УДК 623/518.3/517.5

А.А.Летичевский

ОБ ОДНОМ КЛАССЕ БАЗОВЫХ ПРОТОКОЛОВ

Рассматривается проблема представления требований к поведению интерактивной системы в виде формальных спецификаций, а также ее верификация и генерация трасс, используемых для создания тестовых наборов. Исследуется специальный класс спецификаций, представленный в виде базовых протоколов, в котором рассматриваются возможные виды противоречивости и неполноты. С помощью символьного моделирования требований, производится порождение символьных трасс, используемых для тестирования создаваемой системы по различными критериями.

1. Проблема верификации и формализации требований

Сбор требований является начальным и неотъемлемым этапом процесса разработки и заключается в определении набора функциональностей, который необходимо реализовать в продукте. Процесс сбора реализуется частично в общении с заказчиком, частично посредством мозговых штурмов разработчиков. Результатом является формирование набора требований к системе, именуемой в отечественной практике техническим заданием.

Фиксация требований (Requirement Capturing), с одной стороны, определяется желаниями заказчика в реализации того или иного свойства. С другой стороны, в процессе сбора требований может обнаружиться ошибка, которая приведет к определенным последствиям, устранение которых потребует дополнительное кодирование, редизайн, перепланирование и др.

Ошибка может быть тем серьезней, чем позже она будет обнаружена, особенно если это связано с сотнями и тысячами спецификаций, например в описании телекоммуникационных протоколов или описании работы операционной системы.

Поэтому одной из составляющей этапа фиксации требований наряду со сбором является верификация требований, а именно проверка их на непротиворечивость и полноту.

Автоматизированная верификация требований может проводиться лишь после формального описания или формализации требований.

Формализация включает в себя определение компонентов системы и их состояний, сигналов, с помощью которых взаимодействуют компоненты, и условий в формальном виде, которые должны выполняться при изменении состояний компонентов. Для формального описания поведения системы используются языки инженерных спецификаций MSC [1], SDL [2], UML [3]. В [4-6] в качестве формальной модели для описания требований использовалось понятие системы базовых протоколов. Описание систем с помощью базовых протоколов дополняет языки инженерных спецификаций, поднимая уровень абстракции и позволяя использовать дедуктивные средства верификации в сочетании с моделированием поведения систем путем генерации трасс и проверки моделей, извлекаемых из базовых протоколов. Совокупность сгенерированных трасс является основой для создания множества тестов, которые впоследствии будут использованы для тестирования разработанного продукта. Возможен также синтез модели продукта по формализованным требованиям на том уровне абстракции, на котором описаны требования с использованием средств языков MSC и SDL. Эта модель может быть основой для дальнейшей разработки продукта.

В данной статье рассматривается специальный класс базовых протоколов, называемых К-протоколами и определяемых как асинхронные императивные протоколы с одним ключевым агентом. Для этого класса протоколов строятся специальные алгоритмы верификации, генерации трасс и тестовых последовательностей, более эффективные, чем для базовых протоколов общего вида. Несмотря на строгие ограничения, К-протоколы часто встречаются на практике в таких областях, как телекоммуникации, телематика и др.

2. Базовые протоколы

Базовый протокол является формальной записью утверждения о некоторых событиях, которые должны произойти в программе, алгоритме, протоколе при выполнении определенных условий. Например, в утверждении *"Если после набора номера телефон получает сигнал dial_busy, он переходит в состояние занято"* присутствуют три компонента: предусловие – «*После набора номера*», процесс – «*Получение сигнала dial_busy*» и постусловие – «*Состояние занято*».

Базовый протокол имеет вид аналогичный тройке Хоара $\alpha \rightarrow < P > \beta$, α и β называются предусловием и постусловием соответственно, а P – процессом протокола. Условия α и β представляются с помощью логических выражений некоторого базового языка и определяют условия на множестве состояний системы. Рассматриваются также параметризованные протоколы, имеющие вид $\forall x(\alpha \rightarrow < P > \beta)$, где x – список параметров протокола, а формулы α и β вместе с процессом P могут зависеть от этих параметров.

Общая теория базовых протоколов изложена в [4]. Она содержит общее определение систем базовых протоколов, их семантику и различение конкретных и абстрактных протоколов, а также связей между ними. Первые используются для точного моделирования поведения систем, вторые – для распознавания их свойств с использованием дедуктивных средств.

В данной статье рассмотрим класс базовых протоколов, который является частным случаем конкретных базовых протоколов из [4]. Этот класс можно определить как класс *асинхронных императивных базовых протоколов с одним ключевым агентом*. Базовые протоколы указанного класса будем называть К-протоколами. Асинхронность понимается в том смысле, что обмен информацией осуществляется по буферизованным каналам. Императивность означает, что каждый протокол эквивалентен некоторой императивной программе, а один ключевой агент полностью определяет функционирование базового протокола. Более точные определения следуют ниже.

Система, определяемая базовыми протоколами, представляется в виде композиции среды и агентов, которые в нее погружены [7]. Состояние среды формируется набором значений *атрибутов среды*, играющих роль общей памяти и *параметров (атрибутов) агентов*. Совокупность атрибутов агента образует его локальную память. Агенты имеют типы и имена, а также могут находиться в конкретных состояниях. Состояние агента однозначно устанавливает поведение системы после погружения в нее агента в данном состоянии. Функция погружения определяется как параллельное погружение, т.е. $s[u,v] = s[u \parallel v]$, где $u \parallel v$ обозначает параллельную асинхронную композицию агентов (интерливинг). Примером среды является телефонная сеть. Агенты – это телефоны, а их параметры включают в себя номера телефонов, режимы подключения, отношения с другими телефонами (например, соединение) и другие характеристики, изменяющиеся во времени.

Атрибуты среды и параметры агентов имеют типы: арифметический, логический (булевский) и символьный. Арифметические типы делятся на целые и вещественные, а символьные представляются алгебраическими выражениями в заданной сигнатуре и могут быть определены как абстрактные типы данных путем задания редукций (переписывающих правил, определяющих каноническую форму). Параметр p агента a типа T записывается в виде $T\ a.P$. Набор параметров агента однозначно определяется его типом. Утверждение о том, что агент a типа T находится в состоянии u, записывается в виде $T(a,u)$.

Агенты взаимодействуют между собой путем асинхронного обмена сообщениями. Передача сообщения представляется действием

$$\mathbf{Out}(T\ a, T'\ a', x(p_1,..., p_m)),$$

где a – имя передающего агента типа T; a' – имя агента типа T', которому передается сообщение x; $p_1,..., p_m$ – параметры сообщения (выражения в языке данных). Поскольку имена агентов уникальны, то указатель типа в них может отсутствовать. Предполагается, что каждый агент имеет вход-

ную очередь поступающих к нему сообщений. Прием осуществляется путем чтения первого сообщения из очереди, а передача ставит сообщение в конец соответствующей очереди. Различаем внешние (*наблюдаемые*) и *внутренние* действия, выполняемые агентами. Передача сообщений относится к наблюдаемым действиям и входит в процесс базового протокола. Что же касается приема, то это внутреннее действие, выполняемое неявно, при обращении к первому элементу очереди. Это обращение осуществляется при проверке истинности предиката $\mathbf{In}(T\ a, z)$, используемого в предусловиях. В предикате z есть выражение, которое может зависеть от переменных (параметров протокола). Предикат имеет значение истина, если очередь входных сообщений не пуста, и первое в очереди сообщение сопоставляется с выражением z (мэтчинг). Для определения того, что очередь пуста, используется предикат **emptyq**(n), который принимает значение 1, если в очереди нет сообщений.

Кроме действий, выражающих прием и передачу сообщений, агенты могут выполнять наблюдаемые локальные действия, обозначаемые как **action** X, где X – имя локального действия. Результат выполнения локального действия определяется постусловием.

