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

С. А. Черненок, В. А. Непомнящий

АНАЛИЗ MSC-ДИАГРАММ РАСПРЕДЕЛЕННЫХ СИСТЕМ С ПОМОЩЬЮ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ

Препринт
171

Новосибирск 2013

Стандартный язык диаграмм последовательных сообщений MSC применяется для описания сценариев взаимодействия объектов. В частности, он используется для спецификации поведения в системах реального времени и телекоммуникационных протоколах.

В работе рассматриваются все конструкции MSC-диаграмм и их расширений в виде HMSC-диаграмм, за исключением концепций интерпретируемых данных и времени. Описаны алгоритмы трансляции конструкций MSC в раскрашенные сети Петри (CPN). Представлена программная система, реализующая эти алгоритмы. Приведены примеры трансляции, которые анализируются с помощью известной системы CPNTools.

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

V.A. Nepomniaschy, S.A. Chernenok

ANALYSIS OF MESSAGE SEQUENCE CHARTS OF DISTRIBUTED SYSTEMS USING COLORED PETRI NETS

Preprint
171

Novosibirsk 2013

Message Sequence Charts (MSCs) is a graphical description language for interaction scenarios standardized by the International Telecommunication Union (ITU). In particular, it is used for describing communication behaviors in real-time and telecommunication systems.

This paper presents a translation method from MSC and High-level MSC (HMSC) into Coloured Petri Nets and its implementation. Transformation rules for MSC elements cover the entire MSC standard, except data and time concepts. A software system that implements these algorithms is presented. Illustrated examples of translation are analyzed in the known CPNTools system.

1. ВВЕДЕНИЕ

Разработка сложных систем программного и технического обеспечения требует полного и строгого охвата системных требований на этапе проектирования, чтобы исключить возможные ошибки на поздних этапах разработки системы. Для решения этой задачи на практике широко используются методы формальной спецификации, анализа и верификации программных систем.

Для проектирования и разработки распределенных систем с большим набором действующих компонентов необходимо иметь удобные средства для описания взаимодействия каждого из компонентов системы друг с другом. Проблема спецификации требований может осложняться тем, что пользователи и разработчики системы редко пользуются общими методами описания требований, что может вносить некоторую неоднозначность в их трактовке с обеих сторон. Для преодоления этих трудностей используются сценарно-ориентированные или сценарные спецификации, такие как MSC-диаграммы (Message Sequence Charts) [8], UML-диаграммы последовательностей сообщений (UML Sequence Diagrams) [15] и другие виды диаграмм [2, 12]. В этом случае проектировщик системы может задавать системное поведение при помощи набора сценариев. Каждый сценарий представляет собой возможную трассу (последовательность исполнения событий) или набор трасс исполнения системы. Сценарии имеют выразительную графическую форму, интуитивно понятны и поэтому просты в использовании для обычного пользователя без необходимой математической подготовки.

В настоящей работе рассматриваются диаграммы MSC и HMSC в качестве сценарных языков спецификации, которые предназначены для описания взаимодействия объектов в различных системах реального времени и телекоммуникационных протоколах. Данный выбор обусловлен тем, что язык MSC и его расширение HMSC являются полностью стандартизированными (ITU стандарт Z.120), стандарт (H)MSC имеет формально-описанную семантику и простое графическое и текстовое представление, а также используется на практике для представления системных требований на всех этапах проектирования систем [5–7].

Диаграммы MSC и HMSC нашли широкое применение не только в описании коммуникационных протоколов, но и в разработке современного программного обеспечения широкого профиля [13]. Поэтому развитие

методов и средств анализа и верификации моделей протоколов, описанных на языке MSC, представляет безусловный практический интерес.

Тем не менее, для проведения формального анализа и верификации спецификаций, заданных при помощи (H)MSC, полезен хорошо изученный формализм для представления таких спецификаций. Поэтому в данной работе рассматривается преобразование MSC-диаграмм и их расширения HMSC в раскрашенные сети Петри (CPN) [10], для которых существуют инструментальные средства для анализа и верификации свойств сетевых моделей.

Цель данной работы состоит в разработке алгоритмов трансляции (H)MSC-спецификаций в раскрашенные сети Петри, которые охватывают все конструкции стандарта (H)MSC без расширений в виде интерпретируемых данных и времени. Используемые правила трансляции элементов (H)MSC должны иметь минимум ограничений на стандарт MSC для того, чтобы полученная на выходе алгоритмов CPN наиболее точно моделировала заданную (H)MSC-диаграмму. Кроме того, выходная сеть должна быть, по возможности, оптимальных размеров и иметь формат, совместимый с системой CPNTools для применения известных методов и средств анализа и верификации CPN с целью проверки свойств в заданных (H)MSC-спецификациях.

Данная работа состоит из девяти разделов. В разделе 2 представлено краткое описание стандарта MSC. В разделе 3 дается общее описание алгоритма трансляции. Разделы 4–6 содержат подробные описания базовых и структурных элементов (H)MSC-диаграмм и алгоритмов трансляции этих элементов в раскрашенные сети Петри. В разделе 7 описан процесс обработки графа частичного порядка, получаемого на этапе трансляции. В разделе 8 приведен пример трансляции простой HMSC-спецификации в CPN, для которого выполнен анализ некоторых свойств с использованием инструментария CPNTools. Раздел 9 посвящен заключению и краткому описанию дальнейшей работы в исследуемой области.

2. ЯЗЫК MSC-ДИАГРАММ

Диаграммы последовательных сообщений MSC описывают связь между набором системных компонентов (сущностей или процессов), а также связь между этими компонентами и внешним миром (окружением) [8, 9, 16].

Сущность представляет собой объект или экземпляр объекта моделируемой системы. Каждой сущности в MSC-диаграмме соответствует

ось сущности, представляющая собой вертикальную линию и отражающая течение локального времени вдоль оси сверху вниз.

Взаимодействие между сущностями осуществляется при помощи *сообщений*. В стандарте MSC отправка и прием сообщения являются двумя разными асинхронными событиями. Считается, что окружение MSC также может отправлять и получать сообщения, как и сущности.

Если сущность не содержит регионов неупорядоченных событий и встроенных выражений, то все события данной сущности упорядочены вдоль оси сущности, независимо от других сущностей. Установка упорядоченности событий сразу для нескольких сущностей выполняется при помощи сообщений или элементов обобщенного упорядочивания. Никаких других способов упорядочивания событий в MSC не используется. Таким образом, язык MSC устанавливает *частичное упорядочивание* на множестве содержащихся в нем событий.

Помимо событий отправки и приема сообщений, к *базовым элементам* MSC относятся также события создания/удаления сущностей, установки/сброса/истечения таймера, а также шлюзы обмена сообщениями. К *структурным элементам* MSC-диаграмм относятся следующие:

- *области неупорядоченных событий* (*coregions*) устанавливают неупорядоченное исполнение для заданного подмножества событий конкретной сущности;
- *встроенные выражения* (*inline expressions*) позволяют задавать различные способы композиции (например, параллельную или альтернативную композицию) для заданного набора событий внутри MSC-диаграммы;
- *ссылочные выражения* (*reference expressions*) используются внутри одной MSC-диаграммы для ссылки на другие MSC-диаграммы;
- *HMSC-диаграммы* (*High-level MSC*) представляют возможности для наглядного представления способов синтеза и композиции нескольких MSC-диаграмм в одну.

Более подробное описание перечисленных структурных конструкций будет дано в разделе 5 и 6.

Рассмотрим пример использования диаграмм MSC и HMSC, графическое представление [18] которых приведено на рис. 1.

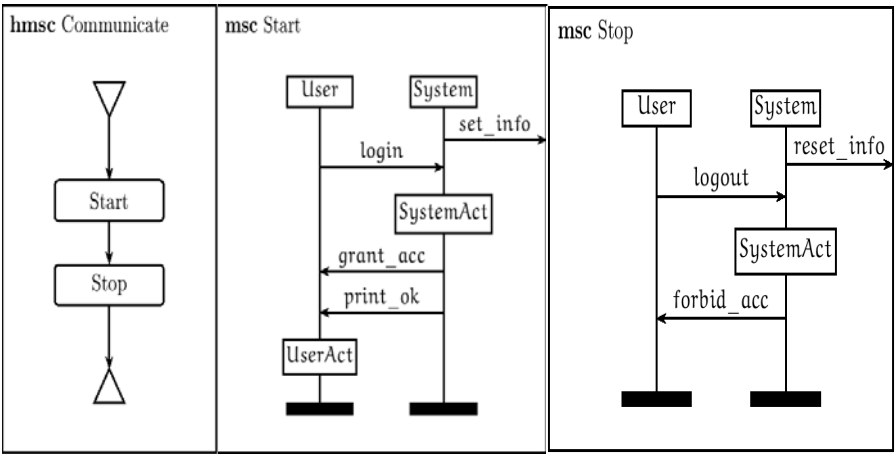

Рис. 1. Графическое представление HMSC-диаграммы *Communicate* и (ссылочных) MSC-диаграмм *Start* и *Stop*

Пример текстового представления для MSC-диаграммы *Start* и HMSC-диаграммы *Communicate* приведен ниже.

msc Start;

User : **instance**;
System : **instance**;

System : **out** set_info **to env**;
User : **out** login **to** System;
System : **in** login **from** User;
System : **action** 'SystemAct';
System : **out** grant_acc **to** User;
User : **in** grant_acc **from** System;
System : **out** print_ok **to** User;
User : **in** print_ok **from** System;
User : **action** 'UserAct';

User : **endinstance**;
System : **endinstance**;
endmsc;

msc Communicate;

expr L_Start;
L_Start : Start **seq** (L_Stop);
L_Stop : Stop **seq** (L_End);
L_End : **end**;

endmsc;

Представленный синтаксис текстового описания является *событийно-ориентированным*, согласно которому набор событий разных сущностей представляется в виде единого списка, например, в порядке появления событий на диаграмме, как это отображено в приведенном примере.

