Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Е. В. Окунишникова

МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКИХ КОНСТРУКЦИЙ ЯЗЫКА ESTELLE ПОСРЕДСТВОМ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ

Препринт
78

Новосибирск 2000

Работа посвящена исследованию проблемы автоматического построения сетевых моделей динамических Estelle-спецификаций распределенных систем. В качестве моделей выбраны раскрашенные сети Йенсена, обогащенные приоритетами. В работе представлен метод трансляции динамических конструкций Estelle-систем в данную сетевую модель.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

E. V. Okunishnikova

COLOURED PETRI NETS APPROACH TO THE VALIDATION OF DYNAMIC ESTELLE-SPECIFICATIONS

Preprint
78

Novosibirsk 2000

This work is concerned to research dealing with automated constructing net models of dynamic Estelle specifications. Coloured Petri nets introduced by Jensen are used as the net models. Describes a method of translating the dynamic construction of Estelle specification into the net models.

ВВЕДЕНИЕ

В настоящее время не вызывает сомнений важность использования формальных методов для спецификации и верификации распределенных систем, таких как, например, коммуникационные протоколы. Для описания последних были созданы языки выполнимых спецификаций Estelle, Lotos и SDL, принятые в качестве стандартов [1–4]. Однако методы анализа для этих языков разработаны не столь хорошо, как для формальных моделей конечных автоматов или сетей Петри и их обобщений. Поэтому зачастую для верификации выполнимые спецификации транслируются в модели, для которых разработаны эффективные методы анализа и/или существуют средства автоматической верификации.

Для Estelle-спецификаций предложен метод автоматического построения конечно-автоматных моделей посредством исчерпывающей симуляции [6]. Известны также методы трансляции Estelle-спецификаций в сети Петри, причем используются как ординарные [7], так и сети Петри высокого уровня (так называемые нумерические) [8]. Методы трансляции SDL-спецификаций в обобщенные сети Петри, такие как SDL- и Pr/T-сети (от английского — predicate/transition), описаны в работах [10–12].

В ИСИ СО РАН ведется работа по отображению в раскрашенные сети двух языков: Estelle и SDL. Для Estelle в работе [13] предложен алгоритм трансляции Estelle-спецификаций с задержками и приоритетами, но без динамических конструкций. На базе этого алгоритма реализована экспериментальная система EPV, которая позволяет автоматически транслировать Estelle-спецификации, имеющие один уровень иерархии модулей, в модифицированные раскрашенные сети и проводить их симуляцию [13–15]. Для SDL разработан алгоритм их трансляции в раскрашенные сети [16, 17], и ведется работа по реализации системы автоматической трансляции.

Настоящая работа является логическим продолжением [13] и предлагает способ перевода в иерархические раскрашенные сети Estelle-спецификаций, содержащих динамические конструкции. Как и в работе [13], в качестве базовой сетевой модели используются раскрашенные сети, предложенные Йенсеном [19]. Эта модель расширена приоритетами [5]. Однако для представления времени вместо временного механизма, предложенного Мерлином [22], используется механизм, основанный на понятиях глобальных часов и временных штампов фишек [20]. Выбор

этого механизма обусловлен тем, что именно он реализован в системе Design/CPN [21, 23], которую в дальнейшем предполагается использовать для симуляции и анализа раскрашенных сетей, моделирующих Estelle-спцификации. Кроме того, временные штампы позволяют различать фишки не только по цвету, но и по времени их создания, что оказывается важным при моделировании динамических конструкций Estelle. Моделирование некоторых конструкций языка Estelle, описанное в данной работе кратко, можно найти в [13].

Данная работа состоит из трех разделов. Первые два содержат описание раскрашенных сетей Петри и базовые понятия языка Estelle. В третьем разделе изложены основные принципы трансляции Estelle-спецификаций, содержащих динамические конструкции.

1. СЕТИ ПЕТРИ

1.1. Ординарные сети Петри

Напомним, что обыкновенную, или *ординарную*, сеть Петри можно определить как размеченный ориентированный граф с вершинами двух типов: *местами* и *переходами*, соединенными дугами таким образом, что каждая дуга соединяет вершины различных типов [5]. Для изображения перехода используется, как правило, прямоугольник или барьер (вертикальная черта), места — окружность, а дуги — направленная стрелка. Места помечаются целыми неотрицательными числами (*разметка* места). В графическом представлении сети разметка изображаются соответствующим числом точек — *фишек* — в месте.

Вершина x называется *входной* для вершины y, если в сети существует дуга, ведущая от вершины x к вершине y. Аналогично, вершина x называется *выходной* для вершины y, если в сети существует дуга, ведущая от вершины y к вершине x. Кроме того, будем называть дуги, ведущие к вершине x и от нее, соответственно *входными* и *выходными* дугами этой вершины, а все вершины, связанные с x дугами, — ее *окружением*.

Сеть Петри функционирует, переходя от разметки к разметке. Функционирование начинается при заданной начальной разметке. Смена разметок происходит в результате срабатывания одного из переходов. Переход может сработать при некоторой разметке, если все входные места перехода содержат хотя бы по одной фишке. Срабатывание перехода изымает по фишке из каждого входного места перехода и по-

мещает по фишке в каждое его выходное место.

Таким образом, сеть Петри моделирует некоторую систему и динамику ее функционирования. При этом места и находящиеся в них фишки представляют состояние моделируемой системы, а переходы — изменение ее состояний.

1.2. Раскрашенные сети Петри

Неиерархические раскрашенные сети Петри являются расширением ординарных сетей Петри, предложенным Йенсеном [18, 19]. В отличие от ординарных сетей Петри, где все фишки считаются неотличимыми друг от друга, каждая фишка в раскрашенной сети обладает индивидуальностью — значением некоторого типа, которое по традиции называется *цветом*. Функционирование раскрашенной сети зависит не только от наличия фишек во входных местах переходов, но и от их цвета.

Неиерархическая раскрашенная сеть состоит из трех частей: структуры сети (которая, по существу, не отличается от структуры ординарной сети), деклараций и пометки сети.

1.2.1. Структура иерархической раскрашенной сети

Иерархическая раскрашенная сеть — это композиция множества неиерархических сетей, называемых *страницами*. Страницы могут содержать вершины специального типа, которые называются *модулями* и соединяются с местами на странице по тому же принципу, что и переходы.

Модуль представляет подсеть, располагающуюся на отдельной странице, которая в свою очередь может содержать модули. Такая страница называется *подстраницей* страницы, на которой располагается модуль. Подстраница содержит копии всех мест, с которыми связан модуль. Место-копия может быть входным для некоторого перехода или модуля на подстранице тогда и только тогда, когда его *прототип* является входным местом для модуля, представляющего подстраницу. Аналогично только копия выходного места-прототипа может быть выходным местом некоторого перехода или модуля на подстранице.

Поведение иерархической сети эквивалентно поведению неиерархической сети, получающейся при замещении всех модулей страницами, которые они представляют. При этом каждый модуль вместе со своими дугами удаляется со страницы, а на его место помещается подсеть, располагавшаяся на подстранице. Соединение сетей происходит по местам:

каждое место-прототип склеивается со всеми своими копиями.

1.2.2. Декларации раскрашенной сети

Декларации состоят из описания множеств цветов (типов) и объявления переменных, каждая из которых принимает значения из некоторого множества цветов. Декларации также могут содержать определение операций и функций. Располагаются декларации, как правило, в верхней части страницы в прямоугольнике, ограниченном пунктирной линией. В иерархических раскрашенных сетях декларации множеств цветов и переменных, общих для всех страниц, часто выносят на отдельную страницу.

В раскрашенных сетях определено четыре базовых множества цветов, соответствующих стандартным типам целый, вещественный, строковый и булевский. Базовым также является специальное множество цветов, состоящее из одного элемента. Множество цветов может описываться путем перечисления всех возможных значений, осуществляемым двумя различными способами:

1) перечисление относительно небольшого числа объектов, каждый из которых обладает уникальным именем:

$$color\ RGB\ =\ with\ red\,|\,green\,|\,blue;$$

2) перечисление класса объектов с общим именем и индивидуальными номерами для отдельных элементов класса:

$$color\ AA\ =\ a\ with\ 1..5.$$

В раскрашенных сетях также имеется несколько механизмов, обеспечивающих возможность конструирования нового множества цветов из уже продекларированных множеств. При трансляции Estelle-спецификаций мы в основном будем использовать два — *product* и *list*. Декларация

$$color\ PP\ =\ product\ AA\ *\ BB\ *\ CC$$

определяет множество цветов, состоящее из всех троек вида (a, b, c), где $a \in AA, b \in BB$ и $c \in CC$. Элементами множества цветов LL, согласно декларации

$$color\ LL\ =\ list\ AA,$$

являются все списки, состоящие из элементов множества AA. Значения последнего множества цветов представляются как $[v_1, v_2, ..., v_n]$ либо в

виде выражения $h :: t$, где h — первый элемент, или *голова*, списка, а t — список без первого элемента, или *хвост* списка. Для пустого списка используются обозначения [ ] или nil.

Наконец, новые множества цветов могут конструироваться путем выделения подмножества ранее определенного множества. Так, декларация

$$color\ LL\ =\ list\ AA\ with\ 3..5$$

описывает списки длиной от 3 до 5 элементов.

В раскрашенных сетях для базовых множеств цветов существуют стандартные операции и функции, применимые к элементам множества, которые не требуется явно описывать в декларациях. Например, из стандартных операций над списками будем использовать следующие:

$l_1\ \hat{}\hat{}\ l_2$	— конкатенация двух списков,
$hd\, l$	— голова списка,
$tl\, l$	— хвост списка,
$length\, l$	— длина списка,
$nth(l, n)$	— n-й элемент списка
$null\, l$	— истина, если список пустой.

1.2.3. Пометка раскрашенной сети

Пометка сети приписывается месту, переходу либо дуге и описывает правила распределения и перемещения фишек по сети. Каждое место имеет три разных типа пометок: имя места, множество цветов и инициализирующее выражение. 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 начальную разметку места. Переходы имеют два типа пометок: имена и спусковые функции, а дуги — один тип: выражения. Спусковая функция перехода — логическое выражение, которое должно быть выполнено до того, как переход сможет сработать. Выражения на дугах могут содержать переменные, константы, функции и операции, определенные в декларациях. Все переменные, входящие в спусковую функцию перехода и выражения на связанных с ним дугах, будем называть *переменными перехода*.

Пример раскрашенной сети приведен на рис. 1. Сеть состоит из перехода $trans$ и четырех мест: $State$, $counter$, i и j. Все места, за исключени-

ем i, являются как входными, так и выходными местами перехода $trans$. Место i — только входное. Декларации сети содержат определения двух множеств цветов и декларации переменных, входящих в выражения на дугах. Рядом с каждым местом приведены имя места, связанное с ним множество цветов и начальная разметка места. Начальная разметка вида $\underline{n`p}$ означает, что в месте содержится n фишек со значением p. Над переходом указаны его имя и спусковая функция.

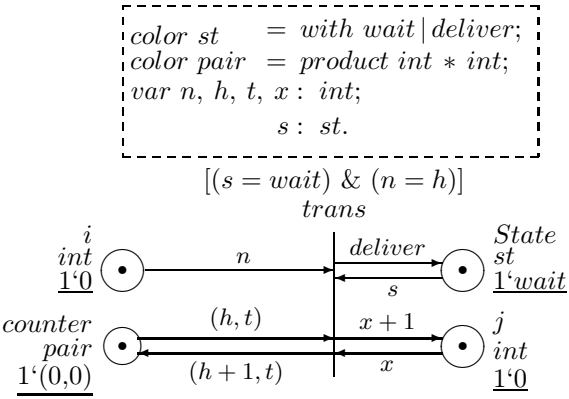

Рис. 1. Пример раскрашенной сети

Далее в работе в большинстве случаев декларации опускаются. Множества цветов, приписанные местам, описываются в тексте; для переменных, входящих в выражение на дуге, предполагается, что они принимают значения из множества цветов, приписанного месту, с которым связана дуга.

1.2.4. Правила функционирования раскрашенных сетей

Функционирование раскрашенной сети отличается от функционирования обыкновенной сети тем, что возможность срабатывания перехода зависит не только от наличия фишек во входных местах перехода, но и от их значений. Чтобы говорить о срабатывании перехода, необходимо определить значения переменных перехода. При этом все вхождения одной и той же переменной замещаются одним и тем же значением. Набор значений переменных, при которых выполнена спусковая функция пе-

рехода, называется *связыванием*. Значение выражения на дуге при выбранных значениях переменных определяет фишку (мультимножество фишек в общем случае), которая может быть "перемещена" по этой дуге. Другими словами, значение выражения на входной дуге определяет, сколько и каких фишек должно содержаться в соответствующем входном месте 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.

Переход раскрашенной сети *возможен*, если можно выбрать такие значения переменных перехода, что в каждом входном месте перехода имеется фишка, определенная значением выражения на соответствующей входной дуге перехода. Возможный переход может сработать. Срабатывание перехода изымает фишки из его входных мест и добавляет в выходные места. Количество и цвет изымаемых/добавляемых фишек определяются выражениями на соответствующих дугах.

В сети на рис. 1 переход *trans* может сработать при начальной разметке. Все входные места перехода содержат по фишке. Значения $s = wait$, $n = 0$, $h = 0$, $t = 0$, $x = 0$ определяют связывание. После срабатывания перехода *trans* место i станет пустым, место *State* будет содержать фишку со значением *deliver*, место *counter* — фишку со значением $(1, 0)$, а место j — фишку со значением 1.

1.3. Временные сети Йенсена

Для моделирования систем, поведение которых явно зависит от времени, предложено множество временных расширений сетей Петри, сильно отличающихся между собой по существу и сложности. Общим для них является наличие внешнего времени, которое наряду с разметкой сети определяет возможность срабатывания переходов.

В работе [20] предложен временной механизм, основанный на понятии *глобальных часов*. Значение часов представляет текущее время в модели, или *модельное время*. Начальное модельное время, т. е. значение часов, при котором сеть начинает функционировать, задается при описании сети. Множества цветов (все или часть) получают признак *timed*. Фишка, которая принимает значения из множества, обладающего признаком *timed*, в дополнение к основному цвету несет временное значение. Это значение, также называемое *временным штампом*, определяет момент времени, раньше которого фишка не может использо-

ваться при срабатывании какого-либо перехода. Фишки, принимающие значения из множеств, не имеющих признака *timed*, временного штампа не несут. Последнее означает, что фишка готова к использованию с момента своего создания.

Временной штамп новой фишки вычисляется как текущее модельное время плюс задержка, величина которой определяется выражением $@+D$. Это выражение называется также *временной пометкой* и может быть связано с переходом и/или с выходной дугой. Если временная пометка связана только с переходом, то во все выходные места перехода помещаются фишки с одинаковыми временными штампами. Временная пометка на дуге определяет значение временного штампа только для фишки, помещаемой в место, для которого данная дуга — входная. Если временная пометка присутствует и на переходе, и на дуге, то значения задержек суммируются.

С помощью временной пометки задаются различные типы задержек. Значение задержки может быть константным, зависеть от связывания, при котором произошло срабатывание, выбираться случайным образом из заданного интервала и т. п. В системе Design/CPN реализовано несколько стандартных статистических функций, которые можно использовать во временной пометке сети [23].

Временной механизм влияет на правило выбора переходов, которые имеют право сработать. Будем называть раскрашенную сеть, которая получается из временной удалением всех временных конструкций, *базовой* для временной сети. Во временных раскрашенных сетях переход называется *возможным по цвету* при выбранном связывании, если он возможен при этом связывании в базовой сети. Однако чтобы возможный по цвету переход мог сработать, временные штампы фишек, необходимых для выбранного связывания, должны быть не меньше, чем текущее модельное время. Переходы, для которых выполнено последнее условие, будем называть *готовыми*. Срабатывание перехода происходит мгновенно.

Текущее модельное время не изменяется до тех пор, пока остаются готовые переходы. Когда в сети остаются только возможные по цвету переходы, происходит изменение значения глобальных часов. При этом новым значением часов будет ближайший момент времени, в который какой-либо из возможных переходов становится готовым.

При описанном подходе фишка становится доступна одновременно для всех переходов, для которых место, содержащее данную фишку, яв-

ляется входным, что создает ряд неудобств при моделировании остановки таймеров до истечения отсчитываемого интервала времени. В работе [23] для моделирования остановки таймеров предложено использовать временную пометку @*ignore* на входных дугах переходов. Такая пометка означает, что при определении готовности перехода временные штампы фишек в соответствующем входном месте игнорируются. Таким образом, один из переходов может использовать фишку раньше, чем она станет доступна остальным переходам.

2. ОБЗОР ЯЗЫКА ESTELLE

Язык формальных описаний Estelle [1, 2, 4] основан на модели расширенного конечного автомата. Estelle-спецификация описывает иерархически структурированную систему недетерминированных компонент, взаимодействующих с помощью сообщений (*примитивов взаимодействий*) по двунаправленным каналам между портами (*точками взаимодействия*). Каждая компонента есть экземпляр модуля. Иерархия компонент и структура связей может изменяться в процессе функционирования системы.

