация создания верифицированных тестовых сценариев на основе гидов

Дробинцев П.Д.*, Котляров В.П.*, Летичевский А.А.**

** Санкт-Петербургский государственный политехнический университет, 195251, Россия, г. Санкт-Петербург, ул. Политехническая, 29*
*** Институт кибернетики им. В.М.Глушкова НАН Украины, 03680МСП, Украина, г. Киев, просп. Академика Глушкова, 40*

e-mail: vpk@spbstu.ru, let@cyfra.net

получена 10 ноября 2013

Ключевые слова: символьная верификация, автоматизация тестирования, конкретизация тестовых сценариев, предикатный трансформер.

В данной работе представлен обзор технологии автоматизированной генерации тестовых сценариев на основе гидов, использование которой позволяет существенно повысить качество разрабатываемого программного обеспечения. В качестве обоснования создания технологии описаны основные проблемы, возникающие при разработке и тестировании крупных промышленных систем, и представлены методики проверки программного обеспечения на соответствие требованиям. Продемонстрированы возможности инструментальных средств по автоматической и полуавтоматической генерации тестового набора с использованием формальной модели, созданной на языке UCM, и средств верификации и автоматизации тестирования.

1. Введение

Среди основных проблем автоматизации разработки и тестирования программного обеспечения промышленных приложений отмечается проблема обработки сложных и больших по объему спецификаций требований. Документы, фиксирующие спецификации требований, пишутся, как правило, на естественном языке и могут содержать сотни и тысячи пунктов требований. В силу этого задача формализации требований для описания поведенческих сценариев, используемых для разработки автоматических тестов или ручных тестовых процедур, характеризуется как задача огромной сложности и трудоемкости.

Применимость формальных методов в промышленности в огромной степени определяется тем, насколько адекватен язык формализации в принятой инженерной практике, в которую вовлечены не только собственно разработчики кода и тестировщики, но и заказчики, руководители проектов разных уровней, маркетологи и другие специалисты. Ясно, что никакой чисто логический язык не подходит для

адекватной формализации требований, которая бы одновременно сохранила семантику разрабатываемого приложения и удовлетворяла всех «причастных лиц» [1].

В современной проектной документации формулировка исходных требований задается либо *конструктивно*, когда из текста требования на естественном языке удается реконструировать процедуру контроля или сценарий проверки выполнения данного требования, либо *неконструктивно*, когда заданное в требовании свойство не содержит пояснения способа его проверки. Процедура проверки неконструктивных требований сводится к конструктивной, для чего необходимо создать согласованный с заказчиком сценарий их проверки.

2. Проверка выполнения требований

Процедура проверки требования — это точная последовательность причин и следствий некоторых активностей (кодируемых действиями, сигналами, состояниями), в результате анализа которой можно утверждать, выполнено данное требование или нет. Подобная процедура проверки может быть использована в качестве критерия выполнения конкретного требования, т.е. стать т.н. критериальной процедурой. В дальнейшем изложении для критериальной процедуры будем использовать термин последовательность или «цепочка» событий.

Отслеживая в поведенческом сценарии системы (гипотетическом, реализованном в модели или в реальной системе) факты выполнения критериальной процедуры, можно утверждать, что соответствующее требование в анализируемой системе удовлетворено.

Процедура проверки требования (цепочка) формулируется путем задания для всех элементов цепочки (событий) следующей информации:

- условий (причин), требующихся для активизации некоторой активности;
- самой активности, подлежащей исполнению при данных условиях;
- следствий — наблюдаемых (измеряемых) результатов исполнения указанной активности.

Для описания причин и следствий используются сигналы, сообщения или транзакции, обычные в коммуникациях объектов реактивной системы [2], а также состояния переменных в виде значений или ограничений на области допустимых значений. Отслеживая изменения в состояниях, производимых активностями цепочек, можно наблюдать за покрытием соответствующих цепочек. При анализе допустимо рассматривать прямой переход из состояния в состояние с пустой активностью, а в случае недетерминизма – альтернативные варианты изменения состояний.

Проблемы неконструктивного задания требований преодолеваются в процессе разработки процедур проверки выполнения требований на пользовательских или межкомпонентных интерфейсах.

Таким образом, критерием выполнения требований могут служить цепочки, содержащие последовательности событий; помимо этого, возможны случаи, в которых критерий выполнения некоторого требования задается не одной, а несколькими цепочками.

