УДК 681.32

Д. М. Белоглазов [1], **В. А. Непомнящий** [2]

Институт систем информатики им. А. П. Ершова СО РАН
пр. Академика Лаврентьева, 6, Новосибирск, 630090, Россия
E-mail: [1] dmitry.beloglazov@gmail.com, [2] vnep@iis.nsk.su

МОДЕЛИРОВАНИЕ И ВЕРИФИКАЦИЯ ВЗАИМОДЕЙСТВИЯ ФУНКЦИОНАЛЬНОСТЕЙ В ТЕЛЕФОННЫХ СЕТЯХ ПРИ ПОМОЩИ КОНЕЧНЫХ АВТОМАТОВ И РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ

Для моделирования, анализа и верификации телекоммуникационных систем обычно применяются такие модели, как конечные автоматы, сети Петри и их обобщения. Цель данной работы – представить новый двухуровневый метод моделирования и верификации телекоммуникационных систем. На первом этапе этого метода телекоммуникационные системы моделируются в виде расширенных конечных автоматов, а на втором этапе автоматные модели транслируются в раскрашенные сети Петри. Данный метод применяется к исследованию проблемы взаимодействия функциональностей в телефонных сетях. В качестве примера рассматривается базовая модель звонков (Basic Call State Model) с дополнительными функциональностями. Для построения графов достижимости раскрашенных сетей Петри используется система CPN Tools, а для верификации методом проверки моделей используется система Petri Net Verifier. Описанные эксперименты позволили выявить некоторые нежелательные взаимодействия функциональностей в телефонных сетях.

Ключевые слова: телекоммуникационные системы, конечные автоматы, раскрашенные сети Петри, взаимодействие функциональностей, телефонные сети, верификация, метод проверки моделей.

Введение

Для моделирования, анализа и верификации телекоммуникационных систем обычно применяются такие модели, как конечные автоматы, сети Петри и их обобщения. Среди этих моделей выделяются раскрашенные сети Петри (РСП) [Jensen, 1996] ввиду их значительной выразительности, а также разработанных мощных средств симуляции и анализа [Kristensen et al., 1998, Ratzer et al., 2003]. Естественно применить эти модели для исследования проблемы взаимодействия функциональностей (feature interaction problem, FIP) [Keck, Kuehn, 1998], которая является одной из наиболее интересных, практически значимых и сложных проблем в области информационных технологий.

Проблема взаимодействия функциональностей состоит в том, что в комплексных системах, включающих в себя несколько модулей со смежной функциональностью, модули могут взаимодействовать друг с другом, вызывая тем самым отклонения от ожидаемого поведения системы в целом. Такие отклонения могут быть безвредными, однако в большинстве случаев они оказываются нежелательными. Классический пример такой системы – телефонные сети с дополнительными функциональностями (сервисами). Моделирование таких телефонных сетей рассматривалось в интересных работах [Calder, Miller, 2001; Capellmann et al., 1999; Cavalli, Maag, 2002; Nakamura, 1999], причем в качестве моделей применялись как расширенные конечные автоматы [Cavalli, Maag, 2002], так и раскрашенные сети Петри [Capellmann et al., 1999; Nakamura, 1999]. В работе [Calder, Miller, 2001] описана верификация взаимодействия функциональностей в телефонных сетях с помощью известной системы SPIN, которая базируется на методе проверки моделей (model checking method) [Кларк и др., 2002]. В работе [Capellmann et al., 1999] описан анализ взаимодействия функциональностей в телефонных сетях, использующий систему Design/CPN. Эта же система применяется для моделирования сети мобильных телефонов фирмы «Nokia» с целью обнаружения нежелательного взаимодействия функциональностей [Lorentsen et al., 2001]. В работе [Nakamura, 1999] представлены алгоритмы обнаружения взаимодействия функциональностей в телефонных сетях, которые используют в качестве моделей подмножество раскрашенных сетей Петри.

Цель данной работы – представить новый двухуровневый метод моделирования и верификации взаимодействия функциональностей в телефонных сетях. Этот метод на первом

этапе использует автоматное представление телефонных сетей, а на втором этапе переводит это представление в РСП, что позволяет провести моделирование и верификацию с помощью программных систем.

Взаимодействующие расширенные конечные автоматы

Расширенным конечным автоматом (РКА) назовем кортеж $A = <S, V, I, T>$,
где S – множество имен всех возможных *состояний* автомата (задается явным перечислением);

V – множество *локальных переменных*, используемых автоматом в работе (задается явным перечислением с указанием типа данных);

I – *очередь входящих сигналов* – выделенная локальная переменная автомата, в которую система помещает сигналы, адресованные данному автомату; благодаря этой переменной осуществляется взаимодействие между автоматами; единичный сигнал имеет вид [Src, Dest, Type, Param], где Src – идентификатор отправителя сигнала, Dest – идентификатор адресата, Type – тип сигнала (перечислимый тип), Param – параметр сигнала (необязательный);

T – *множество переходов* РКА; каждый переход t из T имеет вид $t = \{Ss, Se, O, G(V, I), E(V, I)\}$, где

Ss – начальное состояние (из S);

Se – конечное состояние (из S);

O – набор исходящих сигналов;

$G(V, I)$ – охранная булева функция от локальных переменных автомата и первого входящего сигнала в очереди I;

$E(V, I)$ – вычисление на локальных переменных автомата.