Рассмотрим дополнительные ограничения на базовые протоколы. *Предусловие* ограниченного протокола b имеет вид $\mathbf{pred}(b) = T(a,u) \wedge F$, где a – имя ключевого агента типа T, участвующего в протоколе; u – его состояние; F – логическое выражение, построенное с помощью логических связок, равенств и неравенств из атрибутов среды и параметров ключевого агента. *Процесс* **proc**(b) протокола b представляет собой последовательность $a_1 a_2 \ldots a_m$ действий, выполняемых ключевым агентом. Все элементы этой последовательности представляют собой передачи сообщений от ключевого агента другим агентам и локальные действия. В случае, когда порядок выполнения некоторых действий не существен, вместо последовательности могут рассматриваться выражения алгебры процессов с недетерминированным выбором. *Постусловие* **post**(b) базового протокола b представляет собой конъюнкцию $\mathbf{post}(b) = T(a,u') \wedge G$, где u' – новое состояние агента a, а G – совокупность $x_1 := y_1 \,\&\, x_2 := y_2 \,\&\, \ldots \,\&\, x_m := y_m$ операторов присваивания. Здесь $x_1, x_2, \ldots, x_m$ – атрибуты среды или параметры ключевого агента; $y_1, y_2, \ldots, y_m$ – выражения соответствующих типов, также зависящие только от его параметров. Кроме изменения значений параметров, агент может очистить свою входную очередь, выполняя присваивание **queue**(n):=**empty**. Каждый из операторов рассматривается как высказывание о том, что значение атрибута x_i после завершения протокола будет равно значению выражения y_i, вычисленному для значений атрибутов до начала применения протокола.

В случае, когда в предусловии присутствует предикат $\mathbf{In}(T\ a, z)$, предполагается неявное изменение входной очереди агента. Из нее удаляется первое сообщение. Кроме того, передачи вызывают изменения входных очередей соответствующих агентов, путем включения в очередь отправляемых сообщений.

Для функционирования агента необходимо, чтобы параметры агентов, атрибуты и состояние агента имели начальные значения, определяющие начальное состояние среды. Это состояние однозначно фиксирует поведение среды и агентов, погруженных в нее. Процесс изменения состояний среды устанавливается выполнением базовых протоколов, которые могут функционировать параллельно и недетерминировано. Недетерминизм среды определяется интерливингом при выполнении наблюдаемых действий, относящихся к различным протоколам, а также конкретизацией базовых протоколов, т.е. выбором значений параметров, при которых предусловия становятся истинными. Конкретная история функционирования среды описывается одной из возможных последовательностей наблюдаемых действий и составляет трассу. Для общего случая формальное определение поведения системы, описанной базовыми протоколами, было дано в [1].

В дальнейшем рассмотрим системы К-протоколов, удовлетворяющие требованию

однородности, которое означает, что все базовые протоколы параметризированы именами своих ключевых агентов. Иными словами, каждый протокол начинается квантором общности по имени ключевого агента. Следует заметить, что требование однородности не ограничивает общности, поскольку она сводится к однородному путем более детальной классификации агентов по типам.

3. Пример системы заданной базовыми К-протоколами

В качестве примера системы, заданной базовыми К-протоколами, рассмотрим POTS (Plain Old Telephone System), которая определяет функционирование обычной телефонной сети. В данной сети функционируют телефоны – агенты типа Phone и агент Net типа Network, представляющий коммутационную систему сети. Агент типа Phone может находится в следующих состояниях: idle, offhook, dial, dialing, ringing, busy, fastbusy, ring, offhook_ring, connected. Агенты имеют имена (номера телефонов), и базовые протоколы параметризированы этими именами. Ниже перечислены протоколы, ключевой агент которых имеет тип Phone. Внешний квантор общности, который связывает имя ключевого агента *n*, присутствует неявным образом.

B1: Phone(n,idle)-> < Out(n,Net,offhook n) > Phone(n,offhook);
B2: Phone(n,offhook)& In(n,dialtone) -> < > Phone(n,dial);
B3: Forall m(Phone(n,dial)-> < Out(n,Net, dial(n,m)) > Phone(n,dialing));
B4: Phone(n,idle)& In(n,ring) -> < > Phone(n,ring);
B5: Phone(n,dialing)&In(n,ring)-> < > Phone(n,ringing);
B6: Phone(n,ring)-> <Out(n,Net,offhook n)> Phone(n, offhook_ring);
B7: Phone(n,connected)-> < Out(n, Net, onhook n) > Phone(n, idle);
B8: Phone(n,connected)&In(n, dialtone)-> < > Phone(n, dial);
B9: Phone(n,dialing)&In(n, busy)-> < > Phone(n, busy);
B10: Phone(n,ringing)-> <Out(n, Net, onhook n)> Phone(n, idle);
B11: Phone(n,busy)&In(n, fastbusy)-> < > Phone(n, fastbusy);
B12: Phone(n,busy)-> <Out(n, Net, onhook n)> Phone(n, idle);
B13: Phone(n,fastbusy)-> <Out(n, Net, onhook n)> Phone(n, idle);
B14: Phone(n,dial)&In(n, fastbusy)-> < > Phone(n, fastbusy);
B15: Phone(n,dial)-> <Out(n, Net, onhook n)> Phone(n, idle);
B16: Phone(n,offhook_ring)&In(n, connected)-> < > Phone(n,connected);
B17: Phone(n,ringing)&In(n, connected)-> < > Phone(n, connected).

Наблюдаемые действия телефонов состоят в передаче сообщений (сигналов) в сеть, мгновенно записываемых в очередь агента Net. Так, например, Out(*n*,Net,offhook *n*) есть сигнал о снятой трубке, Out(*n*,Net,dial(*n*,*m*)) – сообщение о том, что абонент *n* набрал номер *m*, Out(*n*, Net, onhook *n*) – сигнал о том, что трубка абонента *n* положена на рычаг. Все сообщения, посылаемые от сети к абоненту, становятся в очередь и считываются абонентом путем неявного выполнения внутреннего действия чтения из своей очереди при проверке предусловия, содержащего предикат In(*n*,*s*). В частности, принятие сигнала dialtone (проверка условия In(*n*, dialtone)) приводит к появлению непрерывного гудка в трубке; проверка In(n,ring) – в телефоне, находящемся в состоянии idle (трубка лежит на рычаге), появляется звонок, что выражается в переходе телефона в состояние Phone(*n*,ring); тот же самый сигнал приводит к появлению длинных прерывистых гудков в снятой трубке и переводит телефон в состояние Phone(n, ringing) и т. д. Соответствие текстовых требований и формальных отражается в следующей таблице.

Таблица

B_1, B_2	При снятии трубки с неактивного телефона из сети должен прийти сигнал длинного непрерывного гудка
B_3	После набора номера сигнал с адресом вызываемого телефона поступает в сеть
B_4,	После звонка при поднятии трубки

B_6, B_{16}	осуществляется соединение с вызывающим абонентом
B_5	После вызова абонента в вызывающий телефон приходит сигнал вызова
B_7	По окончании связи после того, как трубка положена на рычаг, телефон переходит в неактивное состояние
B_8	Если абонент на другом конце провода положил трубку на рычаг, приходит сигнал длинного гудка
B_9	Если после набора номера, вызываемый абонент занят, приходит сигнал «занято».
B_{10}	Если вызывающему абоненту не удается добиться соединения и он кладет трубку, телефон переходит в неактивное состояние
B_{11}	При поднятой трубке после сигнала «занято» может прийти сигнал коротких гудков
B_{12}, B_{13}	Если трубка поднята и приходит сигнал «занято», то после того, как трубка положена на рычаг, телефон переходит в неактивное состояние
B_{14}	При поднятой трубке после длинного гудка в трубке могут появиться короткие гудки
B_{15}	Если трубка поднята и телефон принимает длинный гудок, трубка может быть положена и телефон перейдет в неактивное состояние
B_{17}	После ответа абонента телефоны соединяются

Как видим, в общем случае неформальные требования не содержат в явном виде всех компонентов базового протокола – предусловие, процесс, постусловие. Кроме того, ряд существенных условий, которые входят в предусловие, может быть опущен. Поэтому одной из задач формализации является явное определение этих компонентов, и восстановление пропущенных условий.