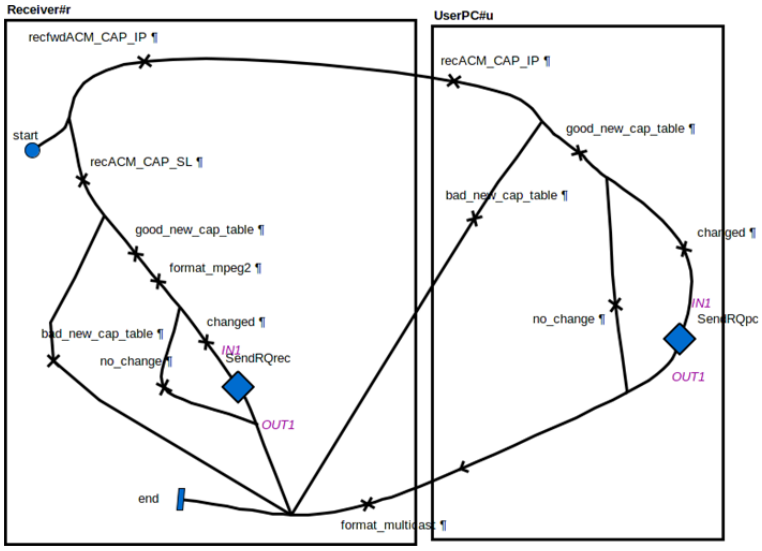


Рис. 1. UCM модель двух объектов Reciver и UserPC

В технологии VRS/TAT [3] для высокоуровневого описания модели используется нотация Use Case Maps (UCM) [4], а инструменты автоматизации проверки и генерации работают с моделью на языке базовых протоколов [5].

UCM модель (рис. 1) содержит описание модели двух взаимодействующих объектов. Каждый путь на графе от события start до события end представляет некоторый сценарий поведения. На каждом пути располагается конкретное множество событий (Responsibilities). События на диаграмме обозначены символом — ×, а элементы Stub, кодирующие вложенную диаграмму, символом ⧫. В результате каждый сценарий содержит определенную последовательность событий. Множество возможных сценариев задается множеством таких последовательностей.

В подобных терминах цепочка определяется как подпоследовательность событий, достаточных для вывода о том, что требование удовлетворено. Путь на UCM диаграмме, содержащий последовательность событий некоторой цепочки, называется трассой, покрывающей соответствующее требование. По трассе можно сгенерировать тесты, необходимые для экспериментального подтверждения выполнимости требования.

3. Матрица отслеживания

Формализация требований верификационного проекта начинается с разработки матрицы отслеживания TRM [1] — Traceability matrix (на рис. 2 TRM для конкретного проекта представлена в виде таблицы). Столбцы Identifier и Requirements содержат идентификатор требования, употребляемый в исходном документе описания требований, и полный текст требования, подлежащий формализации. Столбец Traceability содержит цепочки событий, достаточные для проверки выполнимости соответствующего требования, а столбец Traces — трассы или поведенческие сценарии, которые используются для генерации кода тестов.

Identifier	Requirements	Traceability	Traces
FREQ_GWR.1	Gateway shall transmit ACM_CAP messages repeatedly at an interval of T1 (TBD). This message will carry the ACM Capabilities table.	FREQ_GWR.3	
FREQ_GWR.2	Upon a change in the ACM Capabilities table, the Gateway shall send a new version of ACM_CAP message to the Satellite Terminal.	FREQ_GWR.3	
FREQ_GWR.3	Depending on configuration, ACM_CAP message shall be transmitted either in the MPEG2 private section or in a Multicast message sent across the satellite link.	recACM_CAP_SL recfwdACM_CAP_IP recACM_CAP_IP	FRED_GWR_3-1 FRED_GWR_3-2

Рис. 2. TRM — матрица отслеживания

Например, в строке 3 TRM в столбце Traceability приведены 2 цепочки для покрытия требования FREQ_GWR.3. Для удовлетворения требования достаточно зафиксировать посылку сигнала ACM_CAP по одному из двух возможных сценариев:

FREQ_GWR.3-1: start, **recACM_CAP_SL**, good_new_cap_table, format_mpeg2, no_chanes, end
FREQ_GWR.3-2: start, **recfwdACM_CAP_IP**, **recACM_CAP_IP**, good_new_cap_table, format_multicast, end

Следует заметить, что в сценарии обязательно перечисляются события, входящие в критерий проверки конкретного требования, иные входящие в него события могут быть опущены (как, например, в FREQ_GWR.3-2), кроме того, при создании критериальных цепочек строится модель верифицируемой функциональности, в процессе чего вводится много переменных состояния, типов, агентов и т.п.

4. Структуризация UCM модели системы

В UCM модели в качестве единицы структуризации может быть использован любой элемент или выделенная группа элементов поведения, лишь бы это позволило упростить процесс анализа исходной модели. В рамках настоящей работы в роли единиц структуризации используется элемент Stab или Region — явно выделенное множество элементов UCM диаграммы.

Определим UCM диаграмму как ориентированный граф с вершинами из множества элементов диаграммы и дугами, определяющими достижимость одного элемента диаграммы из других. То есть UCM диаграмма — это четверка:

$$UCM = (U, S, E, R) \ , \tag{1}$$

где U — конечное множество элементов; $S \subseteq U$ — множество стартовых точек; $E \subseteq U$ — множество конечных точек; $R \subseteq U \times U$ — отношение переходов, определяющее достижимость одного элемента диаграммы из других.