Переход *разрешен*, если $G(V, I)$ истинно, и автомат находится в состоянии Ss. *Срабатывание* разрешенного перехода – это переход автомата из состояния Ss в Se, удаление первого входящего сигнала из очереди, изменение значений локальных переменных согласно $E(V, I)$ и отправка исходящих сигналов O. *Срабатыванием автомата* назовем срабатывание любого разрешенного перехода (недетерменированный выбор).

Для обеспечения взаимодействия расширенных конечных автоматов посредством сигналов необходимо снабдить каждый автомат уникальным идентификатором. Назовем *экземпляром автомата A* в системе автомат, построенный по описанию A и снабженный уникальным идентификатором в системе. При этом будем говорить, что такой экземпляр имеет *тип* A. Идентификатор экземпляра фактически является его локальной константой. Для определенности зададим этой константе имя *id*.

Система взаимодействующих РКА – это кортеж:

$\Sigma = <G, A, I>$, где

G – множество глобальных переменных системы, доступных всем автоматам;

A – множество экземпляров РКА;

I – инициализирующая функция, определенная на G и A.

Инициализирующая функция производит:

- присвоение начальных значений глобальным переменным;
- присвоение уникальных идентификаторов (id) всем экземплярам РКА системы;
- присвоение начальных состояний всем экземплярам РКА;
- присвоение начальных значений локальным переменным каждого экземпляра РКА;
- присвоение начального значения очереди входящих сигналов каждого экземпляра РКА.

Для выполнения системы ее необходимо снабдить дискретным временем, т. е. набором *тактов* – целых неотрицательных чисел.

Конфигурация экземпляра автомата на такте t – это тройка $<S(t), V(t), I(t)>$, т. е. состояние экземпляра, значения его локальных переменных и очереди входящих сигналов на такте t.

Конфигурацией системы на такте t называется совокупность конфигураций всех экземпляров автоматов.

Срабатывание системы на такте t – это срабатывание некоторого экземпляра автомата системы на данном такте (недетерминированный выбор), причем все исходящие сигналы, образовавшиеся на данном такте, должны быть записаны в очереди входящих сигналов их адресатов.

Рассмотрим простой пример системы взаимодействующих расширенных конечных автоматов – систему «Отправитель-Получатель». Эта система состоит из двух взаимодействующих РКА – Отправителя и Получателя. Отправитель посылает сообщение Получателю и ожидает подтверждения о получении. Получатель принимает сообщение и возвращает подтверждение Отправителю.

Вначале Отправитель находится в состоянии IDLE. Алгоритм его таков:

1) если состояние IDLE, отправить Получателю сигнал типа *msg* и перейти в состояние AWAITING_ACK (от «awaiting acknowledgement» – ожидание подтверждения);

2) если состояние AWAITING_ACK и получен входящий сигнал типа *ack* (подтверждение), вернуться в исходное состояние IDLE.

Алгоритм Получателя еще проще и состоит из одного правила: если получено сообщение, вернуть Отправителю этого сообщения сигнал типа *ack*.

Таким образом, по определению РКА, для Отправителя имеем $A = <S, V, I, T>$, где

S = {IDLE, AWAITING_ACK}, V = {Receiver_id}, $T = \{t_1, t_2\}$, причем

t_1 = {IDLE, AWAITING_ACK, [id, Receiver_id, msg, null], -, -},

t_2 = {AWAITING_ACK, IDLE, -, I.type = ack, -}.

Здесь и в дальнейшем в переходах I обозначает первый сигнал в очереди.

Автомат Получателя имеет вид

S = {IDLE}, V = {}, $T = \{t_1\}$, причем

t_1 = {IDLE, IDLE, [id, I.src, ack, null], I.type = msg, -}.

Трансляция расширенных конечных автоматов в сети Петри высокого уровня

Раскрашенные сети Петри. Структура сети представляет собой направленный двудольный граф с двумя типами вершин – *местами* и *переходами*, соединенными дугами таким образом, что каждая дуга соединяет вершины различных типов. Места имеют *разметку*. Ненулевая разметка места определяется помещением в место одной или нескольких *фишек*. Места и находящиеся в них фишки представляют состояние системы, моделируемой сетью Петри, тогда как переходы – изменение состояния системы. Места, из которых в переход ведут дуги, называются *входными* местами для данного перехода. Места, в которые ведут дуги от данного перехода, называются его *выходными* местами.

Декларации состоят из описания множеств цветов и объявления переменных, каждая из которых принимает значения из некоторого множества цветов.

Пометка сети приписывается месту, переходу либо дуге. Каждое место имеет три разных типа пометок: имя места, множество цветов и инициализирующее выражения. Имя не имеет формального значения и служит для идентификации. Множество цветов определяет тип фишек, которые могут находиться в месте, т. е. любая фишка, находящаяся в месте, должна иметь цвет, который является элементом данного множества цветов. Инициализирующее выражение определяет мультимножество над соответствующим множеством цветов, а также то, какие фишки находятся в месте в начальный момент. Фишки, находящиеся в месте в начальный момент, называются *начальной разметкой* данного места.

Переходы имеют два типа пометок: имена и охранные функции. Охранная функция перехода является логическим выражением, которое должно быть выполнено до того, как переход сможет сработать.

Дуги имеют один тип пометок: выражения. Выражения на дугах могут содержать переменные, константы, функции и операции, определенные в декларациях. Когда все переменные, входящие в выражение на дуге, связаны, т. е. получили значения из соответствующих множеств цветов, выражение должно определять цвет (или мультимножество цветов) из множества цветов, приписанного месту, с которым связана дуга. Все вхождения одной и той же переменной должны замещаться одним и тем же цветом.