MSC-диаграммы изображаются при помощи рамки, которая содержит графическое представление сущностей описываемой системы, а также имя MSC, задаваемое ключевым словом `msc` и расположенное в верхней части диаграммы.

В MSC-диаграмме *Start* на рис. 1 описано взаимодействие сущностей *User* и *System*, которые представлены вертикальными линиями. Горизонтальные блоки на концах линий обозначают начало и конец описания функционирования сущностей. Названия сущностей должны быть уникальны внутри одной MSC-диаграммы.

Метки *SystemAct* и *UserAct*, помещенные в прямоугольный блок и привязанные к сущностям *System* и *User*, обозначают локальные действия или внутреннюю активность данных конкретных сущностей. Действие *SystemAct* описывает лишь некоторое событие сущности *System*, которое видно извне, но внутренняя логика поведения которого скрыта.

Обмен сообщениями между двумя сущностями изображается при помощи стрелки, исходящей из сущности, которая отправляет сообщение, и входящей в сущность, которая принимает сообщение. На диаграмме *Start* сущность *User* отправляет сообщение *login* сущности *System*, а также принимает сообщения *grant_acc* и *print_ok* от сущности *System*.

Помимо обмена сообщениями между сущностями, существует возможность отправки или приема сообщений от окружения («внешнего мира») MSC-диаграммы. В этом случае стрелки сообщений могут исходить или входить в обрамляющую область диаграммы. Примером такого вида сообщения является сообщение *set_info*, отправленное сущностью *System* в окружение MSC-диаграммы *Start*. Такой вид сообщений может использоваться для обмена информацией между несколькими MSC-диаграммами или структурными элементами MSC, осуществляемого при помощи шлюзов обмена сообщениями.

Все сущности действуют независимо друг от друга, за исключением событий обмена сообщениями, в которых событие приема сообщения должно предшествовать событию его отправки. Из этого следует, что на MSC-диаграмме *Start* сущность *System* может отправить сообщение *grant_acc*, и затем, не дожидаясь его получения сущностью *User*, следом отправить *print_ok*, после чего закончить свое исполнение событий. При этом *User*, отправив сообщение *login*, не может продолжить дальнейшее исполнение

события приема сообщения *grant_acc*, пока сущность *System* не выполнит событие отправки сообщения *grant_acc*.

Отметим, что MSC-диаграммы *Start* и *Stop* на рис. 1 не только описывают события, которые могут быть выполнены сущностями, но также содержат информацию о возможных последовательностях исполнения событий.

Изображенная на рис. 1 HMSC-диаграмма *Communicate* описывает связь между MSC-диаграммами *Start* и *Stop*, представленными в виде ссылок на MSC. Начало и конец диаграммы помечен специальными символами в виде треугольников. Каждая HMSC-диаграмма может иметь только один стартовый символ, причем, согласно статическим ограничениям стандарта, каждый узел внутри HMSC должен быть доступен из стартового символа.

Направленные дуги в HMSC-диаграмме отражают последовательность исполнения между связанными узлами. Если существует более чем одна исходящая из узла дуга, то связанные такими дугами узлы будут являться альтернативными ветвями данной HMSC. Как видно по HMSC-диаграмме *Communicate*, исполнение этой диаграммы начинается с передачи управления ссылочному выражению *Start*, что означает исполнение всех событий из MSC-диаграммы *Start*, после чего выполняется передача управления ссылке *Stop* и исполнение всех событий из MSC-диаграммы *Stop*.

В стандарте MSC определено, что связь всех структурных элементов внутри диаграмм осуществляется при помощи *слабой последовательной композиции (weak sequential composition)*. Это означает, что в HMSC-диаграмме *Communicate* сущность *User* может после выполнения всех своих событий из диаграммы *Start* перейти к выполнению своих событий из диаграммы *Stop*, не дожидаясь исполнения остальных событий диаграммы *Start* другими сущностями. Такое возможно по причине того, что ссылочные выражения *Start* и *Stop* слабо связаны между собой.

Отметим, что в данной работе расширения стандарта MSC, включающие понятия интерпретируемых данных и исчисляемого времени, не рассмотрены. Также в рассмотрение не вошли редко используемые конструкции шлюзов упорядочения (order gates) и декомпозиции сущностей MSC-диаграмм.

3. ОБЗОР АЛГОРИТМА ТРАНСЛЯЦИИ MSC-ДИАГРАММ В CPN

В данном разделе будет рассмотрено общее описание алгоритма трансляции элементов (H)MSC-диаграмм в CPN.

3.1. Входные данные

Входными данными алгоритма трансляции будем считать диаграмму MSC или HMSC, удовлетворяющую следующим требованиям:

- все входные диаграммы должны быть корректны по отношению к синтаксису и статическим требованиям, изложенным в стандарте ITU Z.120. Статические требования для каждого типа элемента (H)MSC-диаграммы приведены в последующих разделах в подробном описании правил трансляции этих элементов;
- все сообщения в MSC-диаграмме должны быть уникальны. Это означает, что каждое сообщение в диаграмме может быть однозначно определено по имени сообщения и именам сущностей, которые являются адресами отправки и приема данного сообщения;
- если входная диаграмма содержит ссылки на другие MSC-диаграммы, то эти диаграммы должны существовать (файлы этих MSC-диаграмм должны находиться в той же директории, что и основной файл MSC).

Внутренним представлением входной (H)MSC-диаграммы является *граф частичного порядка*, представляющий собой ориентированный граф с узлами-событиями (H)MSC, описывающий структурно возможные трассы исполнения в данной (H)MSC-диаграмме. Таким образом, алгоритм трансляции можно рассматривать как перевод одного ориентированного графа в другой.

3.2. Выходные данные

Выходными данными алгоритма трансляции является раскрашенная сеть Петри, заданная при помощи xml-файла, совместимого с входным форматом системы CPNTools. В данной работе используются определения раскрашенных сетей Петри, приведенные в [10].

Отметим, что полученная на выходе алгоритма CPN не является временной сетью. Кроме того, она будет иерархической в том случае, если исходная спецификация была задана при помощи HMSC-диаграммы, либо если входная MSC-диаграмма содержала ссылочные выражения.

3.3. Общее описание алгоритма трансляции

Алгоритм трансляции по входной MSC-диаграмме или HMSC с набором связанных с ней MSC-диаграмм строит CPN, которая позволяет промоделировать все возможные пути исполнения событий MSC-диаграммы. Другими словами, множество всех возможных трасс MSC-диаграммы будет совпадать с множеством всех возможных последовательностей событий (срабатываний переходов) CPN, полученной в результате трансляции. Последнее утверждение будет верно для всех конструкций MSC, за исключением элемента альтернативной композиции со свойством нелокального выбора (см. раздел 5.2.2).

Алгоритм состоит из следующих трех этапов:

Этап 1: Построение графа частичного порядка. Для каждого события в HMSC/MSC-диаграмме создается узел в графе частичного порядка. Данный узел хранит различную информацию: множество предшествующих и последующих узлов, название сущности и номер структурной области, к которой принадлежит данное событие. Таким образом, построенный на первом этапе граф частичного порядка содержит узлы, которые связаны друг с другом так же, как связаны события в HMSC/MSC-диаграмме — на основании заданных диаграммой статических ограничений на возможные пути следования событий друг за другом.

Этап 2: Обработка и оптимизация графа частичного порядка. В процессе обработки происходит создание вспомогательных узлов графа, выполняется удаление лишних узлов и связей, а также может быть выполнена свертка повторяющихся фрагментов MSC-диаграммы. Более подробно процесс обработки и оптимизации графа частичного порядка описан в разделе 7.

Этап 3: Трансляция графа частичного порядка в CPN. Полученную сеть можно охарактеризовать следующим образом:

- каждому узлу графа частичного порядка соответствует переход в CPN;
- каждой дуге графа частичного порядка соответствует место в CPN;
- связь любых двух переходов в CPN осуществляется при помощи места и двух ориентированных дуг. Ориентация дуг совпадает с ориентацией дуги, соединяющей соответствующие переходам узлы в графе частичного порядка;
- исполнению события в MSC-диаграмме соответствует срабатывание соответствующего перехода и перемещение фишек в строящейся сети Петри;

- начальным точкам MSC-диаграммы соответствуют места *start* сети Петри с начальной разметкой.

Отметим, что правила трансляции для большинства элементов (H)MSC действуют независимо от других элементов графа частичного порядка этой диаграммы. Далее более подробно опишем правила трансляции для каждого элемента (H)MSC-диаграммы.

4. ПРЕОБРАЗОВАНИЕ ОСНОВНЫХ ЭЛЕМЕНТОВ MSC-ДИАГРАММ В CPN

В данном разделе будет рассмотрено описание правил трансляции для основных элементов MSC-диаграммы. К набору основных элементов стандарта MSC относятся следующие: сообщения, локальные события, условия, таймеры, события создания и завершения сущностей, элементы упорядочения событий и шлюзы.

