Модел. и анализ информ. систем. Т. **21**, № **6** (2014) 44–56

УДК 004.415

Подход к автоматизации отладки поведенческих сценариев

Дробинцев П.Д.[1], Котляров В.П.[1], Никифоров И.В.[1],Летичевский А.А.[2], Песчаненко В.С.[3]

[1] *Санкт-Петербургский государственный политехнический университет, 195251, Россия, г. Санкт-Петербург, ул. Политехническая, 29*
[2] *Институт кибернетики им. В.М. Глушкова НАН Украины, 03680 МСП, Украина, г. Киев, просп. Академика Глушкова, 40*
[3] *Херсонский государственный университет, 73000 Украина, г. Херсон, ул. 40 лет Октября, 27*

e-mail: [1]*vpk@spbstu.ru,* [2]*let@cyfra.net,* [3]*vladimirius@gmail.com*

получена 30 сентября 2014

Ключевые слова: поведенческая модель, символьная трасса, символьный сценарий, гид, причина ошибки, полуавтоматический анализ, автоматический анализ причин ошибок

В работе представлены два метода, направленных на решение задачи автоматизации отладки поведенческих сценариев приложения: полуавтоматический и автоматический. Первый дает возможность пользователю автоматизировать поиск тех мест в выбранном символьном поведенческом сценарии, в которых кроется возможная причина ошибки. Второй позволяет в едином цикле анализа автоматически идентифицировать не только места, но и возможные причины ошибок в заданном множестве сгенерированных символьных поведенческих сценариев.

1. Введение

В настоящее время индустрия разработки программного обеспечения (ПО) является быстро растущей отраслью, стимулирующей непрерывный рост возможностей аппаратного и программного обеспечения, приложений и инструментальных средств поддержки разработки, в сочетании с растущим спросом на программные продукты (ПП) для множества предметных областей.

Основной задачей создания эффективной промышленной технологии является создание инструментария, обеспечивающего полную автоматизацию, минимизирующую трудозатраты производства ПП, и технологию его применения. В работе описывается часть решения общей задачи, которая заключается в автоматизации генерации тестового набора по верифицированной формальной поведенческой модели, гарантирующей полное покрытие исходных функциональных требований.

Поведенческая модель, используемая в проектировании ПП, представляет собой спецификацию на расширенном графическом языке UCM [1]. Анализ причин выбора языка UCM представлен в [2]. В настоящее время использование UCM для спецификации исходных требований становится заметным фактором в повышении эффективности процесса разработки ПО.

На основе сравнения технологий по степени их поддержки автоматического анализа результатов верификации и генерации тестов [13] выделяются только три технологии: UCM/SDL HLTD [11], UCM CPN [12] и VRS/TAT [10], которые позволяют анализировать генерируемые тестовые сценарии. Тем не менее, процесс анализа не полностью автоматизирован и поэтому требует значительных трудозатрат. Используемые на таблице 1 критерии сравнения проверяют: поддержана ли технология автоматизированными инструментами анализа результатов тестирования и верификации, инструментами отображения сгенерированных символьных поведенческих сценариев, результатов тестирования и анализа на исходную модель.

Таблица 1. Сравнение UCM технологий

Название технологии	Критерии сравнения								
	Исходный язык	Целевой язык	Автоматизированный подход к проектированию	Использование верификации	Генерация исполнимого кода	Генерация тестовых сценариев	Автоматизация исполнения тестовых сценариев	Автоматизация анализа результатов	Автоматизация отладки
SPEC-VALUe	UCM	LOTOS	-/+	+	-	+	+	-	-/+
RT-TROOP	UCM	MSC/SDL	-/+	+	+	-	-	-	-/+
UCM/SDL HLTD	UCM	SDL	-/+	+	+	+	-	-/+	-/+
VRS/TAT	UCM	Базовые протоколы	-/+	+	+	+	+	-/+	-/+
UCM CPN	UCM	CPN	-/+	+	-	-	-	-/+	-

Условные обозначения:
«-» – свойство не поддерживается;
«-/+» – поддерживается частично;
«+» – поддерживается полностью

Анализ по критерию поддержки технологией отладки сгенерированных тестовых сценариев показывает, что во всех случаях (за исключением технологии VRS/TAT) этот процесс в той или иной степени реализуется вручную, то есть не автоматизирован и не имеет полноценной инструментальной поддержки. Поэтому задача автоматизированной отладки тестовых сценариев с автоматическим поиском причин ошибок является актуальной, в частности для разработки высококачественного промышленного ПП.