2.1. Понятие модуля

Модуль определяется заголовком и телом, связанным с заголовком. Тело модуля может включать в себя описания других модулей, называемых *наследниками* данного модуля, которые, в свою очередь, могут содержать описания модулей. Охватывающий модуль называется *родителем* модулей, непосредственно описанных в его теле. Модуль, охватывающий все прочие модули системы, называется *спецификацией* системы. В процессе выполнения Estelle-спецификации одновременно может существовать несколько *экземпляров* модуля, которые создаются как статически — при инициализации всей системы, так и динамически.

С внешней точки зрения экземпляр модуля представляет собой "черный ящик". Взаимодействие с ним происходит через конечное число точек взаимодействия и экспортируемые переменные, доступ к которым имеет только экземпляр родительского модуля. Все экземпляры одного модуля обладают одинаковыми внешними признаками, характеристики которых определяются заголовком модуля, где описаны класс данного модуля, точки взаимодействия, экспортируемые переменные и формальные параметры. Фактически можно считать, что заголовок мо-

дуля определяет тип модулей, а идентификатор, присутствующий в заголовке, можно расценивать как идентификатор данного типа модулей в обычном для языка Pascal смысле.

Внутреннее поведение модуля определяется его телом, содержащим следующие разделы: деклараций, инициализации и описания переходов. Для модуля может быть описано несколько разных тел, т. е. поведение различных экземпляров одного и того же модуля может различаться. Выбор определенного тела для экземпляра модуля происходит при его создании.

В *разделе деклараций* содержатся описания констант, типов, переменных, процедур и функций, а также описания специфических объектов Estelle, таких как каналы, модули и модульные переменные, управляющие (локальные) состояния и точки взаимодействия модуля. Описание канала определяет две роли — по одной на каждое окончание канала, — которые должны играть модули при взаимодействии по этому каналу. Роли называются *оппозитными* по отношению друг к другу. С каждой ролью связан набор примитивов взаимодействий, которые передает модуль, играющий данную роль. Взаимодействие сопровождается списком формальных параметров. Модульные переменные указывают на конкретные экземпляры модулей и позволяют различать их.

С каждой точкой взаимодействия ассоциирована неограниченная FIFO-очередь, в которую поступают все сообщения, полученные экземпляром модуля через эту точку взаимодействия. Несколько точек взаимодействия могут делить между собой одну и ту же очередь. Описание точки взаимодействия определяет, каналом какого типа она может быть связана с другой точкой, роль, которую играет модуль при взаимодействиях через данную точку, и дисциплину — индивидуальную или общую — очереди, ассоциированной с данной точкой. Описание канала дается в родительском модуле. Там же определяется, с какой точкой взаимодействия и какого экземпляра модуля связана данная точка.

В Estelle существует два типа связей между точками взаимодействия, которые определяются операторами установления связи `connect` и `attach`. Первый оператор служит для установления канала между внешними точками взаимодействия модулей одного уровня иерархии или между внутренней точкой взаимодействия охватывающего модуля и внешней точкой одного из его наследников. Связь такого типа будем называть *соединением* двух точек. Второй оператор позволяет установить связь между внешними точками взаимодействия охватывающего

модуля и одного из его наследников. Такой тип связи называется *прикреплением* и используется для организации обмена сообщениями между модулями, находящимися на разных уровнях иерархии. Прикрепление "совмещает" точки взаимодействия. При наличии нескольких точек, последовательно прикрепленных друг к другу, сообщения перемещаются из конца в конец цепочки прикреплений, как если бы промежуточные точки отсутствовали и существовал прямой канал, связывающий первую и последнюю точки цепочки.

Цепочки прикреплений используются для создания *линий связи* между модулями, которые принадлежат разным ветвям дерева иерархии модулей. Линия связи создается в точности одним оператором `connect`, который соединяет точки взаимодействия двух модулей, лежащих на одном уровне, и, возможно, двух цепочек прикрепления. В результате образуется канал связи между модулями, точки взаимодействия которых являются концами линии связи. Когда сообщение посылается через точку взаимодействия, находящуюся на одном конце линии связи, оно попадает в очередь сообщений точки взаимодействия, находящейся на противоположном конце линии связи.

Раздел инициализации определяет значения переменных модуля, с которыми каждый созданный экземпляр данного модуля начинает свое выполнение. Инициализация модульных переменных приводит к созданию экземпляров модулей-наследников. Выполнение операторов установления связей приводит к созданию каналов между точками взаимодействия.

И наконец, *раздел описания переходов* описывает действия модуля в терминах системы состояний. Контрольные состояния системы определяются управляющим состоянием экземпляра модуля, очередями всех точек взаимодействия, значениями переменных, текущего множества наследников модуля (если есть) и текущей структурой связей между ними. Начальное состояние определяется разделом инициализации. Изменение контрольных состояний происходит при выполнении действий, описанных в переходах.

В модуле может быть не описано ни одного перехода. Такой модуль называется *неактивным*. После инициализации неактивный модуль не выполняет ни одного действия, тогда как его наследники могут их выполнять. Модуль с непустым набором переходов называется *активным*. Каждый переход характеризуется *условием возможности* и *действием*. Условие возможности перехода включает текущее со-

стояние (приставка `from`), входное взаимодействие (приставка `when`), предикат возможности (приставка `provided`), приоритет перехода (приставка `priority`) и условие задержки (приставка `delay`). Переход может осуществиться, если выполнены все части его условия возможности.

Действие перехода содержит оператор `to` и блок перехода, состоящий из операторов языка Паскаль с некоторыми ограничениями (например, не используются стандартные операторы `write` и `read`) и специальных операторов Estelle. Выполнение перехода рассматривается как атомарное действие. Начавшееся выполнение перехода не может быть прервано.

2.2. Принцип структуризации и шаг выполнения

Согласно концепциям Estelle описываемая система определяется как множество взаимодействующих модулей. Текстуальная вложенность описаний модулей определяет дерево иерархии модулей. Поведение каждого экземпляра модуля в спецификации зависит от его положения в иерархии модулей и класса, приписанного модулю.

Каждый модуль может иметь класс: *системный процесс, системная активность, процесс* или *активность*. Системные процессы и активности называют также *системными модулями*. Множество экземпляров модулей, описания которых заключены в описании системного модуля, называется *подсистемой*. Классификация модулей подчиняется следующим требованиям:

– каждый активный модуль должен иметь определенный класс;

– системный модуль не может быть вложен в активный;

– каждый модуль класса “процесс” или “активность” должен быть вложен в системный модуль;

– (системный) процесс может включать в себя описания процессов и активностей, а (системная) активность — только описания активностей.

Таким образом, существует единственный уровень системных модулей и модули, охватывающие системные, — неактивны. Допускается существование нескольких экземпляров одного и того же системного модуля. Количество системных модулей и структура связей между ними описываются в охватывающем неактивном модуле и, следовательно, не меняются в течение всего времени выполнения спецификации.

Поведение системных модулей полностью асинхронно. Внутри подсистем оно синхронизируется экземплярами родительских модулей. Каждый экземпляр модуля предлагает один из своих готовых к выпол-

нению переходов модулю-родителю. Переходы модуля-родителя имеют приоритет над переходами наследников: если модуль-родитель предлагает переход к выполнению, то он будет выполняться на следующем такте. Выполнение перехода родителя исключает выполнение переходов всех (не только непосредственных) наследников. Классы "процесс" и "активность" определяют два возможных способа выполнения: параллельное и недетерминированное. Если родитель не предлагает переходов к выполнению и имеет класс "процесс", то на следующем такте будут параллельно выполнены все предложенные наследниками переходы. Если родитель — "активность", то на следующем такте выполняется только один из предложенных наследниками переходов. Выбор перехода для выполнения осуществляется недетерминированно. Такт вычисления в подсистеме завершается, когда выполнены все переходы, предложенные к выполнению. Такты чередуются с фазами управления, в ходе которых после завершения очередного шага вычислений осуществляется проверка условий возможности всех переходов.

Ниже приведена Estelle-спецификация `example`, в которой описаны канал и два типа модулей: `sender` и `receiver`. Каждый из модулей имеет две точки взаимодействия, с которыми ассоциированы общая очередь. Модуль `sender` передает сообщения, нумеруя их натуральными числами. Отправление следующего сообщения происходит после того, как получено подтверждение удачной передачи предыдущего сообщения. Модуль `receiver` принимает сообщения от модуля `sender` и сразу передает подтверждение, содержащее номер принятого сообщения. Раздел инициализации спецификации `example` определяет систему, которая содержит по одному экземпляру `sender` и `receiver`. Точка взаимодействия `s1` модуля `sender` соединена каналом с точкой взаимодействия `r1` модуля `receiver`, а точка `s2` — с точкой `r2`.

```
specification example;
channel CONN(ROLE1,ROLE2);
by ROLE1, ROLE2: mess(i:integer);

module sender systemprocess;
 ip s1:CONN(ROLE2) common queue;
    s2:CONN(ROLE1) common queue;
end;
```

```
body send for sender;
  state  wait, deliver;
  var j : integer;
    initialize to deliver
     begin  j:=0  end;

trans (* transmission of a new message *)
      from deliver to wait
      begin  output s2.mess(j) end;
trans (* reception of acknowledgement *)
      when s1.mess(i)
      provided (i=j)
      from wait to deliver
      begin j:=j+1 end;
end;

module receiver systemprocess;
 ip r1:CONN(ROLE1) common queue;
    r2:CONN(ROLE2) common queue;
end;

body reception for receiver;
var  j:integer;
initialize  begin j:=0 end;

trans (* reception of message and transmission *)
      (* of aknowledgement *)
     when r2.mess(i)
     provided (i=j)
     begin output r1.mess(j); j:=j+1 end;
end;

modvar  ms: sender;
        mr: receiver;

initialize  begin
   init ms with send;
   init mr with reception;
```

```
   connect ms.s1 to mr.r1;
   connect ms.s2 to mr.r2;
end;
end.
```

2.3. Концепция времени в Estelle

Вычислительная модель Estelle описана в терминах, не зависимых от времени. Одним из принципиальных предположений является то, что о времени выполнения перехода ничего не известно, так как скорость выполнения полностью зависит от реализации. Утверждается только, что вычисления происходят не моментально, а на их выполнение затрачивается некоторое время. При этом ход времени одинаков для всех экземпляров модулей. Если в системе существует несколько переходов, выполнение которых было отложено, то по завершении очередного такта вычислений задержки этих переходов уменьшатся на одну и ту же величину.

Для соотнесения процесса вычислений с течением времени используется приставка `delay`, которая может входить в условие возможности некоторых переходов. Данная приставка служит для индикации того, что выполнение перехода (если он возможен) должно быть отложено. Задержка определяется двумя временными значениями: минимальным временем, на которое должно быть отложено выполнение перехода, и максимально возможным временем задержки. Иногда считают, что с каждым переходом, имеющим приставку `delay`, связан таймер, включение и выключение которого происходит во время очередной фазы управления после проверки условия возможности перехода.

Приставка `delay` имеет три синтаксические формы: `delay(d1)`, `delay(d1,d2)` и `delay(d1,*)`. Форма `delay(d1)` семантически эквивалентна `delay(d1,d1)`, а символ `*` служит для обозначения бесконечности. Приставка `delay(d1,d2)` указывает на то, что выполнение перехода должно быть задержано не менее чем на `d1` единиц времени и не может быть задержано более чем на `d2` единиц времени с момента, как его выполнение стало возможно. Если за время, на которое отложено выполнение некоторого перехода, произошло выполнение другого перехода, которое, возможно, изменило значения переменных и локальное состояние модуля, но не нарушило условие возможности отложенного перехода, то задержка продолжает отсчитываться.

2.4. Ограничения на Estelle

В настоящее время процедура трансляции разработана только для Estelle-спецификаций, допускающих последовательное недетерминированное поведение. Другим словами, в транслируемой спецификации все системные модули должны иметь класс “системная активность” и, согласно правилам классификации модулей, все вложенные в них модули имеют класс “активность”.

Другое ограничение связано с разрывом связей, установленных с помощью оператора `attach`. В языке Estelle выполнение операции `detach` подразумевает перемещение очереди сообщений точки, которая является концом разрываемой цепочки прикреплений, к точке модуля, который выполняет операцию. В данной работе операция `detach` подменяется операцией `simple-detach`, при выполнении которой не происходит перемещения очереди сообщений. Вследствие этого ограничения стираются различия между операторами уничтожения модуля `release M` или `terminate M`, поскольку при выполнении операции `release M` прикрепления разрываются с перемещением очереди сообщений, тогда как выполнение `terminate M` предполагает использование операции `simple-detach`.

3. ТРАНСЛЯЦИЯ ESTELLE-СПЕЦИФИКАЦИИ В СЕТЬ ПЕТРИ

В работе [13] описан алгоритм трансляции Estelle-спецификаций, описывающих системы со статической иерархией модулей, в раскрашенные сети Петри, расширенные приоритетами. Кроме того, для моделирования отложенных переходов в Estelle в раскрашенные сети был перенесен временной механизм, предложенный Мерлином [22]. В данной работе мы тоже предполагаем наличие в раскрашенных сетях приоритетов переходов, но используем временной механизм Йенсена, основанный на понятиях глобальных часов и временных штампов. Отличается также подход к поэтапному построению сети. В [13] на каждом этапе трансляции часть переходов в уже построенной сети заменялась более сложными фрагментами сети. В этой работе вместо переходов, которые в дальнейшем подлежат замене, мы используем модули, а соответствующие фрагменты размещаем на страницах, связанных с этими модулями.

Поскольку некоторые термины, такие как модуль и переход, используются и Estelle и сетями, при описании алгоритма трансляции будем

использовать приставки E- и N- для обозначения соответственно объектов (модуля, перехода) Estelle и сети в тех случаях, когда значение термина не очевидно из контекста.

Способ моделирования динамических средств Estelle базируется на принципе, предложенном Лаем в работе [8]. Основанием для моделирования служит то, что текстуальная вложенность описаний модулей образует статический шаблон спецификации. Число экземпляров модуля может изменяться в процессе выполнения, но позиция в общей иерархии фиксирована и соответствует положению модуля в иерархической структуре описаний модулей. При моделировании Estelle-спецификации структура сети, статическая по своей природе, служит для представления иерархической структуры спецификации. Экземпляры модуля моделируются с помощью фишек, число которых меняется при динамическом создании или уничтожении экземпляров модулей.

Однако имеется ряд отличий между предлагаемым нами алгоритмом и способом построения нумерической сети, рассматриваемом в работе [8]. Последний требует предварительной обработки Estelle-спецификации, в процессе которой все переходы в модулях приводятся к такому виду, чтобы моделирование E-перехода требовало ровно одного N-перехода. Например, E-переход, содержащий условный оператор `if`, заменяется на два — по одному для каждой из ветвей условного оператора. При этом приставка `provided` каждого из этих новых переходов модифицируется таким образом, чтобы отражать, какая из ветвей условного оператора образует блок данного перехода. Очевидно, что такое преобразование спецификации требует дополнительных усилий. Кроме того, нет никаких гарантий, что в процессе преобразования не возникнут новые ошибки, которые отсутствовали в исходном тексте спецификации. Наш алгоритм не требует преобразования E-переходов. Вместо этого используются стандартные фрагменты сети, моделирующие выполнение таких операторов, как условный или оператор цикла [13].

Другое отличие заключается в том, что в работе [8] не рассматриваются E-переходы с задержками. Причиной этого, по всей видимости, послужило отсутствие временного механизма в автоматической системе *PROTEAN*, которая использовалась для верификации нумерических сетей, полученных при трансляции Estelle-спецификаций. В более поздних работах [9] используется другой класс сетей высокого уровня, обладающий средствами моделирования поведения, явно зависящего от

времени. Однако авторы в этих работах рассматривают не стандартный вариант Estelle, а предлагают некоторое расширение языка, приближающее Estelle к языкам реального времени.

Напомним основные принципы трансляции Estelle-спецификации в иерархические раскрашенные сети. Сеть, моделирующая спецификацию, строится с помощью последовательного уточнения. Сначала создается страница, которая соответствует основной структуре системы и содержит по одному N-модулю для каждого системного E-модуля; затем для каждого из полученных N-модулей строится дерево страниц, повторяющее иерархию подсистемы с корнем в соответствующем системном E-модуле; и, наконец, происходит трансляция E-переходов.