4.1. Базовые элементы MSC

Трансляция базовых элементов MSC-диаграммы относится к простейшему случаю среди представленных алгоритмов трансляции. Каждый базовый элемент в строящейся сети моделируется одним переходом и одним местом CPN, без необходимости введения вспомогательных переходов. Ниже дано описание каждого базового элемента в отдельности.

Событие отправки/приема сообщения отображает акт обмена сообщениями между двумя сущностями. Сообщение может быть отправлено или принято конкретной сущностью или окружением (*env*) MSC-диаграммы. Также сообщение может иметь статус утерянного (*lost*) или обнаруженного (*found*) в том случае, когда информация о получателе или отправителе неизвестна.

На рис. 2 представлены: обычные сообщения *get2*, *data* и *content*; сообщение *get1*, отправленное сущностью *Client* и утерянное в процессе передачи; сообщение *status*, отправленное процессом *DataGen* во внешнее окружение MSC-диаграммы.

Стандартом MSC предусмотрено только единственное ограничение на сообщения: все сообщения асинхронны, и каждому событию приема сообщения предшествует событие его отправки.

На первой стадии трансляции каждое событие отправки/приема и утери/обнаружения сообщения переводится в узлы графа *Out_msg*/*In_msg* и *Lost_msg*/*Found_msg* соответственно. Согласно ограничениям на сообщения, событие *Out_msg* соединяется направленной дугой с *In_msg* (см. узлы, соответствующие сообщениям *get2*, *data* и *content*, на рис. 3). На третьей стадии трансляции узлы графа, относящиеся к событиям обмена сообщениями, преобразуются в переходы сети Петри. Порядок следования узлов типа *Out_msg*/*In_msg* соблюдается при помощи мест, которые связывают соответствующие этим узлам переходы и гарантируют правильный порядок между их срабатываниями.

Локальные события (*actions*) представляют собой неформальное описание внутренней активности для заданной сущности. Никаких дополнительных ограничений на данный элемент стандарт MSC не предусматривает. Пример локальной активности изображен на рис. 2 в виде события '*DataGeneration*' процесса *DataGen*.

На первой стадии трансляции локальные события переводятся в узлы типа *Act* (см. узел '*DataGeneration*' на рис. 3). Каждый узел типа *Act* преобразуется в переход CPN на третьей стадии трансляции.

Событие создания новой сущности (*Instance creation*) представляет собой динамическое создание новой сущности, осуществляемое другой сущностью. Данное событие похоже на событие отправки сообщения, передача которого приводит к созданию новой сущности. Каждой создаваемой сущности может соответствовать только одно событие создания сущности. Кроме этого, никакие сообщения не могут быть отправлены/приняты сущностью до ее создания.

На рис. 2 показан пример создания нового процесса *DataGen*, который порождается сущностью *Server*. Процесс создается путем отправки сообщения *create* от порождающей сущности к порождаемой сущности.

Трансляция событий создания новой сущности аналогична трансляции событий обмена сообщениями, где событию отправки сообщения соответствует событие создания сущности с типом *Create*, а событию приема сообщения соответствует самое первое событие созданной сущности (см. переходы *Create_DataGen* и *Act_*'*DataGeneration*' на рис. 4). Таким образом, узел *Create* предшествует всем узлам, принадлежащим созданной сущности.

Событие завершения сущности (*Instance stop*) прекращает выполнение созданной сущности. Каждой созданной сущности может соответствовать только одно событие завершения сущности. Кроме этого, никакие сообщения не могут быть отправлены/приняты сущностью после ее завершения.

На рис. 2 событию завершения созданного процесса *DataGen* соответствует перечеркнутая конечная точка оси данного процесса, которая означает завершение жизненного цикла данного процесса.

Трансляция событий завершения созданной сущности аналогична трансляции локальных событий, где локальное событие заменяется на событие завершение сущности с типом *Stop* (см. переход *Stop_DataGen* на рис. 4), причем узел *Stop* графа частичного порядка является конечным узлом данной сущности.

События установки, сброса и истечения таймера носят чисто символический характер, поскольку в данной работе понятие изменяемого времени не рассматривается. Тем не менее, такие события без привязки к динамически изменяемому времени часто встречаются в реальных MSC-спецификациях, поэтому их рассмотрение не лишено смысла.

Каждому событию установки таймера в MSC (*starttimer*) должно соответствовать либо событие истечения таймера (*timeout*), либо событие остановки таймера (*stoptimer*). Кроме того, данные события могут использоваться независимо друг от друга — в случае, когда создание и завершение таймера расположены в разных MSC-диаграммах.

На рис. 2 на оси сущности *Server* изображено событие установки таймера *T1* (в виде символа песочных часов) и соответствующее ему событие истечения времени (тайм-аут). С данными событиями никаких временных характеристик не связано.

При трансляции на первой стадии событиям установки, сброса и истечения таймера ставятся в соответствие узлы с типами *set*, *reset* и *timeout*. Алгоритм трансляции данных событий схож с трансляцией событий отправки/приема сообщения с той лишь разницей, что вместо события передачи сообщения используется событие типа *set*, а вместо события приема сообщения — события типа *reset* и *timeout* (см. переходы *StartTimer_T1* и *StopTimer_T1* на рис. 4).

Шлюзы обмена сообщениями предназначены для передачи сообщений между несколькими MSC-диаграммами.

Шлюзы отправки/приема сообщений преобразуются в узлы графа частичного порядка типа *GateIn* и *GateOut*, которые содержат дополнительную информацию в виде имени шлюза. При трансляции графа в CPN для всех переходов типа *GateIn* с одинаковыми именами шлюзов устанавливается общее входное место, а для всех переходов типа *GateOut* с одинаковыми именами шлюзов — общее выходное место. Если имя шлюза не задано явно, то создается уникальное место для данного перехода. В примере на рис. 2 для сообщения *status*, которое передается в окружение

MSC-диаграммы, не указано имя шлюза отправки сообщения. Поэтому в результате трансляции данного сообщения создается переход *GateIn_status* и уникальное выходное место *gate_status* для этого перехода, которое не является входным ни для одного перехода, т.к. выходной шлюз не определен.

MSC-диаграмма, использующая приведенные в данном разделе элементы, представлена на рис. 2. Ее граф частичного порядка изображен на рис. 3.

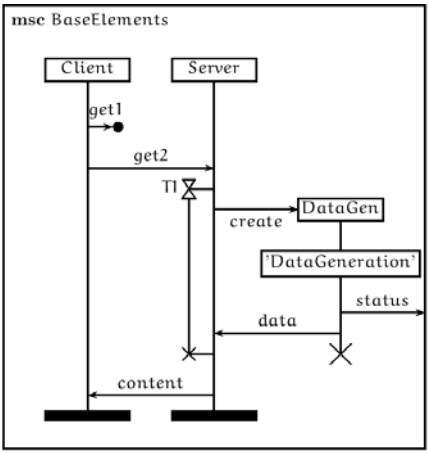

Рис. 2. Пример MSC-диаграммы с базовыми элементами

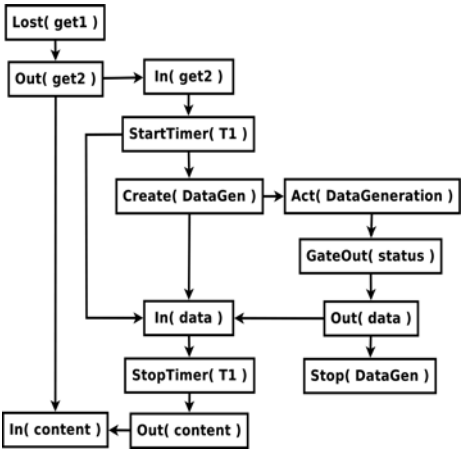

Рис. 3. Граф частичного порядка MSC-диаграммы на рис. 2

CPN, полученная в результате трансляции диаграммы из рис. 2, представлена на рис. 4. Начальной точке MSC-диаграммы соответствует место *start*. Конечным точкам MSC-диаграммы соответствуют места *end*. Таким образом, по описанным выше алгоритмам можно сказать, что каждое событие из базового набора элементов MSC моделируется ровно одним переходом в CPN. При этом в результирующей сети сохраняется частичная упорядоченность, заданная в исходной MSC-диаграмме.

Заметим, что в CPN на рис. 4 используется единственный тип фишек, который может быть произвольным, поскольку не несет никакой дополнительной информации и отражает лишь поток управления в рассматриваемой системе. В данной работе для всех мест, не относящихся к

элементам циклического исполнения, устанавливается по умолчанию следующий строковый тип для CPN:

colset MscMsg = string;

Переменная *d*, являющаяся выражением на дугах в сети на рис. 4, имеет тип *MscMsg*. Место *start* снабжается начальной разметкой: «1` "*msg*"».