Основной вклад настоящей работы заключаются в следующем:

1. В демонстрации важности автоматизированной идентификации мест возможных ошибок в исходной поведенческой модели. Подобная идентификация актуальна, в связи с тем, что автоматизированная генерация сценариев поведения на основе универсальных критериев не всегда обеспечивает ожидаемые пользователем результаты.

2. В реализации полуавтоматического метода отладки, который обеспечивает автоматическую локализацию подозрительных на возникновение ошибки мест в конкретном поведенческом символьном сценарии и предоставляет пользователю возможность анализа причин ошибки, сокращая время на исполнение этой ручной процедуры.

3. В описании метода автоматической отладки, который наряду с локализацией возможных мест ошибок выявляет ее причину и предоставляет пользователю результат объединенного анализа ошибок для целого набора сгенерированных сценариев.

2. Поведенческая модель программного продукта

Приведем определения основных понятий, используемых в поведенческой модели. «Базовый протокол» (Bp) [4] обозначает переход в модели. Он представляет собой атомный параметризованный переход модели из одного состояния в другое. Базовый протокол $B(x)$ — это выражение вида: $\forall x(\alpha(x) \to \langle P(x)\rangle\beta(x))$, где x — список параметров протокола, $\alpha(x), \beta(x)$ — формулы логики первого порядка и $P(x)$ — это процесс протокола (наблюдаемое действие, последовательность действий), в котором возможны изменения параметров, отображаемые в $\beta(x)$. Формула $\alpha(x)$ называется предусловием, а формула $\beta(x)$ — постусловием базового протокола. Постусловие может содержать операторы присваивания. Пред- и постусловия описывают подмножества состояний системы перед и после выполнения процесса, заключающегося в посылке сигналов или изменения значений переменных приложения. Как условия, так и процесс протокола могут зависеть от параметров. Сам базовый протокол может рассматриваться как формула темпоральной логики, выражающая факт, что если (для подходящих значений параметров) состояние системы имеет разметку, удовлетворяющую условию $\alpha(x)$), то процесс $P(x)$ может быть инициирован, и после его завершения разметка будет удовлетворять условию $\beta(x)$. Базирующаяся на Bp поведенческая модель является транзиционной системой [15].

Последовательность $S_0 \xrightarrow{B_0(x_0)} S_1 \xrightarrow{B_1(x_1)} ...S_n$ называется поведенческим сценарием (трассой), где S_i — состояния, B_i — базовые протоколы, а x_i — списки их параметров.

Автоматическая генерация трасс по модели, как правило, сопровождается взрывом состояний (state explosion), что часто приводит к неограниченному времени генерации необходимого набора трасс. Использование гидов (guide) — определенных пользователем последовательностей базовых протоколов управляющих трассовой генерацией — эффективный способ справиться с взрывом состояний. Гид определяет набор и частичный порядок применения базовых протоколов в генерируемой

трассе, т.е. порядок применения базовых протоколов в гиде определяет последовательность целей в виде Bp, которые должны быть достигнуты при генерации трассы.

Спецификация поведения приложений в терминах расширенного языка UCM представлена графом UCM-элементов, а модель поведения представляет собой набор сценариев (путей) в этом графе. Каждый элемент UCM модели может быть представлен как тройка Хоара [3] или как маркированный базовый протокол Летичевского, таким образом, поведенческая модель может рассматриваться как определенная транзиционная система.

Подмножество полного множества конечных трасс, удовлетворяющее некоторому критерию покрытия [5], можно рассматривать как набор сценариев, пригодных для генерации тестов, удовлетворяющих тому же критерию покрытия.

3. Поведенческие сценарии как последовательность UCM элементов

Зафиксируем некоторое начальное состояние транзиционной системы S_0, которое включает состояние окружающей среды (environment) и состояния всех внедренных (inserted) в среду агентов (процессов) [6]. Все возможные трассы (сценарии функционирования системы) можно рассматривать как набор последовательностей вида:

$$S_0 \xrightarrow{b_1(n_1,m_1)} S_1 \xrightarrow{b_2(n_2,m_2)} ...,$$

где $b_1(n_1, m_1)$, $b_2(n_2, m_2)$,... — базовые протоколы соответствующих UCM-элементов. Здесь $n_1, n_2, ...$ — имена ключевых агентов (ключевой агент — базовый протокол агента, состояние которого изменяется на этом протоколе) и $m_1, m_2, ...$ — множества значений других параметров, которые гарантируют, что предусловие соответствующих базовых протоколов истинно. Число всех возможных сценариев определяется числом UCM-элементов и количеством взаимосвязей между ними в данной UCM-спецификации.