3.1. Отображение предопределенных типов Estelle

Для представления стандартных типов целый, булевский и вещественный используются соответствующие множества цветов: *integer*, *boolean* и *real*. Символьному типу данных сопоставляется подмножество множества цветов *string* с ограниченной длиной строк:

$$color\ char\ =\ string\ with\ 1..1.$$

Перечислимые множества цветов сопоставляются перечислимым типам и модульным переменным. Например, для представления массива модульных переменных

```
modvar X: array [1..n] of module_type
```

естественно использовать индексированное множество цветов

$$color\ module\_type\ =\ X\ with\ 1..n.$$

Из четырех базовых механизмов, которые поддерживаются в языке Estelle для обеспечения возможности использования структурированных типов данных, мы допускаем использование записей, массивов и множеств. Записи в моделирующей сети представим множествами цветов, полученных с помощью декларативной приставки *product*. Полям записи при таком представлении соответствуют элементы кортежа. Так, описание типов

```
type UserDataType = (A, B, C, Data, corr);
     PDUType = (CR, DT, CC, DR, DC);
     SDUType = record
                 PDU:   PDUType;
```

```
UData: UserDataType;
end;
```

трансформируется в следующие множества цветов:

$$color\ UserDataType\ =\ with\ A\,|\,B\,|\,C\,|\,Data\,|\,corr;$$

$$color\ PDUType\ =\ with\ CR\,|\,DT\,|\,CC\,|\,DR\,|\,DC;$$

$$color\ SDUType\ =\ product\ PDUType * UserDataType.$$

Для доступа к полям записи требуется указывать имя переменной соответствующего типа в нужной позиции кортежа: $(PDU, UData)$. Присваивание значения какому-либо полю записи в сети представляется кортежем на выходной дуге перехода, где в соответствующей этому полю позиции находится указанное значение. Например, операторам

```
Data.PDU := DT;
Data.UData := A;
```

будет соответствовать кортеж (DT, A).

Среди базовых механизмов, обеспечивающих возможность конструировать новые множества цветов из уже имеющихся, нет такого, который бы позволял перенести описания массивов в декларации сети путем таких же несложных преобразований, какие описаны для записей. Мы предлагаем использовать в качестве массивов списки той же длины, что и исходный массив. Для извлечения значений элементов массива применяется стандартная функция доступа к n-у элементу списка. Присваивание значения l i-у элементу массива m производится с помощью следующей функции, которая должна быть описана в декларациях сети:

$$WAE(i,l,m)\ =\ [\,]\ |\ WAE(i,l,head :: tail)\ =$$

$$if\ i > 1\ then\ \ [head]\hat{}\hat{}WAE(i-1,l,tail)\ \ else\ [m]\hat{}\hat{}tail.$$

3.2. Трансляция иерархии модулей

Трансляция Estelle-спецификации в иерархическую раскрашенную сеть начинается с создания дерева страниц, отражающего текстуальную вложенность описаний модулей. Корнем дерева является страница, на которой представлена общая структура спецификации.

Трансляция осуществляется единообразно для всех модулей. Сначала каждому заголовку E-модуля сопоставляется один N-модуль. Затем на странице, связанной с этим N-модулем, создается столько N-модулей,

сколько тел описано в спецификации для транслируемого E-модуля. И, наконец, на каждой странице, связанной с N-модулем для одного из тел, отображается внутренняя структура этого тела. Такая страница будет содержать N-модули, представляющие E-переходы, и N-модули, соответствующие модулям-наследникам, описанным непосредственно в этом теле. Для каждого из наследников процедура повторяется. Заметим, что если для E-модуля определено только одно тело, то излишне заводить отдельную страницу, на которой бы располагался N-модуль, соответствующий этому телу. Подсеть, которая будет построена в процессе трансляции на такой странице, в точности повторит окружение N-модуля, созданного для заголовка E-модуля.

Процесс построения дерева страниц проиллюстрируем на примере спецификации `SP`, в которой описан тип `SM` системных модулей. Тело `MS_body` включает в себя два типа наследников: `AB` и `AC`, причем для модуля `AB` описано два тела, а для `AC` — только одно.

```
specification SP;
. . .
module SM systemactivity;
ip  N : Channel_type (user)     individual queue;
    P : Channel_type (provider) individual queue;
end;{SM}

body SM_body for SM;
    module AB activity;
    ip  BN : Channel_type (user) individual queue;
    end;{AB}

    module AC activity;
    ip  CN : Channel_type (provider) individual queue;
    end;{AC}

    body B1_body for AB;
    . . .
    end;{B1_body}

    body B2_body for AB;
    . . .
    end;{B2_body}
```

```
    body C_body for AC;
    . . .
    end;{C_body}

modvar B1, B2: AB;
       C: AC;
. . .
end;{SM_body}

modvar S: SM;
. . .
end.{SP}
```

Данной спецификации будет соответствовать дерево страниц, показанное на рис. 2. Отдельные страницы изображены в виде прямоугольников, ограниченных пунктирной линией. Овалы обозначают N-модули. Стрелки демонстрируют соответствие страниц и N-модулей. На верхней странице размещается один N-модуль, соответствующий описанию системного модуля `SM`. Страница, на которой размещался бы N-модуль для тела системного модуля, опущена. N-модули AB и AC соответствуют двум типам модулей-наследников, описанным для `SM`. Подстраницы AB и AC содержат N-модули, сопоставленные описаниям тел для E-модулей `AB` и `AC`. С последними в свою очередь связаны страницы, на которых будут размещаться сети, моделирующие собственно тела E-модулей.

3.3. Идентификация экземпляров модулей

В сети, моделирующей спецификацию, структура которой не изменяется во время функционирования, каждому экземпляру модуля на начальном этапе трансляции ставился в соответствие один N-модуль независимо от положения экземпляра модуля в иерархии Estelle-спецификации. На последующих этапах трансляции происходило построение дерева страниц, корнем которого являлся этот N-модуль. В результате каждому экземпляру модуля соответствовал собственный фрагмент сети, состоящий из страниц, полученных при отображении описания этого E-модуля. Следствием такого подхода был тот факт, что все места, за исключением тех, которые представляли массивы и очереди, содержали не более одной фишки.

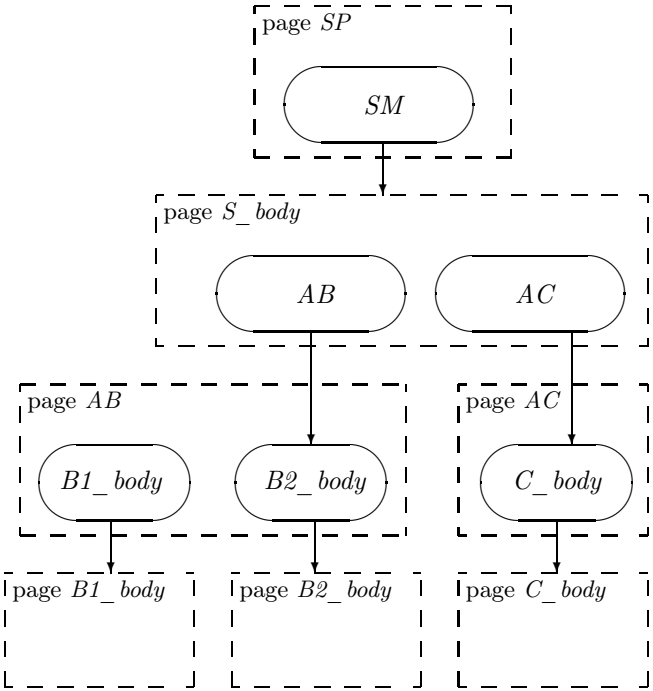

Рис. 2. Соответствие вложенности модулей и иерархии страниц

При трансляции спецификаций, содержащих динамические конструкции, структура сети в целом аналогична той, что описана в работе [13]. Существенно отличается только трансляция точек взаимодействия модулей. Однако подсеть, связанная с N-модулем, который сопоставлен заголовку некоторого E-модуля, соответствует не отдельному экземпляру, а всем модулям данного типа. Экземпляр модуля представляется разметкой этой подсети. Фишки, принадлежащие экземпляру E-модуля, при его создании помещаются в места сети и удаляются оттуда, когда экземпляр модуля прекращает свое существование. Поскольку фишки, принадлежащие экземплярам одного и того же E-модуля, располагаются в одних и тех же местах, требуется механизм, который бы позволял различать фишки, относящиеся к разным экземплярам.

На первый взгляд, для этих целей естественно использовать модульные переменные, которые служат в Estelle для идентификации экзем-

пляров модулей. Следующий пример показывает, почему это невозможно.

```
specification SP;
. . .
module SM systemactivity;
. . .
body SM_body for SM;
    module AB activity;
    ip  BN : Channel_type (user) individual queue;
    end;{AB}

    module AC activity;
    ip  CN : Channel_type (provider) individual queue;
    end;{AC}

    body B_body for AB;
    . . .
    body C_body for AC;
    . . .

modvar B: AB;
       C: AC;
initialize
begin
    init B with B_body;
    init C with C_body;
end;
end;{SM_body}

modvar S1, S2: SM;
initialize
begin
    init S1 with S_body;
    init S2 with S_body;
end;
end.{SP}
```

Каждый из экземпляров S1 и S2 создаст по паре наследников, причем как наследникам S1, так и наследникам S2 будут соответствовать

модульные переменные `B` и `C`. И поскольку для моделирования наследников `S1` и `S2` используются одни и те же подсети, различить фишки, принадлежащие экземпляру `B`, родителем которого является `S1`, и фишки экземпляра `B`, родитель которого — `S2`, не представляется возможным. Заметим также, что использование для идентификации фишек пары модульных переменных вида *(родитель, наследник)* не решает проблемы. Это становится очевидным, если предположить существования третьего уровня вложенности модулей.

Для моделирования динамически создаваемых экземпляров модулей предлагаем ввести уникальные персональные идентификаторы модулей (ПИМ) подобно идентификаторам экземпляров процессов в языке SDL. Для генерации ПИМ используется специальное место, назовем его PID, которое располагается на самой верхней странице моделирующей сети, т. е. на одной странице с N-модулями, которые сопоставлены системным E-модулям. Место PID содержит одну фишку из множества цветов *integer*, значение которой определяет ПИМ экземпляра модуля при его создании во время функционирования сети. Начальная разметка этого места — фишка со значением $n + 1$, где n — количество экземпляров системных модулей, которые создаются при инициализации спецификации. Экземплярам системных модулей ПИМ со значениями от 1 до n присваиваются в произвольном порядке.

Место PID становится входным и выходным для каждого N-модуля, представляющего E-модуль, который порождает наследников. Копия места PID на странице, связанной с этим N-модулем, становится входным и выходным местом для N-модулей, представляющих те из тел E-модуля, в которых происходит инициализация модулей-наследников. Таким образом, копия места PID появляется на странице, где представлена внутренняя структура тела E-модуля. На уровне тела место PID становится входным и выходным местом для всех N-модулей, которые моделируют порождение нового экземпляра наследника, а также для N-модулей, представляющих наследников, в свою очередь имеющих вложенные модули.

Внутри сети, моделирующей тело какого-либо модуля, для идентификации фишек, принадлежащих одному экземпляру, будет использоваться только ПИМ, так как экземпляр модуля не знает своего имени. Однако модуль-родитель при соединении точек взаимодействия или обращении к экспортируемым переменным наследников использует соответствующую модульную переменную. Информация о соответствии мо-

дульной переменной типа `T` и ПИМ, порожденного для экземпляра, на который указывает эта переменная, сохраняется в специальном месте T_inst на уровне тела охватывающего модуля.

Месту T_inst приписывается множество цветов

$$color\ T\_set\ =\ product\ integer * integer * T\_modvar * T\_body,$$

где T_modvar — множество всех модульных переменных, которые указывают на экземпляры модуля типа `T`, а T_body — идентификаторы всех типов тела, которые потенциально может иметь экземпляр. Фишки, принимающие значения из множества T_set, имеют вид (p, p_M, M, b_M), где p — ПИМ экземпляра родительского модуля, p_M — ПИМ наследника, M — модульная переменная, указывающая на экземпляр наследника, которому при создании был сопоставлен ПИМ p_M, а b_M — тип тела, с которым экземпляр наследника был создан.

3.4. Формальные параметры и экспортируемые переменные

Трансляция формальных параметров и экспортируемых переменных структурно не отличается от случая спецификаций, не использующих динамических конструкций. Каждому параметру или переменной ставится в соответствие место, которое является входным и выходным для N-модуля, представляющего E-модуль, где описан параметр или экспортируемая переменная. По правилам построения иерархической сети копия места, соответствующего параметру или переменной, появляется на странице, связанной с этим N-модулем и содержащей по одному N-модулю для каждого тела рассматриваемого E-модуля. Каждый из этих N-модулей соединяется с местом-копией точно так же, как N-модуль для заголовка E-модуля соединен с прототипом. В результате копия места, соотвествующего параметру или экспортируемой переменной, будет присутствовать на каждой странице, отображающей внутреннюю структуру тела E-модуля.

Кроме того, место, соответствующее переменной, является входным и выходным также для каждого N-модуля, представляющего переход модуля-родителя, в блок которого входит оператор, содержащий обращение к экспортируемой переменной наследника. Соединение происходит на странице, где находится сеть, моделирующая тело модуля-родителя.

Отличия возникают при определении множества цветов, из которого принимают значения фишки, находящиеся в месте, сопоставленном

формальному параметру или экспортируемой переменной. Значение из множества цветов, соответствующего типу параметра или переменной, дополняется ПИМ. Например, пусть EV_type — множество цветов, полученное при отображении типа некоторой экспортируемой переменной `RelReq`. Тогда месту, которое соответствует переменной `RelReq`, будет приписано множество цветов

$$color\ RelReq\_type\ =\ product\ integer\ *\ EV\_type.$$

На рис. 3 изображен результат трансляции оператора `B.RelReq := Busy`, где `Busy` — переменная модуля-родителя того же типа, что и `RelReq`. Рис. 3, *а* демонстрирует структуру сети на уровне тела модуля-родителя, а рис. 3, *б* — на уровне E-перехода, в блок которого входит моделируемый оператор присваивания. Место AB_inst сохраняет информацию о соответствии ПИМ и модульных переменных. Чтобы не загромождать рисунок, используется двунаправленная стрелка для обозначения того, что место AB_inst для перехода является входным и выходным. Фишка, изымаемая из этого места при срабатывании перехода, возвращается в него без изменений. Места $co1$ и $co2$ — соединительные места в сети E-перехода[1].

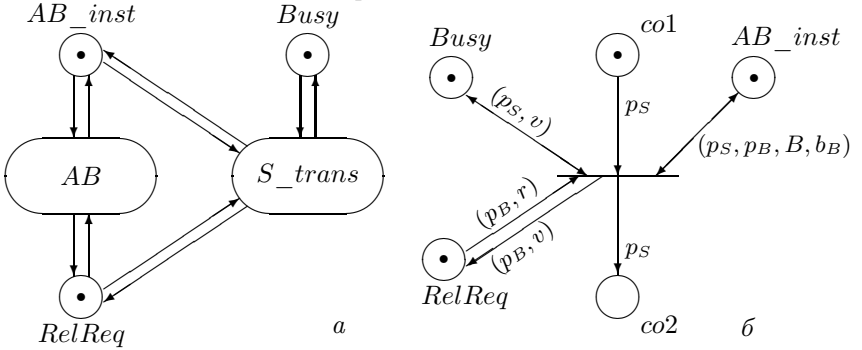

Рис. 3. Моделирование обращения к экспортируемой переменной

Переменные p_B и p_S, входящие в выражения на дугах, обозначают ПИМ наследника `B` и родителя `S` соответственно. Так как все вхождения переменной в спусковую функцию N-перехода и выражения на связанных с ним дугах замещаются одним и тем же значением, в связывание автоматически входят фишки, принадлежащие одному и тому же экземпляру E-модуля `S`.

[1]Трансляция E-переходов подробно описана в [13].

3.5. Точки взаимодействия

Трансляция точек взаимодействия, связи между которыми меняются в процессе функционирования Estelle-спецификации, сильно отличается от статического случая. При отсутствии динамических изменений в структуре связей соединение точек взаимодействия устанавливается на весь период жизни спецификации. Поэтому место, куда в виде фишек помещаются сообщения, отправляемые модулем через некоторую точку взаимодействия, просто сливается с местом, представляющим очередь точки взаимодействия, связанной с первой точкой каналом. Теперь, когда связи между точками могут изменяться, очереди входных и выходных сообщений должны оставаться разделенными. Сообщения перемещаются из очереди выходных сообщений точки взаимодействия во входную очередь той точки, с которой первая точка связана на текущий момент.

Каждая внешняя точка взаимодействия E-модуля представляется тремя местами, которые располагаются на одной странице с N-модулем, соответствующим заголовку E-модуля, т. е. на уровне тела охватывающего модуля. Для обозначения мест, представляющих точку взаимодействия `IP`, будем использовать имена IP_in, IP_out и IP_info. Места IP_in и IP_out представляют соответственно очереди сообщений, входящих и исходящих через точку `IP`, а место IP_info служит для хранения информации о том, какие связи установлены для точки взаимодействия.

Места IP_in, IP_out и IP_info являются одновременно входными и выходными для N-модуля, соответствующего заголовку E-модуля, в котором описана точка взаимодействия. По правилам построения иерархической сети копия каждого из мест, представляющих точку взаимодействия, появляется на следующем уровне, где располагаются N-модули, соответствующие телам рассматриваемого E-модуля. Каждый из этих N-модулей соединяется с местами IP_in, IP_out и IP_info точно так же, как N-модуль для заголовка E-модуля. В результате копии этих мест будут присутствовать на каждой странице, отображающей внутреннюю структуру тела E-модуля. На уровне тела рассматриваемого E-модуля N-модули представляют E-переходы. Их соединение с местами IP_in, IP_out и IP_info определяется наличием в E-переходе приставки `when` или оператора `output`.