В рассматриваемом примере параметры агентов и атрибуты среды отсутствуют, поэтому поведение агента определяется только его состояниями. Телефон является недетерминированным агентом, т.е. в одном и том же состоянии возможно применение различных базовых протоколов. Например, в состоянии dial можно набрать номер либо положить трубку.

Агент-коммутатор Net является полностью детерминированным и его поведение полностью определяется сообщениями, которые передаются телефонами. Эти сообщения, как правило, содержат информацию о новом состоянии телефона, а для того, чтобы установить следующее действие, коммутатор должен знать предыдущее состояние телефона. Этой цели служит параметр Network Net.phone_state *n*, где *n* – номер телефона. Изменение значений этого параметра задается присваиваниями в постусловиях базовых протоколов агента Net. В процессе работы агент типа Network не меняет своего состояния Active.

C1: Forall n(Network(Net,Active)&
&In(Net,offhook n)&(Network
Net.phone_state n =idle)->
-><Out(Net,n,dialtone)>(Network
Net.phone_state n:=offhook));
C2: Forall(m,n)(Network(Net,Active)&
&In(Net,dial(n,m))&(Network Net.phone_state
m=idle)->
-><Out(Net,n,ring)||Out(Net,m,ring)>((Network Net.phone_state n:=ringing m)&
&(Network Net.phone_state m:=ring)));
C3: Forall(m,n)(Network(Net,Active)&
&In(Net,offhook m)&(Network
Net.phone_state n=ringing m)-><Out(Net,n,
connected)||Out(Net,m, connected)> ((Network
Net.phone_state m:= connected n)&(Network
Net.phone_state n:=connected m)));
C4: Forall(m,n)(Network(Net,Active)&
&In(Net,onhook n)&(Network Net.phone_state
n=connected m)-><Out(Net,m, dial-
tone)>((Network Net.phone_state n:=idle)&
&(Network Net.phone_state m:=offhook)));
C5: Forall(m,n)(Network(Net,Active)&
&In(Net,dial(n,m))&~(NetworkNet.phone_stat
e m=idle)->
-><Out(Net,n,busy)>(Network Net.phone_state
n:=busy));
C6: Forall n(Network(Net,Active)&(Network
Net.phone_state n=busy)->
-><Out(Net,n,fastbusy)>(Network
Net.phone_state n:=fastbusy));
C7: Forall n(Network(Net,Active)&In(Net, on-

hook n) -><>(Network Net.phone_state n:=idle));
C8: Forall n(Network(Net,Active)&(Network Net.phone_state n=dial)->
-><(Out(Net,n, fastbusy)>(Network Net.phone_state n:=fastbusy));
C9: Forall n(Network(Net,Active)&In(Net, onhook n)&(Network Net.phone_state n=dial)->
-><>(Network Net.phone_state n:=idle));
C10:Forall n(Network(Net,Active)&In(Net, onhook n)&(Network Net.phone_state n=busy)->
-><>(Network Net.phone_state n:=idle));
C11:Forall n(Network(Net,Active)&In(Net, onhook n) &(Network Net.phone_state n=ringing m)->
-><>(Network Net.phone_state n:=idle)).

4. Непротиворечивость и полнота базовых К-протоколов

После формализации требований появляется возможность провести их автоматическую верификацию, т.е. проверить выполнение корректности требований по различным критериям. С другой стороны, поскольку формальные требования определяют некоторую модель системы, можно верифицировать различные свойства этой модели, пользуясь темпоральной логикой и техникой проверки моделей (model checking). Важными критериями, которые используются при проверке корректности требований, являются их *непротиворечивость* и *полнота*. Содержательная трактовка этих понятий зависит от конкретных приложений и не всегда хорошо формализуется. Однако в ряде случаев можно сформулировать необходимые или достаточные критерии для непротиворечивости и полноты. Проверка этих критериев во многих случаях позволяет отыскать причины истинных ошибок в инженерных решениях на уровне требований.

Рассмотрим критерии, связанные со следующими требованиями: в любой момент времени выбор К-протокола для заданного ключевого агента должен быть однозначным (непротиворечивость), и если система не перешла в состояние успешного завершения своего функционирования, то должен быть хотя бы один протокол для продолжения ее функционирования (полнота). Требование непротиворечивости может быть ослаблено для тех ситуаций, когда агенту позволено осуществлять недетерминированный выбор в своих действиях и тем самым определять свое поведение с помощью различных протоколов.

Достаточными условиями непротиворечивости являются невозможность одновременного выполнения предусловий двух различных протоколов, а достаточное условие полноты выражается как тождественная истинность дизъюнкции всех предусловий всех протоколов. Формально условие непротиворечивости для двух параметризованных базовых протоколов

$$\forall x(\alpha_1 \rightarrow < P_2 > \beta_1) \text{ и } \forall y(\alpha_2 \rightarrow < P_1 > \beta_2)$$

с одним и тем же ключевым агентом, имя которого входит свободно в протоколы, выражается в виде отрицания пересечения формул, выражающих условия применимости протоколов:

$$\neg((\exists x \alpha_1) \wedge (\exists y \alpha_2)) .$$

Два базовых протокола формально противоречивы, если это условие, называемое условием *транзиционной непротиворечивости*, не может быть доказано, т.е. является выполнимым. При проверке транзиционной непротиворечивости необходимо учитывать следующие обстоятельства. Во-первых, оно должно выполняться только на достижимых состояниях системы. Во-вторых, формальная противоречивость может не быть достаточной для содержательного противоречия, поскольку недетерминизм, вызванный этой противоречивостью, может быть допустимым исходя из инженерных соображений. Рассмотрим, например, К-протоколы B7 и B8 формализации требований к телефонной системе. Формально они противоречивы, поскольку их условия могут быть истинными одновременно, и выбор протокола будет происходить недетерминированным образом. Однако этот недетерминизм вполне допустим, поскольку управление трубкой выполняется пользователем и она может быть положена до того, как телефон проверит входную очередь сигналов (сообщений), посланных системой в качестве информации о том, что на другом конце соединения трубка уже повешена и телефон должен перейти в состояние dial.

Другая ситуация, когда формальная противоречивость может быть допустимой, состоит в том, что постусловия протоколов определяют одинаковые преобразования состояния среды.

Автоматическая дедуктивная система, которая применялась для проверки требований к телефонной сети, обнаружила 7 противоречивых пар требований. Примером может служить противоречие между протоколами В3 и В14. Условие их непротиворечивости

~((Phone(n,dial)& Phone(n,dial)& In(n, fastbusy)<=> ~((Phone(n,dial)& In(n, fastbusy))

не может быть доказано, и, поскольку в данном случае недетерминизм недопустим, обнаруженное противоречие указывает на возможность ошибки. Такая ошибка на самом деле имеет место, и она должна быть исправлена путем усиления предусловия К-протокола В3. Исправленный протокол имеет вид

В3: Forall m(Phone(n,dial)&emptyq(n)-> < Out(n,Net,dial(n,m)) > Phone(n,dialing)).

Теперь непротиворечивость становится доказуемой, поскольку из In(n, fastbusy) следует, что очередь сообщений телефона *n* не пуста.

Другим примером может служить обнаруженное противоречие между протоколами С7 и С11. Условие их непротиворечивости имеет вид:

~(Exist n(Network(Net,Active)&(Network Net.phone_state n=busy))&
&Exist n(Network(Net,Active)&
&In(Net,onhook
n)&(Network Net.phone_state n=busy))).