4. Анализ поведения программного проекта

Предположим, что ПП можно разделить на компоненты. Это обычный процесс декомпозиции, применяемый в процессе разработки программного обеспечения. Основная цель процесса — уменьшение сложности описания поведения. Для спецификаций UCM самый простой способ такого разложения основан на элементах Stub в предположении, что каждый Stub описывает поведение отдельной компоненты разрабатываемого ПО.

Анализ поведения компонент осуществляется через анализ трасс, но дизайн и анализ возможных трасс с помощью обхода многоуровневой структуры компонент является трудоемкой и сложной задачей для ПП среднего или большого размера. Для ограничения множества анализируемых трасс предложен метод гидов [14], который управляет генерацией трасс так, чтобы покрыть исходные требования. Гид определяет частичный порядок базовых протоколов или UCM-элементов, которые

принадлежат сгенерированной трассе и следуют в ней в заданном гидом порядке. Метод гидов обеспечивает существенное сокращение набора трасс, гарантирующего покрытие исходных требований.

Генерация поведенческих трасс по методу гидов гарантирует, что поведенческие критерии покрытия ветвей или частичного покрытия путей [5] будут выполнены. Критерий ветвей основан на оценке покрытия всех ветвей программы, при этом ветвь рассматривается как линейный участок кода. В свою очередь критерий путей основан на покрытии путей поведения системы, и условие частичного покрытия обусловливается потенциально бесконечным количеством подобных путей. Однако, автоматическая генерация, управляемая гидами, которые учитывают только потоки управления и не учитывают особенности потоков данных, приводит к генерации подмножества трасс, которые не достигают некоторых UCM-элементов, представленных в рассматриваемых гидах.

Причины недостижимости таких элементов, как правило, следующие:

- Неправильная последовательность элементов в гиде: для исправления последовательность элементов должна быть скорректирована;
- Петля в гиде: для исправления в гиде необходимо ограничить количество итераций, то есть избежать ситуации многократного прохождения через ту или иную точку гида;
- Недостижимый протокол в модели проекта: для исправления необходимо анализировать информацию о допустимых диапазонах значений атрибутов, используемых в соответствующих базовых протоколах, которая должна быть учтена и скорректирована в их предусловиях;
- Неправильная “глубина” (ограничение на число промежуточных базовых протоколов в трассе между соседними базовыми протоколами, указанными в гиде) как результат некорректного ее расчета в сложных гидах: для исправления необходимо или указать явно правильную глубину, или значение глубины указать по умолчанию (то есть задать общую глубину поиска для всех базовых протоколов); другое решение — рассчитать глубину с помощью символьного верификатора [7].

Любой вариант в некоторых случаях может привести к недостижимости базового протокола, что требует выявления ее причин. Таким образом, особую актуальность приобретает задача отладки недостижимости указанного в гиде базового протокола в UCM-трассе.

5. Полуавтоматический метод анализа причин ошибки

Полуавтоматический метод позволяет автоматически искать места (базовые протоколы) в трассе, которые могут являться причиной остановки генерации трассы, управляемой некоторым гидом. Исходной информацией для применения метода является гид и незаконченная трасса той длины, которую удалось сгенерировать под управлением этого гида.

