УДК 519.172

Н.В. ВИЗОВИТИН, В.А. НЕПОМНЯЩИЙ, А.А. СТЕНЕНКО

ВЕРИФИКАЦИЯ UCM-СПЕЦИФИКАЦИЙ РАСПРЕДЕЛЕННЫХ СИСТЕМ С ИСПОЛЬЗОВАНИЕМ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ[1]

Аннотация. Представлена новая система анализа и верификации Use Case Maps (UCM) спецификаций с использованием раскрашенных сетей Петри и системы верификации SPIN. Стандартизованная нотация UCM — удобное графическое средство формального описания функциональных требований. Описаны алгоритмы трансляции UCM-спецификаций в раскрашенные сети Петри, а также трансляции последних во входной язык Promela системы SPIN. Приведен вывод оценки сложности получаемых раскрашенных сетей Петри. Работа представленных алгоритмов и инструментов демонстрируется на примере верификации коммуникационного протокола с локализацией и исправлением ошибок в исходной UCM-спецификации.

Ключевые слова: верификация, спецификация, распределенные системы, нотация Use Case Maps, раскрашенные сети Петри, система верификации SPIN.

ВВЕДЕНИЕ

На ранних стадиях разработки программных проектов, таких как сбор и анализ требований, большое значение имеет недопущение ошибок, которые могут привести к серьезным негативным последствиям. Графическая нотация Use Case Maps (UCM) позволяет формально описывать и анализировать функциональные требования, в то же время сохраняя возможность контроля требований заказчиком. UCM-спецификации носят универсальный характер [1]. Например, они используются для создания тестовых сценариев [2, 3], критериев покрытия тестами [4], а также как язык спецификации свойств [5].

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

Заметим, что инструменты анализа и верификации UCM-спецификаций недостаточно развиты. Стандарт UCM [6] определяет процедуру анализа, реализованную в редакторе jUCMNav [7]. Однако данная техника анализа примитивна и ее сложно эффективно использовать. Этот стандарт формально описывает абстрактный и конкретный синтаксисы вместе с ограничениями для корректного представления UCM-спецификаций. Но семантика описана неформально, с использованием требований к обходу UCM диаграмм. Поэтому цель ряда работ заключается в описании формальной семантики для UCM. J. Hassine и соавторы представили формальную семантику для подмножества UCM, основанную на абстрактных автоматах, вместе с инструментарием для симуляции UCM-спецификаций [8]. Они также исследовали применение временных ограничений в UCM-спецификациях, используя расширение UCM, названное Timed Use Case Maps [9]. Была представлена семантика, основанная на временных конечных автоматах (вместе с инструментарием на основе средства проверки моделей UPPAAL) и системах переходов с часами. Созданный инструментарий работает

[1]Работа частично поддержана грантом Российского фонда фундаментальных исследований № 14-07-00401.

с подмножеством UCM, расширенным информацией о временных задержках, однако он не предоставляет удобного способа интерпретации контрпримеров в терминах исходной UCM-спецификации.

Разработаны также методы верификации для ряда предметных областей. Методы тестирования, анализа и верификации UCM-спецификаций для телекоммуникационных приложений описаны в [10]. В работах [10, 11] предложена формальная семантика UCM-спецификаций на основе теории инсерционного программирования и базовых протоколов, а также их верификатор. В [3, 4] описаны инструментальные средства для автоматизации тестирования программного обеспечения, использующие UCM-спецификации в качестве описания поведенческих требований. UCM-спецификации преобразуются в базовые протоколы. При этом рассматриваются UCM-спецификации со специализированным текстовым вариантом описания базовых протоколов вместо языка данных, определенного в стандарте.

Таким образом, автоматизированная верификация и анализ стандартных UCM-спецификаций является открытой и актуальной проблемой.

В данной статье предложен новый подход для анализа и верификации UCM-спецификаций, базирующийся на раскрашенных сетях Петри (РСП) [12]. Описан алгоритм трансляции UCM-спецификаций в РСП, а также верхняя оценка размера результирующей РСП. Представлена система анализа и верификации UCM-спецификаций, включающая транслятор в РСП, а также верификатор РСП методом проверки моделей, использующий известную систему SPIN [13]. Предварительная версия результатов данной работы приведена в [14].

ВХОДНЫЕ UCM-СПЕЦИФИКАЦИИ

Нотация Use Case Maps. Нотация Use Case Maps — один из языков, описанных в стандарте User Requirements Notation [6]. 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 исследовать архитектуру рассматриваемой системы.

Далее приведен обзор основных элементов нотации UCM. Подробное описание языка, включая графический синтаксис, дано в [15]. Диаграмма (*map*) может содержать произвольное число путей и компонентов. Пути (*path*) выражают причинно-следственные связи между действиями. Пути являются направленными и могут соединять несколько типов узлов (*PathNode*). Пути начинаются в начальных узлах (*StartPoint*) и заканчиваются в конечных узлах (*EndPoint*), отражающих начальные и результирующие состояния системы соответственно, или предусловия и постусловия. Узлы действия (*Responsibility*) отражают шаги, необходимые для выполнения сценария. *Or*-ветвления (*OrFork*) с условиями выбора исходящих ветвей и *Or*-объединения (*OrJoin*) используются для моделирования альтернативного выбора и циклов. *And*-ветвления (*AndFork*) и *And*-объединения (*AndJoin*) выражают параллелизм. Нотация UCM не накладывает никаких ограничений на порядок и вложенность элементов ветвления и объединения. Узлы ожидания (*WaitingPoint*) 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*). Однако в нотации UCM нет понятия времени как такового. Оно присутствует только

в нестандартизованных расширениях [9]. Узлы соединения (*Connect*) и пустые узлы (*EmptyPoint*) используются для синхронного или асинхронного соединения двух путей. Эти элементы обычно не имеют собственного графического представления.