При анализе сценариев поведения системы в процессе верификации или тестирования используется понятие пути (π) в UCM молели, которое определяется как:

$$\pi = u_1 \to u_2 \to \ldots \to u_n \ , \tag{2}$$

где $u_i \in U$.

Причем для тестирования важно условие, что в тестовых сценариях каждый путь начинается в стартовой точке и заканчивается в конечной точке $u_1 \in S \wedge u_n \in E$.

Структурированная UCM диаграмма отличается от исходной только тем, что использует элементы Stub для декомпозиции поведения на основании функциональных или структурных критериев. При этом элемент Stub является вложенной UCM диаграммой и определяется как шестерка:

$$ST = (U_{st}, S_{st}, E_{st}, R_{st}, Sr, Er) \ , \tag{3}$$

в которой определение $U_{st}, S_{st}, E_{st}, R_{st}$ аналогично определению (1), $Sr \subseteq S_{st} \times U$ — отношение переходов, определяющее связь стартовых точек элемента Stub с элементами диаграммы верхнего уровня и $Er \subseteq E_{st} \times U$ — отношение переходов, определяющее связь конечных точек элемента Stub с элементами диаграммы верхнего уровня.

Таким образом, процесс структуризации можно определить как преобразование модели: $UCM \to UCM'$, в котором явно выделены структурные уровни $U' = U \cup ST_1 \cup ST_2 \cup \ldots \cup ST_k$.

Существует множество способов реализации подобного преобразования, однако с точки зрения анализа сценариев поведения системы, структуризацию полезно производить на основании двух дополняющих критериев — функционального и структурного.

Функциональный критерий определяется создателем формальной спецификации и подразумевает выделение в элемент Stub или Region тех элементов исходной UCM диаграммы, которые обеспечивают реализацию определенной функциональности, заданной в требованиях на программное обеспечение. Использование этого критерия требует наряду с анализом исходной спецификации требований к системе также понимания того, какими поведенческими сценариями каждое из требований может быть проверено при тестировании. Ограничением подобного подхода является обязательное участие специалиста предметной области (представителя заказчика), участвующего в формализации и утверждающего предлагаемое преобразование $UCM \to UCM'$. Заметим, что смысловая декомпозиция формализованной спецификации в настоящее время практически не поддается автоматизации и может быть выполнена только ручными методами с участием специалиста в предметной области.

На рис. 3 представлен пример 2 уровней структурированной UCM диаграммы, описывающей протокол взаимодействия между спутником и оператором, предоставляющим IP услуги на основе стандарта ETSI [6].

В данном примере в соответствии с требованиями поведение системы разбито на компоненты первого уровня top и второго configure и message_loss.

Структурный критерий в отличие от функционального использует структуру самой UCM диаграммы для построения элементов Stub (Region) и проведения деком-