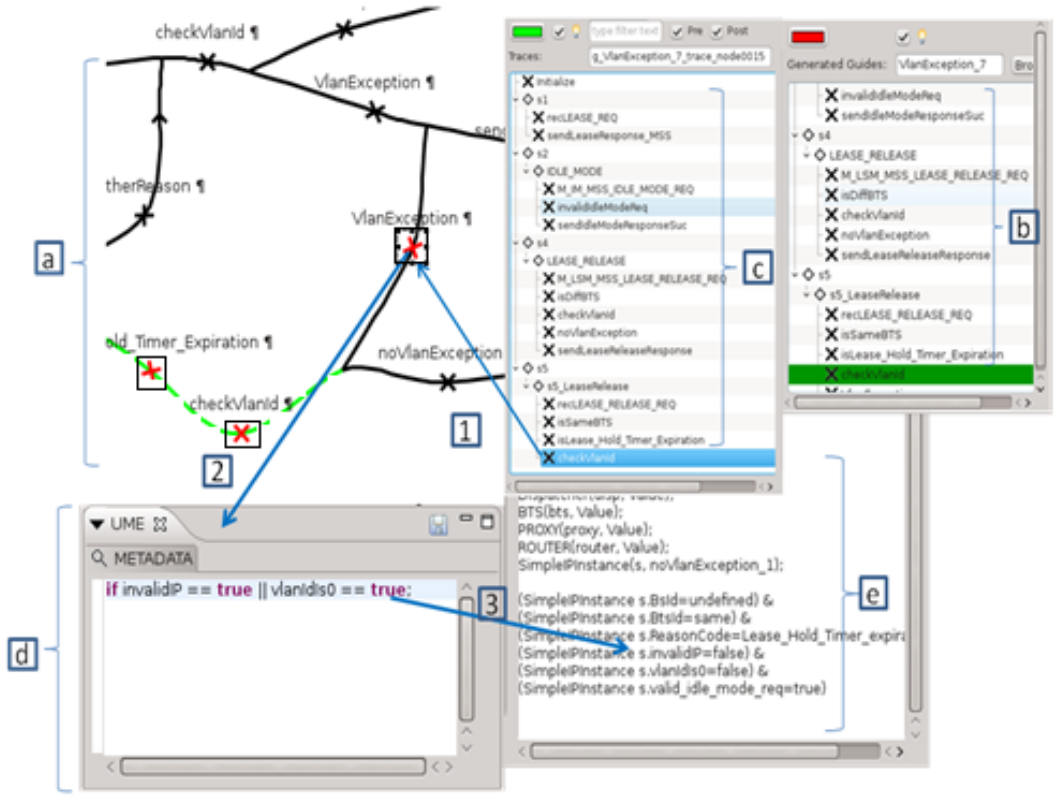

Рис. 1. Пример применения полуавтоматического метода.

Метод проиллюстрирован на рис. 1 и содержит следующие этапы:

Шаг 1. Подсветка гида (b) (окно Generated Guides) и трассы (окно Traces) на UCM-диаграмме (a) используется для идентификации недостижимого протокола. На UCM-диаграмме элементы событий (Responsibility) гида заключены в квадраты, трасса помечена пунктиром и имя недостижимого протокола (следующего в гиде, но не покрытого в трассе) VlanException.

Шаг 2. Идентификация конкретного непокрытого события и пути к нему (1). Путь определяется на основе трассы, а не покрытым событием является событие, описанное в гиде, но не представленное в трассе.

Шаг 3. Анализ переменных (2), используемых в предусловиях Bp, который не удалось применить. Необходимо проанализировать значения этих переменных для применения Bp. Для подобного анализа выделен протокол VlanException и в окне (d) метаданных протокола видно, что в предусловии используется две переменных invalidIP и vlanIdIs, при этом для успешного применения протокола обе переменные должны иметь значение “true”.

Шаг 4. Сравнение ожидаемых значений переменных со значениями, полученными в сгенерированной трассе (3), на основе анализа состояния переменных окружающей среды. Данное сравнение позволит выделить те переменные, которые не позволяют примениться протоколу и провести дальнейший анализ,

направленный на поиск возможных альтернативных путей с иными значениями переменных.

Шаг 5. Определение следующего Вр, двигаясь от неприменeнного Вр "вверх" по сгенерированной трассе к протоколу, в котором применено присваивание или отношение сравнения к искомым переменным. Таким образом, удается обнаружить точку, в которой произошло изменение значений переменных, не позволяющее применить необходимый протокол.

Шаг 6. Если причина ошибки найдена, то необходимо скорректировать Вр (изменив формулы пред-, постусловий или переменных окружения так, чтобы неприменившийся ранее Вр применился) и закончить цикл отладки. Если причина не найдена, то перейти к шагу 5.

Анализ позволяет пользователю определить, почему некоторый Вр в гиде не применился. Тем не менее, поиск места в трассе, где переменные изменили свои значения не так, как предусматривалось неприменeнным Вр в гиде, является проблемой. Чтобы ответить на этот вопрос, необходим анализ постусловий только тех Вр трассы, которые содержат формулы изменения значения или сравнения искомых переменных. Автоматический поиск таких Вр происходит от достигнутого Вр к началу трассы.

Каждый найденный Вр должен быть проанализирован вручную, чтобы понять, какие именно преобразования переменных Вр привели к их недопустимым значениям. После обнаружения такого Вр, UCM-модель или гид могут быть скорректированы, соответственно и эксперимент должен быть повторен.

6. Автоматический анализ ошибок в сгенерированных символьных трассах

Рассмотрим метод автоматического поиска причин ошибок на основе использования обратного предикатного трансформера. Данный подход может быть использован для множества трасс, генерация которых по гидам не была завершена, при этом поиск причины проводится для каждой трассы в отдельности.

Если процесс генерации тестового сценария не был завершен, то точка остановки генерации (Вр) фиксируется, а незавершенная трасса визуализируется на UCM-диаграмме. Хотя факт проявления ошибки успешно зафиксирован, поиск ее причин остается трудоемким процессом, особенно в трассах, содержащих сотни событий, закодированных базовыми протоколами. Для анализа причин ошибки, начиная с зафиксированной точки трассы, применяется инструмент Event Analysis Tool, который использует следующий подход к анализу причин ошибок.