Эта формула не может быть доказана, поскольку для некоторого телефона *n* может быть истинным условие «трубка на рычаге» и сигнал об этом стоит первым в очереди агента Net, а перед этим он воспринимал сигнал «занято». Кроме того, поведение сети, определяемое этими протоколами, различно, что нежелательно. Заметим, что протокол С6 входит в противоречие также и со всеми другими протоколами, предусловия которых могут быть истинными для телефонов, отличных от *n*. Следует отметить, что изменение условия In(Net,onhook n) на его отрицание не спасает, поскольку новое требование входит в противоречие с другими требованиями с позитивным вхождением условия In для других сообщений. Аналогичные противоречия возникают также для протокола С8.

Рассмотренные противоречия легко разрешаются путем введения нового агента типа таймера, который будет устанавливаться при переходе некоторого агента в состояние dial или busy и через некоторое время сигнализировать сети об истечении времени, после которого сеть передает телефону установку на короткие гудки.

Протоколы для таймера выглядят следующим образом:

D1: Timer(T,active)&In(T,start *n*) -><> Timer(T,start *n*);

D2: Timer(T,start *n*) -><Out(Net,expired *n)*> Timer(T,active).

Теперь можно исправить протокол С6 и добавить управление таймером в С7:

C6.1:Forall n(Network(Net,Active)&In(Net, expired n)& ~(Network Net.phone_state n=busy)->
-><Out(Net,n,fastbusy)>(Network(Net, Active)&(Network Net.phone_state n:=fastbusy)));

C6.2:Forall n(Network(Net,Active)&In(Net, expired n)& (Network Net.phone_state n= = dial)-><Out(Net,n,fastbusy)> (Network(Net, Active)&(Network Net.phone_state n:=fastbusy)));

C5:Forall(m,n)(Network(Net,Active)&In(Net,dial(n,m))& ~(Network Net.phone_state m=idle)->
->< Out(Net,n,busy) || Out(T,start n)> (Network Net.phone_state n:=busy)).

Аналогичные изменения вводятся в протоколы С8 и С1. Противоречие между протоколами С4 и С7 устраняется путем уточнения предусловия протокола С7:

C7:Forall n(Network(Net,Active)&
&In(Net,onhook n)&~(Exist m(Network Net.phone_state n=connected m)-><> (Network Net.phone_state n:=idle)).

Однородные системы К-протоколов удобно классифицировать по типам и состояниям ключевых агентов. Достаточным условием полноты требований является полнота для всех агентов, находящихся в одном классе, т.е. имеющих заданный тип и

находящихся в заданном состоянии. Кроме того, удобно ослабить требование полноты, считая, что для агента с пустой входной очередью может не быть протокола, определяющего его функционирование. Таким образом, условие полноты для класса *K*(*T*,*s*) базовых протоколов типа *T*, находящихся в состоянии *s*, формулируется следующим образом:

$$\forall n(T(n,s) \wedge \neg \mathbf{emptyq}(n) \rightarrow \\ \rightarrow (\exists x_1 \alpha_1 \vee \exists x_2 \alpha_2 \vee ...)),$$

где $\alpha_1, \alpha_2, ...$ - все предусловия базовых протоколов класса *K*(*T*,*s*), а $x_1, x_2, ...$ - списки их параметров без общего имени ключевого агента *n*.

Свойства транзиционной непротиворечивости и полноты для базовых протоколов рассматриваются для состояний системы, достижимых из некоторых заранее заданных начальных состояний системы. Обычно начальные состояния системы предполагают пустоту входных очередей, выделенные начальные состояния агентов, ограничения на начальные значения параметров. В некоторых случаях могут быть известны также ограничения на достижимые состояния системы, выраженные в терминах базового языка. Такие ограничения называются условиями целостности и могут быть использованы дедуктивной системой в качестве аксиом для доказательства условий полноты и непротиворечивости. Условия целостности обычно формулируются исходя из содержательных соображений, относящихся к предметной области. Однако они могут нарушаться в силу противоречивости требований и должны быть проверены так же, как и условия транзиционной непротиворечивости и полноты.

5. Трассовая генерация и критерии тестирования

Модель системы, определяемая однородной системой К-протоколов, эквивалентна недетерминированной распределенной (параллельной) программе над двухуровневой памятью (разделяемая память среды и локальная память агентов). Если зафиксировать начальное состояние системы s_0, которое включает в себя состояние среды и состояния всех погруженных в нее агентов, то можем получить все возможные истории функционирования системы как последовательности вида

$$s_0 \xrightarrow{b_1(n_1,m_1)} s_1 \xrightarrow{b_2(n_2,m_2)} \dots,$$

где $b_1(n_1,m_1), b_2(n_2,m_2), ...$ – конкретизированные базовые протоколы, при этом $n_1, n_2, ...$ – имена ключевых агентов, $m_1, m_2, ...$ – наборы значений остальных параметров, выполняющих предусловия базовых протоколов. Поскольку постусловия определяют детерминированные преобразования, то состояния $s_1, s_2, ...$ однозначно определяются начальным состоянием системы и конкретизацией базовых протоколов. История может быть конечной или бесконечной. Рассматривая максимальные (не продолжаемые) истории, получим, что конечные истории оканчиваются либо успешным завершением (все агенты находятся в состоянии Delta), либо тупиковым состоянием. Заметим, что полнота в классах базовых протоколов не гарантирует отсутствия тупиковых состояний, поскольку возможно достижимое состояние, при котором все входные очереди пусты и нет протокола, функционирующего при пустой входной очереди (при заданных состояниях агентов).

Если отбросить из истории состояния системы и оставить только конкретизированные протоколы $b_1(n_1,m_1), b_2(n_2,m_2), ...$ или процессы (последовательности наблюдаемых действий), то получим *трассу*, соответствующую данной истории. Генерация историй и трасс используется для решения двух важных задач обработки требований: проверка достижимости условий и построение тестов. Тестироваться может как система, так и отдельный агент. При тестировании агента остальные агенты рассматриваются как его окружение. Само тестирование осуществляется как воздействие окружения на тестируемый агент. В трассах входные сообщения, посылаемые из окружения агенту, рассматриваются как управление, а выходные сообщения агента – как контролируемая реакция агента.

Генерация трасс может быть организована как последовательный перебор всех возможных вариантов применений базовых протоколов путем движения по дереву поведения системы исходя из некоторых на-

чальных состояний. Количество трасс, которые необходимо сгенерировать для доказательства достижимости условий или покрытия фрагментов поведения тестируемого компонента, даже при конечном множестве состояний может быть очень большим. Поэтому генерация трасс управляется набором фильтров, используемых в алгоритме генерации трасс в качестве входной информации.

К основным фильтрам относятся следующие:

– условие завершения трассы. Оно состоит из двух составляющих: утверждение о состоянии агента и формулы над атрибутами и параметрами агентов. Трасса прекращает генерироваться, если условие завершения трассы истинно. Например, при генерации трасс в вышеуказанной формализации работы телефона условием завершения трассы может быть утверждение о том, что данный телефон находится в состоянии connected. При достижении этого состояния трасса будет завершена;

– условие исключения состояний. Фильтр допускает список условий, при выполнении которых прекращается дальнейшее продвижение по трассе и выполняется переход к другому варианту;

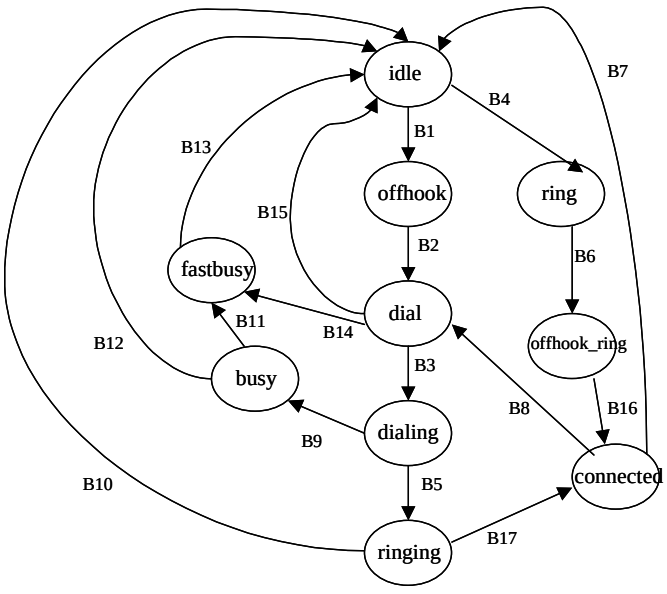
– другие фильтры определяют такие параметры генерации, как общее количество трасс, максимальная длина трассы и некоторые другие числовые ограничения.