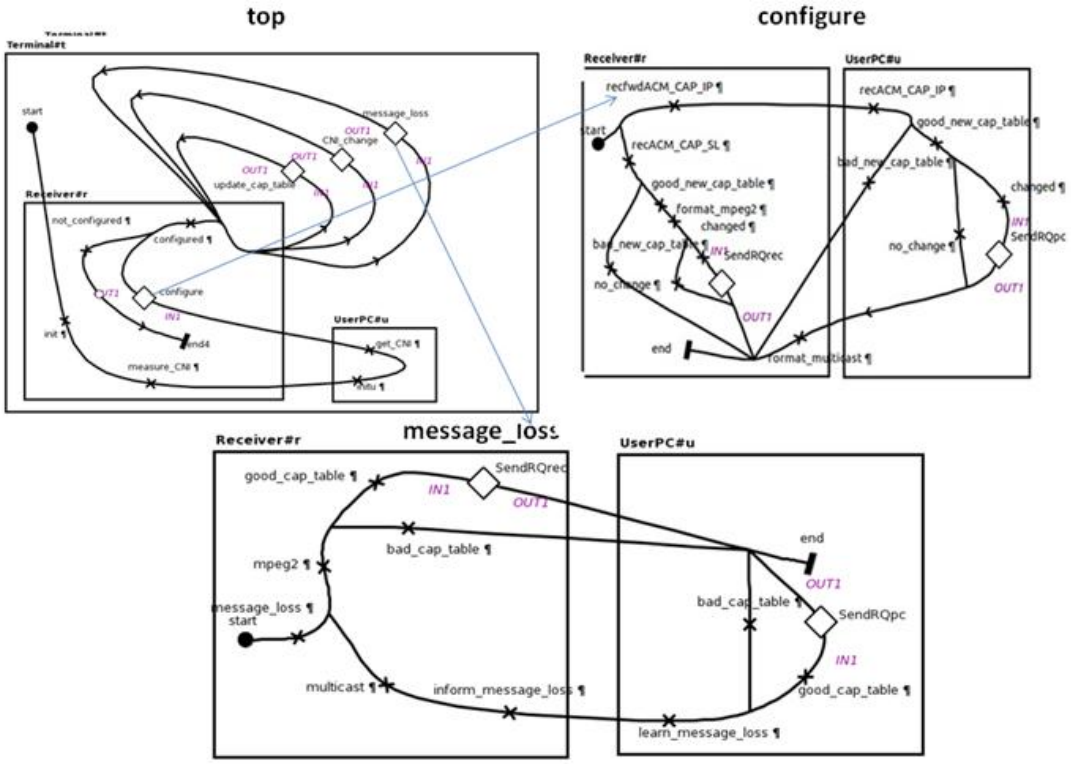


Рис. 3. Пример структурированной UCM диаграммы

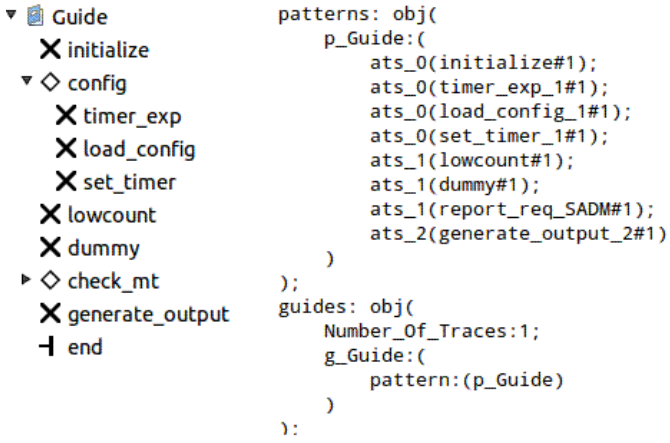
позиции. Характерным примером использования данного критерия является объединение в Stub элементов диаграммы, которые принадлежат различным путям, приводящим систему в одно и то же состояние.

Следует отметить, что композиция поведенческих сценариев вложенных диаграмм разных уровней на основе структурного критерия может быть автоматизирована.

5. Генерация и отбор сценариев, удовлетворяющих выбранному критерию покрытия

Генерация трасс осуществляется символьным и конкретным трассовыми генераторами системы VRS, реализующими достаточно эффективные алгоритмы проверки моделей (Model Checking). Основной проблемой трассовой генерации является “взрыв” вариантов перебора при генерации сценариев (трасс) [7], которые формализуют сценарные события, условия их реализации и соответствующее изменение состояния модели после реализации. Инструментом решения является фильтрация вариантов генерации на основе многочисленных ограничений, выбираемых перед циклом генерации трасс.

Существуют универсальные и специальные ограничения. Например, часто используемыми универсальными являются ограничения на максимальное количество протоколов, используемых в трассе, и максимальное количество трасс, генерируе-



(a) (b)

Рис. 4. Гиды: (a) в нотации UCM, (b) в нотации VRS (Guide Language)

мых в одном цикле генерации. Здесь же можно задать ограничения на число конечных (Goal) и промежуточных (Visited) состояний. Специальные ограничения (т.н. гиды — Guides) задаются последовательностями событий UCM модели, направляющими процесс генерации в область интересующих пользователя вариантов поведения модели. Используется два этапа для генерации тестовых сценариев по гидам. На первом этапе создаются гиды, обеспечивающие заданные критерии покрытия поведения системы. На втором этапе гиды, первоначально заданные в нотации UCM (рис. 4.a) преобразуются в гиды в нотации языка VRS (рис. 4.b) [7], и под их управлением производится генерация трасс. Важно отметить, что в гидах фиксируются лишь главные контрольные точки поведения, в то время как трасса, полученная по гиду, содержит детальную последовательность элементов поведения. Такой подход к генерации существенно уменьшает влияние комбинаторного взрыва во время генерации трасс при обходе UCM модели поведения проектируемой системы.

6. Автоматическая генерация тестового набора по структурной модели

Для автоматической генерации тестового набора по структурной модели необходимо определить возможные критерии покрытия, применение которых будет гарантировать качество тестирования данной модели. В силу того факта, что UCM диаграмма содержит информацию о структуре разрабатываемого программного обеспечения, для обеспечения покрытия удобно использовать структурные критерии. Наиболее известными из них являются критерий покрытия по ветвям, который определяет необходимость покрытия каждой ветви поведения системы хотя бы одним из тестов, и критерий покрытия по путям, который определяет необходимость покрытия каждого пути поведения системы хотя бы одним тестом [8].