UCM-спецификации могут быть иерархически декомпозированы на дочерние диаграммы (*plug-in map*), содержащие переиспользуемые шаблоны поведения и структуры, с помощью узлов расширения (*Stub*). Схема связывания диаграмм (*PluginBinding*) определяет то, как связаны пути на дочерней и родительской диаграммах. Одним из наиболее часто встречающихся узлов расширения является узел статического расширения (*Static Stub*). Он может содержать только одну дочернюю диаграмму, в то время как другие узлы расширения могут содержать их много. Такие узлы называют узлами динамического расширения (*Dynamic Stub*). Среди них также выделяют синхронизирующие (*Synchronizing Stub*) и блокирующие узлы расширения (*Blocking Stub*); в данной работе они не рассматриваются.

Компоненты (*Component*) используются для моделирования структурных особенностей системы. Полагают, что элементы диаграммы, изображенные внутри компонента, связаны (*bound*) с ним. UCM-спецификации без компонентов называют свободными (*unbounded*). Большинство типов компонентов не влияют на семантику, кроме объектов (*Object*) и защищенных (*protected*) компонентов.

В UCM сценарии выражаются на более абстрактном уровне, чем передача сообщений между компонентами. Одна из отличительных особенностей UCM заключается в возможности изобразить сразу несколько сценариев, тем самым позволяя судить об архитектуре системы и ее поведении в рамках нескольких сценариев.

Нотация UCM дает возможность строить модели различной степени формальности, достаточной для текущей задачи. Это отчасти поддерживается возможностью применения как естественного, так и формального языка для задания действий и условий. Язык данных URN [6] используется в качестве формального языка. Он представляет собой достаточно простой текстовый язык, основанный на подмножестве языка выражений SDL. Язык поддерживает три базовых типа данных (булевый, целочисленный и перечисление), логические и арифметические операторы, а также операторы сравнения, присваивание, групповые и условные выражения. Все переменные строго типизированы и глобальны. Каждой переменной, для которой определено начальное значение, поставлена в соответствие константа с суффиксом "_pre", хранящая это значение. Выделяют два типа предложений: выражения (*expression*), в основном используемые в качестве условий в модели, и действия (*action*), определяющие, как состояние системы изменяется при прохождении узлов действия. Только действия могут содержать присвоение переменным, а также условные и групповые выражения.

Ограничения на входные UCM-спецификации. На входные UCM-спецификации налагаются следующие ограничения. Поскольку компоненты используются преимущественно для моделирования структуры системы, в данной работе они не рассматриваются. Считается, что направление всех путей однозначно определено и все узлы модели уникально идентифицируемы. Действительно, модели, построенные в редакторах UCM, таких как jUCMNav [7], удовлетворяют этому требованию. Поддерживаются все структурные элементы UCM за исключением синхронизирующих и блокирующих узлов расширения, а также конструкций для обработки исключительных ситуаций. Атрибуты элементов, предназначенные для поддержки анализа производительности, такие как `probability` и `hostDemand`, не рассматриваются. Специальное значение пе-

ременных undefined также напрямую не поддерживается. Вместо него предлагается использовать значение по умолчанию для данного типа, например 0 или false.

Некоторые ограничения применимы только к узлам определенного типа. Для узлов действия полагаем, что значение атрибута repetitionCount всегда равно единице. Узлы ожидания и таймеры могут быть либо без памяти (*transient*), либо с памятью (*persistent*). Узлы без памяти игнорируют приходящие сигналы, если какой-либо сценарий не находится в ожидании на данном узле. Узлы с памятью регистрируют и накапливают все приходящие сигналы. В данной работе рассматриваются только узлы ожидания и таймеры с памятью. Дочерние диаграммы могут иметь атрибут singleton, означающий, что они существуют в единственном экземпляре. Алгоритм перевода поддерживает только дочерние диаграммы, которые не ограничены одним экземпляром, как наиболее простые и часто используемые.

Указанные ограничения позволяют поддерживать часто используемые элементы и сценарии их применения. Поэтому они не ограничивают существенно спектр поддерживаемых UCM-спецификаций. Для UCM-спецификаций с неподдерживаемыми элементами ограничения, как правило, можно устранить, модифицируя спецификацию так, чтобы использовались лишь поддерживаемые элементы. Алгоритмом игнорируется только информация из исходной модели, не относящаяся к верификации и анализу.

ПЕРЕВОД UCM-СПЕЦИФИКАЦИЙ В РАСКРАШЕННЫЕ СЕТИ ПЕТРИ

Поскольку нотация UCM является достаточно развитой, прямая верификация методом проверки моделей представляется затруднительной. Поэтому для анализа и верификации UCM-спецификаций переводим их в РСП [12]. Алгоритм перевода UCM-спецификаций в РСП детально приведен в [15].

Так как обе нотации описывают аннотированные иерархические ориентированные графы (где некоторым вершинам могут быть сопоставлены графы), входные и выходные данные представляются в виде иерархических ориентированных графов с дополнительной информацией, ассоциированной с вершинами и дугами. Алгоритм трансляции переводит одно представление в другое.