Переходы являются активной компонентой раскрашенных сетей. Переходы срабатывают, изменяя при этом разметку своих входных и выходных мест. Чтобы переход мог сработать, все переменные, входящие в охранную функцию перехода, и выражения на связанных с ним дугах должны получить значения. Выбор для каждой переменной конкретного цвета, при котором охранная функция перехода имеет истинное значение, называется *связыванием*.

При выбранном связывании выражения на входных дугах перехода определяют, сколько и каких фишек должно содержаться во входных местах перехода, чтобы переход мог сработать при выбранных значениях переменных. Выражения на выходных дугах определяют, сколько и каких фишек будет помещено в выходные места перехода, когда он сработает. Если входные места перехода содержат необходимый набор фишек, переход *возможен* при выбранном связывании. Возможный переход может сработать. *Срабатывание* перехода изымает фишки из входных мест и добавляет в выходные места. Количество и цвет изымаемых / добавляемых фишек определяются выражениями на соответствующих дугах.

Алгоритм трансляции РКА в РСП. Ниже приведены основные принципы трансляции модели взаимодействующих расширенных конечных автоматов в РСП [Jensen, 1996]. Общая идея такова: для состояний экземпляров каждого автомата задаются места целого типа (цвета), в которых каждая фишка 1'*x* моделирует экземпляр этого автомата с идентификатором *x*. Переход экземпляра автомата из одного состояния в другое становится переходом соответствующей фишки из одного места в другое.

Цвета. Для построения сети Петри, моделирующей систему взаимодействующих РКА, нам понадобятся следующие цвета (типы данных):

- sType = typedef {…} – для обозначения типов сигналов;
- cInt = Integer – для идентификаторов экземпляров автоматов;
- cBool = typedef {true | false} – для моделирования булевых переменных;
- Signal = Record [cInt Src, cInt Dest, sType Type, cInt Param] – для моделирования сигналов;
- sQueue = queue Signal – для моделирования очереди сигналов.

Также для каждой переменной в системе нужен соответствующий цвет, а для массива – запись типа [cInt ind, <тип массива>], где ind обозначает индекс массива.

Переходы. Переходы РСП создаются последовательно из переходов из описания автоматов – строчек таблиц. Для определенности мы придерживаемся следующего правила именования переходов: имя перехода РСП = <имя автомата>_<номер перехода>.

Места, обозначающие состояния автоматов, имеют цвет cInt. Имена таких мест соответствуют именам состояний. Наличие фишки 1'*x* в месте с именем STATE означает, что экземпляр с идентификатором *x* находится в состоянии STATE. Такой подход обеспечивает удобную симуляцию и последующую верификацию полученной РСП.

Дополнительное место SIGNALS цвета sQueue создается для обмена сигналами.

Для упрощения трансляции объявляем все переменные глобальными. В случае, рассматриваемом нами, это не играет роли, так как конфликтов при использовании переменных возникнуть не может. В общем же случае нетрудно построить типы данных и места для локальных переменных.

Для данного перехода строим входящие и исходящие места с указанными в *Ss* и *Se* именами (если такие еще не построены) и соответствующие дуги. Из входящего места берется id (типа cInt). Так же входящие и исходящие места и дуги строятся для всех используемых в описании перехода локальных и глобальных переменных.

Если в охранной функции перехода автомата используется значение массива $M(x)$, то необходимо добавить дугу из соответствующего места M (если такой дуги еще нет), а также обратную дугу в это место, возвращающую фишку (если это значение не изменяется в E). Согласно цвету массива, эта фишка будет иметь вид $[i, v]$, где i – индекс, v – соответствующее значение. Необходимо сделать в переходе проверку $i = x$ и вместо $M(x)$ в охранной функции РСП подставить v.

Входящий сигнал I. Если в охранной функции G есть проверка входящего сигнала I, добавляем:

- входящую дугу с выражением типа Signal из места SIGNALS;
- в охранную функцию РСП добавляем проверку условия
  I.Dest = id & I.Type = <тип сигнала, заданный в $G(V, I)$>

Исходящие сигналы O. Для каждого из исходящих сигналов O добавляем исходящую дугу в место SIGNALS. При этом выражение на дуге типа Signal содержит идентификаторы отправителя и получателя, а также тип сигнала O и параметр (если он задан).

Охранное условие G. Если для перехода задана охранная функция $G(V, I)$, то добавляем в охранную функцию на переходе РСП проверку условий из G.

Сравнение массивов транслируются следующим образом. Допустим, нас интересует значение выражения $M(x) = y$. Правило 3 описывает построение дуги из соответствующего места M (и обратно). Выражение на этой дуге будет иметь вид $[i, v]$, где i – индекс, v – соответствующее значение. Таким образом, истинность выражения $M(x) = y$ совпадает с истинностью выражения $i = x \,\&\, v = y$.

Вычисление E. Если для перехода необходимо вычисление $E(V, I)$, то посредством выражений на дугах в исходящие места отправляем значения, вычисленные в E.

Вычисление на массивах происходит аналогично тому, как это было описано в предыдущих правилах. Добавляется проверка совпадения индекса массива с аргументом, указанным в E, затем в исходящее место направляется фишка с этим индексом и заданным значением (путем задания выражения на соответствующей дуге).