Рассмотрим критерий покрытия по ветвям. С точки зрения UCM диаграммы ветвь определяется как отрезок пути, удовлетворяющий следующим требованиям:

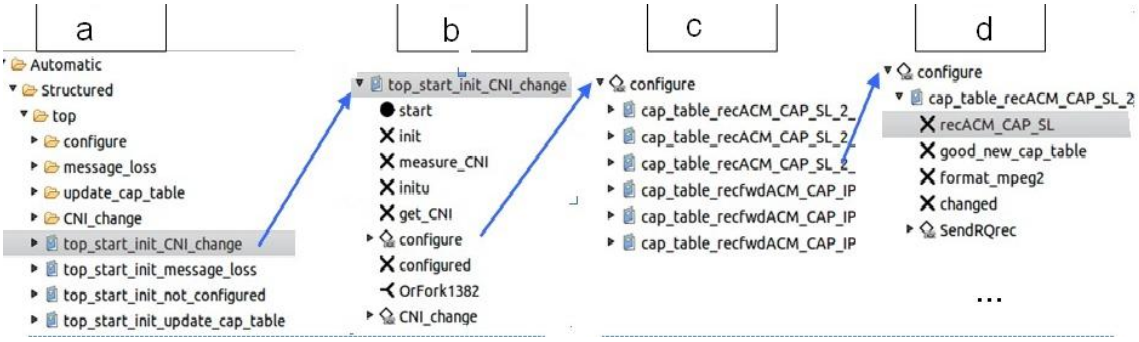


Рис. 5. Структурные гиды UCM проекта

Каждая ветвь начинается в стартовой точке диаграммы или в точке альтернативного выбора OrFork, или в точке начала параллельного поведения AndFork.

Каждая ветвь заканчивается или в конечной точке диаграммы или в точке окончания альтернативного поведения OrJoin, или в точке синхронизации параллельных сценариев AndJoin.

С точки зрения структурного представления UCM модели для выполнения критерия покрытия по ветвям необходимо покрытие всех ветвей основной диаграммы (диаграммы верхнего уровня), а также всех ветвей диаграмм, находящихся в элементах Stub. При этом тестовые сценарии для подобного покрытия могут быть построены независимо для каждого элемента Stub, а затем «склеены» для получения сценариев поведения всей системы.

В рассматриваемом подходе генерация тестов основана на использовании гидов [7], которые могут быть получены путем обхода структурированной модели.

Технология генерации тестовых сценариев для промышленных проектов [9] предусматривает 3 этапа: 1) генерация гидов [7] — кортежей событий, однозначно направляющих генерацию верифицированных тестовых сценариев; 2) генерация множества детальных символьных сценариев [9], обеспечивающих покрытие функциональности системы по заданному критерию; 3) генерация множества конкретных тестовых сценариев или трасс [10], по которым генерируются тестовые наборы для автоматического тестирования.

В настоящей работе рассмотрен первый этап. Например по UCM проекту Satellite генерируются структурные гиды (рис. 5). Для диаграммы top генерируется соответствующая директория top, которая содержит в себе папки с именами элементов Stub и гиды, покрывающие возможные поведения системы на верхнем уровне абстракции (a). Каждый из гидов раскрывается как последовательность UCM элементов (b), и содержит в себе элементы Reference, которые ссылаются на гиды детального описания соответствующего уровня (c). Элемент Reference может содержать множество гидов детального уровня. Каждый из гидов детального уровня так же раскрывается, как последовательность UCM элементов со ссылками на нижние уровни (d). Такая детализация может быть осуществлена вплоть до самого низкого уровня абстракции системы. В результате структурного подхода образуется дерево гидов проектируемой системы.

Полученное множество гидов может быть использовано как для покрытия ветвей отдельного элемента Stub, так и для покрытия системы в целом, в этом случае

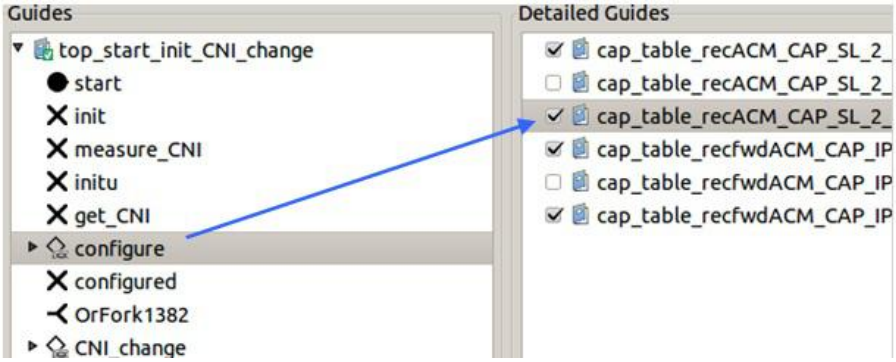


Рис. 6. Выбор гидов из множества гидов, сгенерированных для configure

гиды отдельных структурных уровней «склеиваются» таким образом, чтобы каждый полученный результирующий гид начинался в стартовой точке диаграммы и заканчивался после покрытия определенной ветви системы.

Например, если «склейку» выполнить для всех комбинаций гидов из каждого Stub, то получится нежелательный «взрыв» тестовых сценариев. Поэтому для тестирования необходим избирательный подход к «склейке» гидов отдельных Stub. Для этого предусмотрен механизм селекции актуальных гидов в множестве гидов каждого Stub (рис. 6), реализуемый с помощью пометки check-box в выбранном гиде.