Общее описание алгоритма перевода. На самом верхнем уровне в алгоритме можно выделить пять этапов.

На первом этапе входная UCM-спецификация подвергается предварительной обработке. В частности, входная спецификация преобразуется в свободную (из нее удаляются все компоненты), определяются начальные значения для всех переменных, осуществляется проверка удовлетворения ограничениям алгоритма. 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 следующие служебные цвета: UNIT — стандартный «основной» цвет с единственным возможным значением (), фишки цвета UNIT в основном используются для моделирования передачи сигналов; SATISFIED — булев цвет, используемый для флагов; ACTIVATION — цвет, состоящий из неотрицательных целых чисел, используемый для хранения накапливаемых сигналов. Каждому типу данных, встречающемуся в UCM-спецификации, ставится в соответствие тип данных языка CPN ML. Булевым типам соответствует тип ‘colset BOOL = bool’, целочисленным — ‘colset INT = int’. Перечислению с именем enumName и элементами id0, id1, …, idn ставится в соответствие тип ‘colset enumName = with id0 | id1 | … | idn’. Каждой переменной

с суффиксом "_pre" ставится в соответствие константа CPN ML с таким же именем и значением. Также на данном этапе определяются служебные переменные, такие как 'var act : ACTIVATION'.

На третьем этапе граф UCM-спецификации, полученной в результате предварительной обработки на первом этапе, подвергается преобразованию, упрощающему его дальнейший перевод в двудольный граф и акцентирующему локальную природу алгоритма перевода отдельных элементов. На этом этапе большая часть дуг исходного графа, кроме дуг, инцидентных узлам соединения, разбивается вершинами, соответствующими узлам нового типа — вспомогательным узлам (*FakePathNode*). Эти вершины используются для определения непосредственной окрестности узлов модели и затем преобразуются в места цвета UNIT. Преобразование, осуществляемое на данном этапе, сохраняет аннотации на исходящих дугах.

На четвертом этапе каждый узел вместе со своей непосредственной окрестностью, кроме узлов соединения и вспомогательных узлов, обрабатывается по отдельности. Такая обработка может выполняться независимо от других узлов и параллельно с ними. Непосредственная окрестность узла определяется смежными с ним узлами соединения и вспомогательными узлами. Каждый рассматриваемый узел преобразуется во фрагмент модели РСП — аннотированный граф с дополнительными определениями на языке CPN ML.

На пятом этапе объединяются фрагменты РСП, полученные на четвертом этапе. Элементы модели с одинаковыми именами либо сливаются, либо представляются в виде объединяющих мест (*fusion place*), если это возможно. Места переменных с одинаковым именем в различных модулях РСП всегда становятся объединяющими местами. Заметим, что модель РСП содержит примитивы иерархической декомпозиции, только если исходная UCM-спецификация содержала узлы расширения.

Перевод узлов UCM-спецификации. Рассмотрим подробнее четвертый этап алгоритма перевода, заключающийся в переводе узлов UCM-спецификации с их непосредственной окрестностью во фрагменты РСП. Алгоритм перевода различается в зависимости от типа узла.

Перевод в РСП может породить элементы разных типов. Как правило, узел UCM преобразуется в один или несколько (последнее имеет место только для *Or*-объединений) переходов сети Петри. Узлы соединения, имеющиеся в непосредственной окрестности, преобразуются в места цвета ACTIVATION с начальной разметкой 1`0. Вспомогательные узлы преобразуются в места цвета UNIT с пустой начальной разметкой. В зависимости от типа узла, при переводе может быть добавлено дополнительное место. Так, начальные узлы должны иметь дополнительное начальное место с начальной разметкой 1`(). Для конечных узлов добавляется служебное место цвета SATISFIED, содержащее фишку со значением true, если постусловие конечного узла было удовлетворено, или false, если оно было не удовлетворено в результате выполнения сценария. Для *Or*-ветвлений добавляется служебное место с суффиксом "_OrForkWarnings". Если оно становится непустым, то это сигнализирует о некорректных условиях на исходящих дугах *Or*-ветвления. Имена всех переходов и мест являются производными от имен узлов UCM-спецификации, что значительно облегчает установку соответствия между исходной UCM-спецификацией и полученной моделью РСП после ее анализа.

Создается отдельное место для каждой переменной, встречающейся в действии или выражении, ассоциированном с данным узлом или его исходящими дугами. Место имеет цвет, соответствующий типу переменной, и начальную разметку,

соответствующую ее начальному значению. Имя переменной добавляется к набору определений переменных CPN ML. Места, соответствующие переменным, которым не присваивается нового значения в действиях и выражениях, связанных с данным узлом, соединяются с единственным переходом, соответствующим этому узлу, двунаправленными дугами с именем переменной в качестве надписи. Другие места переменных имеют аналогичные исходящие дуги, но надписи на входящих дугах являются результатом перевода действий для данной переменной в CPN ML. Трансляция выражений заключается в замене имен некоторых операторов и определении пары недостающих инфиксных операторов в CPN ML. Полученные выражения используются как части надписей для дуг или охранные условия в зависимости от типа элемента UCM. Действия транслируются в набор выражений `let-in-end` CPN ML для каждой переменной, которой присваивается значение в данном действии.