Пусть есть некоторая трасса (рис. 2) и базовые протоколы Bp_n, которые не удалось применить в последнем (зафиксированном) состоянии исходной трассы S_n. Необходимо найти причину неприменимости базовых протоколов Bp_n в состоянии окружающей среды S_n и рекомендовать значения атрибутов, при которых такие применения возможны.

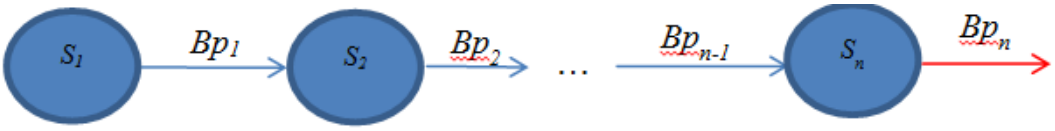

Рис. 2. Исходная трасса и неприменённый базовый протокол

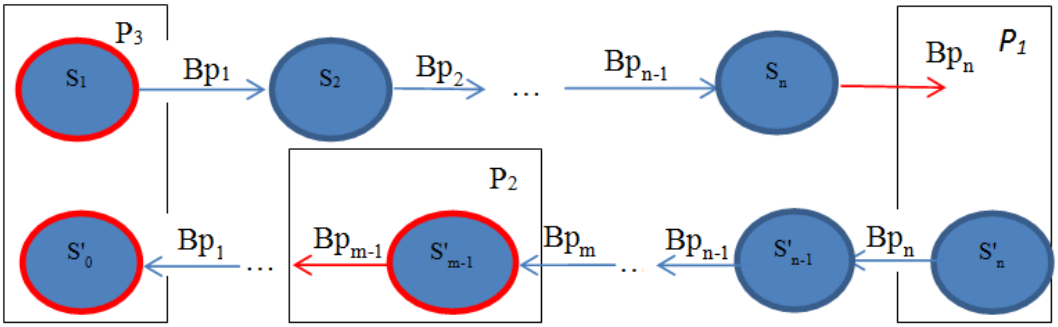

Рис. 3. Построение слабейшего предусловия

Здесь $S_1, S_2, ..., S_n$ — состояние окружения, $Bp_1, Bp_2, ..., Bp_n$ — базовые протоколы. Каждый переход осуществляется с помощью предикатного трансформера. Существуют прямой (Forward) [8] и обратный (Backward) [9] предикатные трансформеры. Действие Forward предикатного трансформера обозначается стрелкой слева направо и реализует алгоритм нахождения сильнейшего постусловия базового протокола и соответствующего состояния окружающей среды для логических формул первого порядка.

Для выявления причин неприменимости базового протокола необходимо построить некоторый интервал. Нижняя граница этого интервала является сильнейшим постусловием, а верхнюю границу предлагается использовать как слабейшее предусловие. Действие Backward предикатного трансформера является реализацией алгоритма нахождения наиболее слабого предусловия для базового протокола и соответствующего состояния окружающей среды и обозначается стрелкой справа налево (рис. 3). Теперь следует попытаться получить слабейшие предусловия для каждого базового протокола исходной трассы и соответствующего состояния окружающей среды. Начальное состояние окружающей среды для этого процесса $S'_n = 1$ (Рис. 3).

Здесь S'_n — отправная точка для построения слабейшего предусловия, $S'_0, ..., S'_{m-1}$, $..., S'_{n-1}$ — слабейшие предусловия для базовых протоколов $Bp_1, ..., Bp_{m-1}, ..., Bp_{n-1}$, (S'_{m-1}, Bp_{m-1}) — неприменившийся базовый протокол при попытке построить слабейшее предусловие, (S_1, S'_0) — невыполнимое объединение двух состояний окружения, P_1 — начальная точка процесса определения неприменимости базового протокола Bp_n в состоянии S_n, P_2 означает, что для некоторого базового протокола Bp_{m-1} в трассе с состоянием окружения S'_{m-1}, невозможно построить слабейшее предусловие, P_3 означает, что результирующее слабейшее предусловие S'_0 и исходное состояние S_1 невыполнимы.

Пусть есть некоторая трасса (рис. 3) и базовый протокол Bp_n, который не применился в последнем состоянии окружающей среды S_n в трассе, а $S'_0, ..., S'_{m-1}$, ..., S'_{n-1} — слабейшие предусловия базовых протоколов $Bp_1, ..., Bp_{m-1}, ..., Bp_n$.

Теорема. Если $S_1 \wedge S'_0 = 0$, то базовый протокол Bp_n неприменим в состоянии окружения S_n.