Инициализация. Согласно определению инициализации системы, мы должны задать значения всех переменных в системе, положив необходимые фишки в соответствующие места. Также необходимо поместить фишки с идентификаторами экземпляров автоматов в места, соответствующие их исходным состояниям.

Итак, схематично *алгоритм трансляции* можно представить следующим образом.

Шаг 1. Согласно правилу трансляции типов данных (правило 1), в декларациях РСП задаем необходимые цвета для всех переменных в системе, а также все необходимые вспомогательные переменные.

Шаг 2. Строим вспомогательное место SIGNALS, а также места для переменных согласно правилу 3.

Шаг 3. Для каждого описания автомата и каждого его перехода проводим следующую циклическую процедуру:

- строим переход с уникальным именем (правило 2);
- строим места для *Se* и *Ss* и соответствующие входящие и исходящие дуги (правило 3);
- для всех переменных, используемых в переходе (в E, O и G), строим соответствующие входящие и исходящие дуги (правило 3), а также, при необходимости, охранную функцию на переход; в выражения на исходящие дуги для переменных из E, которым присваиваются новые значения, добавляем соответствующие вычисления (правило 7);
- для входящего и исходящих сигналов I и O добавляем соответствующие дуги и охранные условия (правила 4 и 5);
- добавляем на переход РСП условие из охранной функции перехода автомата (правило 6).

Шаг 4. Задаем условия инициализации (начальную разметку РСП). Для этого ставим фишки в места, соответствующие глобальным переменным и идентификаторам id экземпляров автоматов, согласно их значениям из описания системы РКА.

На каждом шаге перед построением очередного объекта модели (например, дуги) необходимо добавить проверку его существования, и если объект уже существует, не делать повторного построения.

В результате такой трансляции, полученная РСП будет моделировать все возможные конфигурации системы РКА, так как РСП содержит информацию о состояниях всех экземпляров РКА и значения всех переменных системы. Кроме того, переходы РСП строятся из переходов РКА с сохранением информации об изменении состояния экземпляра РКА и о вычислениях над переменными, а также с сохранением логических условий срабатывания перехода. Это означает, что переходу системы РКА из конфигурации A в конфигурацию B будет взаимооднозначно соответствовать переход из разметки $A_{РСП}$ (эквивалент конфигурации A) в $B_{РСП}$ (эквивалент B). А так как условия инициализации определяют исходную разметку РСП эквивалентной исходной конфигурации системы РКА, то получаем, что построенная РСП эквивалентна исходной системе РКА. Это лишь схематичное доказательство корректности трансляции. Для полного формального доказательства необходим детальный разбор каждого правила и шага трансляции.

Пример. Для системы «Отправитель-Получатель» построим раскрашенную сеть Петри при помощи приведенного алгоритма трансляции.

Шаг 1. В нашей системе есть всего одна переменная – Receiver_id целочисленного типа, поэтому дополнительных цветов не требуется. Тип sType будет иметь вид sType = typedef {msg, ack}.

Шаг 2. Строим вспомогательное место SIGNALS и место Receiver_id.

Шаг 3. Для переходов Отправителя (Sender) строим следующие элементы.

- Для перехода t_1 = {IDLE, AWAITING_ACK, [id, receiver_id, msg, null], -, -}:

а) переход Sender_1;

б) места Sender_IDLE и Sender_AWAITING_ACK, а также дуги Sender_IDLE→Sender_1 и Sender_1→Sender_AWAITING_ACK с выражением id;

в) входящую и исходящую дуги в место receiver_id (так как мы используем эту переменную) с соответствующими выражениями;

г) исходящую дугу в место SIGNALS с выражением 1’[id, receiver_id, msg, null].

- Для перехода t_2 = {AWAITING_ACK, IDLE, -, I.Type = ack, -} строим следующие элементы:

д) переход Sender_2;

е) дуги Sender_AWAITING_ACK→ Sender_2 и Sender_2→Sender_IDLE с выражением id;

ж) дугу из места SIGNALS с выражением 1’[Src, id, Type, Value] и охранным условием I.Type = ack (либо просто выражением 1’[Src, id, ack, Value]).

Аналогично строится переход Получателя.

Шаг 4. Инициализация проводится следующим образом. Присвоим Отправителю идентификатор 1, а Получателю – 2. Таким образом, начальная разметка будет такова:

Sender_IDLE = {1’1}, Sender_AWAITING_ACK = {}, Receiver_IDLE = {1’2},
Receiver_id = {1’2}, SIGNALS = {}.

Ниже приведено графическое представление результирующей сети (рис. 1).