Условия завершения генерации и условия исключения состояний могут быть выражены в терминах линейной темпоральной логики и проверяться синтезированным по этим условиям автоматом, наблюдающим изменение состояний системы в процессе генерации.

При генерации трасс для тестирования одного агента удобно пользоваться диаграммой переходов этого агента. Эта диаграмма есть графическое представление транзиционной системы, состояниями которой являются состояния агента, а действиями – базовые протоколы, конкретизированные по имени этого агента. Переходы из состояния s в состояние s' размечены базовыми протоколами, для которых такой переход возможен чисто синтаксически, т.е. s – состояние ключевого агента, а s' – его новое состояние, которое находится из постусловия. На рисунке изображена диаграмма переходов агента типа Phone.

Рассмотрим все трассы, соответствующие простым путям диаграммы переходов телефона, начинающиеся и оканчивающиеся в состоянии idle. Их всего 24.

Диаграмма поведения агента отражает возможные трассы без учета состояния среды и других агентов, погруженных в нее. Поэтому, вообще говоря, не все трассы данной диаграммы, возможны. Кроме того, применение протокола определяет конкретные значения параметров, от которых он зависит. Поэтому конкретизированных трасс может быть больше, чем на диа-



Рисунок

грамме. В случае рассматриваемого примера только один протокол B3 имеет параметр, отличный от имени ключевого агента. Конкретизируя этот параметр путем выбора произвольного имени агента (на самом деле следует рассмотреть два случая: имя совпадает с именем ключевого агента и отлично от него), получим уже конкретизированные протоколы. Поскольку в рассматриваемом случае все пути возможны, получим, что все 24 трассы могут использоваться для тестирования телефона. При этом сигналы, идущие от агента Network, рассматриваются как управляющие, а сигналы, входящие в него, как контрольные.

В различных проектах, в которых приходится тестировать объекты телекоммуникаций, количество состояний одного агента сравнительно невелико. Экспоненциальный взрыв получается из-за того, что велико количество агентов и, следовательно, состояний всей системы (не говоря уже о трассах). Поэтому здесь возникают следующие вопросы.

- Какое количество трасс необходимо взять для тестирования продукта?
- Как получить минимальное количество трасс, достаточное для тестирования?

Ответы на эти вопросы тесно связаны с критериями, по которым определяются трассы, необходимые для тестирования. Одним из них будет полное покрытие требований, выраженных с помощью базовых протоколов. Это означает, что каждый протокол должен встретиться по крайней мере один раз хотя бы в одной трассе. Следует заметить, что трасс с однократным вхождением базовых протоколов может быть недостаточно для обнаружения некоторых видов ошибок. Например, в протоколах для POTS есть ошибка, состоящая в том, что входная очередь телефона не очищается после того, как трубка кладется на рычаг (что, очевидно, должно делаться путем выполнения присваивания **queue**(*n*):=**empty** в любом протоколе, в котором телефон переходит в состояние idle). Эта ошибка может быть обнаружена только при рассмотрении трасс, в которых телефон по крайней мере два раза попадает в состояние idle. Другой критерий тестирования требует, чтобы любое возможное сочетание пар базовых протоколов входило в некоторую трассу. Такой критерий позволит обнаружить большее количество ошибок по сравнению с первым.

К сожалению, проблема построения полного покрытия требований для произвольных систем однородных К-протоколов оказывается алгоритмически неразрешимой. Действительно, для любой машины Тьюринга не трудно построить систему из двух агентов, которая ее моделирует. Поэтому, чтобы избавиться от трудностей, связанных с неразрешимостью, необходимо накладывать те или иные ограничения на базовые протоколы. На самом деле есть достаточно сильное ограничение, которое выполняется в большинстве практически важных случаев. Оно определяется требованием ограниченности времени реакции агентов на входные воздействия и состоит в следующем. Количество сообщений от одного агента во входной очереди каждого из них ограничено некоторой (достаточно малой) константой. В примере POTS эта константа равна 1. Действительно, сеть не посылает агенту сообщений до тех пор, пока не прочтет от него очередного сообщения. Той же самой дисциплины придерживается и каждый агент по отношению к сети.

В следующем разделе рассмотрен алгоритм генерации минимального множества трасс, обеспечивающих полное покрытие всех базовых протоколов при условии ограниченности очередей и некоторых других предположениях.

6. Прямая и обратная трассовая генерация и покрытие требований

Построение трасс, покрывающих базовые протоколы для одного агента, требует исследования его взаимодействия с окружением. Для того, чтобы не привязываться к конкретному окружению, состоящему из большого количества агентов, а также не рассматривать все возможные наборы значений атрибутов, представляющих возможные начальные состояния среды, можно воспользоваться абстрактной моделью среды и символьной трассовой генерацией. Поведение агента *n* будем рассматривать как конечную транзиционную систему с

переходами, размеченными базовыми протоколами, конкретизированными по ключевому агенту именем *n*. Состояние среды будем рассматривать заданным в виде системы ограничений на значения атрибутов среды и состояния агентов. Эти ограничения заданы в виде бесквантворной формулы. В качестве элементарных высказываний используются равенства и неравенства термов, а также высказывания вида $\mathbf{In}(T\ a, z)$. Неравенства $<, \leq, ...$ могут использоваться только для линейных арифметических выражений. Термы строятся из атрибутов и символов произвольных констант, используемых для конкретизации базовых протоколов. Для унификации действий, выполняемых в соответствии с базовыми протоколами, будем рассматривать входные очереди как атрибуты среды, параметризованные именами агентов, а их изменения сделаем явными, добавив в постусловия протоколов, передающих сообщения, соответствующие присваивания.

Поскольку состояния системы определяются бесквантворными формулами, исчисление предикатов неоднозначно, переходы абстрактной модели системы, используемые для порождения трасс, определяются с помощью предикатных трансформеров [1], т.е. формульных преобразований. Их можно определить следующим образом. Пусть $b(n): \forall x(\alpha(r,x) \rightarrow < P > \beta(r,x))$ есть базовый протокол, конкретизированный именем ключевого агента *n*. Квантор общности связывает переменные типа «имя агента», символьного или арифметического типа. Конкретизируем протокол, выбирая в качестве значений переменных символы произвольных констант соответствующего типа. В качестве значений имен агентов можно использовать имена, уже присутствующие в состоянии среды, или ввести новое имя в виде произвольной константы типа «имя агента». Конкретизированный протокол

$$b(n,m): (\alpha(r,m) \rightarrow < P > \beta(r,m))$$

получается подстановкой параметров в формулы пред- и постусловий. Для К-протоколов постусловие состоит из присваиваний и предположения *T*(*n*,*s*) о новом состоянии агента. Можно предположить, что состояния агента *n* не зависят от параметров. Это не ограничивает общности, поскольку запоминание параметров можно обеспечить путем установки соответствующего атрибута агента. Изменение состояния среды определяется с помощью предикатного трансформера следующим образом. Разделим множество атрибутов $r = (r_1, r_2, ...)$, использованных в протоколе, на те, которые встречаются в левых частях присваиваний, и все остальные. Обозначим список первых $u = (u_1, u_2, ...)$ и остальных $v = (v_1, v_2, ...)$. Изменение состояния среды $\gamma(u,v)$ определяется с помощью *прямого предикатного трансформера*, который имеет следующий вид:

$$\mathbf{pt}(\gamma(u,v), (u_1 := f_1(u,v) \wedge u_2 := f_2(u,v) \wedge ...)) =$$
$$= \exists (c_1, c_2, ...)(\gamma(c_1, c_2, ..., v) \wedge (u_1 = f_1(c,v), u_2 =$$
$$= f_2(c,v), ...)) = \gamma'(u,v)$$