Доказательство:

Доказательство проводим от обратного: если Bp_n применим к состоянию окружающей среды S_n, то $S_1 \wedge S'_0 \neq 0$. Пусть $S_n \xrightarrow{Bp_n} S_{n+1}$. Тогда $S_{n+1} \to S'_n = 1$ и $S_{i+1} \to S'_i$, $i = 0, ..., n$ из монотонности слабейшего предусловия [8] и $S_1 \to S'_0$ и $S_1 \wedge S'_0 \neq 0$. □

Состояние S'_0 не может быть получено, если для некоторого базового протокола Bp_{m-1} в трассе и состояния окружающей среды S'_{m-1} не может быть построено слабейшее предусловие. Это означает, что слабейшим предусловием в данной точке трассы является 0. Продолжая левую стрелку, можно сказать, что в трассе $S'_0 = 0$.

Таким образом, причина неприменимости базового протокола Bp_n при состоянии окружающей среды S_n содержится в позициях P_2, P_3 (Рис. 3). В обоих случаях, поиск причин неприменимости базового протокола можно найти в невыполнимом объединении двух состояний окружения. Пусть E_1, E_2 — состояния окружения, такие, что $E_1 \wedge E_2 = 0$. Найдем причину невыполнимости. Представим состояния окружения в виде дизъюнктивной нормальной формы:

$$dnf(E_1) = u_1 \vee u_2 \vee ..., \qquad dnf(E_2) = v_1 \vee v_2 \vee ...$$

Тогда

$$E_1 \wedge E_2 = u_1v_1 \vee u_1v_2 \vee ... \vee u_2v_1 \vee u_2v_2 \vee ... = 0 \Rightarrow$$
$$u_1v_1 = 0 \wedge u_1v_2 = 0 \wedge ... \wedge u_2v_1 = 0 \wedge u_2v_2 = 0 \wedge ...$$

Каждая пара u_i, v_j представлена в форме

$$u_i = u_i(t_1) \wedge u_i(t_2) \wedge ..., \qquad v_j = v_j(t_1) \wedge v_j(t_2) \wedge ...$$

где $t_1, t_2, ...$ есть множество атрибутов, таких что $t_1 \cap t_2 \cap ... = \emptyset$. Тогда если $u_i(t_m) \wedge v_j(t_m) = 0$, то причина отказа от выполнимости кроется в конъюнкции состояний окружения E_1, E_2.

Рассмотрим пример использования автоматического метода для трассы (рис. 4). Основным преимуществом такого подхода является автоматический поиск протокола, в котором значения переменных изменяются неожиданным способом, не учтенным в гиде [8, 9].

Входной информацией автоматического метода являются непокрытые гиды и соответствующие трассы, генерация которых не была завершена. Базируясь на этой информации, алгоритм определяет непокрытый протокол в гиде и дает рекомендацию для изменения значений (диапазонов значений) переменных, препятствующих покрытию соответствующего базового протокола.

Таким образом, применение автоматического подхода можно условно разделить на три части:

1. Генерация трасс по пакету гидов и выделение не покрытых гидов.

2. Автоматическое выделение причин непокрытия гидов на основе применения обратного предикатного трансформера к полученному на предыдущем этапе пакету трасс.