Представление внутренней точки взаимодействия модуля отличается расположением соответствующих ей мест. Внутренняя точка, анало-

гично внешней, представляется тремя местами. Однако эти места располагаются на странице, отображающей внутреннюю структуру тела Е-модуля, в котором описана точка взаимодействия, а не на странице, где находится сеть, моделирующая тело модуля-родителя.

Множества цветов, приписанные местам IP_in и IP_out, определяются набором примитивов взаимодействия, которые связаны с каналом, указанным в определении точки `IP`. В место IP_out попадают фишки, соответствующие отправляемым Е-модулем сообщениям. Поэтому множество цветов этого места конструируется на основе набора примитивов взаимодействия, которые связаны с ролью, указанной в описании точки взаимодействия. Основой множества цветов места IP_in служит набор примитивов, связанный с оппозитной ролью. Очереди сообщений представляются списками. При этом каждый список должен быть снабжен ПИМ, так как все экземпляры Е-модуля имеют одинаковые точки взаимодействия, которые моделируются одними и теми же местами.

Место IP_info содержит фишки, состоящие из трех полей. Первое поле содержит ПИМ экземпляра, точка взаимодействия которого рассматривается. Второе и третье поля содержат пары вида *(ПИМ, точка взаимодействия)* или выделенное значение *none*. Второе поле определяется связью, которую устанавливает для точки взаимодействия родитель рассматриваемого модуля. Связь такого рода может быть определена только для внешних точек взаимодействия. Это может быть:

– соединение с внешней точкой взаимодействия другого наследника, в том числе и с другим экземпляром того же самого наследника;

– соединение с внутренней точкой взаимодействия модуля-родителя;

– прикрепление к одной из внешних точек модуля-родителя.

Третье поле определяется связью, которая устанавливается для точки самим рассматриваемым модулем. Такими связями являются:

– соединение внутренней точки с внешней точкой взаимодействия одного из наследников;

– прикрепление внешней точки к внешней точке одного из наследников.

Значение *none* используется, когда соответствующая связь не определена. Заметим, что если второе или третье поле фишки в месте IP_info содержит значение *none*, то точка `IP` является концом некоторой линии связи.

3.6. Операторы установления связи

Группа операторов установления связей включает операторы `connect` (соединить) и `attach` (прикрепить), которые могут использоваться в разделе инициализации или в разделе описания переходов. Первый служит для установления канала между внешними точками взаимодействия двух модулей одного уровня иерархии или между внутренней точкой взаимодействия охватывающего модуля и внешней точкой одного из его наследников. Оператор `attach` применяется охватывающим модулем для установления связи между своей внешней точкой и внешней точкой одного из наследников. Применение операторов группы установления связи к связанным точкам игнорируется.

3.6.1. Соединение точек взаимодействия

Каждое соединение, возникающее в течение функционирования спецификации, представляется в сети двумя переходами. Переходы располагаются на странице, которая содержит сеть, представляющая структуру тела модуля, устанавливающего соединение. На рис. 4 показано, каким образом происходит соединение мест и переходов на странице при отображении оператора `connect M1.A to M2.B`, где `M1` и `M2` — модульные переменные типов `T1` и `T2` соответственно. Овалы, помеченные именами типов `T1` и `T2`, представляют N-модули, которые сопоставлены заголовкам E-модулей `M1` и `M2`. Переход B_to_A изменяет значения фишек в местах B_out и A_in в соответствии с перемещением сообщений по каналу от точки `B` модуля `M2` к точке `A` модуля `M1`. Переход A_to_B осуществляет передачу сообщений в обратном направлении.

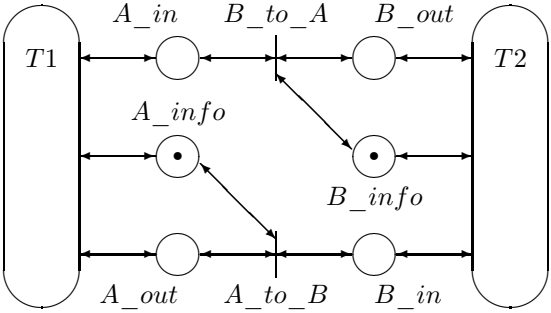

Рис. 4. Моделирование соединения точек взаимодействия

На рис. 5 на примере перехода B_to_A подробно рассматривается изменение значений фишек при перемещении сообщений по каналу, установленному оператором `connect`. Символ $*$ используется для обозначения поля фишки, когда его значение несущественно.

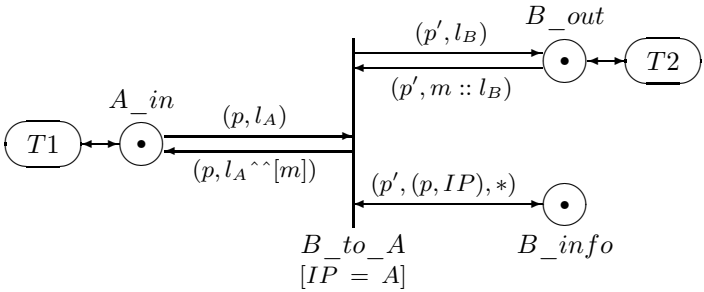

Рис. 5. Моделирование перемещения сообщений

Оператор `connect M1.A to M2.B` входит в блок E-перехода модуля, который устанавливает соединение. Этому блоку на уровне тела E-модуля сопоставлен N-модуль. Сам оператор `connect` представляется переходом сети на странице следующего уровня; N-модуль связан с местами, хранящими информацию о ПИМ экземпляров и связях, установленых для точек взаимодействия. Для нашего примера таким местами будут A_info, B_info, $T1\_inst$ и $T2\_inst$. Все четыре места являются для N-модуля входными и выходными. Соединение копий этих мест с переходом, моделирующим оператор `connect`, повторяет соединение мест-прототипов с N-модулем.

Следует отметить, что оператор `connect` не создает новых фишек для представления очередей и не связан с местами, которые их содержат. Фишки вида (p, nil) помещаются в места IP_in и IP_out при создании нового экземпляра модуля, в заголовке которого описана точка взаимодействия `IP`. "Фишка-очередь" используется в течение всего времени жизни экземпляра и может быть уничтожена только вместе с ним. Установление или уничтожение соединения для точки `IP` изменяет фишку в месте IP_info, определяя тем самым правила доступа к фишкам в местах IP_in и IP_out.

Применение операции соединения к связанным точкам взаимодействия не вызывает никаких изменений в структуре связей спецификации. Поэтому выражения на выходных дугах перехода, моделирующего оператор `connect`, зависят от фишек в местах IP_info. На рис. 6

показано, каким образом происходит соединение мест и переходов для оператора `connect M1.A to M2.B` на уровне блока E-перехода. На дугах присутствуют следующие выражения:

– для мест $T1\_inst$ и $T2\_inst$ на входных и выходных дугах — $(s, p1, M1, b1)$ и $(s, p2, M2, b2)$ соответственно, где s — ПИМ экземпляра, устанавливающего соединение, а $b1$ и $b2$ — тела, с которыми созданы `M1` и `M2`;

– для места A_info на входной дуге — $(p1, link1, *)$, на выходной — $if\ (link1\ =\ none)\ then\ (p1, (p2, B), *)\ else\ (p1, link1, *)$;

– для места B_info на входной дуге — $(p2, link2, *)$, на выходной $if\ (link2\ =\ none)\ then\ (p2, (p1, A), *)\ else\ (p2, link2, *)$.

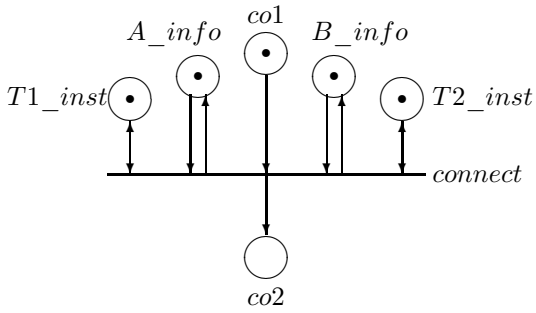

Рис. 6. Моделирование оператора `connect`

В заключение рассмотрим соединение одноименных точек двух экземпляров одного и того же модуля. В этом случае на уровне тела родительского модуля создается только один переход, перемещающий сообщения из выходной очереди одного экземпляра во входную другого. В месте IP_info соединение представляется фишками $(p, (p', IP), *)$ и $(p', (p, IP), *)$, где p и p' — ПИМ экземпляров, а IP — идентификатор точки взаимодействия.

3.6.2. Прикрепление точек взаимодействия

Моделирование прикрепления происходит аналогично моделированию соединения. Каждое прикрепление, возникающее в течение функционирования спецификации, представляется двумя переходами. Переходы также располагаются на уровне тела охватывающего модуля. Отличия в соединении мест и переходов при отображении оператора `attach A to M.B`, где `M` — модульная переменная типа `T`, показаны на

рис. 7. Для большей наглядности сеть, расположенная на уровне тела модуля `M`, помещена внутри соответствующего N-модуля. Одноименные места снаружи и внутри этого N-модуля представляют соответственно прототип и копию одного и того же места. Стрелки показывают направление потока сообщений. Спусковой функцией перехода B_to_A является условие $[IP1 = A]$, которое должно выполняться для второго поля фишки $(p2, (p1, IP1), *)$ в месте B_info; а спусковой функцией A_to_B — условие $[IP2 = B]$, но для третьего поля фишки $(p1, *, (p2, IP2))$ в месте A_info.

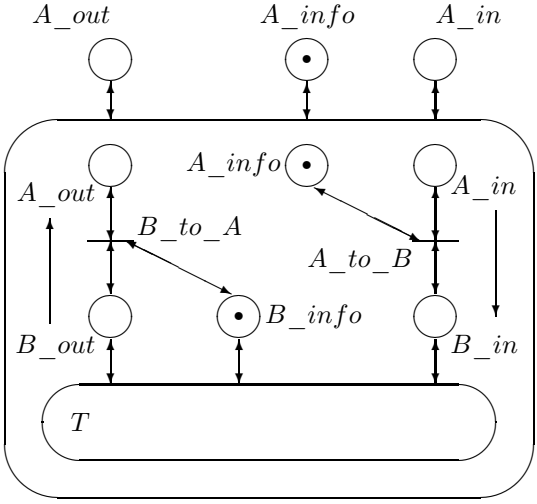

Рис. 7. Моделирование прикрепления точек взаимодействия

Применение оператора `attach` приводит к перемещению очереди сообщений, связанной с точкой взаимодействия охватывающего модуля. Очередь переводится к точке модуля-наследника. Если эта точка уже прикреплена к точке своего охватываемого модуля, то очередь сообщений переводится к соответствующей точке и т. д.

При трансляции Estelle-спецификаций в раскрашенные сети мы ограничиваем применение оператора `attach` точками взаимодействия с индивидуальными очередями сообщений. Тогда после установления прикрепления очередь сообщений, уже полученных модулем, перемещается автоматически при срабатывании переходов, которые моделируют прикрепление. Рассмотрим пример на рис. 7. Перемещение оче-

реди сообщений из места A_in в место B_in произойдет в результате срабатывания перехода A_to_B.

Моделирование самого оператора `attach` происходит аналогично моделированию оператора `connect`, но соответствующий переход связан только с одним местом T_inst. Исходя из информации, содержащейся в этом месте, определяется ПИМ модуля `M`. Второй ПИМ, участвующий в моделировании установления прикрепления, всегда известен — это ПИМ экземпляра, выполняющего оператор. Другое отличие заключается в том, что в выражениях на дугах рассматривается не второе, а третье поле фишки, находящейся в месте A_info.

3.7. Операторы разъединения связей

Группа операторов разъединения связей включает два оператора: `disconnect` и `detach`. В отличие от операторов группы установления связи разъединяющие операторы могут применяться не только к отдельным точкам, но и к модулю в целом. Последнее эквивалентно применению соответствующего оператора к каждой точке взаимодействия модуля.

Операторы `disconnect` и `detach` выполняются только модулями, выдавшими соответствующие `connect` и `attach`. Применение оператора разъединения к несвязанной точке взаимодействия игнорируется.

3.7.1. Отсоединение точек взаимодействия

Рассмотрим отсоединение конкретной точки взаимодействия на следующем примере: `disconnect M.A`, где `M` — модульная переменная типа `T` (см. рис. 8). Для точки взаимодействия `A` создается место dis_A. Каждый оператор вида `connect M.A to M'.IP`, устанавливающий соединение между точками `A` и `IP`, порождает переход dis_A_IP, который уничтожает это соединение. Место dis_A — входное для каждого перехода dis_A_IP, место dis_A_end — выходное, а места A_info и IP_info — входные и выходные. Вся конструкция располагается на уровне тела модуля, осуществляющего отсоединение. Срабатывание перехода dis_A_IP приводит к следующим изменениям разметки:

– из места dis_A изымается фишка со значением $(p1, p2)$;

– из места A_info изымается фишка $(p2, (p3, IP), *)$, помещается — $(p2, none, *)$;

– из места IP_info изымается — $(p3, (p2, A), *)$, помещается — $(p3, none, *)$;

– в место dis_A_end помещается фишка $p1$.

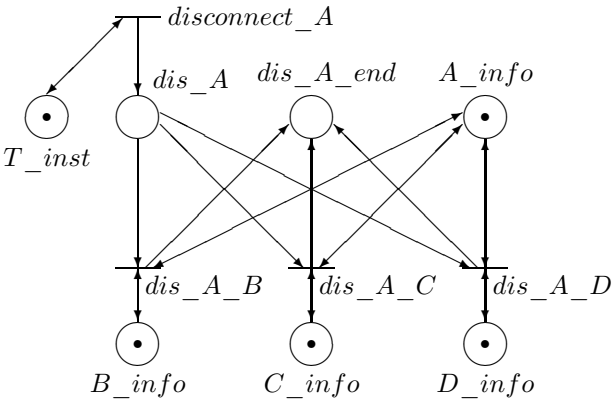

Рис. 8. Моделирование отсоединения точки взаимодействия

Оператор `disconnect` входит в блок некоторого E-перехода. Представляющий этот E-переход N-модуль связан с местами dis_A и dis_A_end. Первое из них является выходным, а второе — входным для N-модуля. Сам оператор `disconnect` моделируется переходом на странице, которая связана с N-модулем, представляющим E-переход. Срабатывание этого перехода сети помещает в копию места dis_A фишку со значением $(p1, p2)$, где $p1$ — ПИМ экземпляра, выполняющего оператор, а $p2$ — ПИМ, который одна из фишек в месте T_inst связывает с модульной переменной `M`. Появление фишки в копии места dis_A_end обозначает, что отсоединение точки взаимодействия завершено и экземпляр, имеющий ПИМ $p1$, может продолжить выполнение своего E-перехода.

Отсоединение всех точек взаимодействия модуля `M` типа `T` осуществляется оператором `disconnect M`. Существование такого оператора требует создания описанной выше конструкции для каждой точки взаимодействия модуля. Переход, моделирующий этот оператор, помещает по фишке в каждое из мест dis_A, dis_B и dis_C, где `A`, `B` и `C` — точки взаимодействия модуля `M` (см. рис. 9).

3.7.2. Открепление точек взаимодействия

Оператор открепления может применяться модулем как к точке взаимодействия наследника, так и к своей собственной точке взаимодействия. В данной работе рассматривается упрощенный вариант операто-

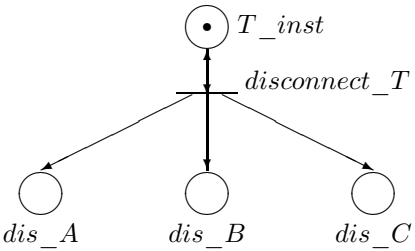

Рис. 9. Моделирование отсоединения модуля

ра `detach`, который отличается от стандартного тем, что при его выполнении не происходит перемещения очереди сообщений. Такой разрыв прикрепления соответствует операции `simple-detach`, которая выполняется при уничтожении экземпляра модуля оператором `terminate`.

Сначала рассмотрим открепление точки взаимодействия наследника, которое осуществляется оператором `detach M.A`, где `M` — модульная переменная типа `T`. В сети создается конструкция, аналогичная той, что создается при моделировании отсоединения (см. рис. 10). Конструкция также располагается на уровне тела модуля, осуществляющего открепление. Для точки взаимодействия `A` создается место dch_A. Каждому оператору, который прикрепляет точку `A` к некоторой точке `IP` охватывающего модуля, сопоставляется переход dch_A_IP, который уничтожает это прикрепление. Срабатывание перехода dch_A_IP приводит к следующим изменениям разметки:

– из места dch_A изымается фишка со значением $(p1, p2)$;

– из места A_info изымается фишка $(p2, (p1, IP), *)$, помещается — $(p2, none, *)$;

– из места IP_info изымается — $(p1, *, (p2, A))$, помещается — $(p1, *, none)$;

– в место dch_A_end помещается фишка $p1$.