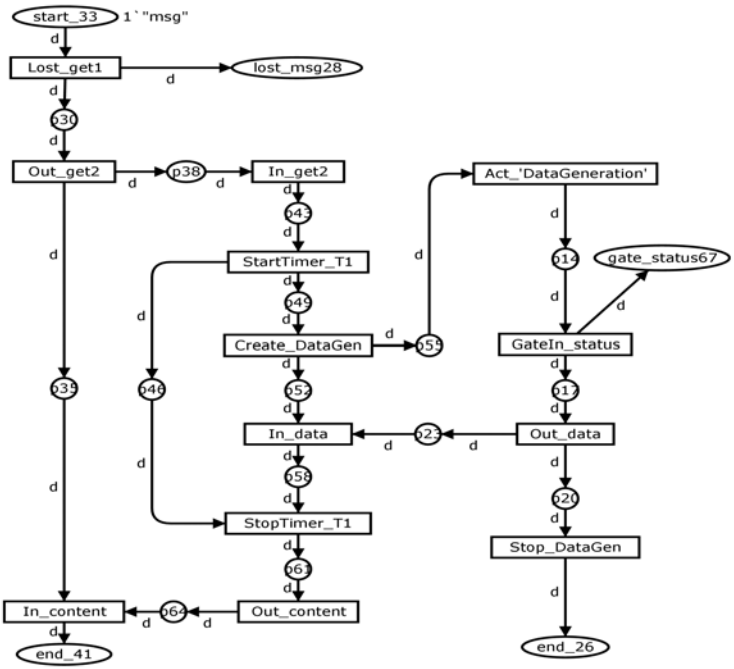

Рис. 4. Результат трансляции MSC-диаграммы из рис. 2

4.2. Условия в MSC

В стандарте MSC начиная с 2000 года введен элемент «условие» двух видов: условия для установки состояния заданных сущностей (*setting conditions*) и проверочные или охранные условия (*guarding conditions*).

Установочное условие выполняет установку текущего состояния меток тех сущностей, для которых оно задано. В диаграммах MSC и HMSC установочное условие обозначается пятиугольником, содержащем список

меток, который оно устанавливает. Например, на рис. 5 установочное условие задается на множестве сущностей *i1* и *i2* со значением *COND*. Следовательно, для данной MSC-диаграммы множество меток {*i1, i2*} будет установлено в значение {*COND*}. Таким образом, при помощи установочных условий можно задавать или изменять текущее состояние описываемой системы.

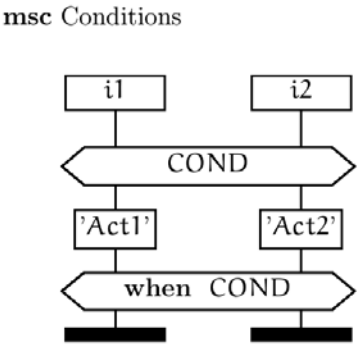

Рис. 5. Пример MSC-диаграммы с установочным и охранным условием *COND*

В MSC-диаграмме можно иметь разное множество меток для различных множеств сущностей в одной и той же точке исполнения диаграммы. Другими словами, каждое множество сущностей уникально определяется по именам сущностей, входящих в данное (неупорядоченное) множество. При этом повторное изменение одного и того же множества меток сущностей приведет к удалению предыдущего значения. В данной работе установочное условие на нескольких сущностях играет роль синхронизирующей конструкции, означающей, что каждая сущность из множества сущностей установочного условия должна выполнить все свои события перед установкой данного условия.

Охранные условия динамически, в процессе исполнения диаграммы, проверяют текущее состояние системы на наличие заданного значения. Точнее, данный тип условий проверяет равенство значений для набора меток, перечисленных в данном условии, и значений из множества меток, установленных для сущностей, участвующих в охранном условии. Если проверочное условие ложно, то продолжение исполнения событий диаграммы, следующих ниже охранного условия и принадлежащих сущностям, входящим в данное охранное условие, прекращается. Охранное

условие на нескольких сущностях не является синхронизирующей конструкцией, т.е. проверка условия для какой-либо сущности осуществляется не одновременно с другими сущностями данного охранного условия. В MSC-диаграмме охранное условие имеет такой же вид, что и установочное условие, за исключением того, что в охранном условии предусмотрено использование ключевого слова when перед списком меток, которые должны быть проверены. На рис. 5 охранное условие имеет вид «when *COND*» и осуществляет проверку того, содержит ли множество сущностей {*i1, i2*}, участвующих в охранном условии на диаграмме, значение {*COND*}. На MSC-диаграмме в примере данное охранное условие будет истинным, и следовательно, будут доступны все события сущностей *i1* и *i2*, расположенные ниже охранного условия.

Каждое установочное и охранное условие, заданное на *N* сущностях, на первой стадии трансляции переводится в один узел типа *SetCond* и в *N* узлов типа *GuardCond* графа частичного порядка. Данные узлы содержат информацию о наборе сущностей, для которых они установлены, а также информацию о наборе меток, указанном в условии.

При трансляции условий в CPN для каждого узла графа с условием создается совмещенное место *"CondVars"* (тип *fusion place* в терминах CPNTools) для хранения установленных меток системы. Это место хранит список кортежей вида "([*i1,..,in*], [*c1,..,cm*])", где "[*i1,..,in*]" – список сущностей и "[*c1,..,cm*]" – список меток условия. Место *"CondVars"* помечается цветовым типом *CONDLIST*:

colset CONDLIST = list COND;
*colset COND = product NAMELIST * NAMELIST;*
colset NAMELIST = list MscMsg;

Для использования списка установленных меток системы вводится переменная *conds:*

var conds: CONDLIST;

Далее в зависимости от типа, — *SetCond* или *GuardCond*, — выполняются следующие действия:

- узел типа *SetCond* преобразуется в соответствующий переход, который имеет входное и выходное место *"CondVars"*. Входная дуга данного места помечается функцией *set* (см. приложение), а выходная дуга — переменной *conds*. При срабатывании данного перехода происходит извлечение фишки из места *"CondVars"*, которая содержит список меток системы, и возвращение фишки с обновленным списком меток обратно в это же место. Обновление

списка меток системы происходит при помощи функции *set*, которая извлекает старое значение кортежа, соответствующего набору сущностей данного условия, и заменяет его новым кортежем с метками установочного условия.

- узел типа *GuardCond* преобразуется в соответствующий переход со спусковой функцией *guard* (см. приложение), который имеет входное и выходное место *"CondVars"*. Входная и выходная дуги данного места помечаются переменной *conds*. Функция *guard* принимает на вход кортеж со списками сущностей и меток, который должен быть проверен охранным условием. *guard* возвращает истинное значение, если кортеж, являющийся входным аргументом функции, будет содержаться во множестве установленных меток системы.

Фрагменты CPN, соответствующие условиям MSC приведены на рис. 6. Каждому установочному условию MSC соответствует один переход в CPN; каждому охранному условию MSC, заданному на *N* сущностях, соответствует *N* переходов в CPN.

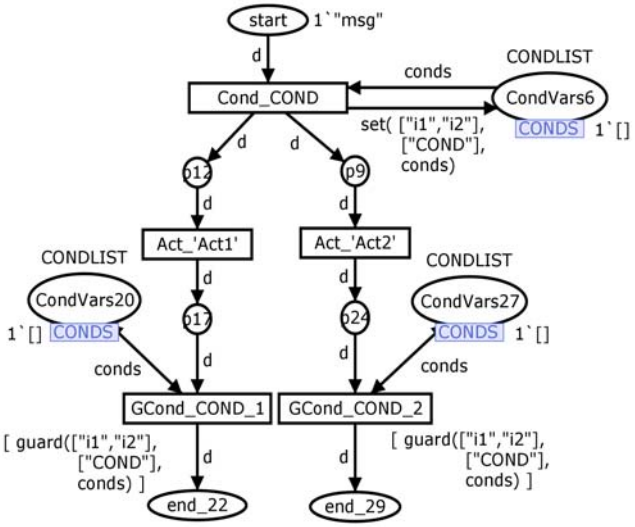

Рис. 6. Результат трансляции MSC-диаграммы с условиями из рис. 5

Как видно из рис. 6, срабатывание перехода *Cond_COND*, соответствующего установочному условию *COND* в MSC-диаграмме, приведет к обновлению фишки с глобальным списком меток системы,

которая хранится в местах *CondVars*. Значение данной фишки при помощи функции *set*([*"i1"*,"*i2*"], ["*COND"*], *conds*) будет установлено в 1`[(["*i1*","*i2*"],"*COND*"])]. Срабатывание любого из переходов *Gcond_COND*, соответствующих охранному условию «`when` *COND*» MSC-диаграммы, извлечет фишку из места *CondVars* для проверки условия при помощи спусковой функции *guard*(["*i1*","*i2*"], ["*COND*"],*conds*). Данная функция вернет значение *true* в том случае, если список текущего состояния меток системы, полученный из места *CondVars*, будет содержать кортеж (["*i1*","*i2*"],["*COND*"]). В результате переходы *Gcond_COND* станут доступными для исполнения только в том случае, если их спусковые функции будут истинны. Отметим, что по причине отсутствия у охранных условий свойства синхронизации в результирующей CPN на рисунке изображены два перехода *Gcond_COND* вместо одного.

4.3. Элементы упорядочения событий в MSC

Элементы упорядочения событий MSC, такие как общее упорядочение и регионы неупорядоченных событий, позволяют задать дополнительный порядок следования элементов MSC, не предусмотренный начальными ограничениями.

Регион неупорядоченных событий (*Coregion*) может быть установлен для конкретной сущности и содержать как минимум одно событие этой сущности. Область неупорядоченных событий в MSC-диаграмме изображается пунктирной линией на оси сущности. На рис. 7 регион неупорядоченных событий установлен для сущности *i1* и включает в себя события отправки сообщения *m1*, приема сообщения *m2* и локального действия *'Local'*. Это означает, что перечисленные события могут быть выполнены сущностью *i1* в произвольном порядке.