При вычислении предикатного трансформера формула $\gamma'(u,v)$ подвергается некоторым упрощениям, зависящим от предметной области. В частности, могут применяться известные процедуры элиминации кванторов для линейной арифметики и абстрактных типов данных, используемых при формализации требований. Если состояние среды конкретно, т.е. все атрибуты имеют конкретные значения, $\gamma'(u,v)$ получается простой подстановкой этих значений. Действительно, в этом случае

$$\gamma(u,v) = (u_1 = d_1 \wedge u_2 = d_2 \wedge ... \wedge v_1 =$$
$$= e_1 \wedge v_2 = e_2 \wedge ...)$$

и, следовательно,

$$\gamma'(u,v) = (u_1 = f_1(d,v) \wedge u_2 =$$
$$= f_2(d,v) \wedge ... \wedge v_1 = e_1 \wedge v_2 = e_2 \wedge ...).$$

Отметим важное свойство предикатного трансформера – его монотонность. Оно означает следующее: если $\gamma_1 \rightarrow \gamma_2$, то $\mathbf{pt}(\gamma_1, \beta) \rightarrow \mathbf{pt}(\gamma_2, \beta)$.

Определим теперь отношение переходов среды:

$$\frac{\gamma(u,v) \rightarrow \mathbf{pre}(b(n,m)),\ \gamma''(u,v) \rightarrow \gamma'(u,v)}{\gamma(u,v) \xrightarrow{b(n,m)} \gamma''(u,v)},$$

где $\gamma'(u,v) = \mathbf{pt}(\gamma(u,v), \mathbf{post}(b(n,m)))$. Применяется это правило следующим образом. Пусть среда находится в состоянии $\gamma(u,v)$. Выберем базовый протокол $b(n)$ и его кон-

кретизацию $b(n,m)$ таким образом, что предусловие конкретизированного протокола выполняется на состоянии $\gamma(u,v)$ ($\gamma(u,v) \to \mathbf{pre}(b(n,m))$). Вычислим значение предикатного трансформера $\gamma'(u,v)$ на состоянии среды и постусловии конкретизированного протокола, затем возьмем некоторую формулу $\gamma''(u,v)$, из которой следует полученное значение. В результате получим, что среда может переходить с помощью протокола $b(n,m)$ из состояния $\gamma(u,v)$ в новое состояние $\gamma''(u,v)$. Таким образом, при переходе среды в новое состояние оно может усиливаться. Различные стратегии ослабления зависят от приложений, но здесь можно отметить некоторые из них. Если γ' имеет вид дизъюнкции, то можно оставить только один из дизъюнктивных членов, что даст возможность применить базовый протокол, предусловие которого может быть сопоставлено с этим членом. Заметим, что каждое условие покрывает некоторое множество состояний (именно тех, на которых условие истинно), а применимость базового протокола означает его применимость на всех состояниях покрываемого множества. Усиление условия сокращает множество состояний, которые им покрываются и, следовательно, расширяет возможности применения базовых протоколов.

Определим теперь среду для генерации трасс. Она рассматривается как среда $\gamma[s]$ с погруженным в нее одним ключевым агентом. Отношение переходов в этой среде определяется следующими правилами:

$$\frac{\gamma \xrightarrow{b(n,m)} \gamma',\ s \xrightarrow{b(n,m)} s'}{\gamma[s] \xrightarrow{b(n,m)} \gamma'[s']},$$

(переход агента)

$$\frac{\gamma \xrightarrow{b(n',m)} \gamma'}{\gamma[s] \to \gamma'[s]},$$ (переход среды)

где n' – имя агента, отличное от n. Генерация трасс управляется критерием покрытия (однократное покрытие всех протоколов, покрытие всех парных сочетаний и т.д.), условием завершения и подходящими фильтрами. В условии завершения можно использовать критерий покрытия (процесс генерации завершается, если покрытие уже обеспечено). Для этой цели в состояние среды следует ввести дополнительный компонент, в котором накапливается информация о степени покрытия (например, множество уже пройденных протоколов, которое пополняется при каждом переходе). Фильтруя трассы по максимальной допустимой длине, можно получить процесс, при котором генерация происходит по дереву поведения агента, обходя его в глубину или в ширину. При этом в точках возврата при недетерминированном выборе следующего шага можно отдавать предпочтение еще не пройденным протоколам.

В общем случае сложность процесса генерации определяется недетерминизмом при выборе подходящей конкретизации протоколов и работой алгоритмов дедуктивной системы (прувер и солвер). Значительное повышение эффективности можно получить, если накладывать те или иные ограничения на базовые протоколы.

Рассмотрим, например, следующие ограничения.

- Области значений символьных атрибутов представляют собой конечные области без операций и предикатов (перечислимые типы).
- Параметризация протоколов (внешний универсальный квантор) осуществляется только по символьным переменным и именам агентов.
- Каждый протокол изменяет входную очередь одного агента и читает из своей очереди не более чем по одному разу.
- Глобальные атрибуты отсутствуют.
- Длина любой входной очереди не превосходит 1.

При этих ограничениях алгоритм генерации трасс может быть описан следующим образом.

Прямая генерация трасс.

Вход: начальное состояние s_0 ключевого агента n, ограничения (формула γ_0), определяющие начальное состояние среды (в случае отсутствия ограничений формула равна 1), критерий покрытия, условие завершения трассы и другие фильтры.

Выход: множество сгенерированных трасс, обеспечивающих максимальное покрытие требований по заданному критерию.

В процессе генерации строятся текущая история

$$\gamma_0[s_0] \xrightarrow{b_1} \gamma_1[s_1] \xrightarrow{b_2} \ldots \xrightarrow{b_k} \gamma_k[s_k]$$

и текущее состояние системы $\gamma_k[s_k]$. В текущей истории выделены состояния, отличные от текущего и называемые точками возврата. Для каждой из точек возврата задано множество всех переходов из этой точки из всех возможных, которые еще не пройдены. Кроме того, в каждый момент генерации задано множество уже построенных трасс и объект **cover**, определяющий текущее покрытие требований (например, множество уже пройденных базовых протоколов для критерия однократного прохождения). Процесс генерации завершается, если выработано условие окончания генерации или осуществлено покрытие (объект **cover** находится в состоянии завершения).