Сам оператор `detach` моделируется переходом, входящим в сеть соответствующего блока E-перехода. Срабатывание этого перехода помещает в место dch_A фишку со значением $(p1, p2)$, где $p1$ — ПИМ экземпляра, выполняющего оператор, а $p2$ — ПИМ, который одна из фишек в месте T_inst связывает с модульной переменной `M`. Появление фишки в месте dch_A_end обозначает, что открепление точки взаимодействия завершено и экземпляр, имеющий ПИМ $p1$, может продолжить выполнение своего перехода.

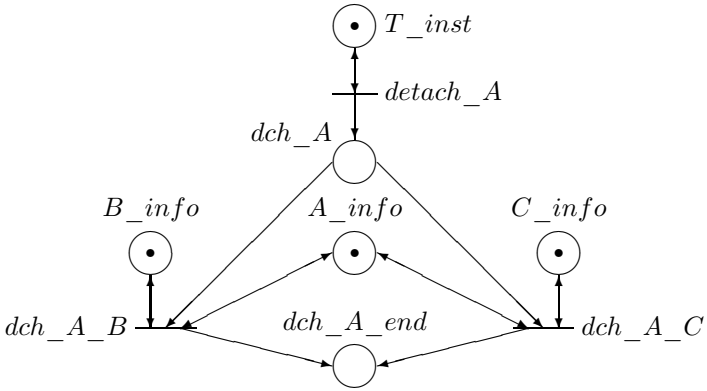

Рис. 10. Моделирование открепления точки взаимодействия

Если `A` — собственная точка взаимодействия модуля, а `IP` принадлежит одному из наследников, то изменится только схема перемещения фишек. В dch_A помещается ПИМ модуля-родителя. В месте A_info происходит изменение не второго, а третьего поля: $(p1, *, (p2, IP))$ на $(p1, *, none)$. Соответственно в IP_info изменяется значение второго поля: $(p2, none, *)$ заменяет $(p2, (p1, A), *)$.

Открепление всех точек взаимодействия модуля-наследника `M` типа `T` осуществляется оператором `detach M`. Моделирование открепления модуля полностью аналогично моделированию отсоединения модуля.

В заключение рассмотрим случай применения операторов группы разъединения связей к несвязанным точкам. В этом случае необходимо удалить фишку из места dis_IP/dch_IP (см. рис. 11). Если представить фишку в месте IP_info в виде $(p, link1, link2)$, то для перехода, который удаляет фишку из места dis_IP, спусковой функцией будет условие $(link1 = none)$. Спусковая функция перехода, который удаляет фишку из места dch_IP, зависит от принадлежности точки взаимодействия `IP`: если `IP` — собственная точка модуля, то спусковой функцией будет $(link2 = none)$, если `IP` — точка взаимодействия одного из наследников, то $(link1 = none)$.

3.8. Организация ввода/вывода

Для передачи примитива взаимодействия через точку взаимодействия `IP` в языке Estelle используется оператор `output IP.m`, где `m` —

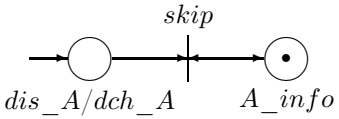

Рис. 11. Моделирование применения оператора разъединения связи к несвязанной точке

идентификатор взаимодействия, возможно, со списком параметров.

Оператор `output` представляется в сети переходом с тем же именем, который располагается на странице, соответствующей блоку E-перехода, куда входит данный оператор. Пусть m — значение, соответствующее в сети взаимодействию `m`. На рис. 12 изображен фрагмент сети, моделирующий выполнение оператора `output IP.m`. Список l представляет собой очередь сообщений, накопившуюся в месте IP_out, p — ПИМ экземпляра, который осуществляет вывод. Выражение $endline_{out}$ определяет, является ли точка взаимодействия концом некоторой линии связи:

$$endline_{out}(line1, line2) = ((line1 \neq none)\ and\ (line2 = none))\ or$$

$$((line1 = none)\ and\ (line2 \neq none)).$$

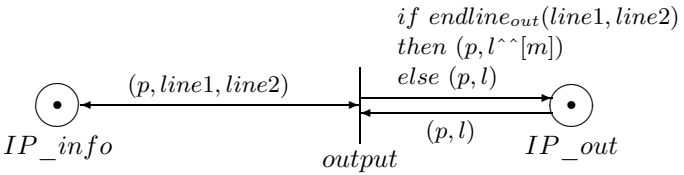

Рис. 12. Моделирование оператора `output`

Прием примитивов взаимодействия осуществляется входными E-переходами, т. е. такими, в условие возможности которых входит приставка `when`. Как и в статическом случае, если моделируемый E-переход входной, то соответствующая ему сеть использует место, которое представляет очередь точки взаимодействия (см. [13], п. 3.4). Незначительные отличия возникают из-за разных подходов к представлению FIFO-очередей.

В гл. 3 работы [13] для представления очереди сообщений использовалось место, при помещение в которое фишки нумеровались. Фишка

с наименьшим номером представляла голову, а с наибольшим — хвост очереди. При реализации алгоритма в рамках экспериментальной системы EPV использовались места специального типа — места-очереди. Фишка, поступающая в такое место, помещается в очередь и остается недоступной, пока из места не будут извлечены все фишки, поступившие до нее. В обоих случаях выполнение входного E-перехода уменьшало разметку места, представляющего ассоциированную с точкой взаимодействия очередь, на одну фишку.

В данной работе для представления очередей сообщений мы используем множества цветов типа *list*. В результате очереди соответствует одна фишка. Следствием является тот факт, что любое место, представляющее очередь, будет входным и выходным для N-перехода, который обращается к этой очереди. Изъятию фишки из головы очереди соответствует изъятие из "места-очереди" IP_in фишки со списком, представляющим эту очередь, и помещение в него фишки, значением которой служит тот же список без первого элемента. Место IP_info также участвует в моделировании приема сообщений, так как считывать сообщение из головы очереди имеет право только модуль, точка взаимодействия которого является концом линии связи.

На рис. 13 приведена сеть, моделирующая следующий E-переход:

```
trans
     from S1 to S2
     when V.mess  provided F(a,b)
       begin
          . . .
       end;
```

Так как модуль сохраняет очередь полученных через точку сообщений после того, как канал к этой точке уничтожен, условие $endline_{in}$ проще, чем $endline_{out}$:

$$endline_{in}(line1, line2) = (line1 = none)\ or\ (line2 = none).$$

Если несколько точек взаимодействия модуля имеют общую очередь входных сообщений, то прием сообщений моделируется несколько иначе. Схема моделирования общей очереди представлена на рис. 14. Пометка сети, относящаяся к точке `B`, полностью аналогична пометке, связанной с точкой `A`. Общая очередь представляется местом *Queue*, которое содержит фишку (p, l) для каждого экземпляра модуля. Сообщения в очередь, представляемую списком l, помещаются в виде пары

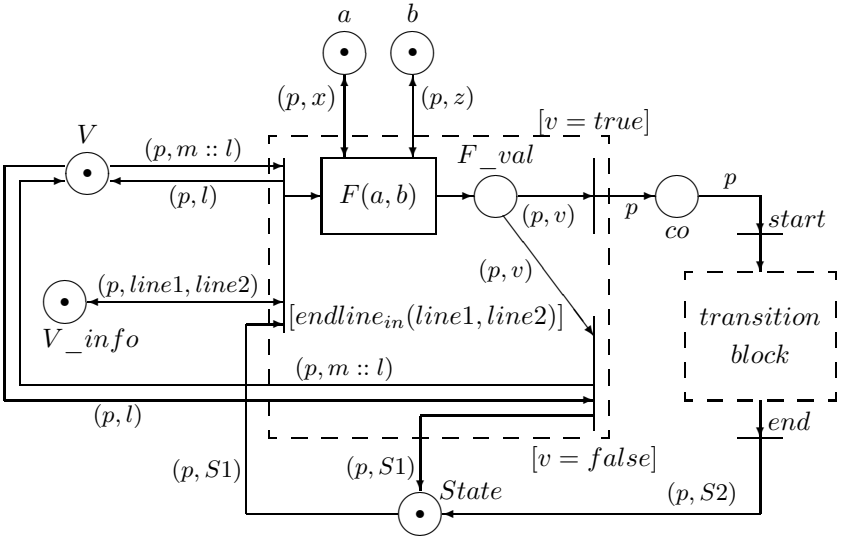

Рис. 13. Моделирование входного E-перехода

(IP, m), где IP — идентификатор точки взаимодействия, через которую получено исходное сообщение m. Для каждой из точек, делящих очередь, создается переход rec_IP, который перемещает сообщения из места IP_in в общую очередь. Спусковой функцией этого перехода является условие $endline_{in}$ от значений второго и третьего полей фишки в месте IP_info.

При моделировании общей очереди место *Queue* играет для входных E-переходов ту же роль, что и место IP_in в случае индивидуальной очереди. Отличие заключаются в том, что первый переход сети, моделирующей входной E-переход, не проверяет условие $endline_{in}$, но должен проверить, совпадают ли идентификатор точки взаимодействия, которым помечено сообщение в голове очереди, и точки, указанной в приставке `when`. На рис. 14 переход *provided* представляет первый переход сети, моделирующей входной E-переход с приставкой `when A.msg`.

3.9. Трансляция тела модуля

Напомним, что модуль Estelle-спецификации представлен в иерархической сети на трех уровнях. Впервые он появляется в виде N-модуля

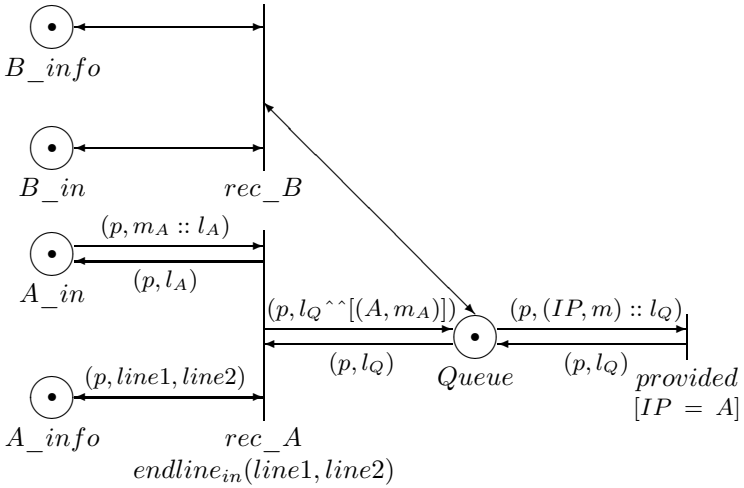

Рис. 14. Моделирование общей очереди

на уровне тела охватывающего модуля. Страница, связанная с этим N-модулем, содержит по одному N-модулю для каждого тела, описанного для E-модуля в спецификации. Выделение такой страницы приводит к тому, что сеть, моделирующая каждое из описанных тел, размещается на отдельной странице.

Вершины сети на уровне тела E-модуля можно условно разделить на две части. Первая состоит из N-модулей, представляющих наследников E-модуля, и связанных с ними местами. Ко второй относятся E-переходы, переменные и сетевые конструкции, призванные моделировать функционирование модуля.

3.9.1. Моделирование модулей-наследников

Как уже говорилось выше, каждому типу модулей-наследников, описанных в теле, на странице, которая этому телу соответствует, сопоставляется N-модуль. Согласно правилам, описанным в п. 3.5–3.7, осуществляется моделирование точек взаимодействия модулей-наследников и возможных связей между ними.

Помимо представляющих точки взаимодействия мест с N-модулем, сопоставленным модулям-наследникам типа `T`, связываются места T_inst, T_init и T_kill. Первое подробно описано в п. 3.3. Помеще-

ние фишки в место T_init приводит к созданию нового экземпляра модуля типа T, а в месте T_kill — напротив, к уничтожению уже существующего. Значениями фишек в местах T_init и T_kill являются пары (p, b), где p — ПИМ создаваемого/уничтожаемого экземпляра, а b — тип тела этого экземпляра.

Соединение мест T_inst, T_init и T_kill с N-модулями на странице происходит по следующим правилам:

– место T_inst не связано с N-модулем, сопоставленным модулям типа T. Фишки из этого места используются E-переходами, в которых происходит обращение к ресурсам наследников (например, установление или разъединение связи между точками взаимодействия). Поэтому место T_inst — входное и выходное для всех N-модулей, представляющих такие E-переходы. Кроме того, место T_inst — выходное для каждого N-модуля, который создает новый экземпляр модуля типа T, и входное для каждого N-модуля, который уничтожает один из существующих экземпляров;

– место T_init является входным для N-модуля, соответствующего модулю-наследнику типа T, и выходными для N-модулей, которые представляют E-переходы охватывающего модуля, где происходит создание экземпляров наследника;

– место T_kill является входным для N-модуля, соответствующего модулю-наследнику типа T, и выходными для N-модулей, которые представляют E-переходы охватывающего модуля, где происходит уничтожение экземпляров наследника.

3.9.2. Моделирование функциональности тела модуля

Построение той части сети, которая определяет функционирование E-модуля при выборе данного тела, не зависит от его положения в иерархии Estelle-спецификации и мало отличается от статического случая (см. [13], п. 3.3). E-переходы представляются N-модулями. Каждой переменной в сети соответствует одно место. В качестве имени моделирующего места, как правило, выбирается имя соответствующей переменной. Множеством возможных цветов для фишек в месте-переменной становится множество цветов, составленное из ПИМ экземпляра и типа данной переменной. На странице также создается место *State*, типом которого является множество локальных состояний E-модуля, расширенное полем для хранения ПИМ. Если в модуле не определено множество локальных состояний, то место *State* содержит по одной фишке

для каждого экземпляра модуля. Значением каждой фишки является ПИМ соответствующего экземпляра. Место $State$ — входное и выходное для всех N-модулей, представляющих E-переходы. Таким образом, место $State$, помимо представления локальных состояний E-модуля, служит для обеспечения неделимости выполнения E-перехода.

Правила соединения мест, представляющих переменные, и N-модулей, представляющих E-переходы, сохраняется: если некоторая переменная используется в E-переходе, то соответствующее данной переменной место будет входным и выходным для N-модуля, моделирующего этот E-переход. Правило обращения к экспортируемым переменным модулей-наследников несколько изменяется. Наряду с местом, соответствующим переменной, входным и выходным для N-модуля становится место T_inst, где `T` — тип модуля-наследника, которому принадлежит переменная. Кроме того, место T_inst — входное и выходное для каждого N-модуля, представляющего E-переход, в блок которого входят операторы `all`, `forone` или `exist`, если областью действия этих операторов являются все экземпляры модуля типа `T`.

Поскольку очереди сообщений представляются списками, изъятие сообщения из очереди или добавление к ней нового сообщения моделируется изменением значения в соответствующем месте. Поэтому места представляющие точки взаимодействия, как и места-переменные, всегда являются как входными, так и выходными. Место IP_in соединяется с N-модулем, представляющим входной E-переход, где `IP` — точка взаимодействия, указанная в приставке `when` и имеющая индивидуальную очередь входных сообщений. N-модуль, соответствующий E-переходу, в блок которого входит оператор `output IP.msg`, соединяется с местом IP_out. Это место также будет всегда входным и выходным для N-модуля. В п. 3.8 подробно описано, в каких случаях N-модули будут соединены еще и с местом IP_info.

Новым по отношению к статическому случаю является наличие на уровне тела модуля служебных мест и N-модулей, управляющих созданием и уничтожением экземпляров. На странице, где расположена сеть для тела с именем `B`, присутствуют места-копии TB_init и TB_kill. Фишки в этих местах несут только ПИМ экземпляра без имени тела, которое используется уровнем выше при определении, на какую из страниц с телом модуля передать управление.

В статическом случае раздел инициализации E-модуля определял начальную разметку сети, моделирующей тело модуля. При трансля-

ции спецификаций, содержащих динамические конструкции, переходы раздела инициализации присутствуют на уровне тела модуля в виде N-модулей. Это связано с тем, что наборы фишек, принадлежащие экземплярам модулей, могут создаваться и размещаться по местам сети в произвольный момент функционирования спецификации. Окружение N-модулей, представляющих переходы раздела инициализации, определяется по следующим правилам:

– место TB_init — входное;

– если при создании нового экземпляра модуля используется значение какого-либо параметра модуля, то место, соответствующее параметру, будет входным и выходным;

– место $State$, а также места, соответствующие внутренним точкам взаимодействия и переменным E-модуля, являются выходными;

– если переход раздела инициализации создает экземпляр модуля-наследника типа `T`, то

- место PID — входное и выходное для N-модуля, представляющего этот переход;
- места T_inst, T_init, а также все места, представляющие параметры и внешние точки взаимодействия наследника — выходные;

На странице, где расположена сеть для тела модуля,присутствует также N-модуль, предназначенный для уничтожения экземпляров модуля типа `T` с телом `B`. Входными для этого N-модуля являются места $State$, TB_kill, T'_inst для всех типов наследников, а также места, представляющие параметры, переменные и точки взаимодействия модуля, выходными — T'_kill для всех типов наследников. Подробности создания и уничтожения модуля будут рассмотрены ниже.

3.10. Трансляция Estelle-перехода

Схема трансляции E-переходов также в целом сохраняется (см. [13], п. 3.4). Основное отличие заключается в том, что все фишки, используемые при выполнении E-перехода, "помечены" ПИМ экземпляра, переход которого был выбран для выполнения.