Механизм селекции позволяет из всего множества поведений системы выбрать только интересующие, а именно режимы функционирования и соответствующие им поведенческие сценарии, обеспечивающие полное покрытие исходных требований.

Как в случае покрытия ветвей, так и в случае покрытия выбранных пользователем режимов поведения, использование структурированного представления облегчает анализ модели и дает пользователю возможность контролировать процесс генерации гидов, используемых для получения тестового набора.

7. Автоматический и ручной процесс создания гидов

Существует два возможных подхода к созданию гидов по высокоуровневому описанию системы на языке UCM: ручной и автоматический.

Автоматический подход позволяет генерировать множество гидов, обеспечивающих покрытие поведения системы по критерию ветвей. Каждый из гидов содержит информацию о ключевых точках сценария поведения, начиная от начального состояния модели, моделируемого элементом StartPoint, и заканчивая конечным состоянием системы, моделируемым элементом EndPoint. Процесс генерации гидов осуществляется генератором UCM в MSC [11].

Автоматический подход к созданию гидов можно рассматривать как быстрый способ получения тестового набора, удовлетворяющего критерию ветвей, но такой подход не всегда является достаточным. Часть функциональности можно проверить, используя критерий путей. Поскольку этому критерию временами соответствует неопределенность в выборе тестового набора, достаточного для покрытия

требований, его, как правило, применяют частично, только для покрытия фиксированного множества требований. Создание гидов для покрытия подмножества требований по критерию путей требует использования в процессе генерации дополнительной информации, вводимой вручную. Кроме того, у заказчика и инженера-тестировщика может возникнуть желание создать особые сценарии поведения системы для проверки, например, каких-либо специфических требований. Такие сценарии разработчик задает вручную, и они могут быть созданы в инструменте Анализа событий UCM (UCM EVA) [9].

Важным моментом создания гидов для промышленных систем является минимизация объма ручной работы. Для этого предлагается использовать язык GDL (Guides Definition Language), специализированный на описании гидов [10]. В результате специфицированные на GDL гиды обеспечивают автоматизированный перевод в тестовые сценарии и в код тестов.

8. Язык описания гидов

В рамках языка описания гидов GDL вводится понятие формулы, описывающей множество гидов, обеспечивающих покрытие набора ветвей в заданных регионах в определенной пользователем последовательности. Под регионом подразумевается подмножество элементов UCM диаграммы и связей между ними. Таким образом, регион определяется как шестерка:

$$RG = (U_{rg}, S_{rg}, E_{rg}, R_{rg}, Srg, Erg) \ , \tag{4}$$

в которой определение $U_{rg}, S_{rg}, E_{rg}, R_{rg}$ аналогично определению (1), $Srg \subseteq S_{rg} \times U$ — отношение переходов, определяющее связь стартовых точек региона с элементами диаграммы верхнего уровня и $Erg \subseteq E_{rg} \times U$ — отношение переходов, определяющее связь конечных точек элемента регион с элементами диаграммы верхнего уровня. Частным случаем региона является Stub. Единственное отличие заключается в том, что элемент Stub является явно определенной структурной единицей UCM диаграммы и принадлежит множеству элементов диаграммы, в то время как элемент регион является дополнительным элементом и может не иметь непосредственного графического отображения на диаграмме. Регион задается перечислением множества входящих в него элементов, заданного в виде $Region\_Name = (u_1, u_2, \ldots, u_k)$, где $u_1, u_2, \ldots, u_k$ — элементы UCM диаграммы. При определении данного множества в регион включаются все явно указанные элементы диаграммы, а также автоматически добавляются элементы, лежащие на линейных путях, находящихся между явно заданными элементами. Если на некотором пути региона встречается элемент типа Stub, то все его содержимое добавляется в соответствующий регион.

Stub, заданный на исходной диаграмме, по умолчанию считается регионом и может быть идентифицирован именем. Таким образом, множество UCM элементов исходной диаграммы может быть описано как $U = U_{st} \cup U_{rg} \cup U_{free}$, где U_{st} — множество элементов, принадлежащих элементам Stub, U_{rg} — множество элементов, принадлежащих регионам, и U_{free} — множество остальных элементов, не принадлежащих элементам Stub и регион.

Регион применяется для определения части UCM диаграммы, которую предполагается использовать для генерации гидов.

Для описания множеств гидов или путей покрытия некоторого региона R вводится два префикса, соответствующих кванторам всеобщности и существования, где $\forall' R$ определяет необходимость покрытия всех ветвей в указанном регионе R и $\exists' R$ определяет необходимость покрытия хотя бы одного из множества возможных путей региона R. В регионе можно указать конкретный гид (путь), перечислив кортеж элементов UCM, однозначно определяющих этот путь. Например, $g = \{a, b, c, d\}$, где g — имя кортежа.