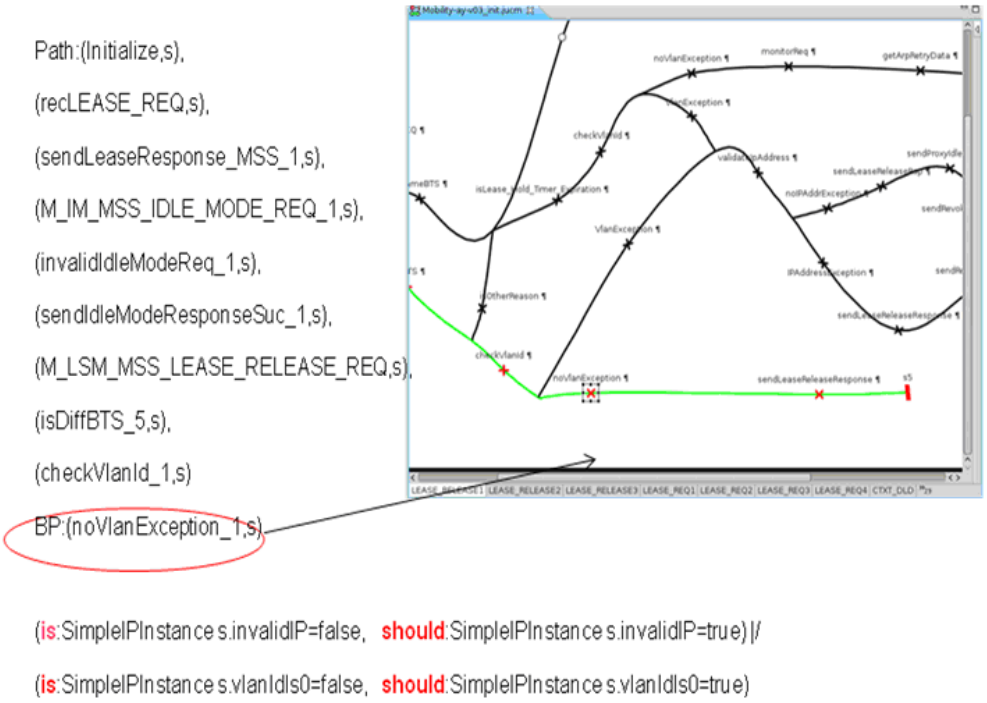

Рис. 4. Пример применения автоматического метода

3. Поиск альтернативных путей, ведущих к применению необходимого протокола без потери покрытия.

На рис. 4 представлена UCM-диаграмма и результат применения обратного предикатного трансформера. Значение переменной SimpleIPInstances.ivalidIP, вычисленное на основе прохождения трассы равняется “false”. В то же время для применения следующего протокола, описанного в гиде, данное значение должно равняться “true”. Из рисунка видно, что данное присвоение определяется в рамках примененного в трассе протокола noVlanException. Таким образом, автоматически идентифицирован протокол, содержащий проблемное присвоение. В результате, если причина ошибки найдена, то протокол, являющийся причиной ошибки, должен быть исправлен или альтернативный путь со значением переменной “true” должен быть вычислен. При дальнейшем анализе возможно использование двух методов для покрытия необходимого протокола. Аналогично описывается и анализ переменной vlanIdIs0.

Следует отметить, что доказанная теорема формулирует достаточное условие неприменимости протокола, что обеспечивает частичную корректность применяемого подхода, но не его полноту. В этом случае надо использовать полуавтоматический метод поиска ошибки.

Совместное использование полуавтоматического метода (для анализа трасс небольших проектов и для трасс больших проектов, затруднительных для автоматического анализа) и автоматического метода (для анализа пакетов трасс больших проектов) помогает существенно сократить затраты на анализ потоков управления и поиск ошибок при генерации трасс по гидам.

7. Поддерживающий инструментарий

Предлагаемый подход к автоматизации отладки поведенческих сценариев реализован в инструментальной системе VRS / TAT [7], которая сочетает инструменты символьной верификации, генерации тестовых наборов и прогона тестов в едином автоматизированном процессе с минимальными трудозатратами на этапах разработки и выполнения тестов.

Экспериментальная проверка эффективности технологии и поддерживающего инструментария была проведена на телекоммуникационных проектах, где размер моделей варьировался от 50 до 1500 базовых протоколов. В результате было зафиксировано снижение трудоемкости на фазе тестирования более чем на 30% по сравнению с традиционными технологиями.

8. Заключение

Автоматизация отладки очень важна для производства высококачественного программного обеспечения. Два метода к реализации отладки были описаны в этой статье. Полуавтоматический метод позволяет автоматизировать процесс поиска мест возможных причин ошибок в конкретной символьной трассе, однако анализ ошибок в определенных таким образом местах трассы производится вручную, что, тем не менее, упрощает процесс исправления. В то же время автоматический метод позволяет, хотя и с ограничениями, найти причину многочисленных ошибок в заданном множестве трасс в одном цикле анализа, однако их исправление тоже должно осуществляться вручную. Объединение двух методов в процессе отладки дает наибольший эффект.

Проверка технологии и поддерживающего инструментария подтвердила свою эффективность на ряде промышленных проектов по разработке программного обеспечения в области телекоммуникаций.

Список литературы

1. Z.151 : User requirements notation (URN) — Language definition
`http://www.itu.int/rec/T-REC-Z.151-200811-I/en`

2. Drobintsev P.D., Nikiforov I.V., Kotlyarov V.P. Translation of UCM Real-Time Constructs into Basic Protocols // University Journal. 2013. №5. P. 193–201.

3. Hoare C.A.R. Communicating sequential processes. Prentice Hall, 1985.

4. Letichevsky A.A., Kapitonova J.V., Kotlyarov V.P., Letichevsky O.O., Volkov V.V., Baranov S.N., Weigert T. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications // Proc of ISSRE04 Workshop on Integrated Reliability Engineering (ISSRE04:WITUL), IRISA. Rennes France, 2004.