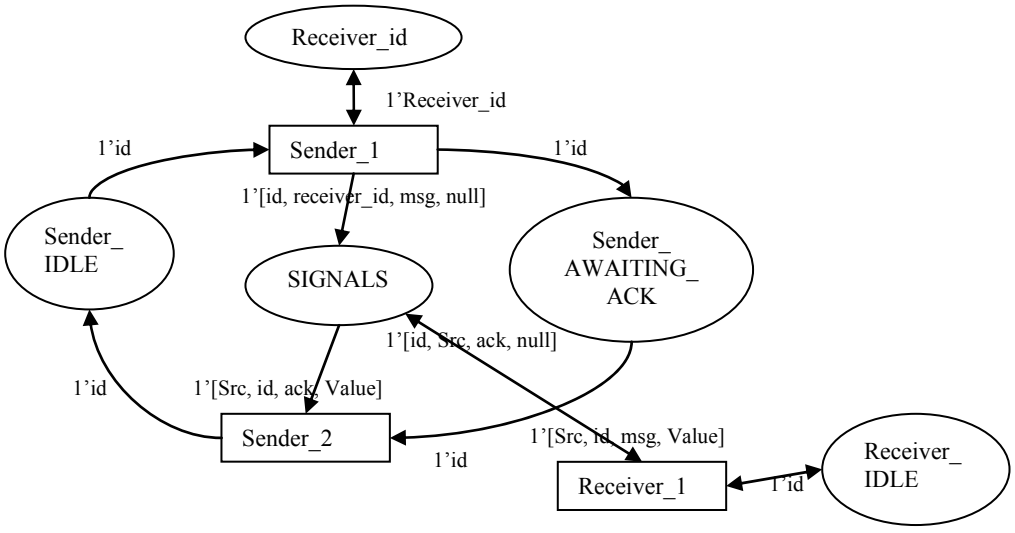


Рис. 1. Схема результирующей РСП

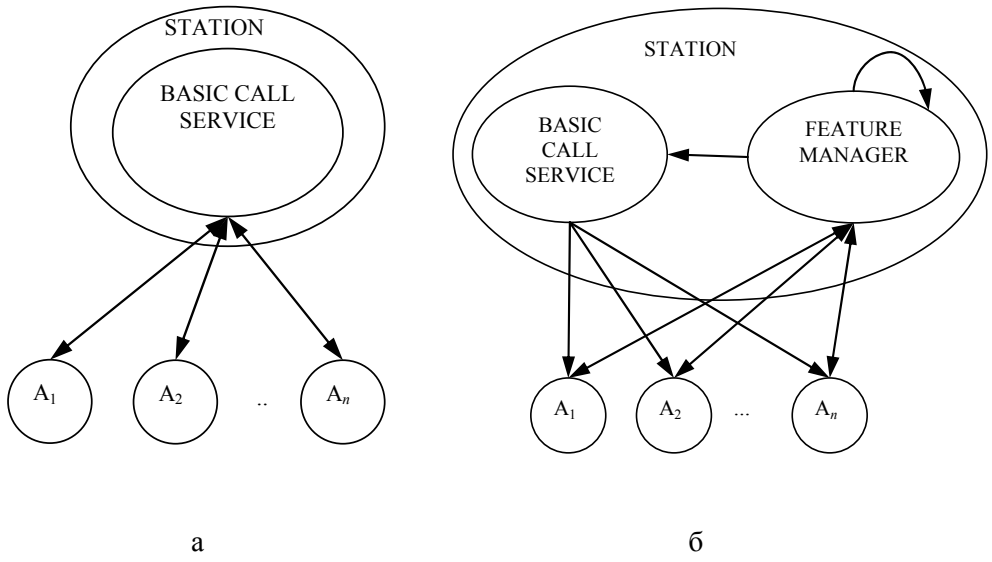
Моделирование телефонных сетей посредством расширенных конечных автоматов

Базовая модель звонков. Базовое взаимодействие абонентов со станцией описывается так называемой «базовой моделью звонков» (Basic Call State Model, BCSM) [Nakamura, 1999]. Это модель простейшего телефонного сервиса (Basic Call Service, BCS), который позволяет абонентам общаться – набирать номер, отвечать на звонки и т. д.

В нашей работе для моделирования мы используем взаимодействующие посредством сигналов расширенные конечные автоматы, выделяя абонентов и станции в отдельные сущ-

ности, т. е. отделяя логику абонента и логику станции в отдельные автоматы. Абоненты посылают запросы станции, которая обрабатывает запросы и выдает абонентам результаты в виде определенных сигналов. Модель BCSM схематично представлена на рис. 2, *а*.

Функциональности моделируются отдельным элементом Feature Manager, который становится посредником между абонентами и BCS. Этот элемент принимает и обрабатывает сигналы абонентов с учетом того, подключена ли определенная функциональность у вызывающего (или вызываемого) абонента. Feature Manager либо самостоятельно посылает сигналы абонентам, либо передает управление Basic Call Service. Схематично общая модель представлена на рис. 2, *б*.



а б

Рис. 2. Базовая модель звонков: *а* – без функциональностей, *б* – с функциональностями

Модель абонента. Для простоты изложения мы приводим модель абонента в виде таблицы, где каждая строка обозначает переход, ячейки содержат значения в соответствии с обозначениями из определения перехода. Вычисления у абонентов отсутствуют. Пустое значение параметра сигнала обозначаем как *null*.

Для взаимодействия со станцией абонент должен знать ее идентификатор, и так как, согласно нашей схеме, абонент отправляет сигналы модулю Feature Manager, зададим переменную для абонентов: Dim fm_id as Integer (табл. 1).

Дадим некоторые пояснения по состояниям модели абонента:

- IDLE – абонент не совершает никаких действий, трубка не снята;
- OFFHOOK – абонент снял трубку;
- DIALTONE – абонент слышит длинный непрерывный гудок;
- BUSYTONE – абонент слышит короткие гудки (сигнал «занято»);
- DIALLING – абонент набирает номер другого абонента;
- CALLING – происходит соединение данного абонента с вызываемым;
- TALKING – абонент соединен и разговаривает с другим абонентом;
- PHONE_RINGS – телефон абонента звонит;
- READY_TO_TALK – абонент, телефон которого звонил, принял звонок, сняв трубку.

Исходящие сигналы обозначают: *offhook* – «снял трубку», *onhook* – «повесил трубку», *dial* – «набрал номер».