В случае простого E-перехода на соответствующей ему странице будет присутствовать один переход сети. Приставки `from`, `when` и `provided` образуют спусковую функцию перехода. На странице присутствуют копии всех мест, которые связаны с N-модулем, представляющим E-переход. Входные/выходные дуги повторяют соединение мест-прототипов с N-модулем. Блок E-перехода определяет выражения на дугах.

Для E-переходов, которые невозможно представить одним N-переходом, так же как и в статическом случае, применяется тактика разбиения блока E-перехода на подблоки. В результирующей сети E-переход представляется сетью, которая располагается на отдельной странице и состоит из последовательно связанных фрагментов. Каждый из фрагментов моделирует одну из приставок `provided` или `delay` условия E-перехода или один из операторов блока E-перехода. Фрагмент может быть отдельным переходом сети или более сложной подсетью. Цепочку фрагментов, представляющих блок E-перехода, ограничивают два служебных перехода — *start* и *end*. Рассмотрим пример. На рис. 15 схематично изображена сеть для следующего E-перехода:

```
trans
     from S1
     when US.DT
     provided F
       to S2
       begin
            . . .
       end;
```

Заметим, что моделирование приставок `from` и `when` и оператора `to` происходит с помощью выражений на дугах, а не самостоятельных фрагментов, как для приставки `provided`.

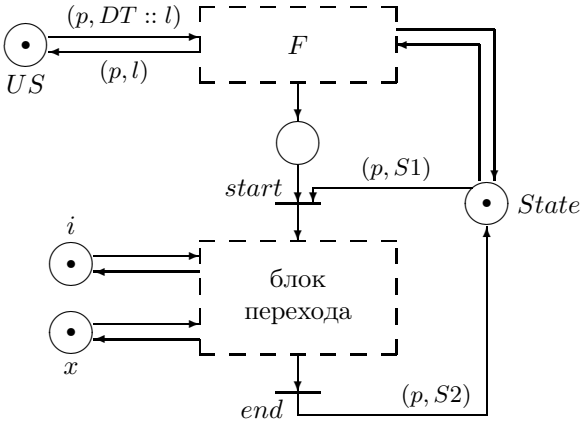

Рис. 15. Схема сети, моделирующей E-переход

3.10.1. Трансляция специфических операторов Estelle

Под специфическими понимаются операторы `all`, `forone` и `exist`. Для моделирования этих операторов используются специальные фрагменты сети.

Оператор `all` представляет собой своеобразный оператор цикла, подобный циклу `for`. Однако в отличие от цикла `for`, в котором изменение переменной цикла происходит последовательно в определенном направлении, оператор `all` предполагает недетерминированный выбор значения переменной цикла для следующей итерации. В обобщенной форме оператор `all` выглядит следующим образом:

`all` *область_действия* `do` *оператор*;

Областью действия может быть либо множество всех экземпляров модуля некоторого типа `T`, либо вектор значений любого скалярного типа. Если область действия пуста, выполнение оператора `all` равноценно выполнению пустого оператора.

На рис. 16 изображена сеть, в которую транслируется оператор `all`, если областью его действия служит множество всех экземпляров модуля некоторого типа `T`. Результатом использования фишек с временными штампами и выражений @*ignore* является приоритет перехода *all* над переходом *empty* и перехода *return* над *all_end*. Пока место *T_inst* не пусто, появление фишки в месте *co*1 приводит к немедленному срабатыванию перехода *all* и, следовательно, выполнению оператора, следующего после `do`. Переход *empty* получает возможность сработать только после того, как все фишки переместятся из места *T_inst* в место *keep*. Срабатывание перехода *empty* перемещает фишку из места *co*1 в место *co*2, после чего та же схема повторяется для переходов *return* и *all_end*. Первый срабатывает, пока не вернет все фишки в место *T_inst*, а срабатывание второго завершает выполнение оператора `all`.

Если область действия оператора `all` пуста, оба места *T_inst* и *keep* не содержат фишек. Следовательно, переходы *all* и *return* невозможны, а срабатывание *empty* и *all_end* не имеет никакого стороннего эффекта.

В случае, когда областью действия оператора `all` служит вектор значений скалярного типа, место *T_inst* заменяется местом *domain*, начальной разметкой которого становятся фишки с этими значениями. С другой стороны, можно использовать сеть, предложенную в п. 3.4.4 работы [13] для моделирования цикла `for`, поскольку результат выполнения оператора `all` не зависит от порядка выполнения итераций.

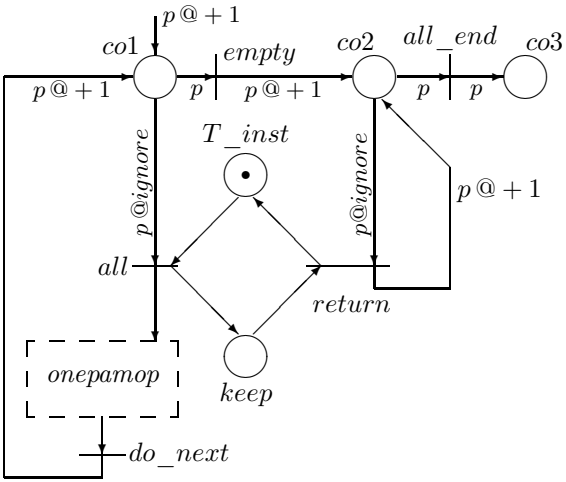

Рис. 16. Моделирование оператора `all`

Оператор `forone` является условным оператором и имеет следующий синтаксис:

`forone` *область_ действия* `suchthat` *булевское_ выражение* `do`
оператор-1
`otherwise` *оператор-2*;

Областью действия также может быть либо множество всех экземпляров модуля некоторого типа `T`, либо вектор значений любого скалярного типа, кроме вещественного. Рассмотрим первый случай, так как второй легко сводится к условному оператору `if` или оператору ветвления `case` и не представляет особого интереса.

Если областью действия оператора `forone` является множество всех экземпляров модуля некоторого типа `T`, оператор находит экземпляр модуля, удовлетворяющий *булевскому_ выражению*, и выполняет для него *оператор-1*. Если нет экземпляра модуля, удовлетворяющего *булевскому_ выражению*, выполняется *оператор-2*, если он присутствует, поскольку часть оператора `forone`, начинающаяся со служебного слова `otherwise`, необязательна.

Схема моделирования оператора `forone` изображена на рис. 17. Спусковую функцию F перехода $forone$ образует *булевское_ выражение*. Кроме места T_inst с переходом $forone$ связываются также места, необходимые для вычисления значения спусковой функции, напри-

мер места, соответствующие экспортируемым переменным наследника. Использование фишек с временными штампами гарантирует, что срабатывание перехода *otherwise* произойдет только в случае, когда *булевское_выражение* не выполняется ни для одного экземпляра модуля типа `T`.

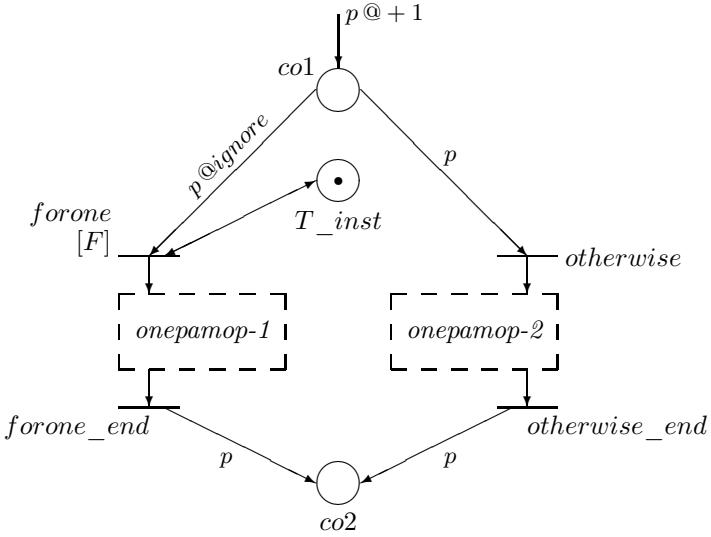

Рис. 17. Моделирование оператора `forone`

Выражение `exist` имеет вид:

`exist` *область_действия* `suchthat` *булевское_выражение*

Если областью действия выражения `exist` является множество всех экземпляров модуля некоторого типа `T`, то оно вырабатывает булевское значение, зависящее от того, найден ли экземпляр модуля, удовлетворяющий *булевскому_выражению*.

Следующий пример показывает, каким образом выражение `exist` представляется в сети. Пусть один из E-переходов неоторого модуля типа `T` имеет приставку `provided exist X: T suchthat X.RelReq = 1`. В сети, моделирующей этот E-переход, выражению `exist` не сопоставляется отдельного перехода (см. рис. 18). Оно определяет соединение перехода, соответствующего приставке `provided`, с местами T_inst и $RelReq$. Условие `X.RelReq = 1` преобразуется в спусковую функцию.

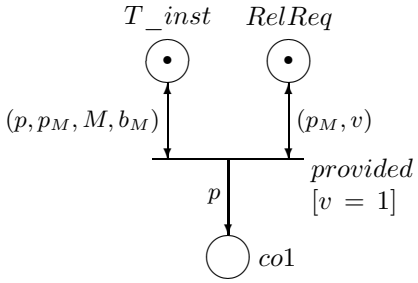

Рис. 18. Моделирование выражения `exist`

3.10.2. Трансляция процедур и функций

Если блок E-перехода содержал вызов процедуры или функции, то в сети для E-перехода ему соответствует N-модуль. Места, моделирующие переменные, которые участвуют в вызове процедуры/функции, являются входными и выходными местами N-модуля, который представляет этот вызов. Кроме того, для каждого вызова функции создается дополнительное место для хранения результата функции. Множеством возможных цветов для фишек в таком месте становится множество цветов, составленное из ПИМ экземпляра и типа результата функции. Процедуры и функции транслируются таким же способом, что и блок перехода. Сеть, моделирующая соответствующую процедуру или функцию, размещается на странице, связанной с N-модулем, который представляет оператор вызова. Аналогично сети, представляющей блок E-перехода, сеть, которая моделирует тело процедуры/функции, представляет собой цепочку фрагментов, ограниченную двумя служебными переходами *begin* и *end*.

Все места для хранения результата функции создаются пустыми. Каждое место такого рода — входное и выходное для N-модуля, представляющего вызов функции, которому сопоставлено это место, а также выходное для перехода *start* и входное для перехода *end*, которые ограничивают сеть, моделирующую блок E-перехода. Срабатывание перехода *start* помещает в место для хранения результата функции фишку. По завершении выполнения E-перехода переход *end* изымает фишку из этого места. Таким образом, начальная разметка места для хранения результата функции определяется не при создании нового экземпляра модуля, которому принадлежит E-переход, а когда начинается выполнение E-перехода.

При таком подходе трансляция вызова процедуры/функции, который содержится в теле другой процедуры/функции, практически не отличается от трансляции оператора вызова, входящего в блок E-перехода. Единственное отличие заключается в том, что роль переходов *start* и *end*, ограничивающих сеть для блока E-перехода, выполняют переходы *begin* и *end*. На рис. 19 приведена схема моделирования вложенного вызова функций $F1$ и $F2$.

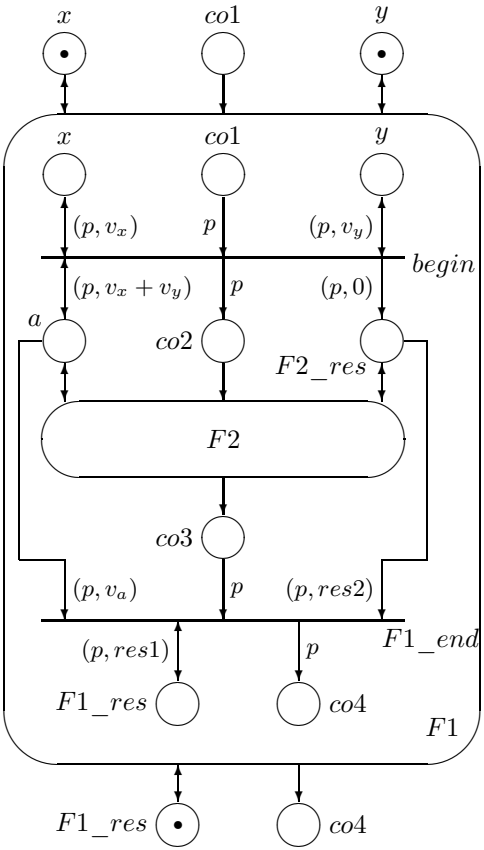

Рис. 19. Моделирование вложенного вызова функций

3.10.3. Моделирование E-переходов с задержками

До сих пор за рамками рассмотрения оставались временные характеристики E-переходов. Приставка `delay` всегда представляется фрагментом сети, который помещается между фрагментом для приставки `provided` и переходом *start*, так как включение таймера происходит, когда условие возможности E-перехода проверено и выполняется. На рис. 20 изображена подсеть для приставки `delay(d1,d2)`. Чтобы не загромождать рисунок, сеть не содержит мест, не участвующих в моделировании задержки.

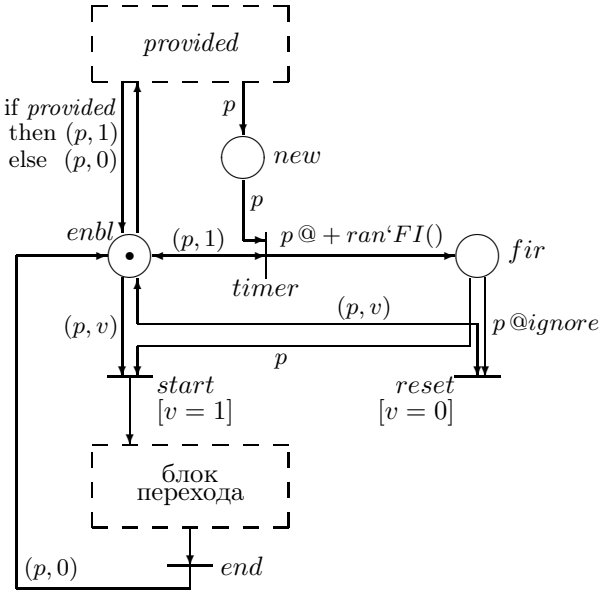

Рис. 20. Моделирование приставки `delay(d1,d2)`

Временные штампы несут только фишки в месте fir. При определении временного штампа фишки используется специальное множество цветов:

$$color\ FI\ =\ int\ with\ d1\,..\,d2\ declare\ ran;$$

Для множества FI (от английского термина *firing interval* — интервал срабатывания) определена функция $ran`FI()$, которая возвращает произвольное целое значение между $d1$ и $d2$. Фишки во всех остальных местах не несут временных штампов.

Любая фишка в месте *enbl* представляет собой пару, где первое поле предназначено для хранения ПИМ экземпляра, которому принадлежит E-переход, а значение второго совпадает со значением приставки `provided`. При инициализации нового экземпляра модуля в место *enbl* помещается фишка (ПИМ,0), где ПИМ — персональный идентификатор, с которым создается экземпляр. Фишки в месте fir несут временной штамп и имеют в качестве значения ПИМ соответствующего экземпляра. Переход *timer* может сработать, когда второе поле фишки в месте *enbl* равно единице. При этом место *new* гарантирует, что срабатывание перехода *timer* происходит только один раз, когда E-переход становится возможным. Фишка со значением ПИМ помещается в это место, когда в месте *enbl* фишка с тем же ПИМ и нулем во второй позиции заменяется на фишку (ПИМ,1). Изымает фишку из места *new* срабатывание перехода *timer*, которое соответствует запуску таймера E-перехода. Если перевычисление `provided` после завершения выполнения другого E-перехода показало, что рассматриваемый E-переход остался возможным, место *new* остается пустым и повторного запуска таймера не происходит.

Срабатывание перехода *timer* помещает фишку в место fir. Таким образом, временной штамп фишки в месте fir определяется как текущее модельное время плюс некоторое значение из интервала $[d1, d2]$. Фишка станет доступна переходам сети не ранее чем через $d1$, и не позднее чем через $d2$ единиц времени с момента своего создания. Если нарушения условия возможности E-перехода не происходит, переход *start* срабатывает сразу, как только фишка в месте fir становится доступной.

Если произошло нарушение условия возможности рассматриваемого E-перехода в результате выполнения какого-то другого E-перехода, фишку из места fir необходимо удалить. На рис. 20 изображена сеть, в которой фишка из места fir удаляется с помощью временной пометки @*ignore*. Данная пометка приписывается входной дуге перехода *reset*, который срабатывает сразу после нарушения условия возможности E-перехода. Таким образом, для каждого из экземпляров модуля, которому принадлежит E-переход, в месте fir находится всегда только одна фишка, помещенная туда при последнем включении таймера E-перехода.