Узлы расширения преобразуются в подстановочные переходы вместе со смежными им гнездами (*socket*). Дочерние диаграммы преобразуются в модули модели РСП. Начальные и конечные узлы, являющиеся частью схемы связывания диаграмм, переводятся отлично от остальных узлов. Переход, соответствующий начальному узлу, вместо начального места имеет входное место (порт) цвета `UNIT` с пустой начальной разметкой аналогично пустому узлу. Фрагмент РСП, соответствующий конечному узлу, в дополнение к служебному месту цвета `SATISFIED` имеет исходящее место (порт) цвета `UNIT`. Соответствие гнезд и портов модели РСП определяется исходя из схемы связывания UCM диаграмм.

В зависимости от сложности схемы связывания может потребоваться промежуточный модуль. В частности, такой модуль вводится в случае наличия свободных начальных или конечных узлов дочерней диаграммы, т.е. не связанных с родительской диаграммой. Для узлов статического расширения промежуточный модуль состоит из подстановочного перехода, ассоциированного с модулем, соответствующим дочерней диаграмме, а также смежных с ним портов для связанных начальных и конечных узлов, входных гнезд цвета `UNIT` с начальной разметкой `1`()` для свободных начальных узлов и выходных гнезд цвета `UNIT` с пустой начальной разметкой для свободных конечных узлов. Порты промежуточного модуля ассоциируются с гнездами, смежными с подстановочным переходом, соответствующим узлу расширения.

Промежуточный модуль, направляющий входящие сигналы к подстановочным переходам и модулям, соответствующим активным дочерним диаграммам, вводится также при переводе узлов динамического расширения. В данном случае он используется для отделения логики выбора активной дочерней диаграммы.

ОЦЕНКА РАЗМЕРА РЕЗУЛЬТИРУЮЩИХ МОДЕЛЕЙ РСП

Рассмотрим вопрос повышения сложности модели при переводе из UCM в РСП. Сначала исследуем UCM-спецификацию без узлов расширения. Пусть v — произвольный из рассматриваемых узлов UCM-спецификации. Обозначим $d^+(v)$ полустепень захода для вершины v, т.е. число дуг, входящих в вершину v, а $d^-(v)$ — полустепень исхода для вершины v, т.е. число дуг, исходящих из вершины v. Обозначим множество узлов UCM-спецификации V, множество дуг — E, множество переменных — *Vars*. Существует связь между суммой полустепеней исхода и захода всех вершин и количеством дуг графа, известная как лемма о рукопожатиях: $\sum_{v \in V} d^+(v) + d^-(v) = 2|E|$. Здесь и далее $|A|$ обозначает мощность множества A или, в данном случае, количество элементов в нем, поскольку все рассматриваемые множества конечны.

При трансляции узла исходной UCM-спецификации во фрагмент РСП можно сделать следующие оценки. Сначала рассмотрим оценки без учета мест переменных и мест, соответствующих узлам соединения. Количество переходов ограничено сверху величиной $d^+(v)$ и достигается только для *Or*-объединений, для остальных узлов количество переходов равно единице. Количество мест ограничено $d^+(v)+d^-(v)+1$, другими словами, в места переводятся все смежные вспомогательные узлы, число которых не превышает числа инцидентных дуг, а также может появиться не более одного дополнительного места. Для *Or*-объединений количество дуг в соответствующем фрагменте РСП равно $2d^+(v)$. Для всех остальных узлов количество дуг ограничено $d^+(v)+d^-(v)+1$. Примем $2d^+(v)+d^-(v)+1$ как общую оценку сверху количества дуг без учета мест переменных и мест цвета ACTIVATION.

Каждый узел соединения в РСП представляется одним местом цвета ACTIVATION с ровно четырьмя инцидентными дугами. В UCM-спецификации такие элементы всегда имеют одну входящую и одну исходящую дугу. Поэтому оценки для отдельных элементов, приведенные выше, справедливы и для узлов соединения.

Просуммируем полученные оценки по всем узлам UCM-спецификации. Обозначим максимальную степень захода среди всех узлов UCM-спецификации $d^+_{\max}$. Тогда общее количество переходов полученной РСП ограничено величиной $d^+_{\max}|V|$. Теперь рассмотрим общее количество мест. Оно ограничено суммой оценок для всех узлов, которая по лемме о рукопожатиях равна $2|E|+|V|$. Суммы $d^+(v)+d^-(v)$ в оценках для узлов исходной UCM-спецификации представляют места, соответствующие вспомогательным узлам, каждое из которых встречается в двух фрагментах РСП. Следовательно, в результирующей сумме $2|E|$ каждое из них учтено дважды. Поэтому общее количество мест РСП без учета мест переменных ограничено величиной $|E|+|V|$. С учетом мест переменных эта величина равна $|E|+|V|+|Vars|$. Отметим, что в полученной оценке рассматриваем все объединяющие места, указывающие на одно место как единственное.

Оценка общего количества дуг РСП с учетом дуг, инцидентных местам переменных, приведена ниже. Каждый фрагмент РСП использует не более $|Vars|$ мест переменных, связывая каждое из них с единственным переходом двумя дугами. Суммируя оценку $2|Vars|$ дуг, инцидентных местам переменных, с суммой оценок дуг по всем узлам и применяя лемму о рукопожатиях, получаем общую оценку количества дуг: $2|E|+(d^+_{\max}+2|Vars|+1)|V|$.