Если кортеж элементов UCM g задается формулой $\forall'(g)$, то он обозначает множество всех гидов UCM диаграммы, включающих кортеж $\{a, b, c, d\}$, заданный параметром g, в то время как $\exists'(g)$ — как минимум один гид, включающий кортеж, заданный параметром g.

Для описания процедуры склейки путей регионов вводится бинарный оператор "*", определяющий склейку путей регионов на основании описанной пользователем формулы. В частном случае этот оператор описывает объединение кортежей: $g = \{a, b, c, d\}$, $h = \{i, j, k\}$, $g * h = \{a, b, c, d, i, j, k\}$.

Синтаксис языка формул (GDL) определяется как:

```
<formula>       ::= {<region formula>}{{<MUL>|<PAR>}{<region formula>}}*;
<region formula>::= [<ALL>|<EXIST>]'<region name>[(<formula>)]}*;
<ALL>           ::= ∀;
<EXIST>         ::= ∃;
<region name>   ::= <name>;
<name>          ::= <identifier>;
<identifier>    ::= {<alfa>|<UNDERLINE>}{<alfa>|<UNDERLINE>|<digit>}*;
<alfa>          ::= a..z, A..Z;
<UNDERLINE>     ::= _;
<digit>         ::= 0|1|2|3|4|5|6|7|8|9;
<GLU>           ::= * ;
<PAR>           ::= ||;
```

Определение:

1. Если R — регион, то $\forall' R$ — формула, определяющая множество путей региона R, обеспечивающих полное покрытие ветвей региона R. $\exists' R$ — формула, определяющая некоторый путь региона R, обеспечивающий покрытие хотя бы одной ветви региона.

2. Если r — элемент Responsibility, то r — формула, определяющая необходимость прохода генерируемого гида через элемент r.

3. Если $R1$ и $R2$ — регионы, то $\forall' R1 * \forall' R2$ — формула, определяющая генерацию множества гидов для покрытия всех ветвей региона $R1$ и региона $R2$ с их последующей склейкой. $\exists' R1 * \forall' R2$ — формула, определяющая гид для генерации некоторого пути по $R1$ и генерации гидов для множества путей, покрывающих все ветви $R2$ с их последующей склейкой. $\exists' R1 * \exists' R2$ — формула, определяющая хотя бы один гид для генерации путей, сначала проходящих через $R1$, а затем через $R2$.

4. Если R — регион, а r — элемент Responsibility, то $\forall' R * r$ — формула, определяющая покрытие всех ветвей региона R с последующим проходом через элемент r.

5. Если g — формула, а R — регион, то $\forall' R(g)$, $\exists' R(g)$ — формулы, определяющие генерацию гидов, обеспечивающих покрытие в регионе R всех тех ветвей в случае $\forall'$ или одной в случае $\exists'$, которые удовлетворяют формуле g.

Рассмотрим примеры спецификаций для системы, представленной на рис. 3, которые могут быть определены с использованием введенного аппарата формул языка GDL и регионов:

1. Формула $\forall'$configure — приводит к генерации 6 гидов, покрывающих все ветки в регионе configure с последующим выходом в конечную точку Stub.

2. Формула $\exists'$configure — ведет к генерации 1 произвольного гида, проходящего через Stub configure. Гид заканчивается на выходе из Stub.

3. Формула get_CNI*not_configured — приводит к генерации гидов, покрывающих все ветви, лежащие на путях, проходящих через элементы get_CNI и not_configured (в результате будет сгенерировано 6 гидов для покрытия всех ветвей в configured с дальнейшим движением в точку not_configured). Генерация останавливается после достижения точки not_configured.

4. Формула $\forall'$configure$*\exists'$update_cap_table$*\forall'$message_loss — ведет к генерации множества гидов, покрывающих все ветви региона configure, одну из ветвей update_cap_table и все ветви региона message_loss, путем полного перебора всех возможных последовательностей.

5. Формула $\forall'$configure(bad_new_cap_table) ведет к генерации множества всех гидов, проходящих через configure и имеющих в своем составе прохождение через точку bad_new_cap_table. Генерация заканчивается после выхода из Stub (в результате в данном примере будет сгенерировано 2 гида).

В сгенерированном на основе анализа формулы гиде присутствует интересующая нас ветвь в явном виде, а проход до нее и после определяется предшествующими и последующими частями формулы.

В рассмотренных примерах по гибко заданным на языке GDL гидам генерируются пути, удовлетворяющие критерию частичного покрытия путей и контролирующие важные для проектируемой системы режимы работы, что позволяет выявить дефекты некорректной обработки сложных поведенческих сценариев реальных проектов.

9. Заключение

Настоящая работа рассматривает подход к решению актуальной проблемы автоматизации тестирования программного обеспечения промышленных приложений с сложными и большими по объему спецификациями требований, который характеризуется следующими особенностями.

1. Для сформулированных на естественном языке исходных требований разрабатываются специализированные критерии их проверки в виде наблюдаемых последовательностей событий или критериальных цепочек.

