Моделирование и анализ информационных систем. Т. 23, №6 (2016), с. 688–702
Modeling and Analysis of Information Systems. Vol. 23, No 6 (2016), pp. 688–702

DOI: 10.18255/1818-1015-2016-6-688-702

УДК 519.172

Применение раскрашенных сетей Петри для верификации конструкций управления сценариями языка UCM

Визовитин Н.В., Непомнящий В.А, Стененко А.А.

получена 15 марта 2016

Аннотация. В данной работе представлен метод анализа и верификации моделей Use Case Maps (UCM) с конструкциями управления сценариями — защищенными компонентами и конструкциями обработки ошибок. Анализ и верификация UCM моделей проводится с помощью раскрашенных сетей Петри (РСП) и верификатора SPIN. Приводятся описания алгоритмов трансляции UCM в РСП и РСП во входной язык Promela системы SPIN. Впервые представлены оценки для количества элементов результирующих РСП моделей в зависимости от количества элементов исходных UCM моделей с конструкциями управления сценариями, а также количества состояний Promela моделей. Представленный метод анализа и верификации демонстрируется на примере верификации программы обновления прошивки маршрутизатора.

Ключевые слова: верификация, трансляция, нотация Use Case Maps, раскрашенные сети Петри, верификатор SPIN, защищенный компонент, обработка ошибок

Для цитирования: Визовитин Н.В., Непомнящий В.А, Стененко А.А., "Применение раскрашенных сетей Петри для верификации конструкций управления сценариями языка UCM", *Моделирование и анализ информационных систем*, **23**:6 (2016), 688–702.

Об авторах:
Визовитин Николай Валерьевич, orcid.org/0000-0001-7420-2711, младший научный сотрудник,
Институт систем информатики им. А.П. Ершова СО РАН,
проспект Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: vizovitin@gmail.com

Непомнящий Валерий Александрович, orcid.org/0000-0003-1364-5281, канд. физ.-мат. наук, доцент,
Институт систем информатики им. А.П. Ершова СО РАН,
проспект Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: vnep@iis.nsk.su

Стененко Александр Александрович, orcid.org/0000-0002-9538-6887, аспирант,
Институт систем информатики им. А.П. Ершова СО РАН,
проспект Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: stirpar@mail.ru

Благодарности:
Работа частично поддержана грантом РФФИ 14-07-00401.

Введение

Предотвращение ошибок на ранних стадиях разработки программных проектов, таких как сбор и анализ требований, является важной проблемой в силу высокой стоимости их исправления на более поздних стадиях. Графическая нотация Use Case Maps (UCM) [12] позволяет формально описывать и анализировать функциональные требования. В то же время она поддерживает возможность контроля

требований заказчиком. UCM модели могут использоваться для решения широкого спектра задач. Они применяются для генерации тестовых сценариев [5, 6], формирования критериев покрытия тестами [4] и как язык спецификации свойств [10] для верификаторов.

UCM модели представляют набор сценариев как совокупность причинно-следственных связей между действиями в моделируемой системе. Действия могут быть связаны с компонентами системы, отражая ее архитектуру. Нотация UCM описывает взаимодействие архитектурных сущностей, уделяя внимание причинно-следственным связям и абстрагируясь от некоторых деталей передачи сообщений и обработки данных.

Средства для анализа и верификации UCM моделей недостаточно развиты. Стандарт UCM [12] описывает процедуру анализа, реализованную в редакторе jUCMNav [13]. Но эта техника анализа является достаточно примитивной и сложна в использовании. Стандарт дает семантику языка в неформальном виде, используя процедуры обхода UCM моделей. Поэтому часть работ, посвященных UCM, рассматривают формальную семантику [8]. В нескольких работах рассматривается проблема верификации UCM моделей. Среди них отметим работу [9], в которой предлагается метод моделирования и валидации UCM моделей с временными конструкциями. Методы верификации UCM моделей были разработаны для отдельных предметных областей. В работе [1] представлены методы тестирования, анализа и верификации для телекоммуникационных систем, основанные на UCM моделях.

В наших работах [16, 17] был представлен новый подход для анализа и верификации UCM моделей, базирующийся на раскрашенных сетях Петри (РСП). В рамках этого подхода UCM модель транслируется в РСП. Полученная РСП может быть верифицирована с помощью разработанного верификатора РСП методом проверки моделей, использующего известную систему SPIN [11], и проанализирована с помощью системы CPN Tools [7]. В работе [17] расширяется набор поддерживаемых конструкций UCM защищенными компонентами и конструкциями для обработки ошибочных ситуаций. В настоящей работе, наряду с описанием метода трансляции UCM моделей в РСП, представлены оценка размера результирующих РСП, а также оценка числа состояний программы на входном языке Promela системы верификации SPIN.

1. Обзор нотации Use Case Maps

Нотация Use Case Maps является одним из языков, описанных в стандарте User Requirements Notation [12]. UCM является высокоуровневым сценарно-ориентированным графическим средством моделирования, описывающим поведение системы через причинно-следственные связи между событиями и действиями, которые могут быть связаны со структурой компонентов системы. Нотация упрощает моделирование и анализ функциональных требований для распределенных и параллельных систем, а также позволяет рассуждать об архитектуре рассматриваемой системы.