Рассмотрим UCM-спецификацию с узлами расширения. Если при переводе в РСП не использовались промежуточные модули, то приведенные выше общие оценки сохраняют свою корректность. Действительно, оценки для каждого узла остаются справедливыми, в частности для самих узлов расширения, а также начальных и конечных узлов, связанных с родительской диаграммой. Трансляция остальных узлов не изменяется, а следовательно, не изменяются и оценки сложности для них.

Промежуточный модуль, введенный для моделирования узла статического расширения v, всегда включает один подстановочный переход, а также $d(v)$ дополнительных мест и дуг, где $d(v)$ — степень вершины v. Верхние оценки количества переходов, мест и дуг промежуточного модуля, введенного для моделирования узла динамического расширения v с $c(v)$ дочерними диаграммами, приве-

дены ниже. Количество подстановочных переходов равно $c(v)$, общее количество переходов не превышает $c(v)+d^+(v)+d^-(v)c(v) \le c(v)(1+d(v))$. Количество мест не превышает $d^+(v)+d^+(v)c(v)+d^-(v)c(v)+d^-(v) = d(v)(1+c(v))$. Количество дуг, инцидентных местам переменных, не превышает $d^+(v)|Vars|$. Количество остальных дуг не превышает $d^+(v)+2d^+(v)c(v)+3d^-(v)c(v) \le 3d(v)c(v)$. Заметим, что все полученные оценки для промежуточных модулей узла динамического расширения превышают соответствующие оценки для узла статического расширения.

Пусть d_s — максимальная степень среди узлов расширения, d_s^+ — максимальная полустепень захода 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 $d^+_{\max}|V| + n_s c_s(1+d_s) = O(d^+_{\max}|V| + n_s c_s d_s)$. Общее количество мест ограничено $|E|+|V|+|Vars|+n_s d_s(1+c_s) = O(|E|+|V|+|Vars|+n_s c_s d_s)$. Общее количество дуг ограничено $2|E|+|V|(d^+_{\max}+2|Vars|+1)+n_s(d_s^+|Vars|+3d_s c_s) = O(|E|+d^+_{\max}|V|+$ $+|Vars||V|+n_s d_s(|Vars|+c_s))$.

Таким образом, общее количество вершин РСП (мест и переходов, в том числе подстановочных) ограничено величиной $O(|E|+d^+_{\max}|V|+|Vars|+n_s c_s d_s)$.

ТРАНСЛЯЦИЯ РСП В ЯЗЫК PROMELA

Ограничения на входную модель РСП. Поддерживаются следующие типы данных (так называемые наборы цветов): целые числа, булевы, вырожденный тип данных `UNIT`, перечисления, кортежи и списки. Мультимножества и списки финитны, т.е. для каждого из соответствующих типов данных установлена емкость — максимальное число элементов, которые могут в них содержаться. Временные сети в текущей версии транслятора не поддерживаются.

Поддерживаемое транслятором подмножество языка выражений CPN ML включает переменные, целочисленные константы, элементы перечислений, логические константы, константу типа `UNIT`, константы пустого мультимножества и пустого списка, арифметические операторы и операторы сравнения для целых чисел, логические операторы, подмножество функций для работы со списками, конструкторы кортежей и операторы работы с ними, операторы конструирования и суммы мультимножеств, условный оператор. Рекурсивные функции транслятором не поддерживаются.

Сетевая модель должна позволять находить все допустимые значения переменных переходов по разметке входных мест и выражений входных дуг переходов без перебора большого количества значений переменных.

Заметим, что модели РСП, полученные из UCM-спецификаций приведенным выше алгоритмом, удовлетворяют всем ограничениям, описанным в этом разделе.

Алгоритм трансляции. Трансляция осуществляется поэтапно. По очереди транслируются типы данных, места РСП (в том числе их начальная разметка), а затем переходы РСП. Выходная Promela-модель содержит вставки кода на языке C, что позволяет исключить служебные переменные из вектора состояния модели.

Типы данных языка CPN ML преобразуются в типы данных языка C, примитивные типы данных — в целые числа. Кортежи переводятся в структуры с полями соответствующих типов. Списки переводятся в структуры с двумя полями: одно из них — массив, содержащий элементы списка, другое — целочисленная переменная, хранящая длину списка.

Места РСП переводятся в переменные, представляющие мультимножества. Способ представления мультимножества определяется в зависимости от класса типа его элементов. Для типов, имеющих небольшое множество значений (`UNIT`, булевы, перечисления), мультимножество представляется как структура со счетчиками вхождений каждого элемента из области значений типа. Для типов, которые имеют большое множество значений (целые, кортежи, списки), оно представляется как структура, содержащая массив, хранящий элементы мультимножества, и целочисленную переменную, хранящую общее количество элементов (или размер мультимножества). В последнем случае элементы мультимножества упорядочены, а неиспользованная часть массива содержит нулевые значения. Поэтому значение мультимножества, представленное в виде последовательности байтов, определяется только значениями его элементов и не зависит от порядка, в котором они были добавлены, удалены, или каких-либо иных факторов. РСП-модели часто содержат места, которые всегда имеют ровно одну фишку. Такие места переводятся в переменные, типом которых является не мультимножество, а соответствующий тип элемента. Выражения начальной разметки переводится в код на языке C, который выполняется один раз в начале выполнения модели, изменяя ее состояние с неинициализированного на начальное. Инициализация выполняется атомарно, т.е. не генерирует промежуточных состояний модели. Трансляция выражений CPN ML описана ниже.