Начало и конец региона неупорядоченных событий переводятся в узлы *Coregion_Begin* и *Coregion_End* соответственно. В остальном алгоритм трансляции данного элемента совпадает с алгоритмом трансляции элемента параллельного исполнения, заданного на одной сущности (см. раздел 5.3).

Общее упорядочение (*General Ordering*) позволяет устанавливать дополнительный порядок между событиями, который не определен MSC-диаграммой по умолчанию. Графически дополнительный порядок изображается при помощи пунктирной стрелки, отражающей направление потока управления в данной диаграмме. В текстовом представлении для этих же целей предусмотрены ключевые слова `before` и `after`. На рис. 7 установлен следующий дополнительный порядок событий: *«out_m1* `before`

Local» и *«Local* `after` *out_m2»*. Данные отношения означают, что событие отправки сообщения *m1* сущностью *i1* должно произойти перед локальным событием *'Local'* сущности *i1*, а локальное действие *'Local'*, в свою очередь, может произойти только после события отправки сообщения *m2* сущностью *i2*.

Установка общего упорядочения и *шлюзов упорядочения* (*Order Gates*) не имеет представления в виде узлов графа частичного порядка, а лишь задает связь между узлами. Связанные узлы транслируются по тем же правилам, что и для базовых элементов MSC. Как было отмечено ранее, трансляция шлюзов упорядочения в данной работе не рассмотрена.

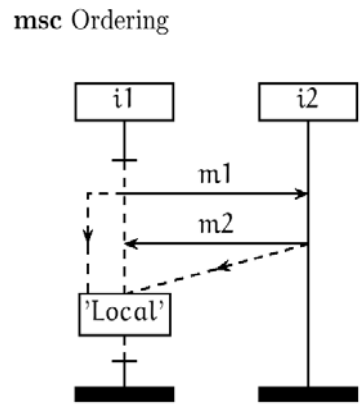

Рис. 7. Пример MSC-диаграммы с регионом неупорядоченных событий и дополнительным упорядочением сообщений (изображены пунктирными линиями)

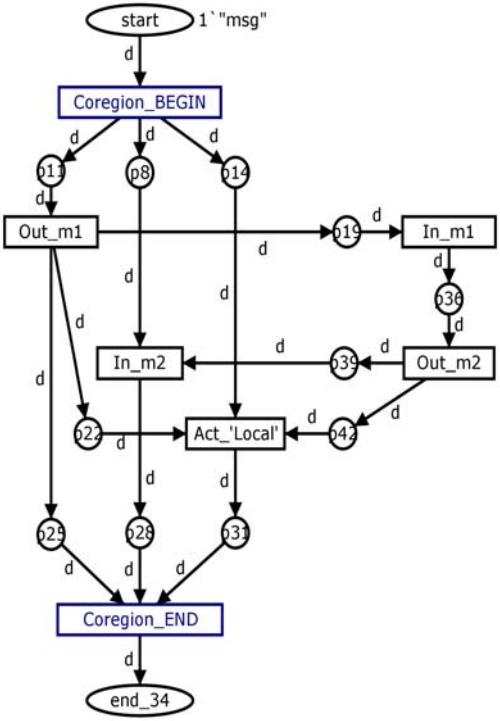

Рис. 8. Результат трансляции MSC-диаграммы с элементами упорядочения событий, изображенной на рис. 7

Фрагменты CPN, соответствующие обобщенному упорядочению и неупорядоченным регионам, приводятся на рис. 8. Как видно из рисунка, каждый регион неупорядоченных событий MSC моделируется двумя вспомогательными переходами в CPN (переходы *Coregion_Begin* и *Coregion_End*).

5. ПРЕОБРАЗОВАНИЕ ВСТРОЕННЫХ ВЫРАЖЕНИЙ MSC-ДИАГРАММ В CPN

Встроенные выражения (*inline expressions*) позволяют описывать композицию наборов событий (по-другому, анонимных MSC-диаграмм) внутри MSC. Встроенные выражения в MSC-диаграмме изображаются графически при помощи прямоугольной рамки, обрамляющей определенный набор сущностей (как минимум одну сущность). Операнды встроенного выражения отделяются друг от друга при помощи пунктирной линии. В левом верхнем углу рамки указывается тип встроенного выражения при помощи одного из следующих доступных ключевых слов: `seq`, `alt`, `opt`, `exc`, `par` и `loop`. Далее будет подробно рассмотрен каждый из перечисленных типов встроенных выражений, представляющий отдельный вид композиции.

5.1. Последовательная композиция

Данная композиция является фундаментальной композицией для MSC, т.к. связь между различными элементами MSC по умолчанию осуществляется при помощи последовательной композиции. В стандарте MSC последовательная композиция является *слабой*. Последовательная композиция может задаваться явно при помощи ключевого слова `seq` только в ссылочных выражениях MSC.

5.2. Альтернативная композиция

Альтернативная композиция в MSC-диаграмме может быть представлена при помощи встроенного или ссылочного выражения `alt`. Данный вид композиции, заданный для нескольких операндов, позволяет выполнить только один из операндов, исключая выполнение других.

Пример элемента альтернативного выбора (встроенного выражения `alt`) изображен на рис. 9. В данной диаграмме локальное событие *Stop* и события отправки и приема сообщения *Interrupt* являются взаимоисключающими с событиями отправки и приема сообщения *Continue*.

Определение. *Стартовыми событиями* MSC-диаграммы или операнда встроенного выражения MSC будем называть события, которые могут быть выполнены первыми (доступны для исполнения в первую очередь) среди набора событий, принадлежащих данной MSC-диаграмме или операнду встроенного выражения MSC.

Как следует из определения, стартовыми событиями MSC-диаграммы могут быть только события отправки сообщения, локального действия, создания новой сущности и установки таймера.

По аналогии со стартовыми событиями будем определять также и *конечные события* MSC. К ним относятся такие события, которые завершают каждую сущность внутри данной MSC-диаграммы или операнда встроенного выражения MSC.

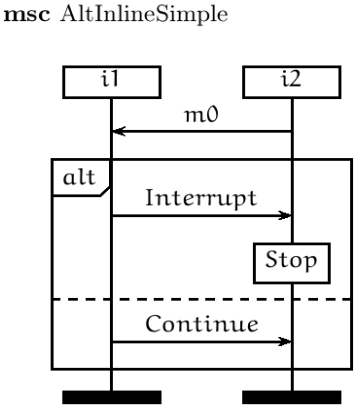

Рис. 9. Пример MSC-диаграммы с элементом альтернативного выбора

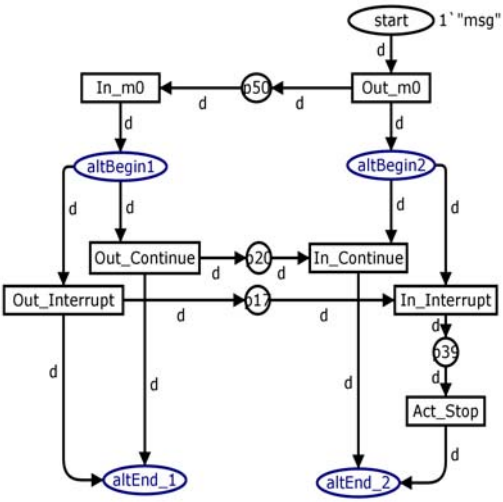

Рис. 10. Результат трансляции MSC-диаграммы с элементом альтернативного выбора, изображенной на рис. 9

Определение. Будем говорить, что элемент альтернативной композиции MSC обладает *свойством нелокального выбора* (*non-local choice*) [8, 14], если существуют такие ветви данной альтернативы, которые имеют стартовые события, локализованные на различных сущностях, входящих в альтернативу.

Элемент альтернативного выбора на рис. 9 обладает свойством локального выбора. В диаграмме на рисунке выбор альтернативной ветви может быть осуществлен только локально, посредством сущности *i1*. В зависимости от выполнения события отправки сообщения *Interrupt* или *Continue* будет выбрана первая или вторая альтернатива, соответственно. Сущность *i2* никак не может повлиять на выбор, поскольку каждая альтернативная ветвь сущности *i2* начинается с событий приема сообщения, которые не могут быть выполнены до событий их отправки.

MSC-диаграммы могут оставаться синтаксически корректными, даже если будут содержать элементы `alt`, не обладающие свойством локального выбора. Отсутствие данного свойства может приводить к неоднозначной трактовке MSC-диаграммы, а также к незапланированному поведению и возможным дедлокам в результирующей CPN.

Методика автоматического преобразования произвольных HMSC-спецификаций в локальные HMSC-спецификации, т.е. обладающие свойством локального выбора, была подробно рассмотрена в [1]. В этой работе локализация осуществляется за счет добавления вспомогательных сообщений или активных сущностей (т. е. сущностей со стартовыми событиями) в MSC-диаграммы, входящие в HMSC-спецификацию. Отметим, что в нашей работе нет необходимости каким-либо образом модифицировать исходные MSC-диаграммы, а достаточно лишь локализовать транслированную HMSC-спецификацию, представленную в виде CPN.