Моделирование приставок `delay(d1)` и `delay(d1,*)` отличается от моделирования приставки `delay(d1,d2)` выбором временной пометки

на дуге от перехода *timer* к месту fir. Приставке `delay(d1)` соответствует пометка $p\,@ + d1$, т. е. к текущему модельному времени добавляется всегда одно и то же значение — $d1$. Сложнее обстоит дело с приставкой `delay(d1,*)`, так как каждая разметка в раскрашенной сети, снабженной временным механизмом Йенсена, существует в рамках замкнутого временного интервала [20]. Нам представляется разумным использовать при моделировании приставки `delay(d1,*)` выбор значения задержки из интервала [$d1$, $d2$], где значение $d2$ много больше, чем $d1$. Такое допущение не является сильным нарушением временной семантики Estelle, так как, несмотря на существование обозначения для бесконечности, E-переход не может вечно оставаться возможным и не выполниться [4]. Если E-переход с задержкой был возможным хотя бы в течение одного такта вычислений, то он будет предлагаться к выполнению на каждом следующем такте и рано или поздно выполнится, если только не перестанет быть возможным.

3.10.4. Моделирование E-перехода с приоритетом

Если спецификация содержит E-переходы с приставкой `priority`, то модель раскрашенных сетей Петри необходимо расширить приоритетами [5]. Для этого переходы получают новый тип пометки — приоритет, который задается целым неотрицательным числом. Как и в Estelle, считается, что чем меньше число, тем выше приоритет перехода. Отсутствие приоритета соответствует наименьшему приоритету. Возможность срабатывания перехода определяется так же, как в раскрашенных сетях. Из нескольких возможных переходов срабатывает любой, приоритет которого не меньше, чем у остальных возможных переходов. При наличии в сети временных конструкций срабатывает любой из готовых переходов, приоритет которого достаточно велик.

Моделирование E-переходов с приоритетами не требует дополнительных мест или переходов сети и отображается приоритетами переходов сети. Если E-переход простой, то пометка приписывается моделирующему его N-переходу, в противном случае — служебному переходу *start*. Значение пометки совпадает со значением приоритета E-перехода. Все остальные переходы не помечаются, что соответствует наименьшему приоритету.

3.11. Создание и уничтожение новых экземпляров модулей

Создание или уничтожение новых экземпляров модулей происходит в охватывающем модуле, причем создание может выполняться как статически, в разделе инициализации, так и динамически, в разделе описания переходов охватывающего модуля.

3.11.1. Создание экземпляра модуля

Любое создание модуля-наследника осуществляется его родителем с помощью оператора `init M with B (par_1,..,par_n)`, где `M` — модульная переменная типа `T`, `B` — тип тела, с которым создается новый экземпляр, а `par` — список фактических параметров (если есть). Оператору `init` соответствует переход в сети, представляющей блок E-перехода, которому принадлежит этот оператор (см. рис. 21). Этот N-переход помещает в место T_inst новую фишку со значением (p_S, p, M, B), где p_S — ПИМ модуля, который выполняет оператор `init`, а p — значение фишки в месте PID, которое становится ПИМ создаваемого экземпляра. В каждое место, представляющее один из параметров модуля-наследника, в виде фишки со значением (p, par_i) помещается соответствующий фактический параметр. Помимо этого переход $init\_M$ помещает новые фишки в места, моделирующие точки взаимодействия. Для каждой точки `IP` в место IP_info помещается фишка $(p, none, none)$, а в места IP_in и IP_out — по фишке (p, nil). В место T_init помещается фишка (p, B). И наконец, значение фишки в месте PID увеличивается на единицу.

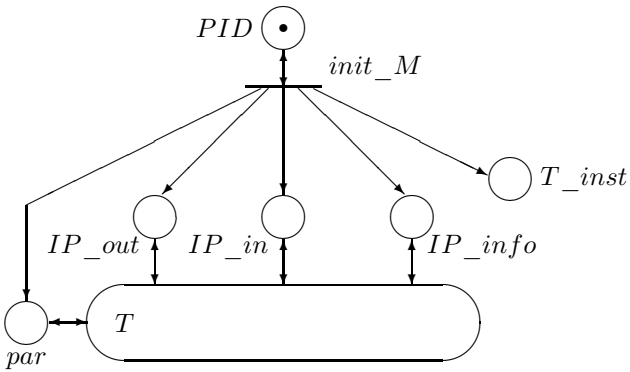

Рис. 21. Моделирование оператора `init`

Однако на этом создание нового экземпляра модуля не заканчивается, но все дальнейшие действия происходят на страницах более низкого уровня иерархии. Сначала на странице, где каждому из определенных для модуля тел соответствует один N-модуль, фишка (p, B) в месте T_init заменяется фишкой p в месте TB_init (см. рис. 22). Это определяет, на какой из страниц, представляющих структуру тела, произойдет окончательная активизация нового экземпляра.

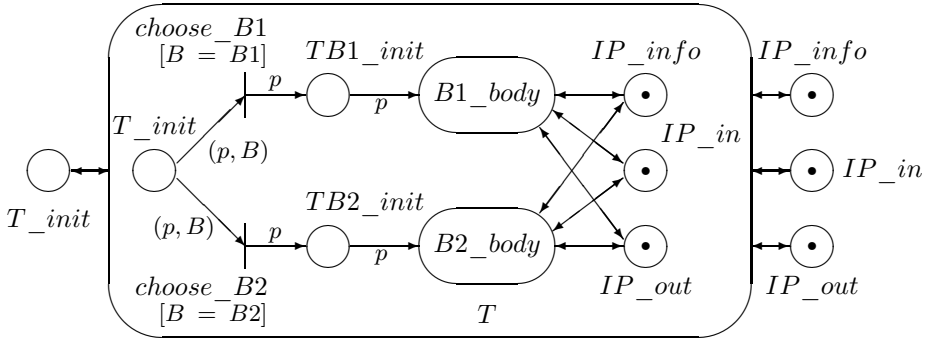

Рис. 22. Моделирование выбора тела для нового экземпляра

Когда фишка p появляется в месте TB_init, фишка p возникает также в копии этого места на странице, где представлена структура выбранного тела. Последнее делает возможным выполнение одного из E-переходов раздела инициализации, которые представлены на странице N-модулями. Сеть, моделирующая блок перехода, осуществляет активизацию нового экземпляра. Во все места-переменные заносятся новые фишки, причем для инициализации используется либо "нулевое" значение[2], либо значение, указанное для переменной в разделе инициализации. Все фишки помечены ПИМ создаваемого экземпляра. Если модуль имеет точки взаимодействия, которые делят общую очередь, то в место *Queue*, моделирующее эту очередь, помещается фишка (p, nil), что соответствует пустой очереди, принадлежащей экземпляру модуля `M` с ПИМ p. Место *State* получает фишку (p, s_i), где s_i — начальное состояние модуля, определенное в разделе инициализации, или p, если

[2]Под нулевым подразумевается значение 0 для тех типов данных, где оно присутствует, или значение, минимальное относительно порядка, который определен для соответствующего типа, например: для перечислимого типа таким значением будет первое из перечисленных.

для модуля не определено множество локальных состояний.

И наконец, если в момент своей активизации модуль может порождать экземпляры собственных модулей-наследников, в сети, моделирующей E-переход раздела инициализации, производятся те же действия, которые выполнил охватывающий модуль по отношению к порождаемому, т. е. согласно типам наследников заносятся новые фишки в места T'_inst и T'_init и размечаются места, соответствующие точкам взаимодействия. На этом процесс активизации нового экземпляра завершается.

3.11.2. Уничтожение экземпляра модуля

Уничтожение экземпляра модуля может осуществляться одним из операторов — `release M` или `terminate M`. В результате выполнения одного из них все связи, в которых участвуют точки взаимодействия модуля `M`, разрываются, экземпляр модуля уничтожается. Если уничтожаемый экземпляр содержит вложенные модули, то происходит их рекурсивное уничтожение. Особенность оператора `release M` состоит в том, что его выполнение эквивалентно последовательному выполнению операторов `detach M` и `terminate M`. Однако, как было сказано в п. 3.7.2, в данной работе уничтожение прикрепления происходит с помощью операции `simple-detach`, которая не подразумевает переноса очередей сообщений. Поэтому результат выполнения оператора `release` ничем не отличается от результата выполнения оператора `terminate`, и можно считать, что для уничтожения модулей в спецификации используется только оператор `terminate`.

В сети моделирование уничтожения экземпляра модуля, как и моделирование создания модуля-наследника, происходит на трех уровнях. Пусть `T` — тип модуля `M`, а `B` — тип тела, с которым модуль `M` был создан. На странице, где представлен блок E-перехода, в который входит оператор `terminate`, моделирование уничтожения модуля `M` осуществляется последовательным срабатыванием трех переходов.

Срабатывание первого перехода изымает из места T_inst фишку со значением (p_S, p_M, M, b_M), где p_S — ПИМ модуля, который выполняет оператор `terminate`, а p_M — ПИМ уничтожаемого экземпляра. В место T_kill помещается фишка (p, b_M), которая будет использоваться на следующем уровне. Вторым срабатыванием фишки со значением p_M помещаются в места dis_IP и dch_IP для всех точек взаимодействия модуля `M`. Третий переход может сработать, когда в каждом

месте IP_info будет находиться фишка $(p_M, none, *)$, т. е. все связи, установленные для точек взаимодействия модуля M, будут разрушены. Срабатывание этого перехода изымает фишки, принадлежащие уничтожаемому экземпляру, из мест IP_in, IP_out и IP_info для каждой точки взаимодействия IP модуля M.

На странице, где каждому из определенных для модуля тел соответствует один N-модуль, все происходит полностью аналогично случаю создания модуля. Фишка (p_M, b_M) в месте T_kill заменяется фишкой p_M в месте TB_kill при условии, что $b_M = B$.

Когда фишка p_M появляется в месте TB_kill, фишка p_M возникает также в копии этого места на странице, где представлена структура выбранного тела. На странице присутствует N-модуль, призванный завершить уничтожение экземпляра. Схема сети, которая располагается на странице, связанной с этим N-модулем, показана на рис. 23. Фишки, принадлежащие уничтожаемому экземпляру, изымаются из мест $State$, $Queue$, мест-переменных и мест-параметров.

Затем, если уничтожаемый экземпляр содержит вложенные модули, то для каждого типа наследников T' производятся те же действия, которые выполнил охватывающий модуль по отношению к уничтожаемому. Заметим, что поскольку должно произойти уничтожение всех модулей-наследников типа T', выполнение трех переходов, которые выполняют первую фазу уничтожения модулей, происходит циклически. До тех пор, пока в месте T'_inst остаются фишки, переходы $terminate\_T'$, $unbind\_T'$ и $destroy\_IP$ будут выполняться. После того, как процесс уничтожения будет запущен для всех модулей-наследников, переход $stop\_T$ удалит из сети последнюю фишку, в которой еще присутствует ПИМ модуля M. На этом процесс уничтожения модуля завершается.

3.12. Моделирование такта вычисления

Как было отмечено в п. 2.2, поведение модулей в ходе функционирования Estelle-спецификации определяется текстуальной вложенностью описаний модулей и классами, которые приписаны модулям. Переходы охватывающих модулей имеют приоритет над переходами своих наследников. Классы "(системный) процесс" и "(системная) активность" определяют два возможных способа выполнения: параллельное и последовательное недетерминированное. Поведение системных модулей полностью асинхронно.

В данной работе рассматриваются только спецификации, допуска-

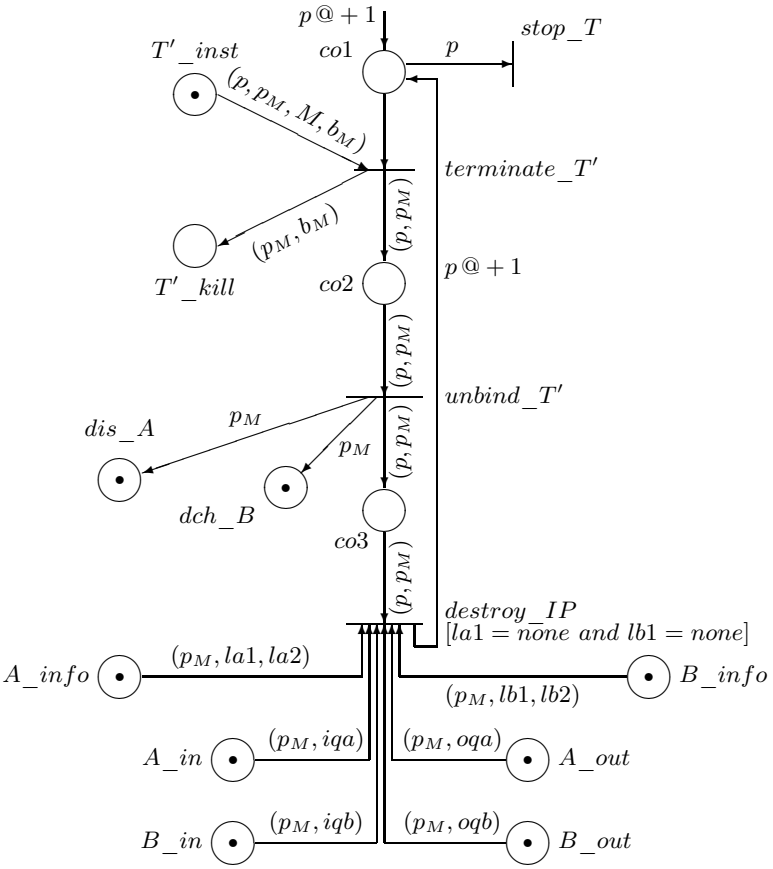

Рис. 23. Моделирование уничтожения экземпляра модуля

ющие последовательное недетерминированное поведение. Другим словами, все системные модули в спецификации должны иметь класс “системная активность” и, согласно правилам классификации модулей, все вложенные в них модули имеют класс “активность”. В этом случае на каждом такте вычисления выполняется только один переход из каждой подсистемы, независимо от того, сколько экземпляров модулей существует на текущий момент.

Правило срабатывания раскрашенных сетей Петри не может гарантировать, что поведение моделирующей сети будет полностью соответ-

ствовать законам, по которым функционирует спецификация. Неделимость выполнения E-перехода в сети обеспечивается местом *State*, являющимся входным для первого перехода каждой сети, которая моделирует блок E-перехода. Возникает конфликт, который исключает возможность параллельного выполнения E-переходов, принадлежащих одному модулю. Однако в процессе выполнения переходов, которые представляют блок некоторого E-перехода, ничто не мешает выполнению переходов сети, моделирующих E-переходы других модулей.

В работе [13] для обеспечения корректности выбора использовались специальные сетевые конструкции, которые связывают модули внутри подсистем. Между собой подсистемы связаны только через места, моделирующие точки взаимодействия. Таким образом, функционирование подсистем остается независимым. Конструкция, которую мы используем в данной работе для организации такта вычисления, аналогична предложенной в работе [13]. Наличие в спецификации динамических конструкций не влияет на организацию такта вычисления, так как при полностью последовательном выполнении из подсистемы для выполнения на следующем такте выбирается всегда ровно один переход.

Для организации такта вычисления на каждой странице сети, представляющей тело системного модуля, создается место *step*. На странице это место связано со всеми N-модулями, которые соответствуют E-переходам системного модуля. Для каждого N-модуля место *step* является входным и выходным, а на странице, связанной с N-модулем, оно связывается с переходами подобно тому, как это происходит для места *State*, т. е. становится входным для служебного перехода *start* и выходным — для перехода *end*. Начальная разметка места *step* — одна бесцветная фишка, несущая временной штамп.

Соединение сетей, моделирующих родителя и его наследников, происходит следующим образом. На уровне тела охватывающего модуля дополнительно создается два места — *serv* и *out* — и переход *next_step*. Кроме того, на каждой странице, представляющей тело некоторого модуля, создается переход *pass*. Чтобы различать места и переходы с одинаковыми именами, их нумеруют. Место *serv* становится входным, а место *out* — выходным для каждого N-модуля, представляющего одного из модулей-наследников. На уровне тела наследника копии мест *serv* и *out* аналогичным образом связываются со всеми N-модулями, соответствующими E-переходам. В сети для E-перехода место *serv* будет входным для перехода *start*, а место *out* — выходным для перехода *end*. Таким образом, места *serv* и *out* гарантируют, что из всех готовых пе-

реходов модулей с общим родителем может выполниться только один переход подобно тому, как место *State* обеспечивает выполнение только одного перехода внутри модуля.

Схема построения конструкции, связывающей модули приведенной ниже спецификации SP, показана на рис. 24. Заметим, что на странице CB присутствует два места *serv*. Одно из них — обозначенное на рисунке как $serv1$ — является копией места, созданного на уровне тела охватывающего модуля. Другое — $serv2$ — впервые появляется на странице CB, наличие фишки в этом месте предоставляет переходам модуля DM возможность быть выполненными на следующем такте вычисления.

```
specification SP;
. . .
module AM systemactivity;
ip P1 : Channel_type(user);
end; {AM}

body AB for AM;
     module BM activity;
     body BB for BM;
     . . .
     end; {BB}

     module CM activity;
     body CB for CM;
         module DM activity;
         body DB for DM;
         . . .
         end; {DB}
     end; {CB}
end; {AB}
end. {SP}
```

Переход *pass* передает право предложить переход для выполнения наследникам модуля. На уровне тела системного модуля переход *pass* перемещает фишку из места *step* в место *serv*. На более низких уровнях этот переход перемещает фишку из места *serv*, относящегося к модулю-родителю, в место *serv*, передающее право выбора перехода наследни-

кам, как это делает переход *pass*2 на странице CB. Если у модуля нет наследников, то переход *pass* перемещает фишку из места *serv* в место *out*, что позволяет избежать тупика, если в модуле не окажется готовых к выполнению E-переходов. Переходы *next_step* возвращают фишку в место *step* после завершения очередного такта.