Ниже приведен обзор основных элементов нотации UCM. Подробное описание языка, включая графический синтаксис, приведено в [12]. *Диаграмма (map)* может содержать произвольное число путей и компонентов. *Пути (path)* выража-

ют причинно-следственные связи между действиями. Пути являются направленными и могут соединять несколько типов *узлов (Path Node)*. Пути начинаются в *начальных узлах (Start Point)* и заканчиваются в *конечных узлах (End Point)*, которые отражают начальные и результирующие состояния системы соответственно, или предусловия и постусловия. Начальные узлы также могут обозначать начало сценариев для обработки ошибок и исключений. Такие узлы называют *начальными узлами обработчиков ошибок (Failure Start Points)* и *исключений (Abort Start Points)* соответственно. Тип начального узла определяется атрибутом `failureKind`, а атрибут `failureList` определяет список ошибочных и исключительных ситуаций, которым он соответствует. Начальный узел обработчика исключений является начальным узлом обработчика ошибок, который также останавливает все сценарии в своей области действия — диаграмме, на которой он находится, а также всех дочерних диаграммах в соответствии с иерархией узлов расширения (см. ниже). *Узлы действия (Responsibility)* отражают шаги, необходимые для выполнения сценария. *Or-ветвления (Or-Fork)* с условиями выбора исходящих ветвей и *Or-объединения (Or-Join)* используются для моделирования альтернативного выбора и циклов. *And-ветвления (And-Fork)* и *And-объединения (And-Join)* выражают параллелизм. Нотация UCM не накладывает никаких ограничений на порядок и вложенность элементов ветвления и объединения. *Узлы ожидания (Waiting Place)* и *таймеры (Timer)* обозначают точки, в которых исполнение сценария приостанавливается, пока не будет выполнено некоторое условие, или до прибытия сигнала. Как правило, у таймеров два исходящих пути: *основной (regular path или release path)* и *альтернативный (timeout path). Узлы соединения (Connect)* и *пустые узлы (Empty Point)* используются для синхронного или асинхронного соединения двух путей. Эти элементы, как правило, не имеют собственного графического представления. При достижении сценарием *узла прерывания (Failure Point)* он может прерваться, если произойдет ошибка или исключение. Каждый узел прерывания имеет условие срабатывания, а также имя, идентифицирующее произошедшую ошибку или исключение. Имя определяет начальные узлы обработчиков, с которых должно продолжиться исполнение сценария, если условие срабатывания оказалось истинным. UCM модели могут быть иерархически декомпозированы на *дочерние диаграммы (plug-in map)*, содержащие переиспользуемые шаблоны поведения и структуры, с помощью *узлов расширения (Stub)*.

Компоненты (Component) используются для моделирования структурных особенностей системы. Говорят, что элементы диаграммы, изображенные внутри компонента, *связаны (bound)* с ним. UCM модели без компонентов называют *свободными (unbounded)*. Компоненты могут содержать вложенные компоненты и иметь различные типы. Большинство типов компонентов не влияют на семантику и служат лишь для передачи архитектурных особенностей системы. Исключения составляют *объекты (Object)* и *защищенные (protected)* компоненты, ограничивающие количество одновременно исполняемых сценариев внутри них. В стандарте максимальное количество одновременно исполняемых сценариев внутри защищенного компонента равно 1. Следовательно, защищенные компоненты работают как механизм взаимного исключения для одновременно исполняемых сценариев.

2. Метод трансляции UCM моделей в раскрашенные сети Петри

Для анализа и верификации UCM моделей они транслируются в раскрашенные сети Петри [14]. Входные и выходные модели представляются в виде иерархических ориентированных графов с дополнительной информацией, ассоциированной с вершинами и дугами. Алгоритм трансляции переводит одно представление в другое.

2.1. Ограничения на входные UCM модели

На входные UCM модели налагаются следующие ограничения. Считается, что направление всех путей однозначно определено и все узлы модели уникально идентифицируемы. Особая семантика обхода узлов, связанных с компонентами типа объект, не поддерживается. UCM модели с начальными узлами обработчиков исключений отвергаются. Эти узлы редко используются при моделировании и вызывают комбинаторный взрыв количества состояний РСП модели в силу семантики остановки множества сценариев.

Также для трансляции защищенных компонентов введем дополнительное ограничение на использование узлов Or-ветвления, Or-объединения и таймеров. Каждый такой узел должен либо одновременно со всеми своими смежными узлами быть связанным с защищенным компонентом, либо нет. Это ограничение не является существенным, т.к. может быть удовлетворено простой модификацией исходной модели, например, введением дополнительных пустых узлов.

Указанные ограничения позволяют поддерживать наиболее часто используемые элементы и сценарии их использования. Поэтому они не ограничивают спектр поддерживаемых UCM моделей существенным образом.

2.2. Обзор алгоритма трансляции UCM моделей в РСП

Алгоритм трансляции UCM моделей в раскрашенные сети Петри на верхнем уровне состоит из пяти этапов. Алгоритм трансляции подробно описан в [15 – 17].

На первом этапе входная UCM модель подвергается предварительной обработке. В частности, осуществляется проверка удовлетворения ограничениям алгоритма, определяются начальные значения для всех переменных, входная модель преобразуется в свободную (т.е. из нее удаляются все компоненты). Информация о защищенных компонентах сохраняется в атрибутах узлов, связанных с ними. Пользователь уведомляется о любых неявных преобразованиях на данном этапе.