1. В качестве начальной текущей истории выбираем историю, состоящую из одного текущего состояния $\gamma_0[s_0]$. Множество пройденных трасс пусто, объект **cover** находится в начальном состоянии (нулевое покрытие), генерация не закончена. Далее выполняется цикл, включающий следующие пункты.
2. Если состояние объекта **cover** говорит о том, что покрытие осуществлено, то процесс генерации останавливается, возвращает построенные трассы и сообщает о полном покрытии.
3. Если выполняется условие завершения трассы, то трасса добавляется к множеству уже построенных трасс. Если в текущей истории есть точки возврата, то происходит «откат» к ближайшей из них, иначе вырабатывается условие завершения генерации. Цикл повторяется.
4. Если один из фильтров запрещает текущее состояние и есть точки возврата, то происходит «откат» к ближайшей из них, иначе вырабатывается условие завершения генерации. Цикл повторяется.
5. Если текущее состояние есть точка возврата, то выбирается следующий переход, история продолжается на один шаг, цикл повторяется. При этом если переход был последним, то признак точки возврата снимается.
6. Находим все переходы агента n из текущего состояния, выбирая применимые базовые протоколы и их допустимые конкретизации. Если при своем переходе агент посылает сообщение, то к состоянию среды добавляем соответствующие ограничения на очереди других агентов.
7. Если входная очередь ключевого агента пуста, то рассматриваем все допустимые переходы среды, которые изменяют входную очередь ключевого агента. Для этого, возможно, потребуется добавить дополнительные ограничения к состоянию среды. Для любого такого перехода добавляем все допустимые переходы ключевого агента и новые переходы к множеству уже построенных.
8. Если количество построенных переходов больше 1, объявляем текущее состояние точкой возврата.
9. Если в результате выполнения пп. 6 и 7 не найдено ни одного перехода, выполняем откат к точке возврата или вырабатываем признак окончания генерации, если точек возврата больше нет.
10. Если переходы есть, то выбираем один из переходов в текущем состоянии и продолжаем текущую историю на один шаг. Повторяем цикл.

В рассмотренном алгоритме дополнительных пояснений требуют п. 6 и 7. Первый соответствует правилу перехода агента, второй – правилу перехода среды.

Переход агента (вместе со средой, к которой относим заполнение его входной очереди) выполняется с помощью предикатного трансформера. Рассматриваем состояние среды γ и протокол b, параметризованный именем ключевого агента. Предполагаем, что состояние среды имеет вид конъюнкции. Если протокол параметризирован, то сначала находим значения параметров $m = (m_1, m_2, \ldots)$ такие, что формула $\gamma(u, v) \to \alpha(u, v, m)$ истинна. Для символьных параметров решения находятся пере-

бором, а для имен агентов выбираем решения из набора имен, которые участвуют в состоянии среды плюс одно новое имя. Для уменьшения перебора можно рассматривать символьные переменные типа неизвестные и откладывать присваивание им значений до тех пор, пока возможно.

После построения формулы $\gamma'(u,v)$ элиминируем кванторы существования и упрощаем полученную формулу. Затем приводим ее к дизъюнктивной нормальной форме и каждый из конъюнктов выбираем в качестве нового состояния среды.

Переходы среды выполняются так же, как и переходы агента, но выбираются только такие, которые изменяют состояние входной очереди ключевого агента. Если таковых нет, то следует рассмотреть переходы среды некоторой ограниченной длины, приводящие в конце концов к изменению входной очереди ключевого агента.

Обратная генерация трасс.

Другой подход к генерации трасс состоит в обратном прохождении трассы с помощью *обратного предикатного трансформера*, который определяется следующим образом:

$$\mathbf{pt}^{-1}(\gamma(u_1,u_2,...,v),(u_1 := f_1(u,v) \wedge u_2 := \\ := f_2(u,v) \wedge ...)) = \gamma(f_1(u,v), f_2(u,v),...,v)$$

(при тех же обозначениях, что и в определении **pt**). Связь между двумя трансформерами выражается следующим образом:

$$\gamma \Rightarrow \mathbf{pt}^{-1}(\mathbf{pt}(\gamma,\beta),\beta);$$

$$\mathbf{pt}(\mathbf{pt}^{-1}(\gamma,\beta),\beta) \Rightarrow \gamma.$$

Действительно, пользуясь очевидными сокращениями, получаем

$$\mathbf{pt}^{-1}(\mathbf{pt}(\gamma(u,v),\beta),\beta) \Leftrightarrow \mathbf{pt}^{-1}(\exists c(\gamma(c,v) \wedge u_1 = \\ = f_1(c,v) \wedge u_2 = f_2(c,v) \wedge ...),\beta) \Leftrightarrow \\ \Leftrightarrow \exists c(\gamma(c,v) \wedge f_1(u,v) = f_1(c,v) \wedge f_2(u,v) = \\ = f_2(c,v) \wedge ...);$$

$$\gamma(u,v) \Leftrightarrow \gamma(u,v) \wedge (f_1(u,v) = f_1(u,v) \wedge f_2(u,v) = \\ = f_2(u,v) \wedge ...) \Rightarrow \exists c(\gamma(c,v) \wedge f_1(u,v) = \\ = f_1(c,v) \wedge f_2(u,v) = f_2(c,v) \wedge ...) \Leftrightarrow \\ \Leftrightarrow \mathbf{pt}^{-1}(\mathbf{pt}(\gamma(u,v),\beta),\beta);$$

$$\mathbf{pt}(\mathbf{pt}^{-1}(\gamma(u,v),\beta),\beta) \Leftrightarrow \mathbf{pt}(\gamma(f_1(u,v), \\ f_2(u,v),...,v),\beta) \Leftrightarrow \exists c(\gamma(f_1(c,v), \\ f_2(c,v),...,v) \wedge u_1 = f_1(c,v) \wedge u_2 = \\ = f_2(c,v) \wedge ...) \Leftrightarrow \exists c(\gamma(u,v) \wedge u_1 = f_1(c,v) \wedge \\ \wedge u_2 = f_2(c,v) \wedge ...) \Leftrightarrow \gamma(u,v) \wedge \exists c(u_1 = \\ = f_1(c,v) \wedge u_2 = f_2(c,v) \wedge ...) \Rightarrow \gamma(u,v).$$

Обратную генерацию трасс можно получить в системе, определяемой следующими правилами:

$$\frac{\gamma \xrightarrow{b(n,m)} \gamma', s \xrightarrow{b(n,m)} s'}{\gamma'[s'] \xrightarrow{b^{-1}(n,m)} \gamma[s]},$$

$$\frac{\gamma \xrightarrow{b(n',m)} \gamma'}{\gamma'[s] \to \gamma[s]}.$$

В этой среде все стрелки перевернуты в обратном направлении, а переходы размечены теми же самыми переходами, но с отрицательным показателем, указывающим на то, что речь идет об обратных преобразованиях (от постусловий к предусловиям). При этом

$$\gamma = \mathbf{pt}^{-1}(\gamma',\beta) \wedge \mathbf{pre}(b(n,m)) \ \textbf{или}$$

$$\gamma = \mathbf{pt}^{-1}(\gamma',\beta) \wedge \mathbf{pre}(b(n',m))$$

соответственно. Условием завершения обратной генерации является переход в начальное состояние, начинать следует от состояний, в которые ведут еще не пройденные переходы.

Преимущества обратной генерации состоят в следующем:

- Обратный предикатный трансформер обычно вычисляется легче, чем прямой (подстановка вместо решения уравнений и неравенств)
- Достигнув начального состояния, получаем значения атрибутов, с которых следует начинать тестирование по данной трассе.

Другим применением обратной генерации является возможность определения начальных значений для трасс, полученных с помощью прямой генерации.

7. Факторизация пространства состояний агента

Критерия покрытия всех требований может быть недостаточно, чтобы создать исчерпывающий тестовый набор, так как он не будет содержать всех поведений и изме-

нений атрибутов и параметров. Использование более точных критериев покрытия приводит к «экспоненциальному взрыву» количества трасс, необходимых для выполнения такого критерия. Существенное уменьшение количества трасс можно получить путем факторизации пространства состояний агента (разбиения этого пространства на классы), построения полного набора трасс внутри каждого из классов, а затем связывания классов с начальными и заключительными состояниями системы небольшим количеством трасс.

Разбиение пространства состояний агента на классы может происходить различными способами. Один из практических подходов состоит в том, чтобы строить разбиение в соответствии с классификацией требований по функциональностям. Это разбиение фактически может быть определено структурой разделов исходной документации, которая используется при формализации требований. Другой подход заключается в том, чтобы осуществлять разбиение пространства состояний по количеству связей, концентрируя в разных классах состояния с большим количеством связей и уменьшая их между классами.