Каждому из переходов исходной РСП в транслированной модели соответствует фрагмент кода на Promela, проверяющий, является ли данный переход допустимым в текущем состоянии модели. В нем происходит перебор возможных в данном состоянии значений переменных перехода. Если найдены значения, при которых переход может осуществиться, т.е. входные места содержат фишки, значения которых совпадают с вычисленными значениями входных дуг, охранное условие истинно, а выходные места могут вместить фишки, являющиеся значениями выражений выходных дуг, то номер перехода и найденные значения сохраняются в служебных переменных Promela-модели. Далее происходит недетерминированный выбор перехода, который будет реализован, после чего выполняется код, осуществляющий переход и изменяющий состояние модели. Он использует ранее сохраненные в служебных переменных значения для вычисления значений дуговых выражений и изменяет значения переменных-мультимножеств, соответствующие входным и выходным местам перехода, согласно семантике модели.

Такой процесс поиска допустимых переходов и их выполнения происходит до тех пор, пока модель не достигнет состояния, в котором нет допустимых переходов, или не придет в состояние, которое уже встречалось. При верификации модели рассматриваются все пути исполнения модели. Каждый переход выполняется атомарно и не генерирует промежуточных состояний. Каждое новое состояние является результатом осуществления некоторого перехода.

Чтобы транслировать охранные условия, дуговые выражения и выражения начальной разметки из CPN ML в язык C, транслятор производит обход дерева синтаксического разбора выражения CPN ML. Во время обхода он генерирует код на C, вычисляющий значения подвыражений и сохраняющий их во временные переменные. Сгенерированный код также проверяет допустимость значений аргументов, принимаемых операторами и функциями. Например, если в процессе вычисления выражения возникнет ситуация деления на ноль, система верификации сообщит о найденной в модели ошибке.

ПРИМЕР ВЕРИФИКАЦИИ КОММУНИКАЦИОННОГО ПРОТОКОЛА

Продемонстрируем работу представленных алгоритмов и инструментов на примере. Исходная UCM-спецификация описывает простой сетевой протокол, предназначенный для надежной передачи неотрицательного целого числа по ненадежной сети, способной передавать по одному разряду числа за один раз. Данные для передачи в этом примере являются целочисленными ввиду ограничений языка данных URN. UCM-спецификация транслируется в РСП, затем — в модель на языке Promela, которая исполняется в целях проверки желаемого свойства. Если оно не реализуется, то при исполнении модели генерируется контрпример. Контрпример является последовательностью элементов связывания (сработавшие переходы и связывания их переменных), которая либо ведет в конечное состояние, не удовлетворяющее верифицируемому свойству, либо не удовлетворяет данной формуле логики линейного времени (если свойство сформулировано в таком виде). Контрпример можно использовать для поиска ошибок в исходной спецификации. После исправления ошибок процесс повторяется до тех пор, пока верификация не пройдет успешно.

Спецификация сетевого протокола. Назовем начальную версию сетевого протокола протоколом A. На рис. 1 представлена диаграмма верхнего уровня UCM-спецификации протокола A. Все узлы UCM, кроме узлов ветвления и объединения, подписаны. Выражения на языке данных URN, а именно условия ветвления и постусловия, также приведены внутри квадратных скобок. Описание узлов действия (изображенных в виде крестиков) на языке данных URN и некоторые выражения здесь не показаны. UCM-спецификация содержит три компонента: Sender, Unreliable Network и Receiver, или отправитель, ненадежная среда и получатель соответственно. Отправитель разбивает значение `SendData` на разряды и посылает их в сеть, при необходимости отсылая заново. Ненадежная среда представлена двумя узлами статического расширения, моделирующими сетевые соединения для передачи пакетов от отправителя к получателю и наоборот. Оба узла расширения содержат одну дочернюю диаграмму Connection, моделирующую недетерминированную потерю пакета. Тем не менее общее число потерянных таким образом пакетов ограничено, для того чтобы избежать бесконечного исполнения модели. Получатель принимает пакеты, как только они прибывают, и собирает из них исходные данные. Получатель подтверждает отправителю каждый обработанный пакет посылкой порядкового номера следующего ожидаемого

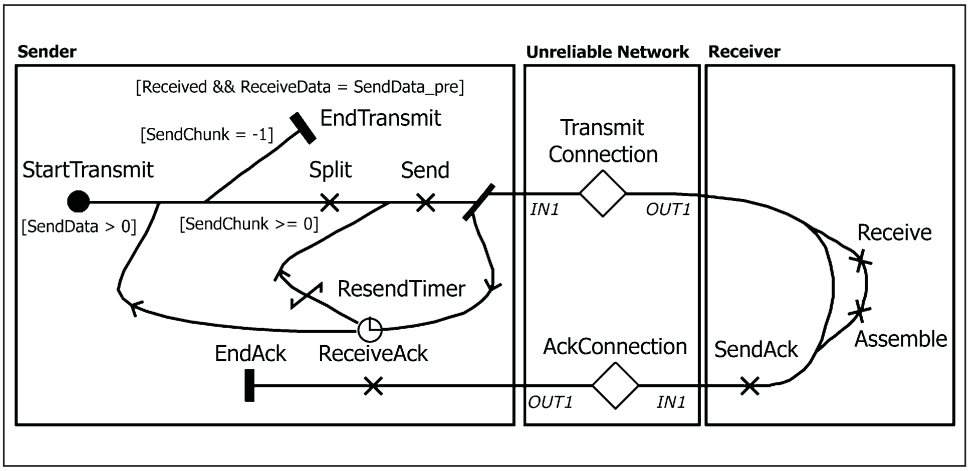

Рис. 1. Диаграмма верхнего уровня UCM-спецификации протокола A

пакета. Отправитель получает пакеты подтверждения асинхронно и обновляет порядковый номер следующего пакета, который он отошлет. Отправитель полагается на то, что порядковые номера пакетов могут только увеличиваться.