На втором этапе создаются определения на языке CPN ML, общие для всей модели. На этом этапе определяются цвета, константы и некоторые переменные. Вводятся служебные цвета, включая `UNIT` — стандартный «основной» цвет с единственным возможным значением `()`. Фишки цвета `UNIT` в основном используются для моделирования передачи сигналов и выполнения сценариев.

На третьем этапе дуги графа UCM модели, полученной в результате предварительной обработки на первом этапе, кроме дуг, инцидентных узлам соединения, разбиваются вершинами, соответствующими узлам нового типа — *вспомогатель-*

ным узлам. Преобразование, осуществляемое на данном этапе, сохраняет аннотации на исходящих дугах.

На четвертом этапе каждый узел вместе со своей непосредственной окрестностью, определяемой узлами соединения и вспомогательными узлами, обрабатывается по отдельности. Каждый рассматриваемый узел преобразуется во фрагмент модели РСП — аннотированный граф с дополнительными определениями на языке CPN ML. Фрагмент РСП также создается для каждого имени ошибки из UCM модели.

На пятом этапе объединяются фрагменты РСП, полученные на четвертом этапе. Элементы полученной модели с одинаковыми именами либо сливаются, если это возможно, либо представляются в виде объединяющих мест (fusion place).

3. Трансляция узлов, связанных с защищенными компонентами

Для эффективной верификации UCM моделей с помощью РСП необходимо, чтобы количество одновременно исполняющихся сценариев в любой заданной точке было ограничено. Иначе, в транслированной РСП появляются места с неограниченной емкостью, моделирующие транспорт сигналов.

Стандарт UCM предлагает способ для моделирования механизма взаимного исключения исполнения сценариев для подмножества путей модели. Таким механизмом являются защищенные компоненты, изображаемые с двойным контуром. Элементы UCM модели, связанные с любым экземпляром защищенного компонента, попадают под его действие. Исполнение любого сценария может быть продолжено внутрь защищенного компонента, только если никакой другой сценарий не исполняется внутри него.

Заметим, что предлагаемый стандартом механизм является слишком ограниченным для того, чтобы одновременно представлять широкий спектр взаимодействий различных сценариев и ограничивать количество исполняющихся сценариев. Поэтому мы предлагаем расширить стандарт, позволяя задавать максимальное количество одновременно исполняющихся сценариев внутри защищенного компонента. Это можно сделать, либо добавив новый целочисленный атрибут `scenarios` в класс *Component* абстрактной грамматики UCM, либо используя связанные с компонентом элементы комментария. Второй способ может быть использован для того, чтобы избежать необходимости модификации существующих редакторов UCM.

Далее будет рассмотрена трансляция компонентов с атрибутом `protected = true` и положительным значением атрибута `scenarios`, и других элементов UCM модели, связанных с ними. Заметим, что если `scenarios = 1`, то такой компонент является защищенным компонентом в терминах действующего стандарта UCM.

Каждый защищенный компонент, как и другие элементы UCM, имеет имя. Также мы ввели дополнительное ограничение на использование узлов Or-ветвления, Or-объединения и таймеров. Оно позволяет избежать неоднозначности трактовки семантики UCM моделей и существенного усложнения алгоритма трансляции.

На первом этапе алгоритма трансляции дополнительно необходимо проверить ограничение `scenarios > 0` для всех защищенных компонент. Если оно не соблю-

дается, модель считается некорректной. Далее проверим дополнительное ограничение на узлы Or-ветвления, Or-объединения и таймеры. Если оно не выполняется, то модель считается непригодной к верификации и пользователю предлагается исправить ее путем добавления дополнительных пустых узлов на дуги, инцидентные узлам, вызвавшим проблему, или любым другим подходящим способом. Для каждого связанного узла UCM модели с ним ассоциируется информация (имя и значение атрибута `scenarios`) о каждом защищенном компоненте, с которым он связан. Заметим, что один узел может быть связан с несколькими различными защищенными компонентами. После этого все компоненты из модели можно удалить.

Второй и третий этапы алгоритма трансляции не меняются по сравнению с этими этапами, описанными в разделе 2.2.

Защищенные компоненты в РСП будут моделироваться с помощью так называемых антимест. Введение антимест является распространенным шаблоном моделирования в РСП, используемым для ограничения количества фишек в заданном фрагменте РСП. Начальная разметка создаваемых антимест состоит из такого же количества фишек типа `UNIT`, как и максимальное количество сценариев, одновременно исполняющихся в данном компоненте (значение атрибута `scenarios`).

На четвертом этапе алгоритма трансляция каждого узла, связанного в исходной модели, может породить необходимое количество антимест (по количеству связанных с ним защищенных компонентов) и дуги к переходу, соответствующему узлу, или от него. Только узлы, способные породить или остановить сценарии, проходящие через них и защищенный компонент, породят дополнительные дуги и антиместа. К таким узлам относятся And-ветвления, And-объединения, начальные узлы и конечные узлы. Между именами защищенных компонентов и соответствующих им антимест поддерживается взаимно однозначное соответствие.

Пятый этап алгоритма фактически остается без изменений — все дополнительные антиместа будут объединены в соответствии со своими именами, аналогично другим местам объединяемых фрагментов РСП.