5. Baranov S., Kotlyarov V., Weigert T. Varifiable Coverage Criteria For Automated Tesdting. SDL2011: Integrating System and Software Modeling // LNCS. 2012. Vol. 7083. P. 79–89.

6. Letichevsky A.A., Kapitonova J.V., Kotlyarov V.P., Letichevsky A.A.,Jr., Nikitchenko N.S., Volkov V.A., and Weigert T. Insertion modeling in distributed system design // Software problems (Проблеми програмування). 2008. S. 13–38.

7. 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. Вып. 3 (26). С. 349–383. [I.S. Anureev, S.N. Baranov, D.M. Beloglazov, E.V. Bodin, P.D. Drobintsev, A.V. Kolchin, V.P. Kotlyarov, A.A. Letichevsky, O.A. Letychevsky, V.A. Nepomnyashchii, I.V. Nikiforov, S.V. Potiyenko, L.V. Priima, B.V. Tyutin. Tools of Integrated Technology for Analysis and Verification of Telecom Application Specs // Tr. SPIIRAN. 2013. 3(26). P. 349–383 (in Russian)].

8. Letichevsky A.A., Godlevsky A.B., Letichevsky Jr., A.A., Potienko S.V., Peschanenko V.S. Properties of Predicate Transformer of VRS System // Cybernetics and System Analyses. 2010. 4. P. 3–16.

9. Godlevsky A.B., Potienko S.V. Backward transformation of formulas in symbolic modeling: from the result to the source formula // Problems of Programming. 2010. 4. P. 363–368.

10. 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, Russia, 2005. P. 81–86.

11. Sales I. A Bridging Methodology for Internet Protocols Standards Development: Ph.D. Master of Computer Science Thesis. Ottawa, Ontario, Canada, School of Information Technology and Engineering — S.I.T.E., September 30 2001. P. 119.

12. Vizovitin N.V., Nepomniashcy V.A., Stenenko A.A., Verification of UCM-Specifications of Distributed System Using Colored Petri Nets
`www.iss.nsk.su/files/preprints/168.pdf`

13. Никифоров И.В. Методы автоматизации построения поведенческой модели программого продукта на основе UCM-спецификаций: Дис. ... канд. техн. наук. СПб.: СПбГПУ, 2014. 150 с. [Nikiforov I.V. Metody avtomatizatsii postroeniya povedencheskoy modeli programmogo produkta na osnove UCM-spetsifikatsiy: Dis. ... kand. tekhn. nauk. SPb.: SPbGPU, 2014. 150 s. (in Russian)].

14. Колчин А.В. Разработка инструментальных средств для проверки формальных моделей асинхронных систем: Дис. ... канд. физ.-мат. наук. Киев, 2009. 140 с. [Kolchin A.V. Razrabotka instrumentalnykh sredstv dlya proverki formalnykh modeley asinkhronnykh sistem: Dis. ... kand. fiz.-mat. nauk. Kiev, 2009. 140 s. (in Russian)].

15. Wan Fokking. Introduction to Process Algebra, Computer science monograph 2d edition. Springrer-Verlag, 2007. 169 p.

Approach to Behavior Scenarios Debugging

Drobintsev P.D.[1], Kotlyarov V.P.[1], Nikiforov I.V.[1], Letichevsky A.A.[2], Peschanenko V.S.[3]

[1]*St. Petersburg State Polytechnical University*
Polytechnicheskaya st., 29, St. Petersburg, 195251, Russia
[2]*Glushkov Institute of Cybernetic of NAS of Ukraine,*
Glushkova av., 40, Kyiv, 03187, Ukraine
[3]*Kherson State University,*
40 rokiv Zhovtnya St., 27, Kherson, 73000 Ukraine

Keywords: behavior model, symbolic trace, symbolic scenario, guide, causes of errors, semi-automatic analysis, futomatic analysis of error causes

The paper presents two approaches to debugging the application model behavior scenarios: semi-automatic and automatic. The first approach allows a user to automatize the process of finding the place in a concrete behavioral scenario that is suspicious of being a cause of an error. The second approach allows, in a single cycle of the analysis, to automatically identify not only the place, but also possible causes of errors in a given set of generated behavioral symbolic scenarios

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;
Никифоров Игорь Валерьевич
Санкт-Петербургский государственный политехнический университет, аспирант;
Летичевский Александр Адольфович,
Институт кибернетики им. В.М. Глушкова НАН Украины,
зав. отделом теории цифровых автоматов, академик, д-р физ.-мат. наук
Песчаненко Владимир Сергеевич,
Херсонский государственный университет, Херсон, Украина,
канд. физ.-мат. наук, доцент