После посылки пакета отправитель ожидает получения подтверждения на элементе `ResendTimer`. Если текущий порядковый номер пакета совпадает с порядковым номером следующего пакета, то отсылается тот же самый пакет. Иначе отправитель извлекает очередной разряд для посылки и отправляет его в новом пакете вместе с его порядковым номером. Конец данных обозначает пакет, содержащий значение `−1`. Передача данных считается завершенной, когда отправитель получает подтверждение получения такого пакета. После завершения передачи данных осуществляется проверка постусловия [`Received && ReceiveData = SendData_pre`] конечного узла `EndTransmit`, где `SendData_pre` — начальное значение переменной `SendData`. Постусловие считается удовлетворенным, если данные были получены (соответствующий флаг принял значение `true`) и полученное значение равно значению, которое было отправлено.

Начальная UCM-спецификация подробно описана в [16], включая модель, построенную в редакторе jUCMNav.

Верификация и построение корректной UCM-спецификации. Для транслированной модели РСП формулируется и верифицируется следующее свойство: в конечном состоянии (таком, в котором нет активных переходов) постусловие `EndTransmit` удовлетворено и все места с суффиксом "_OrForkWarnings" пусты. Это свойство проверяет факт, что протокол всегда завершается в ожидаемом корректном состоянии и исходная UCM-спецификация консистентна в отношении условий ветвления. Промежуточные модели РСП и Promela вместе с результатами верификации и подробным описанием приведены в [16].

Для эффективной верификации UCM-спецификации пространство состояний должно быть ограничено. На уровне модели Promela без ограничения общности введено ограничение финитности, предписывающее не рассматривать исполнения модели, в которых одновременно более трех пакетов находятся в сети и в обработке получателем. Начальное значение переменной `SendData` также понижено до 120 для уменьшения пространства состояний и получения небольшого и простого для анализа контрпримера. Верификация заняла 583 с и использовала 2.7 Гб памяти. Эксперименты проводились на персональном компьютере с центральным процессором Intel Core i7-3520M, работающим на тактовой частоте 2900 МГц с 8 Гб памяти DDR3 RAM на частоте 1600 МГц под управлением 64-битовой операционной системы Ubuntu 12.04 Linux 3.5.0-28.

Контрпример сгенерирован в виде последовательности элементов связывания и проанализирован посредством трассировки в CPN Tools. Анализ показал, что значение внутренней переменной `NextReceive`, содержащей следующий ожидаемый порядковый номер пакета, проверяется и обновляется неатомарно. Это может привести к повторяющимся и отсутствующим разрядам в десятичной записи величины `ReceiveData`. Например, в построенном контрпримере в конечном состоянии значение `ReceiveData` равно 100 вместо 120. Контрпример показал, что получатель был некорректно синхронизован. После исправления UCM-спецификации процесс повторили, после чего обнаружилась аналогичная ошибка в отправителе.

Обе найденные ошибки исправлены введением собственных циклов обработки данных для отправителя и получателя. Таким образом, получатель обрабатывает только один входящий пакет за один раз, а отправитель не обрабатывает

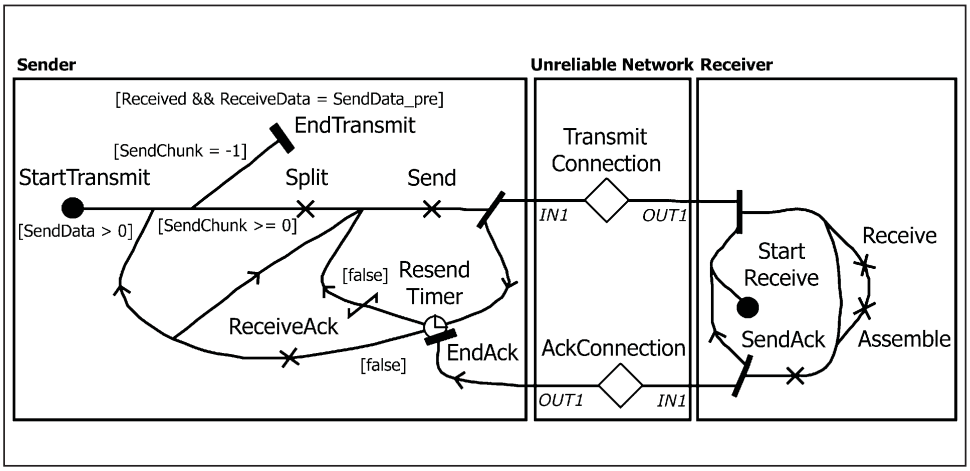

Рис. 2. Диаграмма верхнего уровня UCM-спецификации протокола B

пакеты подтверждения одновременно с отсылкой новых пакетов. Назовем исправленный сетевой протокол протоколом B. На рис. 2 изображена UCM-спецификация протокола B. Дочерняя диаграмма Connection не изменилась. Для проверки исправленной спецификации без ограничения общности использовано следующее ограничение: в каждый момент времени не более трех пакетов могут быть в сети. С начальным значением `SendData`, равным 123450, верификация заняла 388 с, использовала 4 Гб памяти и была успешно завершена. UCM-спецификация протокола B, промежуточные модели и результаты верификации приведены в [16].