Алгоритм трансляции узлов, связанных с защищенными компонентами, рассмотрен подробнее в [17].

4. Трансляция конструкций для обработки ошибок

Для обработки ошибок в UCM моделях используются узлы прерывания и начальные узлы обработчиков ошибок. Узел прерывания представляет собой точку пути, в которой продолжение исполнения сценария зависит от происхождения ошибки или исключения. Если произошла ошибка, то текущий сценарий прерывается и начинаются новые с соответствующих начальных узлов обработчиков ошибок. Соответствие между узлами прерывания и начальными узлами обработчиков ошибок определяется через имя ошибки.

Описание первого, второго и третьего этапов алгоритма трансляции дано в разделе 2.2.

На четвертом этапе алгоритма трансляции каждому имени ошибки в модели сопоставляется фрагмент РСП — переход и место с соответствующими именами. Место имеет тип `UNIT` и пустую начальную разметку. От места к переходу ведет

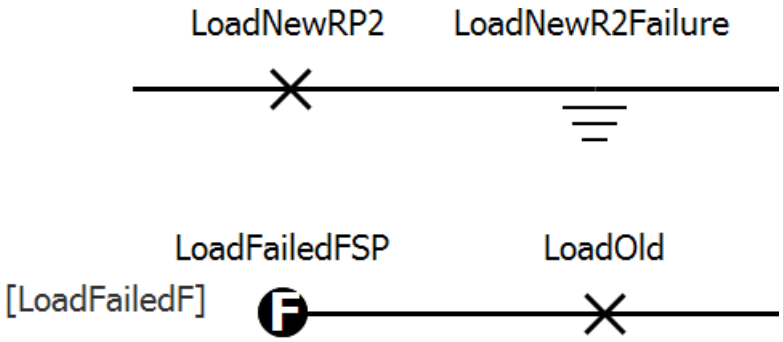

Рис. 1. Фрагмент UCM модели обработчика ошибок загрузки прошивки маршрутизатора
Fig. 1. UCM fragment with router firmware load errors handler

дуга с надписью `()`. Дуги также ведут от перехода ко всем местам, соответствующим начальным узлам обработчиков ошибок с соответствующим именем ошибки в списке ошибок, которые они могут обрабатывать. Эти дуги также имеют надпись `()`. Эта процедура трансляции во многом похожа на трансляцию узлов And-ветвления [16].

Трансляция начальных узлов обработчиков ошибок аналогична трансляции начальных узлов на дочерних диаграммах в случае, если они связаны со входами узла расширения [16]. Переход, соответствующий начальному узлу обработчика ошибок, связывается с местом типа `UNIT` с пустой начальной разметкой. Дуга от места к переходу имеет надпись `()`.

Трансляция узлов прерывания аналогична трансляции узлов Or-ветвления [16] с двумя выходными путями. Дуга, соответствующая одному из этих путей, продолжает штатное исполнение сценария, другая дуга ведет к месту, имя которого соответствует имени ошибки для этого узла прерывания. Условия на исходящих дугах основаны на условии срабатывания ошибки — одна из дуг переносит фишку, если условие истинно, другая – если оно ложно. Поэтому при трансляции узлов прерывания не возникает необходимости в дополнительных местах `*_OrForkWarnings`, которые используются для проверки того, что условия на выходных дугах узла Or-ветвления являются взаимоисключающими.

Заметим, что узел прерывания и начальный узел обработчика ошибок, ему соответствующий, могут быть на разных диаграммах. Этот случай разрешается естественным образом на пятом этапе алгоритма посредством преобразования отдельных мест, смежных с переходом, соответствующим имени ошибки, в объединяющие места при слиянии фрагментов РСП.

На Рис. 1 представлен пример фрагмента UCM модели с узлом обработчика ошибок. На Рис. 2 представлен соответствующий ему фрагмент РСП.

5. Оценка размера результирующих моделей РСП

В [16] были приведены оценки размера результирующей модели РСП в зависимости от размера исходной UCM модели. В этой работе рассматривались UCM модели без конструкций для обработки ошибок и без защищенных компонентов.