Определение того, обладает ли встроенное выражение `alt` или альтернативная композиция в HMSC свойством локального выбора, может быть выполнено автоматически, при помощи следующего алгоритма:

1. Для заданного альтернативного элемента `alt` строится матрица S размера $N \times M$. Матрица может содержать следующие значения s_{ij}: «1» — сущность i в альтернативе j имеет стартовое событие; «–1» — сущность i в альтернативе j не представлена (не имеет событий); «0» — в противном случае.
2. Если заданный элемент `alt` содержит вложенный элемент альтернативного выбора, расположенный в начале

альтернативной ветви, то вложенный `alt` трактуется как элемент со свойством локального выбора.

3. Если матрица содержит хотя бы одно значение «–1», то считаем, что данный альтернативный элемент обладает свойством нелокального выбора. Иначе переходим на пункт 3.
4. Если все столбцы матрицы, за исключением единственного — нулевые, то данный альтернативный элемент обладает свойством локального выбора.
5. В противном случае элемент `alt` обладает свойством нелокального выбора.

Рассмотрим пример на рис. 11, опустив для простоты все охранные условия. Выбор альтернативной ветви в данном примере может быть осуществлен нелокальным способом. Например, сущность *i2* после выполнения события отправки сообщения *m0* может выполнить локальное событие '*a*' (стартовое событие альтернативы). В то же время сущность *i1*, выполнив событие приема сообщения *m0*, может выбрать альтернативу, выполнив событие отправки сообщения *m2* (стартовое событие альтернативы). Таким образом, каждая из сущностей произвела выбор разных ветвей одной и той же альтернативы, образуя тем самым ситуацию с дедлоком.

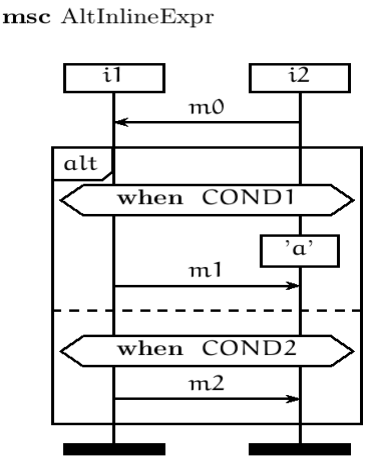

Рис. 11. Пример элемента альтернативного выбора с охранными условиями

Поэтому в случае, если альтернативная композиция не обладает свойством локального выбора, применяется модифицированная версия алгоритма, описанная в разделе 5.2.2.

5.2.1 Альтернативная композиция с локальным выбором

На первой стадии трансляции элемента альтернативной композиции со свойством локального выбора `alt`, содержащего *M* альтернатив и примененного к *N* сущностям, выполняются следующие операции с графом частичного порядка:

- Каждое событие i-й сущности ($1 \le i \le N$), стоящее перед элементом альтернативной композиции, соединяется направленной дугой с каждым j-м стартовым событием альтернативы ($1 \le j \le M$) i-й сущности;
- Каждое j-е конечное событие альтернативы ($1 \le j \le M$) i-й сущности соединяется направленной дугой с каждым событием i-й сущности ($1 \le i \le N$), стоящим после элемента альтернативной композиции;
- Все остальные события, содержащиеся внутри альтернативного элемента, переводятся согласно правилам для данного типа событий.

На третьей стадии трансляции для каждого перехода, соответствующего стартовому или конечному событию рассматриваемой альтернативы, устанавливается общее входное место *altBegin* или выходное место *altEnd* соответственно. Использование общих мест дает возможность выполнить только один из переходов, соответствующих событиям из разных альтернативных ветвей одной сущности.

CPN, полученная в результате трансляции MSC-диаграммы из рис. 9, которая содержит элемент `alt` со свойством локального выбора, представлена на рис. 10. В сети на рисунке за счет использования общего места *altBegin* возможно срабатывание только одного из переходов *Out_Continue* или *Out_Interrupt*, каждый из которых логически принадлежит отдельной ветви.

5.2.2 Альтернативная композиция с нелокальным выбором

Для элемента альтернативной композиции со свойством нелокального выбора `alt`, содержащего *M* альтернатив и примененного к *N* сущностям создается *M* начальных узлов *AltCond* и *N* конечных узлов *Alt_END*. Все события, содержащиеся внутри альтернативного элемента, переводятся

согласно правилам тех элементов, к которым они относятся. Каждый узел *AltCond* соединяется со всеми начальными узлами (стартовыми событиями) одной конкретной альтернативной ветви.

Если ни одна из альтернатив встроенного выражения `alt` не снабжена охранным условием, то выбор произвольной альтернативы осуществляется недетерминированно. Если для какого-либо из операндов элемента альтернативного выбора установлено охранное выражение, и это выражение ложно, то в этом случае данный операнд (ветвь) не может быть выбран для исполнения. Операнды без охранных условий интерпретируются по умолчанию как операнды с истинными охранными условиями. Если все операнды имеют ложные охранные условия, то элемент `alt` является динамически некорректным, поскольку через данный элемент не будет проходить ни одной трассы. Для элемента альтернативного выбора допускается использовать охранное условие с ключевым словом `otherwise`, которое интерпретируется как конъюнкция отрицаний всех остальных охранных условий операндов альтернативной композиции.

На третьей стадии трансляции каждый элемент *AltCond* и *Alt_END* преобразуется к виду, представленному на рис. 12. Следует отметить, что в месте соединения MSC-диаграммы с началом нелокального альтернативного элемента используется *строгая последовательная композиция* для моделирования локального выбора. На рисунке такое соединение изображено при помощи места *altBegin*. Все переходы, соответствующие событиям непосредственно перед началом альтернативной композиции, соединяются с местом *altBegin*. Место *altBegin*, в свою очередь, соединяется дугами со всеми переходами *AltCond*, используемыми для выбора альтернативной ветви. Каждая исходящая из места *altBegin* дуга помечается выражением *«x`d»*, где *x* — число сущностей, вовлеченных в альтернативный элемент. Завершению альтернативного элемента в CPN для конкретной сущности соответствует переход *Alt_END.* Для всех переходов, соответствующих конечным событиям альтернативных ветвей заданной сущности, устанавливается общее выходное место *altEnd*. Таким образом, моделирование локального выбора в элементе `alt` за счет использования строгой композиции вместо слабой последовательной композиции, установленной по умолчанию, нарушает эквивалентность между множеством трасс исполнения событий исходной MSC-диаграммы и множеством последовательностей срабатываний переходов, соответствующих данным событиям в строящейся сети.

Если какой-либо из операндов элемента `alt` снабжен охранным условием, то на третьем этапе трансляции соответствующий этому операнду

переход *AltCond* помечается спусковой функцией *guard,* реализующей данное условие в соответствии с правилами для охранных условий из раздела 4.2). При этом сами переходы, представляющие охранное условие, удаляются из результирующей сети с целью избежать повторных проверок условий.

CPN, полученная в результате трансляции MSC-диаграммы из рис. 11, которая содержит элемент `alt` со свойством нелокального выбора и охранными условиями, представлена на рис. 12. Для моделирования локального выбора одной из двух альтернативных ветвей в сети на рис. 12 начало каждой ветви предваряется переходами *AltCond_COND1* и *AltCond_COND2*, которые имеют общее входное место *altBegin*. Поскольку элемент `alt` в MSC на рис. 11 снабжен охранными условиями *COND1* и *COND2*, каждый из переходов *AltCond_COND1* и *AltCond_COND2* преобразуется по правилам, описанным для охранных условий в разделе 4.2. Таким образом, после срабатывания переходов *Out_m0* и *In_m0*, будут доступны для исполнения только те из переходов *AltCond_COND1* или *AltCond_COND2*, для которых спусковая функция будет истинной.

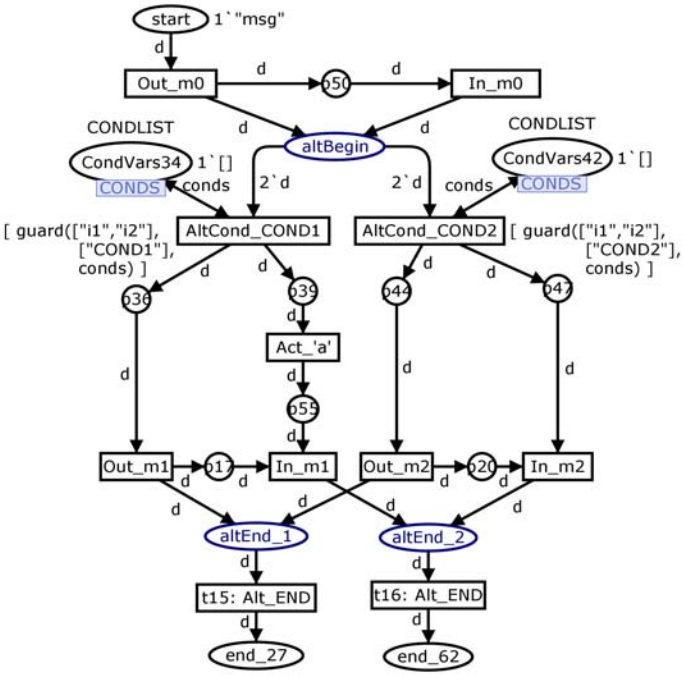

Рис. 12. Результат трансляции элемента альтернативного выбора с охранными условиями, изображенного на рис. 11

Отметим, что элемент альтернативной композиции с локальным выбором моделируется без вспомогательных переходов. В то же время на каждый элемент альтернативной композиции с нелокальным выбором приходится *2M* вспомогательных переходов в CPN, где *M* — число альтернатив.