Таким образом, такту вычисления соответствует в сети следующая последовательность действий. Первым предложить переход для выполнения имеет возможность системный модуль. Выполнение одного из его готовых переходов приводит к началу следующего такта. Однако в языке Estelle модуль может не предлагать для исполнения никакого перехода, даже если среди его переходов есть готовые к выполнению, а передать право выбора перехода для следующего шага выполнения своим наследникам. В сети этому случаю соответствует срабатывание перехода *pass*, которое помещает фишку в место *serv*, связанное с N-модулями, моделирующими непосредственных наследников системного модуля. Наличие фишки в месте *serv* делает возможным выполнение одного из готовых переходов, предлагаемых наследниками. Те в свою очередь могут воспользоваться правом выбора перехода или передать его своим наследникам.

Фишка в месте *out* на каком-нибудь уровне может появиться также в результате выполнения некоторого E-перехода в одном из модулей. Как только это произойдет, может сработать переход *next_step*, связанный с данным уровнем. При его срабатывании фишка перемещается в *out*, которое связано с модулями более высокого уровня, если этот уровень не системный. В противном случае фишка появится в месте *step* в системном модуле, и станет возможным следующий вычислительный такт.

Рассмотрим организацию фазы управления между тактами вычисления. Переход *next_step*, расположенный на уровне тела системного модуля, определяет временной штамп фишки, которая помещается в место *step*. Временной штамп определяется как текущее модельное время плюс некоторое значение из интервала $[\delta, \Delta]$, где δ и Δ соответственно обозначают минимальное и максимальное время, которое предположительно может потребоваться для выполнения E-перехода. Таким образом, гарантируется постоянный прогресс времени в модели. Кроме того, пока фишка в месте *step* остается недоступной, автоматически организуется фаза управления. Это происходит благодаря тому, что фишки, которые участвуют в моделировании потока сообщений, проверке условий возможности E-переходов, а также включении и выключении тай-

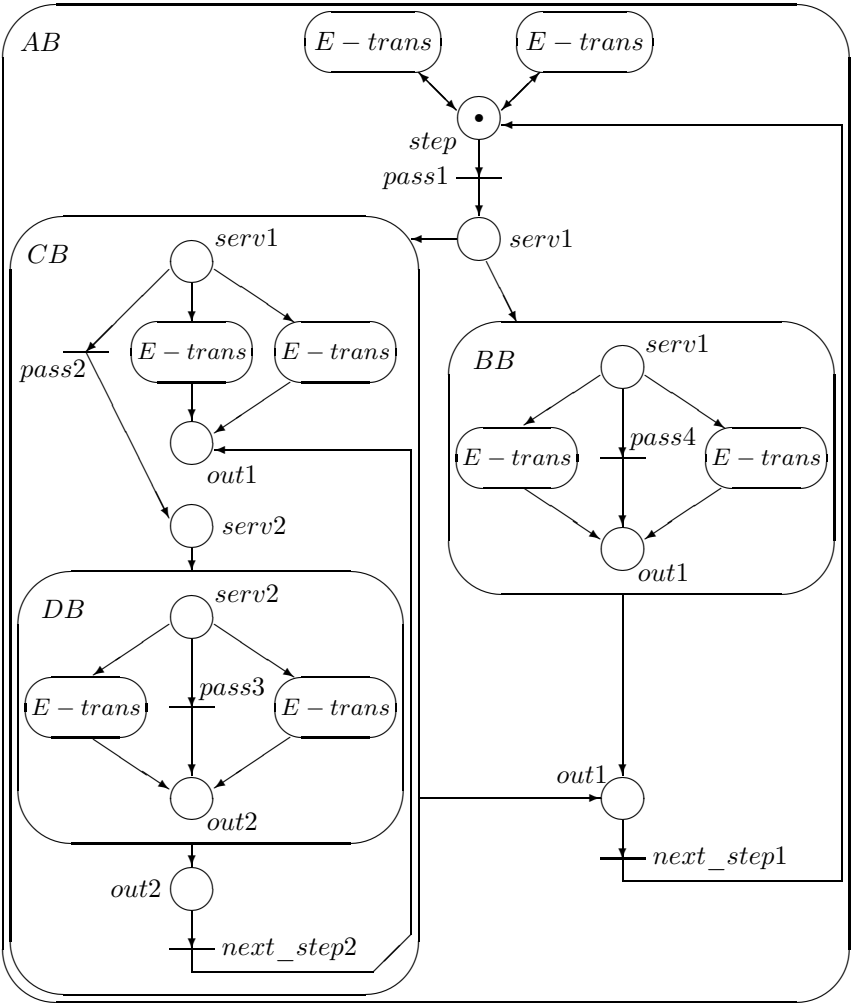

Рис. 24. Организация такта вычисления

меров, не несут временных штампов. Следовательно, использующие их переходы оказываются готовыми на текущий момент модельного времени, тогда как выполнение какого-нибудь E-перехода станет возможно только спустя некоторое время.

Однако может возникнуть тупиковая ситуация, если в некотором модуле не окажется ни одного возможного E-перехода. В этом случае в сети будут срабатывать только переходы, моделирующие условия возможности переходов этого модуля. Время не сможет сдвинуться и начала нового такта не произойдет. Чтобы подобного не случилось, в сети для каждого E-перехода, который не является входным, создается место s_ch — входное место первого перехода подсети для приставки `provided`. Кроме того, место s_ch становится выходным для служебных переходов *end* всех подсетей, которые моделируют E-переходы, принадлежащие тому же модулю, что и рассматриваемый E-переход (см. рис. 25). При начальной разметке место s_ch содержит бесцветную фишку.

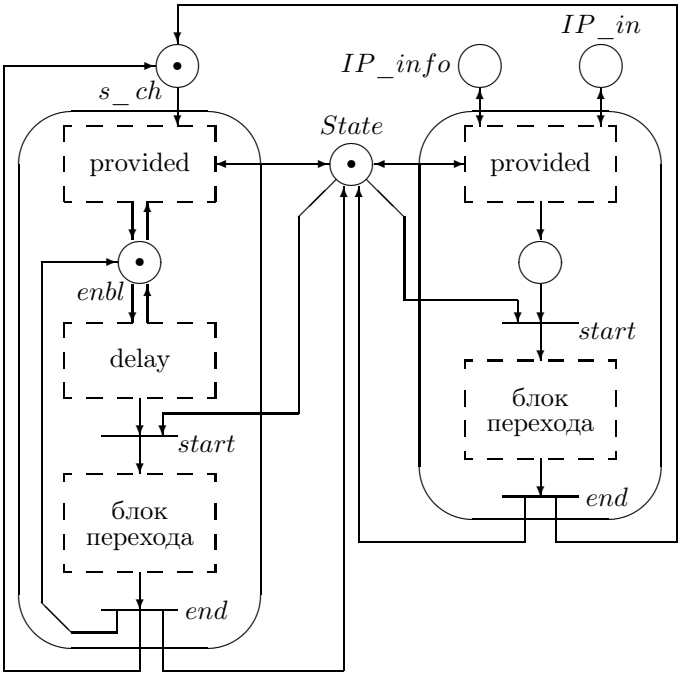

Рис. 25. Моделирование фазы управления

Таким образом, проверка истинности приставки `provided` E-перехода, не являющегося входным, первый раз происходит в начале моделирования спецификации с использованием фишки, содержащейся в ме-

сте s_ch при начальной разметке. Далее, проверка происходит каждый раз после того, как завершается выполнение некоторого E-перехода. Входные E-переходы от бесконечных перепроверок условия возможности ограждают очереди сообщений, ассоциированные с точкой взаимодействия. Если очередь пуста, то проверка условия возможности не сможет начаться, так как пустую очередь невозможно представить в виде $head :: tail$ (см. рис. 13).

3.13. Вид и оценка размера сети

В результате трансляции Estelle-спецификаций, которые не содержат динамических конструкций, получались квазибезопасные сети, т. е. такие, в которых почти все места, за исключением мест, соответствующих массивам и точкам взаимодействия, могут содержать не более одной фишки. Теперь, когда фишки, принадлежащие разным экземплярам модуля, располагаются в одних и тех же местах сети, сеть не может быть безопасной. Однако можно утверждать, что все места, которые относятся к одному модулю, содержат не более одной фишки, принадлежащей конкретному экземпляру. Причем последнее верно также для мест, соответствующих массивам и точкам взаимодействия, так как очереди сообщений и массивы моделируются списками, которые представляются в раскрашенных сетях одной фишкой.

В работе [13] приведена оценка размера неиерархической сети, получающейся при трансляции Estelle-спецификаций, которые не содержат динамических конструкций. В данной работе описывается алгоритм построения иерархической сети, но если рассмотреть эквивалентную неиерархическую сеть, то можно привести аналогичную оценку в динамическом случае.

Независимо от наличия динамических конструкций общая оценка размера сети представляет собой сумму оценок всех модулей, входящих в спецификацию, и оценки построений, организующих такт выполнения. Однако в динамическом случае каждое слагаемое соответствует описанию модуля, т. е. паре заголовок — тело. Если для модуля определено несколько тел, то каждое тело будет представлено отдельной подсетью, размер которой оценивается по приведенной ниже формуле. Дополнительные построения добавляют не более $2 * N$ мест и $2 * N$ переходов, где N — общее число модулей в спецификации. Кроме того, в сети присутствует место PID для генерации персональных идентификаторов модулей.

Рассмотрим оценку размера сети для модуля. Предположим, что модуль Estelle-спецификации содержит var переменных, ip точек взаимодействия, par параметров и t E-переходов. Кроме того, среди общего числа n операторов данного модуля выделим k_1 циклов, k_2 условных операторов, k_3 операторов установления и разрыва связи, k_4 операторов создания и уничтожения модулей, а также k_5 операторов `all` и `forone` и C вызовов процедур и функций. Введем обозначения

$k = k_1 + k_2 + k_3 + k_4 + k_5$ — число операторов, для моделирования которых заведомо требуется более одного перехода, и

$att = var + 8 + 5 * ip + par$ — число "значимых" (не соединительных мест).

Тогда сеть, моделирующая модуль, может максимально иметь TN переходов и PN мест, где

$TN_d = (n + 5 * k) * (C + 1) + 2 * t$,

$PN_d = (n + att + 4 * k) * (C + 1) + 4 * t$.

Если модуль не содержит процедур и функций, то оценка становится линейной.

ЗАКЛЮЧЕНИЕ

Настоящая работа является логическим продолжением работы [13] и предлагает процедуру трансляции Estelle-спецификаций с динамическими конструкциями в раскрашенные сети Йенсена, обогащенные приоритетами. В настоящее время процедура трансляции разработана только для Estelle-спецификаций, допускающих последовательное недетерминированное поведение. Однако даже при этом ограничении возможность использования динамических средств Estelle существенно расширяет класс спецификаций, которые могут быть отображены в раскрашенные сети.

Способ моделирования основан на том, что позиция каждого экземпляра модуля в общей иерархии спецификации остается неизменной, что позволяет использовать структуру сети, статическую по своей природе, для представления иерархической структуры спецификации. Экземпляры модуля моделируются с помощью фишек, число которых изменяется при динамическом создании или уничтожении экземпляров модулей.

Заметим, что хотя доказательств в этой работе не приводится, однако на содержательном уровне поясняется, как семантика конструкций языка Estelle представляется в формальной семантике раскрашенных

сетей Йенсена.

Мне хочется поблагодарить В. А. Непомнящего за постоянное внимание к работе и поддержку. Я также благодарна Т. Г. Чуриной за множество полезных обсуждений и критических замечаний, сделанных по ходу написания этой работы.

СПИСОК ЛИТЕРАТУРЫ

1. Зайцев С. С. Описание и реализация протоколов сетей ЭВМ. — М.: Наука, 1989.
2. Budkowski S., Dembinski P. An introduction to Estelle: a specification language for distributed systems // Computer Networks. — 1988. — Vol. 14. — P. 3—23.
3. Ferenc B., Hogrefe D., Sarma A. SDL with applikations from protocol specification. — Englewood Cliffs, NJ: Prentice Hall, 1991.
4. Information Processing Systems — Open Systems Interaction — ESTELLE: A Formal Description Technique Based on an Extended State Transition Model: Inernational standard. — ISO 9074, 1989.
5. Котов В. Е. Сети Петри. — М.: Наука, 1984.
6. Richier J. L., Rodriguez C., Sifakis J., Voiron J. Verification in XESAR of the Sliding Window protocol // Proc. IFIP Intern. Sympos. on Protocol Specification, Testing and Verification VII. — Amsterdam: North-Holland, 1987. — P. 235—248.
7. Dimitrov V., Petkov A. Verification oriented Estelle specification of communication protocols // Research into Networks and Distributed Applications. — Amsterdam: North-Holland, 1988. — P. 953—960.
8. Lai R., Jirachiefpattana A. Verifying Estelle protocol specifications using numerical Petri nets // Comput. Syst. Sci & Eng. — 1996. — Vol. 11, N 1. — P. 15—33.
9. Lai R., Tsang T. Specification and verification of multimedia synchronisation scenarios using Time-Estelle // Software Practice and Experience, John Wiley and Sons. — 1998. — Vol. 28, N 11. — P. 1185—1211.
10. Fisher J., Dimitrov E. Verification of SDL'92 specifications using extended Petri nets // Proc. IFIP 15th Intern. Conf. on Protocol Specification, Testing and Verification. — Warsaw, Poland, 1995. — P. 455—458.
11. Kettunen E., Montonen E., Tuuliniemi T. An interactive PrT-Net tool for verification of SDL-specifications: Techn. Rep. No. 3. — Helsinki Univ. of Technology, Digital System Laboratory, 1988.
12. Bause F., Kabutz H., Kemper P., Kritzinger P. SDL and Petri net perfomance analysis of communicating systems // Proc. IFIP 15th Intern. Symp. on Protocol Specification, Testing and Verification. — Warsaw, Poland, 1995. — P. 259—272.
13. Верификация Estelle-спецификаций распределенных систем посредством раскрашенных сетей Петри // Непомнящий В. А., Быстров А. В. и др. — Новосибирск, 1998. — 140 с.
14. Churina T. G., Okunishnikova E. V. Coloured Petri nets approach to the validation of Estelle specifications // Proc. of Workshop on Concurrency, Specification and Programming. — Warsaw, Poland, 1997. — P. 25—36.
15. Churina T. G., Okunishnikova E. V. Modelling Estelle specifications using coloured Petri nets //Joint NCC&IIS Bull., Comp. Science. — 1998. — N 8. — P. 19—38.
16. Чурина Т. Г. Способ построения раскрашенных сетей Петри, моделирующих SDL-системы. — Новосибирск, 1998. — 56 с. — (Препр./ РАН. Сиб. отд-ние. ИСИ;

N 56).
17. Чурина Т. Г. Моделирование динамических конструкций языка SDL посредством раскрашенных сетей Петри. — Новосибирск, 2000. — 36 с. — (Препр./ РАН. Сиб. отд-ние. ИСИ; N 71).
18. Jensen K. Coloured Petri nets: A high level language for system design and analysis // Lect. Notes Comput. Sci. — 1991. — Vol. 483 — P. 343—416.
19. Jensen K. Coloured Petri nets: Basic concepts, analysis methods and practical use. — Berlin a. o.: Springer-Verlag, 1996. — Vol. 1. Basic concepts.
20. Jensen K. Coloured Petri nets: Basic concepts, analysis methods and practical use. — Berlin a. o.: Springer-Verlag, 1996. — Vol. 2. Analysis methods.
21. Jensen K. Coloured Petri nets: Basic concepts, analysis methods and practical use. — Berlin a. o.: Springer-Verlag, 1997. — Vol. 3. Practical use.
22. Berthomieu B., Diaz M. Modelling and verification of time dependent systems using time Petri nets // IEEE Transact. on Software Eng. — 1991. — Vol. 17, N 3. — P. 259—273.
23. Kristensen L. M., Christensen S., Jensen K. The practitioner's guide to coloured Petri nets // International Journal on Software Tools for Technology Transfer — 1998. — Vol. 2, N 2. — P. 98—132.

Е. В. Окунишникова

МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКИХ КОНСТРУКЦИЙ ЯЗЫКА ESTELLE ПОСРЕДСТВОМ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ

Препринт
78

Рукопись поступила в редакцию 15.06.2000
Рецензент А. В. Вотинцева
Редактор Л. А. Карева

Подписано в печать 27.06.2000
Формат бумаги 60×84 1/16 Объем 4 уч.-изд.л., 4,3 п.л.
Тираж 50 экз.

НФ ООО ИПО "Эмари" РИЦ, 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6