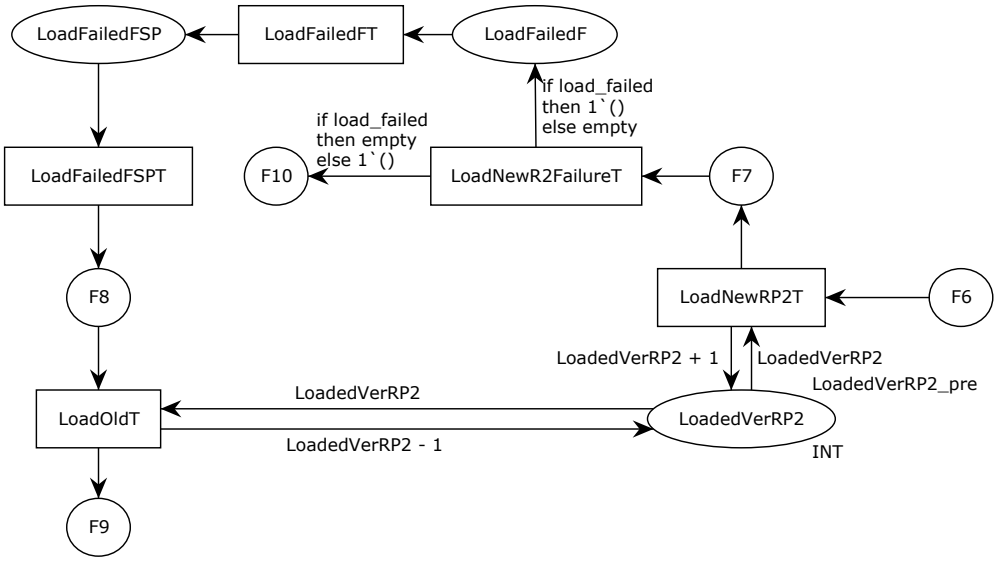

Рис. 2. РСП, полученная трансляцией фрагмента UCM, представленного на Рис. 1
Fig. 2. CPN translated from the UCM fragment presented on Fig. 1

Обозначим множество узлов UCM модели через V, множество дуг через E, множество переменных через $Vars$. Обозначим через $|A|$ количество элементов множества A. Обозначим максимальную степень захода среди всех узлов UCM модели через d^+_{max}. Также пусть d_s — максимальная степень среди узлов расширения, c_s — максимальное количество дочерних диаграмм среди всех узлов расширения, n_s — общее количество узлов расширения. Тогда общее количество переходов РСП ограничено величиной $O(d^+_{max}|V|+n_sc_sd_s)$. Общее количество мест ограничено $O(|E|+|V|+|Vars|+n_sc_sd_s)$. Общее количество дуг ограничено $O(|E|+d^+_{max}|V|+|Vars||V|+n_sd_s(|Vars|+c_s))$.

Пусть C — множество защищенных компонентов, F — множество имен ошибок в исходной UCM модели. Каждый защищенный компонент при трансляции создает одно дополнительное место и количество дуг, ограниченное удвоенным количеством узлов, связанных с этим компонентом. Использование защищенных компонентов не порождает дополнительных переходов. Для каждого имени ошибки при трансляции создается одно дополнительное место и один переход. Также, если V_f — множество начальных узлов обработчиков ошибок с соответствующим именем, то количество дополнительных дуг равно $|V_f|+1$ или $O(|V|)$. Для целей оценки количества элементов РСП узлы прерывания допустимо рассматривать как узлы Or-ветвления, а начальные узлы обработчиков ошибок как начальные узлы.

Поэтому, если в исходной UCM модели были использованы рассмотренные в данной работе новые конструкции — защищенные компоненты и конструкции для обработки ошибок, то общее количество переходов РСП ограничено величиной $O(d^+_{max}|V|+|F|+n_sc_sd_s)$. Общее количество мест ограничено $O(|E|+|V|+|Vars|+|C|+|F|+n_sc_sd_s)$. Общее количество дуг ограничено $O(|E|+|V|(d^+_{max}+|Vars|+|C|+|F|)+n_sd_s(|Vars|+c_s))$.

Таким образом, общее количество вершин РСП (то есть мест и переходов, в том

числе подстановочных) ограничено величиной $O(|E| + d^+_{max}|V| + |Vars| + |C| + |F| + n_s c_s d_s)$.

Заметим, что максимальное количество дочерних диаграмм c_s среди всех узлов расширения часто ограничено небольшой константой, что следует из естественного требования удобного представления исходной UCM модели.

Пусть N — общее количество элементов в исходной UCM модели (включая узлы, переменные, компоненты и имена ошибок). Тогда оценка может быть упрощена до $O(N^2)$ вершин РСП, т.е. является квадратичной в зависимости от количества элементов UCM модели.

Полученная оценка может быть уменьшена при наложении дополнительных ограничений на UCM модель. Так, если в UCM модели нет узлов Or-объединения и узлов расширения, то оценка принимает вид $O(|E| + |V| + |Vars| + |C| + |F|)$, т.е. $O(N)$.

Приведенные оценки показывают эффективность алгоритма трансляции в РСП. Меньший размер РСП модели также упрощает процесс анализа контрпримеров.

6. Верификация раскрашенных сетей Петри

6.1. Метод верификации раскрашенных сетей Петри

Формальная верификация может проводиться для РСП, у которых ограничена ёмкость мест и длина списков. Для проведения формальной верификации свойств РСП была разработана система CPNVer [2], которая включает транслятор из РСП во входной язык Promela системы верификации SPIN [11] и систему SPIN. Система CPNVer позволяет верифицировать свойства, представленные в виде формул линейной темпоральной логики LTL, а также в виде постусловий. Для постусловий проверяется истинность заданного предиката в конечных состояниях модели, то есть в таких состояниях, в которых нет допустимых переходов.

Заметим, что язык логики LTL достаточно выразителен. В частности, на этом языке можно представить многие свойства коммуникационных протоколов. Система SPIN широко применяется как для проведения исследований, так и для решения практических задач. Регулярно выходят новые версии этой системы.

Результатом верификации является либо сообщение системы SPIN о том, что проверяемые свойства выполняются, либо контрпример. Он является последовательностью срабатывания переходов РСП, которая приводит к нарушению проверяемого свойства. Контрпример может быть проанализирован с использованием системы CPN Tools [7]. Симуляция контрпримера в этой системе позволяет во многих случаях локализовать ошибку и внести изменения для её устранения.