2. На основе требований и критериальных цепочек создается поведенческая модель приложения на языке UCM, которая верифицируется и корректируется. В результате поведение приложения описывается набором сгенерированных символьных сценариев, которые обеспечивают полноту покрытия поведения в соответствии с заданными критериями.

3. Автоматическое преобразование символьных сценариев в тестовый набор осуществляется в процессе конкретизации, проводящемся в соответствии с планом тестирования.

4. Для сокращения количества тестов в наборе и уменьшения трудоемкости разработки тестовых наборов совместно используется автоматическая генерация и язык проектирования тестов GDL.

Результат применения описанной технологии продемонстрировал сокращение трудозатрат на создание тестовых наборов более чем в 1,4 раза.

Список литературы

1. Баранов С., Котляров В., Летичевский А. Индустриальная технология автоматизации тестирования мобильных устройств на основе верифицированных поведенческих моделей проектных спецификаций требований // Труды междунар. науч. конф. “Космос, астрономия и программирование”. СПб.: СПбГУ, 2008. С. 134—145.

2. Карпов Ю.Г. Теория автоматов. СПб.: Питер, 2003. 208 с.

3. Baranov S., Kotlyarov V., Letichevsky A., Drobintsev P. The technology of Automation Verification and Testing in Industrial Projects // Proc. of St.Petersburg IEEE Chapter, International Conference, May 18—21. St.Petersburg, 2005. P. 81—86.

4. Z.151: User Requirements Notation (URN) — Language Definition `http://www.itu.int/rec/T-REC-Z.151-200811-I/en`

5. Letichevsky A., Kapitonova J., Letichevsky A. jr., Volkov V., Baranov S., Kotlyarov V., Weigert T. Basic Protocols, Message Sequence Charts, and Verification of Requirements Specifications // Proc of ISSRE04 Workshop on Integrated-reliability with Telecommunications and UML Languages (ISSRE04:WITUL), 02 Nov 2004. IRISA, Rennes, France, 2004. P.30—38.

6. Digital Video Broadcasting (DVB). DVB-S2 Adaptive Coding and Modulation for Broadband Hybrid Satellite Dialup Applications // `http://www.etsi.org/deliver/etsi_ts/102400_102499/102441/01.01.01_60/ts_102441v010101p.pdf`

7. Летичевский А.А., Колчин А.В. Генерация тестовых сценариев на основе формальной модели // Проблемы программирования. 2010. № 2–3. С. 209—215.

8. Baranov S., Kotlyarov V., Weigert T. Verifiable Coverage Criteria For Automated Testing. SDL2011: Integrating System and Software Modeling // LNCS. 2012. Vol. 7083. P. 79—89.

9. Ануреев И.С., Баранов С.Н., Белоглазов Д.М., Бодин Е.В., Дробинцев П.Д., Колчин А.В., Котляров В.П., Летичевский А.А., Летичевский А.А. мл., Непомнящий В.А., Никифоров И.В., Потиенко С.В., Прийма Л.В., Тютин Б.В. Средства поддержки интегрированной технологии для анализа и верификации спецификаций телекоммуникационных приложений // Труды СПИИРАН. 2013. №1—28Р.

10. Дробинцев П.Д., Котляров В.П., Черноруцкий И.Г. Автоматизация тестирования на основе покрытия пользовательских сценариев // Научно-технические ведомости СПбГПУ. 2012. Т. 4(152). С. 123—126.

11. Recommendation ITU-T Z.120. Message Sequence Chart (MSC), 11/2000.

The Guide-based Automatic Creation of Verified Test Scenarious

Drobintsev P.D., Kotlyarov V.P., Letichevsky A.A.

St. Petersburg State Polytechnical University
Polytechnicheskaya st., 29, St. Petersburg, 195251, Russia
Glushkov Institute of Cybernetic of NAS of Ukraine,
Glushkova av., 40, Kyiv, 03187, Ukraine

Keywords: symbolic verification, testing automation, concretization of test scenarios, predicative transformer

This paper presents an overview of technology of the automated generation of test scenarios based on guides. The usage of this technology can significantly improve the quality of the developed program products. In order to ground the technology creation, the main problems that occur during the development and testing of the large industrial systems, are described, as well as the methodologies of software verification on conformity to product requirements. The potentialities of tools for automatic and semi-automatic generation of a test suite by using a formal model in UCM notation are demonstrated, as well as tools for verification and automation of testing.

Сведения об авторах:
Дробинцев Павел Дмитриевич,
Санкт-Петербургский государственный политехнический университет,
доцент, канд. техн. наук;
Котляров Всеволод Павлович,
Санкт-Петербургский государственный политехнический университет,
профессор, канд. техн. наук;
Летичевский Александр Адольфович,
Институт кибернетики им. В.М. Глушкова НАН Украины,
зав. отделом теории цифровых автоматов, академик, д-р физ.-мат. наук