Например, переход 1 обозначает, что абонент снял трубку, отправив при этом модулю Feature Manager сигнал типа offhook.

Переход 5 отправляет сигнал типа Dial с параметром Rnd *X*, это означает, что *X* – случайный идентификатор другого абонента в сети. Возможность генерации такого идентификатора должна предоставлять система, ответственная за организацию взаимодействия автоматов.

Для реализации конструкции Rnd *X* при трансляции из РКА в РСП мы построили специальное место ABONENTS_POOL цвета cInt. При трансляции мы пользовались следующим дополнительным правилом: если в качестве параметра сигнала указана переменная *X* типа Rnd, то соответствующее значение берется из места ABONENTS_POOL, при этом необходимо добавить условие *X* != id. Инициализируется место ABONENTS_POOL полным набором номеров абонентов.

Таблица 1

Модель абонента

#	*Ss*	*Se*	*G*	*O*
1.	IDLE	OFFHOOK	–	[id, fm_id, off-hook, null]
2.	IDLE	PHONE_RINGS	I.Type = incom-ing_call	–
3.	OFFHOOK	DIALTONE	I.Type = dialtone	–
4.	DIALTONE	IDLE	–	[id, fm_id, on-hook, null]
5.	DIALTONE	DIALLING	–	[id, fm_id, dial, Rnd *X*]
6.	DIALLING	BUSYTONE	I.Type = busytone	–
7.	DIALLING	CALLING	I.Type = callingtone	–
8.	BUSYTONE	IDLE	–	[id, fm_id, on-hook, null]
9.	CALLING	IDLE	–	[id, fm_id, on-hook, null]
10.	CALLING	TALKING	I.Type = call_accepted	–
11.	TALKING	BUSYTONE	I.Type = call_dropped	–
12.	TALKING	IDLE	–	[id, fm_id, on-hook, null]
13.	PHONE_RINGS	READY_TO_TALK	–	[id, fm_id, off-hook, null]
14.	PHONE_RINGS	IDLE	I.Type = call_dropped	–
15.	READY_TO_TALK	TALKING	I.Type = connect	–
16.	OFFHOOK	PHONE_RINGS	I.Type = incom-ing_call	–

Модель Basic Call Service использует три массива:

Dim busy as Array of Boolean;

Dim caller as Array of Integer;

Dim callee as Array of Integer.

Эти переменные используются совместно с Feature Manager, поэтому они должны быть объявлены глобальными.

Массив busy хранит информацию о состоянии абонента (занят / свободен) с точки зрения станции. Занятость абонента с идентификатором id определяется истинностью busy(id). Далее, caller(*X*) = *Y* обозначает «абоненту *Y* звонит абонент *X*». Выражение callee(*Y*) = *X* фактически обозначает то же самое. Массив callee является «обратным» по отношению к массиву caller и вводится для удобства.

Basic Call Service всегда имеет одно и то же состояние, поэтому мы опускаем *Ss* и *Se* (табл. 2).

Поясним типы исходящих сигналов:

- *dialtone* – непрерывный гудок;
- *call_dropped* – оповещение абоненту о том, что его оппонент положил трубку;
- *busytone* – короткие гудки (сигнал «занято»);
- *callingtone* – длинные гудки;

- *incoming_call* – входящий звонок;
- *connect* – вспомогательный сигнал для установления соединения между абонентами;
- *call_accepted* – оповещение абоненту о том, что его оппонент (чей номер он набрал) принял звонок, сняв трубку.

Например, переход 1 обозначает следующее. Если мы получили от абонента *X* (*X* = I.Src) сигнал типа offhook (что означает, что абонент снял трубку), и при этом абоненту *X* никто не звонит, отправляем ему сигнал типа dialtone (непрерывный гудок), записав информацию о том, что абонент *X* занят.

Таблица 2

Модель Basic Call Service

#	*G*	*O*	*E*
1.	I.Type=offhook & caller(I.Src)=null	[id, I.Src, dialtone, null]	busy(I.Src)=true
2.	I.Type=onhook & calee(I.Src)=null & caller(I.Src)=null	–	busy(I.Src)=false
3.	I.Type=onhook & caller(I.Src)!=null	[id, caller(I.Src), call_dropped, null]	busy(I.Src)=false callee(caller(I.Src))=null caller(I.Src)=null
4.	I.Type=onhook & calee(I.Src)!=null	[id, callee(I.Src), call_dropped, null]	busy(I.Src)=false caller(callee(I.Src))=null callee(I.Src)=null
5.	I.Type=dial & busy(I.Param)	[id, I.Src, busytone, null]	–
6.	I.Type=dial & !busy(I.Param)	[id, I.Src, callingtone, null], [id, I.Param, incoming_call, null]	caller(I.Param)=I.Src callee(I.Src)=I.Param busy(I.Param)=true
7.	I.Type=offhook & caller(I.Src)!=null	[id, I.Src, connect, null], [id, caller(I.Src), call_accepted, null]	busy(I.Src)=true callee(caller(I.Src))=I.Src

Модель Feature Manager (табл. 3). Моделируем следующие три функциональности:

1) прямое соединение – Direct Connect (DC) – если абонент *X* подключен к DC с направлением на номер *Y*, то, как только *X* снимает трубку, он автоматически соединяется с абонентом *Y*;

2) запрет входящих – Denied Termination (DT) – если абонент *X* подключен к DT, то все звонки на номер *X* запрещены, абонент, набирающий номер *X*, автоматически получает сигнал «занято»;

3) перевод звонков, когда номер занят – Call Forwarding when Busy (CFB) – если абонент *Y* звонит абоненту *X*, а *X* занят и подключен к CFB с перенаправлением на номер *Z*, то *Y* соединяется с абонентом *Z*.