6.2. Трансляция раскрашенных сетей Петри в язык Promela

Язык Promela оперирует понятиями процессов и переменных. Значения переменных входят в состояние программы. Процессы могут содержать инструкции недетерминированного выбора, а при верификации система SPIN последовательно выполнит каждую из альтернатив такого выбора. Программа на языке Promela может содержать вставки на языке C, а переменные языка C могут быть включены в состояние

программы. Инструкции процесса могут быть сгруппированы в атомарно выполняющиеся блоки.

Программа на языке Promela, полученная в результате трансляции, содержит набор переменных, каждая из которых хранит значение, соответствующее разметке определённого места исходной РСП, и единственный процесс, изменяющий значения этих переменных. В начале работы этот процесс инициализирует переменные, присваивая им значения, транслированные из начальной разметки мест. Затем процесс начинает выполнять главный цикл, каждая итерация которого состоит из двух этапов. На первом этапе происходит поиск допустимых в текущем состоянии переходов. На втором этапе происходит моделирование срабатывания одного из допустимых переходов, выбранного недетерминированно. Если допустимых переходов в текущем состоянии нет, то программа проверяет постусловие и завершает работу. Непосредственно перед завершением работы программа обнуляет значения переменных, это делается для того, чтобы избежать появления избыточных состояний. Каждая итерация главного цикла выполняется атомарно и не порождает промежуточных состояний.

Алгоритм трансляции состоит из нескольких этапов, на которых последовательно транслируются типы данных, функции, переменные, места и переходы исходной РСП. Типы данных РСП транслируются в типы данных языка C. При этом записи и кортежи транслируются в структуры, содержащие поля соответствующих типов, а списки транслируются в структуры, одно из полей которых — это массив для хранения элементов списка, а другое — это переменная, хранящая длину списка.

Для каждого типа данных РСП на языке C определяется тип «мультимножество элементов такого типа». Представление мультимножеств различается в зависимости от типа его элементов. Для типов `UNIT`, булевских типов и перечислений мультимножество представляется набором счётчиков числа вхождений каждого из значений типа данных. Для прочих типов мультимножество представляется массивом входящих в него элементов, а также переменной, хранящей ёмкость мультимножества. Элементы в указанном массиве хранятся упорядоченно, чтобы обеспечить единственность представления мультимножества и, таким образом, не допустить роста числа состояний программы.

Места РСП транслируются в переменные-мультимножества, а выражения начальной разметки мест транслируются в код на языке C, который выполняется на этапе инициализации программы и присваивает мультимножествам их начальные значения.

Каждый переход РСП транслируется в два фрагмента кода на языке Promela. Первый фрагмент проверяет, является ли данный переход допустимым, второй фрагмент моделирует срабатывание перехода, при условии, что он является допустимым. При генерации этих фрагментов кода, выражения на дугах РСП транслируются в код на языке C.

При проверке допустимости перехода выражения на дугах, инцидентных данному переходу, используются для того, чтобы определить, какие значениях переменных могут сделать его допустимым. С этой целью для каждой входящей в переход дуги производится перебор возможных значений выражения на данной дуге. В качестве возможных значений рассматриваются элементы, которые входят в разметку инцидентного этой дуге места. По значению выражения на дуге программа опреде-

ляет значения некоторых переменных, входящих в выражение. При этом значение каждой из переменных перехода должно определяться во время рассмотрения хотя бы одной из входных дуг перехода. Для найденного таким образом набора значений переменных перехода программа проверяет допустимость перехода следующим образом. Программа вычисляет, что охранное значение истинно, а также что для каждого из входных мест перехода его разметка содержит элемент, значение которого определяется при вычислении значения на соответствующей входной дуге перехода.

Второй фрагмент кода моделирует срабатывание перехода при известных значениях переменных перехода. Моделирование срабатывания перехода заключается в изменении значений переменных-мультимножеств, в соответствии с вычисленными при данных значениях переменных перехода выражениями на инцидентных переходу дугах. Во время моделировния срабатывания перехода программа показывает название данного перехода и значения его переменных для того, чтобы в случае обнаружения ошибки можно было определить, какая последовательность срабатывания переходов привела к нарушению свойств.

Подробное описание алгоритма трансляции дано в препринте [2].

6.3. Оценка числа состояний транслированной модели

Состояние программы на языке Promela включает значения переменных и значения указателей на инструкции каждого из процессов. Программа, полученная в результате трансляции, содержит единственный процесс. Указатель на инструкцию этого процесса может принимать одно из следующих значений: начало инициализации переменных, начало итерации главного цикла, завершение работы процесса. Эти значения не зависят от исходной РСП, так как инструкции процесса сгруппированы в атомарные блоки, которые при верификации исключают промежуточные состояния из рассмотрения. Переменные программы имеют ненулевые значения только в тех состояниях, когда указатель на инструкцию находится в начале итерации главного цикла. Каждое такое состояние, достижимое в процессе работы программы, взаимно однозначно отображается на достижимое состояние исходной РСП. Кроме этих состояний программа может находиться в начальном состоянии, когда переменные не инициализированы, и в конечном состоянии, в которое программа переходит, если нет достижимых переходов. Таким образом, число состояний программы составляет $N + 2$, где N — число достижимых состояний исходной РСП.