5.2.3 Альтернативная композиция вида `opt` и `exc`

В стандарте MSC, помимо встроенного выражения альтернативной композиции `alt`, представлены два частных случая этого элемента: встроенное выражение опционального исполнения `opt` и ошибочного исполнения `exc`. Оба выражения могут иметь только один операнд.

Выражение `opt` эквивалентно выражению `alt` с двумя операндами, второй из которых является пустым. Выражение `exc` является удобным способом для описания исключительных ситуаций в MSC-диаграмме. После выполнения событий внутри `exc` исполнение остальных событий MSC-диаграммы прекращается. Выражение `exc` можно рассматривать как выражение `alt`, в котором второй операнд содержит всю оставшуюся часть MSC-диаграммы. Поэтому выражение `exc` может быть установлено только для всех сущностей MSC-диаграммы.

Встроенные выражения `opt` и `exc` переводятся по правилам трансляции элемента `alt` с той поправкой, что данные элементы содержат две фиксированные альтернативы.

5.3. Параллельная композиция

Параллельная композиция в MSC-диаграмме может быть представлена при помощи встроенного или ссылочного выражения `par`. Параллельная композиция, заданная для нескольких операндов, позволяет одновременно выполнить все события в каждом из операндов данной композиции. Выход из элемента `par` возможен только после завершения всех событий параллельных секций.

Пример элемента параллельного исполнения (встроенного выражения `par`) изображен на рис. 13. В данной диаграмме внутри элемента `par`, сущность *i1* может одновременно выполнять события отправки сообщения *m1* и *m2*, а сущность *i2* — локальное событие '*a*' с последующим выполнением события приема *m1* и событие приема сообщения *m2*.

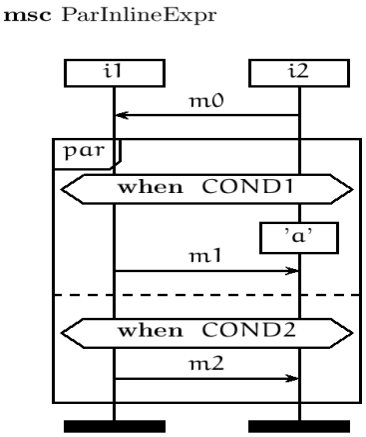

Рис. 13. Пример MSC-диаграммы с элементом параллельного исполнения

Для элемента параллельного исполнения `par`, примененного к N сущностям системы, создается N начальных узлов типа *Par_begin* и N конечных узлов *Par_end*. Все события MSC, содержащиеся внутри элемента параллельного исполнения, переводятся согласно правилам для данного типа событий. Каждый узел *Par_begin* и *Par_end* заданной сущности соединяется с начальными и конечными узлами параллельных секций этой сущности.

На третьей стадии трансляции для каждого перехода *Par_begin* создается M выходных мест, где M — число операндов/секций элемента `par`, по одному для каждого перехода, соответствующего стартовому событию каждой параллельной секции данной сущности. Срабатывание перехода *Par_begin*, означающее вход в конструкцию параллельного исполнения, осуществляет перемещение фишек в выходные места, моделирующие состояние начала одновременного выполнения сразу всех секций данного параллельного элемента. Для перехода *Par_end* создается M входных мест, в каждое из которых может попасть фишка только после исполнения всех переходов из определенной параллельной секции. Таким образом, срабатывание перехода *Par_end*, означающее выход из конструкции параллельного исполнения для заданной сущности, становится возможным только после срабатывания переходов, соответствующих событиям MSC-диаграммы из всех параллельных секций этой сущности.

Фрагмент CPN, который получается в результате трансляции элемента `par` (MSC-диаграмма на рис. 13 с опущенными охранными условиями) приведен на рис. 14.

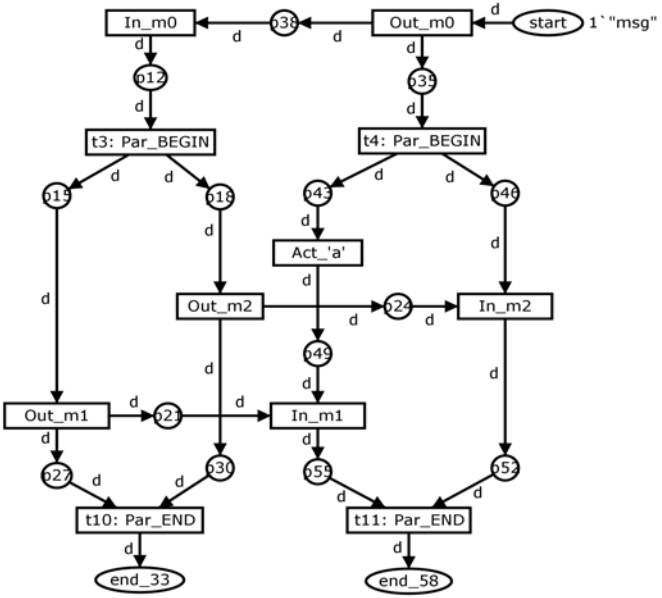

Рис. 14. Результат трансляции MSC-диаграммы с элементом параллельного исполнения, изображенной на рис. 13

Если для какого-либо из операндов параллельного элемента установлено охранное выражение, и это выражение ложно, то в этом случае данный операнд (секция) исключается из параллельного элемента. Если все операнды во время исполнения MSC-диаграммы будут иметь ложные охранные выражения, то параллельный элемент будет пустым (`empty`), т.е. ни одного события во время выполнения данного элемента не произойдет.

Элемент `par`, снабженный охранными условиями, на третьей стадии трансляции преобразуется в CPN по следующим правилам:

- для перехода *Par_begin* каждой входящей в элемент `par` сущности создается *2M* выходных мест, где *M* — число операндов/секций элемента `par`. Для каждого параллельного операнда создается одно место-вход для случая истинности

охранного условия и одно место-выход — для выхода из параллельной секции, если охранное условие ложно. Для имитации охранных условий параллельных секций создаются функции на дугах. Дуги, соединяющие каждый переход *Par_begin* с местом-входом помечаются функцией *func*, а с местом-выходом — функцией *nfunc*. Все перечисленные здесь и далее функции описаны в приложении. Функция *func* аналогична действию функции *guard* с той лишь разницей, что *func* в случае истинности охранного условия возвращает не значение логического типа, а значение типа, установленного для фишек по умолчанию. Так же обстоит дело с функциями *nfunc* и *eguard*. Заметим, что переходы, соответствующие охранным условиям элемента `par`, удаляются из результирующей сети;

- переход *Par_end* имеет *M* входных мест, каждое из которых является местом-выходом, описанным в предыдущем пункте;
- для каждой сущности создается дополнительный переход *Empty*, моделирующий пустое выполнение элемента `par`. Правила трансляции секции с переходом *Empty* аналогичны описанным в предыдущих двух пунктах, за исключением использования других функций на дугах. Вместо функции *func*, управляющей возможностью исполнения перехода *Empty*, используется функция *efunc*, возвращающая значение, соответствующее типу фишки по умолчанию только в том случае, если отрицание охранных условий всех секций будет истинным. Вместо функции *nfunc* используется функция *enfunc*, возвращающая непустое значение в том случае, если хотя бы одно охранное условие элемента `par` истинно.

CPN, которая получается в результате трансляции элемента `par` (из диаграммы на рис. 13) с охранными условиями, приведена на рис. 15. На данном рисунке выражениями на дугах *f1, f2, f3* и *f4* для краткости обозначены функции *func*, *nfunc*, *efunc*, *enfunc*, описанные в приложении. Каждый элемент параллельного исполнения моделируется при помощи *2N* вспомогательных переходов в CPN, где *N* — число сущностей, входящих в `par`.

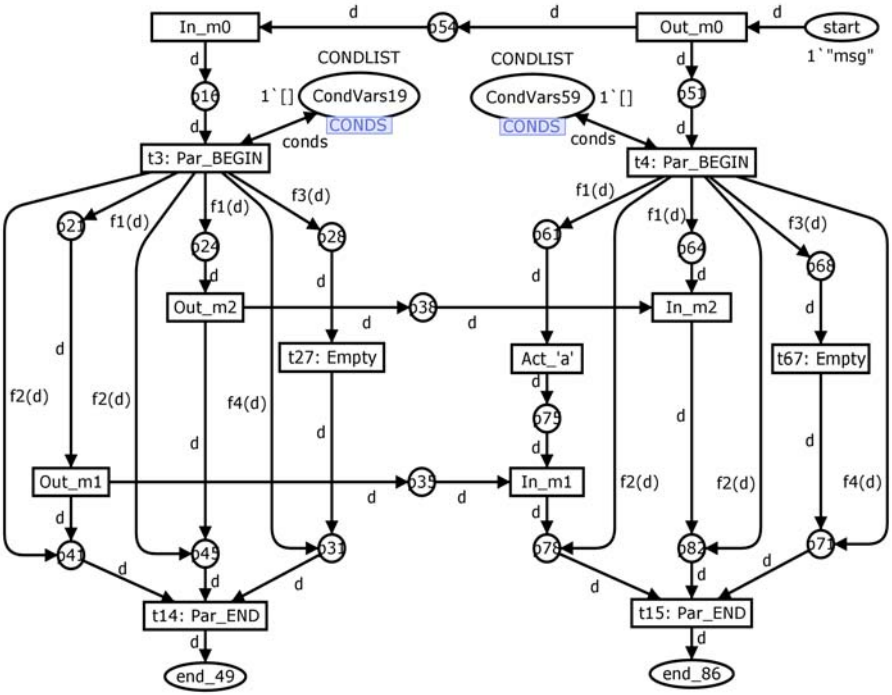

Рис. 15. Результат трансляции элемента параллельного исполнения с охранными условиями, изображенного на рис. 13