Для моделирования функциональностей вводим дополнительные глобальные переменные:

Dim DC as Array of Integer;

Dim DT as Array of Boolean;

Dim CFB as Array of Integer.

Эти переменные обозначают наличие или отсутствие данной функциональности у данного абонента. Дополнительно модулю Feature Manager нужно знать id модуля Basic Call Service, для этого заводим переменную: Dim bsc_id as Integer.

Здесь вводятся вспомогательные сигналы типов DC, DT и CFB. Эти сигналы обозначают подключение соответствующей услуги. Сигналы, типы которых имеют знак «-», обозначают отключение соответствующей услуги.

Таким образом, Feature Manager отвечает за срабатывание функциональностей при соответствующих сигналах. Все остальные сигналы транслируются модулю BCS.

Таблица 3

Модель Feature Manager

#	*G*	*O*	*E*
1.	I.Type=offhook & DC(I.Src)!=null	[I.Src, id, dial, DC(I.Src)]	–
2.	I.Type=dial & DT(I.Param)	[id, I.Src, busytone, null]	–
3.	I.Type=dial & CFB(I.Param)!=null & busy(I.Param)	[I.Src, id, dial, CFB(y)]	–
4.	I.Type=DC	–	DC(I.Src)=I.Param
5.	I.Type=DT	–	DT(I.Src)=true
6.	I.Type=CFB	–	CFB(I.Src)= I.Param
7.	I.Type=DC-	–	DC(I.Src)=null
8.	I.Type=DT-	–	DT(I.Src)=false
9.	I.Type=CFB-	–	CFB(I.Src)=null
10.	Не выполнены условия 1–9	[I.Src, bcs_id, I.Type, I.Param]	–

Для возможности использования услуги DT автомату абонента необходимо добавить переход t = {OFFHOOK, DIALLING, -, I.Type = calling_tone, -}. Кроме того, для возможности подключения услуг абонентами необходимы дополнительные переходы в автоматах абонентов. Они представлены в табл. 4, расширяющей табл. 1.

Таблица 4

Модель абонента: дополнительные переходы

#	*Ss*	*Se*	-	*O*
17.	OFFHOOK	DIALLING	-	I.Type = calling
18.	IDLE	IDLE	-	DC(id, Rnd *X*)
19.	IDLE	IDLE	-	DC-(id, Rnd *X*)
20.	IDLE	IDLE	-	CFB(id, Rnd *X*)
21.	IDLE	IDLE	-	CFB-(id, Rnd *X*)
22.	IDLE	IDLE	-	DT(id)
23.	IDLE	IDLE	-	DT-(id)

Эксперименты

Используя модель РКА и алгоритм трансляции РКА в РСП, мы построили раскрашенную сеть Петри, моделирующую телефонную сеть. При помощи полученной модели мы провели симуляцию и программную верификацию интересующих нас свойств для небольших моделей (до 5 абонентов).

Симуляция проводилась средствами CPN Tools. Мы исследовали всевозможные сценарии для базовой модели звонков для небольшого числа абонентов. Также при добавлении функциональностей, были выявлены некоторые нежелательные взаимодействия.

Для верификации мы воспользовались программным комплексом STSV [Nepomniaschy et al., 2008], представляя проверяемые свойства на языке μ-исчисления [Кларк и др., 2002].

Верификация проводилась для следующих свойств.

Наличие тупиков. Тупиком считается такое состояние системы, из которого переход в какое-либо другое состояние невозможен. Для спецификации этого свойства на модели используется формула $\neg$ <to>*true*, где *true* – предикат тождественной истинности.

Зацикливание – наличие таких циклов, из которых невозможно вернуться в исходное состояние. Если имеется набор P подозрительных на зацикливание состояний, то наличие циклов проверяется μ-формулой $P \rightarrow \mu X.(<to>(X \vee P))$. Однако для моделей без тупиков ситуация упрощается: зацикливание совпадает с нарушением условия «возможность вернуться в

исходное состояние», выражаемого формулой $\mu X.(\langle to \rangle(begin \vee X))$, где begin – предикат, истинный в начальном состоянии.

Недетерминизм – наличие в сети конфликтующих переходов, соответствующих двум функциональностям. Недетерминизм для двух функциональностей проверяется следующим образом. Заводятся предикаты, соответствующие разметкам сети, в которых возможно срабатывание перехода для каждой функциональности. Обозначаем эти предикаты, например, по названиям исследуемых функциональностей с добавлением префикса en- (от enabled). Заводим также предикаты для разметок, полученных в результате срабатывания функциональностей по названиям функциональностей. Для функциональностей DT и CFB проверяемая формула будет иметь вид (enDT * enCFB) & <to>(DT & ¬(enCFB)). Формула истинна в моделях, у которых существует переход из разметки, в которой возможно срабатывание обеих функциональностей, в разметку, которая получена в результате срабатывания DT и в которой невозможно срабатывание CFB. Это и есть недетерминизм.