При проверке свойств, заданных LTL-формулами, система SPIN добавляет в программу ещё один процесс, представляющий собой автомат Бюхи для этой формулы [11]. В этом случае в состояние программы включается также состояние этого вспомогательного процесса. Если число состояний этого процесса составляет K, то число состояний программы не превосходит $(N + 2)K$.

Таким образом, число состояний программы на языке Promela, которая проверяет заданное свойство, линейно зависит от числа состояний исходной РСП, что позволяет применять систему SPIN для широкого класса моделей.

7. Пример верификации программы обновления прошивки маршрутизатора

Доступность сети становится все более важной проблемой для поставщиков услуг и их клиентов. Основной причиной простоя сетевого оборудования и недоступности сети часто является запланированное техническое обслуживание и обновление программного обеспечения. Многие производители маршрутизаторов, такие как Cisco Systems, предлагают оборудование с возможностью обновления программного обеспечения без прерывания обслуживания (ISSU), т.е. с минимальными перебоями для проходящего сетевого трафика клиентов.

Одна из тактик для обеспечения высокой доступности заключается в использовании избыточности компонент (активного и пассивного обработчика маршрутов). В [15] приведен пример UCM модели обновления маршрутизатора с двумя обработчиками маршрутов (RP) со старой версии прошивки на новую. Маршрутизатор работает в режиме нагруженного резерва, где RP1 является активным обработчиком, а RP2 – резервным. Первым шагом для обновления прошивки является ее копирование на файловые системы обоих обработчиков. Затем новая версия загружается на резервный обработчик RP2, что может привести к ошибке в силу несовместимого или поврежденного программного обеспечения. В случае возникновения ошибок загружается старая версия прошивки. Фрагмент UCM модели, описывающий эту ситуацию, представлен на Рис. 1.

Если новая версия была успешно загружена, RP2 перезагружается и входит в режим переключения с сохранением состояния (SSO). Если переключение происходит успешно и в течение некоторого времени новая версия принимается администратором, то ставший пассивным обработчик RP1 также обновляется на новую версию и процедура обновления успешно завершается. Иначе, происходит возврат к предыдущей версии прошивки и процедура обновления считается завершенной неуспешно.

Для формализации требований к процедуре обновления прошивки маршрутизатора по этим требованиям была построена UCM модель. При верификации UCM модели проверялись следующие свойства:

1. Во всех состояниях модели один обработчик маршрутов является активным. Это свойство, сформулированное в виде LTL-формулы [15], позволяет убедиться, что нет перебоев в обслуживании.
2. В конечном состоянии проверяются постусловия конечных узлов UCM модели. Эти условия позволяют убедиться, что обновление маршрутизатора завершилось либо корректным возвратом к предыдущей версии в случае ошибок, либо корректным обновлением до новой версии прошивки.

В результате верификации с помощью системы CPNVer после трансляции модели из UCM в РСП, была обнаружена ошибка, приводящая к нарушению первого свойства. В результате анализа контрпримера исходная UCM модель была модифицирована и повторно верифицирована. Верификация второй модели прошла успешно. Подробное описание UCM моделей и их верификации приведено в [15]. В полученной UCM модели зафиксированы корректные требования к системе. Эта UCM модель может быть использована для проверки построенного программного обеспечения на соответствие требованиям, а также для автоматического создания тестовых сценариев для проверки программного обеспечения [1, 4].

Заключение

Графическая нотация Use Case Maps является выразительным средством описания функциональных требований к программным системам и протоколам. В данной работе мы представили алгоритмы и средства для трансляции UCM моделей в раскрашенные сети Петри и модели на входном языке Promela известной системы верификации SPIN. Описанный метод позволяет анализировать и верифицировать более сложные UCM модели по сравнению с методом, описанным в работе [16], благодаря поддержке конструкций для обработки ошибочных ситуаций и защищенных компонентов.

Защищенные компоненты с расширенной семантикой особенно полезны для верификации. Они позволяют ограничить количество одновременно исполняемых сценариев, тем самым ограничивая емкость мест в соответствующей РСП модели. Это свойство позволяет обеспечить финитность модели и верифицировать её с помощью системы SPIN.

Текущая версия нашей системы поддерживает трансляцию файлов редактора jUCMNav [13] в файлы CPN Tools [7]. UCM модели, транслированные в РСП, могут быть верифицированы, используя систему CPNVer [2]. Процедура верификации показывает, является ли верифицируемая модель корректной по отношению к заданному свойству. Если это не так, необходимо локализовать ошибку. Для этой цели можно использовать контрпримеры, полученные системой SPIN. Для их анализа можно использовать систему CPN Tools.

Алгоритм трансляции UCM моделей в РСП является эффективным, так как он имеет квадратичную сложность в отношении размера результирующей РСП в зависимости от количества элементов UCM модели.

Важной задачей является обоснование корректности трансляции UCM в РСП. Однако это требует формальной семантики для UCM, которую стандарт [12] не предоставляет.

Мы планируем применить представленный метод для временных расширений [9] нотации UCM.

Список литературы / References

[1] 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”, *Тр. СПИИРАН*, **26** (2013), 349–383; [Anureev I.S., Baranov S.N., Beloglazov D.M., Bodin E.V., Drobitsev P.D., Kolchin A.V., Kotlyarov V.P., Letichevsky A.A., Letichevsky A.A. jr., Nepomniaschy V.A., Nikiforov I.V., Potiyenko S.V., Priyma L.V., Tyutin B.V., “Tools of Integrated Technology for Analysis and Verication of Telecom Application Specs”, *SPIIRAS Proceedings*, **26** (2013), 349–383, in Russian].