ЗАКЛЮЧЕНИЕ

Графическая нотация Use Case Maps является выразительным средством описания функциональных требований к программным системам и протоколам. UCM изображает сценарии как набор причинно-следственных связей между действиями в системе, которые могут быть связаны со структурой компонентов системы. В данной работе приведены алгоритмы и инструменты для перевода UCM-спецификаций в раскрашенные сети Петри и модели на входном языке Promela известной системы верификации методом проверки моделей SPIN. Предложенные алгоритмы поддерживают большинство элементов UCM, описанных в стандарте, кроме сложных, редко используемых примитивов декомпозиции, некоторых редко используемых семантик примитивов ожидания и семантики компонентов. Работа алгоритмов продемонстрирована на примере.

Прототипная реализация инструмента для перевода поддерживает перевод файлов редактора jUCMNav [7] в файлы CPN Tools [17]. UCM-спецификации, транслированные в РСП, можно проанализировать либо с помощью встроенных средств CPN Tools, либо с использованием нового верификатора на основе SPIN, описанного в данной работе. Сгенерированные им контрпримеры могут легко трассироваться как на уровне CPN, так и на уровне UCM.

Планируется расширение поддерживаемого подмножества нотации UCM путем добавления элементов для моделирования возникновения и обработки исключительных ситуаций [6], а также опробирование построенного инструмента на других UCM-спецификациях.

СПИСОК ЛИТЕРАТУРЫ

1. Amyot D., Mussbacher G. User requirements notation: The first ten years, The next ten years (Invited Paper) // J. Software. — 2011. — **6**, N 5. — P. 747–768. — http://ojs.academypublisher.com/index.php/jsw/article/view/4659.
2. Amyot D., Weiss M., Logrippo L. Generation of test purposes from use case maps // Computer Networks. — 2009. — **49**, N 5. — P. 643–660.
3. 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 Anniv. of Radio Invention Conf.; IEEE Press, St. Petersburg, 2005. — P. 81–89.
4. Baranov S., Kotlyarov V., Weigert T. Verifiable coverage criteria for automated testing // Proc. SDL-Forum 2011; Lect. Notes Comput. Sci. — 2011. — **7083**. — P. 79–89.
5. Hassine J., Rilling J., Dssouli R. Use case maps as a property specification language // Software and Systems Modeling. — 2009. — **8**, N 2. — P. 205–220.
6. ITU-T, Recommendation Z.151 (10 /12), User Requirements Notation (URN) — Language definition. — http://www.itu.int/rec/T-REC-Z.151/en.
7. jUCMNav — Расширение интегрированной среды разработки Eclipse для поддержки User Requirements Notation. — http://jucmnav.softwareengineering.ca /ucm /bin /view /ProjetSEG/WebHome.
8. Hassine J., Rilling J., Dssouli R. Abstract operational semantics for use case maps // Proc. FORTE 2005; Lect. Notes Comput. Sci. — 2005. — **3731**. — P. 366–380.
9. Hassine J., Rilling J., Dssouli R. Formal verification of use case maps with real time etensions // Proc. SDL-Forum 2007; Lect. Notes Comput. Sci. — 2007. — **4745**. — P. 225–241.
10. Средства поддержки интегрированной технологии для анализа и верификации спецификаций телекоммуникационных приложений / И.С. Ануреев, С.Н. Баранов, Д.М. Белоглазов, Е.В. Бодин, П.Д. Дробинцев, А.В. Колчин, В.П. Котляров, А.А. Летичевский, А.А. Летичевский мл., В.А. Непомнящий, И.В. Никифоров, С.В. Потиенко, Л.В. Прийма, Б.В. Тютин // Тр. СПИИРАН. — СПб, 2013. — **26**, N 1. — С. 349–383.
11. Baranov S., Kapitonova J., Letichevsky A., Volkov V., Weigert T. Basic protocols, message sequence charts, and verification of requirements specifications // Computer Networks. — 2005. — **49**, N 5. — P. 661–675.
12. Jensen K., Kristensen L.M. Coloured petri nets: modelling and validation of concurrent systems. — Berlin: Springer, 2009. — 384 p.
13. Holzmann G.J. The SPIN model checker. Primer and reference manual. — Boston: Addison-Wesley, 2004. — 596 p.
14. Vizovitin N.V., Nepomniaschy V.A., Stenenko A.A. Verification of UCM-Specifications of distributed systems using coloured Petri nets // Proc. IV Intern. Workshop PSSV 2013. — Yaroslavl, 2013. — P. 70–80.
15. Визовитин Н.В., Непомнящий В.А. Алгоритмы трансляции UCM-спецификаций в раскрашенные сети Петри. — Новосибирск, 2012. — 55 с. — (Препр. / СО РАН. Ин-т систем информатики; 168). — http://www.iis.nsk.su/files/preprints/168.pdf.
16. Vizovitin N.V. Verification of UCM-specifications of distributed systems using coloured Petri nets. Appendix. — http://bitbucket.org/vizovitin/ucm-verification-examples.
17. Средство разработки и анализа раскрашенных сетей Петри CPN Tools. — http://cpntools.org/.

Поступила 24.02.2014