Нарушение условий. В данном случае условие есть только у функциональности Denied Termination: «никто никогда не может звонить абоненту, подключившему услугу DT». В терминах автоматной модели нарушение этого условия будет означать, что на некотором такте для некоторого X из множества идентификаторов абонентов выполнено DT[X] = true, и для X есть входящий сигнал типа Incloming_call. Выявляется это свойство по наличию в графе достижимости таких состояний сети, которые имеют соответствующую разметку: фишку [X, true] в месте DT и фишку [Src, X, Incoming_call, null] в месте SIGNALS. Для проверки этого свойства нет необходимости проводить программную верификацию, так как оно выявляется на этапе построения предикатов.

Результаты проведенных экспериментов представлены в табл. 5.

Таблица 5

Взаимодействие функциональностей

Функциональности	Тупики	Зацикливание	Недетерминизм	Нарушение условий
CFB + DC	false	false	False	false
CFB + DT	false	false	true	false
DC + DT	false	false	false	false
CFB + CFB	false	true	false	false
All Features	false	true	true	false

Поясним некоторые результаты. Если у абонента подключены одновременно CFB и DT, возникает недетерминизм, и не ясно, какая именно функциональность должна сработать: должен ли входящий звонок быть сброшен, либо он должен быть переведен. В случае CFB + CFB возникает зацикливание, если два абонента настроили эту услугу, указав номера друг друга в качестве номеров для перевода звонка.

Заключение

В данной работе представлен новый двухуровневый метод моделирования и верификации телекоммуникационных систем. Этот метод на первом этапе моделирует телекоммуникационные системы в виде расширенных конечных автоматов, а на втором этапе переводит эти модели в раскрашенные сети Петри. Данный метод применяется к исследованию известной проблемы взаимодействия функциональностей в телефонных сетях. В качестве примера рассматривается базовая модель звонков с дополнительными функциональностями. Для построения графов достижимости раскрашенных сетей Петри используется система CPN Tools, а для верификации методом проверки моделей используется система PNV, которая входит в программный комплекс STSV. Описанные эксперименты позволили выявить некоторые нежелательные взаимодействия функциональностей в телефонных сетях.

В качестве дальнейшего развития данной работы предполагается реализовать трансляцию автоматной модели во входной язык Promela системы SPIN для последующей верификации методом проверки моделей. Также предполагается исследовать проблему взаимодействия

для других известных функциональностей, используемых в мобильных телефонных сетях с помощью программных систем CPN Tools, STSV и SPIN.

Список литературы

Кларк Э. М., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. М.: Изд. МЦНМО, 2002.

Calder M., Miller A. Using SPIN for Feature Interaction Analysis – A Case Study // Lecture Notes in Computer Science. Proc. SPIN. 2001. Vol. 2057. P. 143–162.

Capellmann C., Dibold H., Herzog U. Using High-Level Petri Nets in the Field of Intelligent Networks // Lecture Notes in Computer Science. 1999. Vol. 1605. P. 1–36.

Cavalli A., Maag S. A New Algorithm for Service Interaction Detection // Lecture Notes in Computer Science. Proc. ICFEM-2002. 2002. Vol. 2495. P. 371-382.

Jensen K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Springer, 1996. Vol. 1–3.

Keck D. O., Kuehn P. J. The Feature and Service Interaction Problem in Telecommunications Systems: A Survey // IEEE Transactions on Software Engineering. 1998. Vol. 24. No. 10.

Kristensen L. M., Christensen S., Jensen K. The Practitioner's Guide to Coloured Petri Nets // Internat. J. Software Tools for Technology. 1998. Transfer 2 (2). P. 98–132.

Lorentsen L., Tuovinen A., Xu J. Modelling Feature Interaction Patterns in Nokia Mobile Phones using Coloured Petri Nets and Design/CPN // Proc. 3^{d} Workshop on Practical Use of Coloured Petri Nets (CPN'01), Aarhus Univ., DAIMI PB-554. 2001. P. 1–14.

Nakamura M. Design and Evaluation of Efficient Algorithms for Feature Interaction Detection in Telecommunication Services: PhD dissertation. Osaka, 1999.

Nepomniaschy V., Beloglazov D., Churina T., Mashukov M. Using Coloured Petri Nets to Model and Verify Telecommunications Systems // Lecture Notes in Computer Science. Proc. CSR-2008. Vol. 5010. P. 360–371.

Ratzer A. V., Wells L., Lassen H. M., Laursen M., Qvortrup J. F., Stissing M. S., Westergaard M., Christensen S., Jensen K. CPN Tools for Editing, Simulating and Analysing Coloured Petri Nets // Lecture Notes in Computer Science. Proc. ICATPN. 2003. Vol. 2679. P. 450–462.

Материал поступил в редколлегию 26.08.2008

D. M. Belogolazov, V. A. Nepomniaschy

Modeling and Verification of Feature Interaction in Telephone Networks Using Finite Automata and Coloured Petri Nets

For modeling, analysis and verification of telecommunication systems, the models such as finite automata, Petri nets and their generalizations are usually applied. The goal of our work is to represent a new two-level approach for modeling and verification of telecommunication systems. On the first level telecommunication systems are modeled by extended finite automata, while on the second level the automata models are translated into coloured Petri net (CPN). This method is applied to investigation of feature interaction problem in telephone networks. As an example, Basic Call State Model with additional features is considered. CPN Tools is used for construction of reachability graphs of CPN. We used the Petri Net Verifier for verification of the net models by model checking method. Described experiments allowed us to detect some undesirable feature interactions in telephone networks.

Keywords: telecommunication systems, finite automata, coloured Petri nets, feature interaction, telephone networks, verification, model checking method.