[2] Стененко А.А., Непомнящий В.А., *Верификация раскрашенных сетей Петри методом проверки моделей*, Препринт 178, http://www.iis.nsk.su/files/preprints/stenenko_nepomniaschy_178.pdf, Ин-т систем информатики СО РАН, Новосибирск, 2015, 28 с.; [Stenenko A.A., Nepomniaschy V.A., *Model Checking Approach to Verifiaction of Coloured Petri Nets*, Preprint 178, http://www.iis.nsk.su/files/preprints/

stenenko_nepomniaschy_178.pdf, Institute of Informatics Systems RAS, Novosibirsk, 2015, 28 pp., in Russian].

[3] Визовитин Н.В., Непомнящий В.А., *Алгоритмы трансляции UCM-спецификаций в раскрашенные сети Петри*, Препринт 168, http://www.iis.nsk.su/files/preprints/168.pdf, Ин-т систем информатики СО РАН, Новосибирск, 2012, 55 с.; [Vizovitin N.V., Nepomniaschy V.A., *UCM-Specifications to Colored Petri Nets Translation Algotithms*, Preprint 168, http://www.iis.nsk.su/files/preprints/168.pdf, Institute of Informatics Systems RAS, Novosibirsk, 2012, 55 pp., in Russian].

[4] Kotlyarov V., Weigert T., "Verifiable Coverage Criteria for Automated Testing", *SDL 2011, LNSC 7083*, 2011, 79–89.

[5] Baranov S.N., Drobintsev P.D., Kotlyarov V.P., Letichevsky A.A., "The Technology of Automated Verification and Testing in Industrial Projects", *Proc. IEEE Russia Northwest Section*, 110 Anniversary of Radio Invention Conference, IEEE Press, St.Petersburg, 2005, 81–89.

[6] Boulet P., Amyot D., Stepien B., "Towards the Generation of Tests in the Test Description Language from Use Case Map Models", *SDL 2015, LNCS 9369*, Springer, 2015, 193–201.

[7] *CPN Tools Homepage*, http://cpntools.org/.

[8] Hassine J., Rilling J., Dssouli R., "Abstract Operational Semantics for Use Case Maps", *FORTE 2005, LNCS 3731*, 2005, 366–380.

[9] Hassine J. Early modeling and validation of timed system requirements using Timed Use Case Maps, *Requirements Engineering*, **20**:2 (2015), 181–211.

[10] Hassine J., Rilling J., Dssouli R, "Use Case Maps as a Property Specification Language", *Software and Systems Modeling*, **8**:2 (2009), 205–220.

[11] Holzmann G.J, *The SPIN model checker. Primer and Reference Manual*, Addison-Wesley, 2004.

[12] *ITU-T, Recommendation Z.151 (10/12), User Requirements Notation (URN) — Language definition*, http://www.itu.int/rec/T-REC-Z.151/en.

[13] *jUCMNav — Eclipse plugin for the User Requirements Notation*, http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome.

[14] Jensen K., Kristensen L.M, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, Springer, 2009.

[15] Vizovitin N.V., *Application of coloured Petri nets for verification of scenario control structures in UCM notation. Appendix*, http://bitbucket.org/vizovitin/ucm-verification-examples-3.

[16] Vizovitin N.V., Nepomniaschy V.A., Stenenko A.A., "Verifying UCM Specifications of Distributed Systems Using Colored Petri Nets", *Cybernetics and Sys. Anal.*, **51**:2 (2015), 213–222.

[17] Vizovitin N.V., Nepomniaschy V.A., Stenenko A.A., "Verification of UCM Models with Scenario Control Structures Using Colored Petri Nets", *System Informatics*, **7** (2016), 11–22.

Vizovitin N.V., Nepomniaschy V.A., Stenenko A.A., "Application of Coloured Petri Nets for Verification of Scenario Control Structures in UCM Notation", *Modeling and Analysis of Information Systems*, **23**:6 (2016), 688–702.

DOI: 10.18255/1818-1015-2016-6-688-702

Abstract. This article presents a method for the analysis and verification of Use Case Maps (UCM) models with scenario control structures — protected components and failure handling constructs. UCM models are analyzed and verified with the help of coloured Petri nets (CPN) and the SPIN model checker. Algorithms for translating UCM scenario control structures into CPN and CPN into SPIN input language Promela are described. The number of elements of the resulting CPN model and the

number of Promela model states are estimated. The presented algorithm and the verification process are illustrated by the study of a network router firmware update.

Keywords: verification, translation, UCM, coloured Petri nets, SPIN, protected component, error handling

About the authors:
Nikolay V. Vizovitin, orcid.org/0000-0001-7420-2711, junior researcher,
A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences,
6 Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: vizovitin@gmail.com

Valery A. Nepomniaschy, orcid.org/0000-0003-1364-5281, PhD,
A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences,
6 Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: vnep@iis.nsk.su

Alexander A. Stenenko, orcid.org/0000-0002-9538-6887, graduate student,
A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences,
6 Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: stirpar@mail.ru

Acknowledgments:
This work was partially supported by RFBR grant 14-07-00401.