Факторизация пространства состояний агента индуцирует факторизацию состояний всей системы на классы, каждый из которых состоит из достижимых состояний, для которых состояние заданного (ключевого) агента находится в одном и том же классе состояний этого агента.

Снова для генерации трасс будем проводить символьное моделирование, рассматривая состояния системы заданными в виде $\gamma[s]$, где γ – формула, ограничивающая атрибуты, а s – состояние агента. Абстрактные состояния такого типа разбиваются на классы по принадлежности определенному классу состояния ключевого агента. В каждом классе абстрактных состояний выделим в качестве начальных те, в которые можно попасть из других классов, а в качестве заключительных такие, из которых возможны переходы в другие классы. Более точно, пусть $s \xrightarrow{b(n)} s'$, где s и s' – состояния агентов, которые принадлежат разным классам, $b(n): \forall m(\alpha(r,m) \rightarrow < P > \beta(r,m))$ – базовый протокол, параметризованный именем ключевого агента. $\exists m\, \alpha(r,m)[s]$ будем рассматривать как заключительное состояние класса, которому принадлежит s, а $\exists m\, \mathbf{pt}(\alpha(r,m), \beta(r,m))[s']$ – как начальное состояние класса, которому принадлежит s'. Для того чтобы уменьшить количество начальных и заключительных состояний в классе, воспользуемся следующим приемом. Пусть $\gamma_1[s], \gamma_2[s], ...$ – все начальные (заключительные) состояния с одним и тем же состоянием s ключевого агента. Заменим их одним $(\gamma_1 \vee \gamma_2 \vee ...)[s]$. Теперь для каждого состояния ключевого агента будет только одно начальное состояние системы в классе.

Перейдем к построению тестовых трасс. Выберем начальное состояние системы $\gamma_0[s_0]$. Класс K_0, содержащий его, объявим начальным классом и добавим это состояние к начальным состояниям класса, определенным выше. В качестве критерия покрытия требований выберем критерий однократного прохождения всех базовых протоколов (независимо от конкретизации). В каждом классе с помощью обратного символьного моделирования построим множество трасс соединяющих начальные и заключительные состояния класса, так, чтобы обеспечить покрытие всех требований внутри класса. Естественно, что все трассы построить невозможно, поскольку могут быть циклы. Поэтому накладываем некоторые ограничения с помощью фильтров, а также на повторяемость цикла. Заключительные состояния, из которых начальные оказались недостижимыми, отбрасываем вместе со всеми переходами, ведущими в них, отмечая наличие таких состояний как предупреждение о возможной ошибке.

Для каждой трассы внутри класса обратное моделирование дает возможность усилить начальное состояние этой трассы и, проходя повторно ту же самую трассу в прямом направлении, усилить также и заключительное состояние этой трассы с помощью предикатных трансформеров. Пусть $\gamma_0[s_0] \xrightarrow{b_1} \gamma_1[s_1] \xrightarrow{b_2} ... \xrightarrow{b_k} \gamma_k[s_k]$ – история обратного прохождения трассы $(b_1, b_2, ..., b_k)$ из заключительного состояния

$\gamma_k[s_k]$ в начальное $\gamma_0[s_0]$. Переход из $\gamma_{i+1}[s_{i+1}]$ в $\gamma_i[s_i]$ выполняется с помощью обратного трансформера, а применение прямого трансформера усиливает условие γ_{i+1}.

Определим на множестве классов отношение переходов, полагая, что $K \xrightarrow{b(n)} K'$, если для некоторого заключительного состояния $s \in K$ класса K существует начальное состояние $s' \in K'$ такое, что $s \xrightarrow{b(n)} s'$. Если в этой системе есть классы, не достижимые из K_0, то отбросим их вместе с переходами, сгенерировав сообщение о возможной ошибке. Поскольку классы определяются состояниями ключевых агентов, то протоколы, соединяющие разные классы, не входят в множество протоколов, действующих внутри классов. Поэтому для каждого класса *K* продолжим все трассы, построенные для него, на один шаг, добавляя соответствующие протоколы, ведущие в соседний класс. Теперь для каждого класса *K* задано множество трасс, соединяющих начальные состояния этого класса с начальными состояниями всех смежных, и для каждого начального состояния $\gamma[s]$ любого из классов *K*, достижимых из K_0, можно построить историю, соединяющую начальное состояние системы с состоянием $\gamma[s]$. Действительно, по определению, для состояния $\gamma[s]$ найдется заключительное состояние $\gamma'[s']$ в некотором смежном классе K_n такое, что $\gamma'[s'] \xrightarrow{b} \gamma[s]$. Поскольку K_n достижим из начального состояния, можно построить историю, соединяющую K_0 с *K*:

$$K_0 \xrightarrow{b_1} K_1 \xrightarrow{b_2} \ldots \xrightarrow{b_k} K_n \xrightarrow{b} K .$$

Далее, выбирая подходящие истории внутри классов, получим историю, соединяющую $\gamma_0[s_0]$ и $\gamma[s]$. Теперь для каждого класса возьмем все построенные для него трассы, соединим их с начальным состоянием и получим набор трасс, обеспечивающих однократное покрытие. Количество трасс в наборе не будет превосходить суммы количества трасс в каждом классе.

Заключение

Описанная технология была успешно применена в проекте VRS на примерах индустриальных проектов. Входом для первого этапа технологической цепочки являлась документация, в которой требования были выражены на естественном языке. Необходимо было формализовать их и представить в виде базовых протоколов посредством MSC-диаграмм.

Создание базовых протоколов осуществлялось через специальную оболочку для их ввода. Сначала определялись агенты и их параметры в виде описания среды. Синтаксис грамматики представления агентов проверялся специальным валидатором, что уменьшало появление ошибок на этапе верификации требований. Базовые протоколы вводились в графическом виде. После того как данные были подготовлены, на втором этапе проводилась проверка транзиционной непротиворечивости, полноты и достижимости требований. При проверке выдавались трассы с указанием ошибок и неточностей в требованиях. После отладки требований на третьем этапе выполнялась генерация трасс, предназначенных для тестирования. Полученные трассы использовались в тестировании уже готовой системы, созданной на основе требований.

Благодарности Ю.В. Капитоновой, А.А. Летичевскому, В.П. Котлярову и А.Б. Годлевскому за ценные советы и помощь в подготовке материала.

1. *International* Telecommunications Union. Recommendation Z. 120–Message Sequence Chart (MSC). –1999.
2. *International* Telecommunications Union. Recommendation Z. 100–Specification and Description Language (SDL). –1999.
3. *Object* Management Group. Unified Modeling Language Specification, 2.0. –2003.
4. *Спецификация* систем с помощью базовых протоколов / А.А. Летичевский, Ю.В. Капитонова, А.А. Летичевский (мл.) и др. // Кибернетика и системный анализ. – 2005. – №4. – С.3–21.
5. *Basic* Protocols, Message Sequence Charts, and theVerification of Requirements Specifications / A.A. Letichevsky, J.K. Kapitonova, A.A. Letichevsky Jr. et all.// Proc. Intern. Workshop, WITUL'04, Rennes (France). – 2004. – P.30–38.
6. *Leveraging* UML to deliver correct telecom applications in UML for Real: Design of Embedded Real-Time Systems / S. Baranov, C. Jervis, V. Kot-

lyarov, A. Letichevsky, T. Weigert // L.Lavagno, G. Martin, and B. Selic, Kluwer Academic Publ. – 2003. – P.323–342.

7. *Инсерционное* программирование / А.А. Летичевский, Ю.В. Капитонова, А.А. Летичевский (мл.) и др. // Кибернетика и системный анализ. – 2003. – №1. – С.19–32.

Получено 02.06.05

Об авторе

Летичевский Александр Александрович
ведущий программист

Место работы автора:
ООО «Информационные Программные Системы»
03680, Киев, ул. Боженко, 15
Тел. 490 5424 (раб.), 257 6788 (дом.)
E-mail: lit@iss.org.ua