Т.к. элемент `par` в MSC на рис. 13 снабжен охранными условиями *COND1* и *COND2*, для каждого из переходов *Par_BEGIN* на рис. 15 создаются входные/выходные места *CondVars* по правилам, описанным для охранных условий в разделе 4.2. Проверка охранных условий выполняется при помощи функций *func*, *nfunc*, *efunc* и *enfunc*, которыми помечаются выходные дуги переходов *Par_BEGIN*. Таким образом, после срабатывания перехода *Par_BEGIN* произойдет извлечение фишек из входного места данного перехода и места *CondVars* для проверки охранных условий. В результате выполнения функций на дугах фишки будут помещены в те выходные места перехода *Par_BEGIN*, для которых функции на инцидентных им входящих дугах вернули непустое значение. Если какая-либо из функций возвращает пустой результат, то соответствующая данной параллельной

секции фишка будет перемещена во входное место перехода *Par_END*, что будет означать пропуск исполнения событий из данной секции.

5.4. Циклическая композиция

Циклы в MSC-диаграмме могут быть представлены при помощи встроенного или ссылочного выражения `loop`. Встроенное выражение `loop` может иметь только один операнд, представляющий собой тело цикла. Цикл может иметь несколько форм:

- *loop<n, m>*, где *n, m* — натуральные числа, означает, что операнд встроенного выражения может быть выполнен как минимум *n* и как максимум *m* раз. Вместо натурального числа можно использовать ключевое слово `inf`. Так, выражение *loop<n,* `inf`*>* будет означать, что оператор встроенного выражения может быть выполнен как минимум *n* раз.
- *loop<n>* (эквивалент *loop<n, n>*) означает, что операнд встроенного выражения должен быть выполнен ровно *n* раз. Выражение *loop<inf>* обозначает бесконечный цикл.
- *loop* (без аргументов), что эквивалентно выражению *loop<1, inf>*.

Важной особенностью циклов в MSC-диаграмме является то, что каждый новый проход цикла связывается с предыдущим проходом посредством слабой последовательной композиции.

Если встроенное выражение `loop` снабжено охранным условием, то тело цикла может быть выполнено любое число раз, пока будет оставаться истинным охранное условие, и пока будут удовлетворяться границы цикла.

Пример MSC-диаграммы с циклом представлен на рис. 16. Данный цикл может быть выполнен от *n* до *m* раз, пока истинно охранное условие *COND*.

Определение. Будем говорить, что MSC-диаграмма обладает *свойством глобальной согласованности* (*globally cooperative*) [8, 14], если в теле каждого цикла (элемента циклической композиции) каждая сущность или группа сущностей связана с остальными сущностями, участвующими в этом цикле, посредством сообщений.

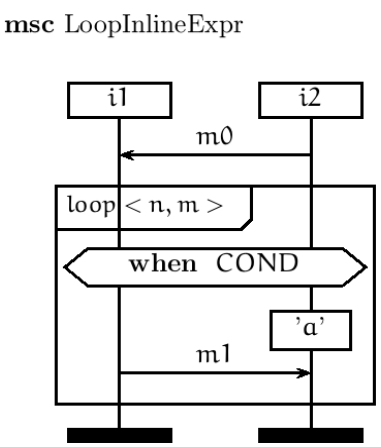

Рис. 16. Пример MSC-диаграммы с циклом вида *loop<n, m>*

Описывая спецификацию при помощи MSC-диаграмм, следует уже на этапе их составления строить диаграммы таким образом, чтобы они обладали свойством глобальной согласованности. Соответствие этому правилу исключает недвусмысленную трактовку диаграмм в случае их дальнейшей трансляции в другие представления.

Тем не менее, некоторые корректно составленные MSC-диаграммы могут случайно или намеренно не обладать свойством глобальной согласованности. Поэтому описываемый далее алгоритм трансляции циклических элементов на выходе строит CPN, моделирующую исполнение MSC-диаграммы без проблемы глобальной согласованности, независимо от того, обладала ли таким свойством исходная MSC-диаграмма.

На первом этапе трансляции элемент циклического исполнения `loop`, примененный к *N* сущностям, преобразуется в *N* начальных узлов *Loop_begin* и *N* конечных узлов *Loop_end.* Все остальные события, содержащиеся внутри элемента циклического исполнения, переводятся согласно правилам для данного типа событий.

На третьей стадии трансляции происходит обработка узлов *Loop_begin* и *Loop_end*, которая варьируется в зависимости от типа цикла. Заметим, что для моделирования элемента `loop` внутри CPN используется модифицированный тип фишки (*PACKET*), который предназначен для хранения списка, содержащего номера текущих итераций циклов (стек LIFO для вложенных циклов):

*colset PACKET = product MscMsg * NOLIST;*

colset NOLIST = list NO;
colset NO = int;

Если задан цикл без охранного условия с типом *loop<n>*, то выполняются следующие преобразования (см. рисунок 17):

- каждая дуга, исходящая из перехода *Loop_begin*, помечается функцией *CreateLoop*() (см. приложение), которая возвращает значение типа *PACKET*, содержащее новый итератор цикла в списке *NOLIST*. Таким образом, срабатывание перехода *Loop_begin* инициализирует цикл;
- переход *Loop_end* соединяется дугой, помеченной функцией *NextLoop*() (см. приложение), с начальным местом циклического выражения *loopIn*. Данная функция увеличивает счетчик цикла на один, если он не превышает заданного числа *n*; в противном случае функция возвращает пустое значение (*empty*);
- каждая дуга, исходящая из перехода *Loop_end*, помечается функцией *EndLoop*() (см. приложение). Данная функция возвращает значение типа *PACKET* с удаленным значением текущего итератора цикла из списка *NOLIST* в том случае, если текущее значение счетчика не меньше заданного числа *n* цикла. Таким образом, срабатывание перехода *Loop_end* моделирует проверку условия завершения цикла.

В сети на рис. 17 срабатывание перехода *Loop_BEGIN* приведет к добавлению в список *nlist* фишки нового итератора цикла, равного единице. Срабатывание перехода *Loop_END* будет приводить к перемещению фишки в место *loopIn* (начало цикла) до тех пор, пока номер итерации цикла в списке *nlist* не станет равен заданному числу *N*.

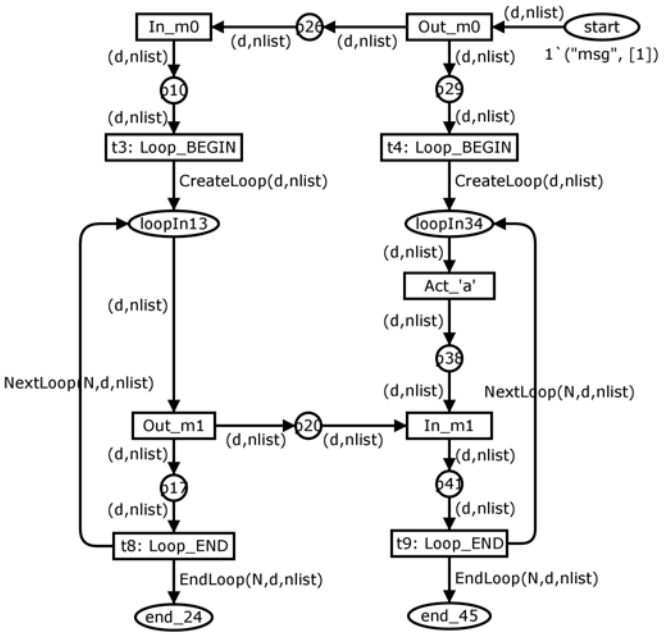

Рис. 17. Результат трансляции элемента циклического исполнения из рис. 16 без охранных условий

Если задан цикл с охранным условием и/или типом *loop<n, m>*, то выполняются следующие преобразования (см. рисунок 18):

- каждая дуга, исходящая из перехода *Loop_begin* помечается функцией *CreateLoop*(), которая возвращает значение типа *PACKET*, содержащее новый итератор цикла в списке *NOLIST*;
- переход *Loop_end*, стоящий в конце циклического фрагмента перемещается в начало фрагмента цикла; место *loopIn*, соответствующее началу циклического выражения, соединяется с переходом *Loop_end*;
- переход, соответствующий конечному событию цикла, связывается дугой, помеченной функцией *UpdLoop*() (см. приложение), с местом *loopIn*. Данная функция осуществляет увеличение счетчика цикла;

- создается совмещенное место *LoopStatus* (тип *fusion place* в терминах CPNTools), содержащее текущий счетчик цикла и флаг, сигнализирующий о выходе из цикла одной из сущностей. Данное место помечается типом *NOxFLAG*:
  *colset NOxFLAG = product NO * FLAG;*
  colset FLAG = bool;
  Место *LoopStatus* соединяется с переходами *Loop_begin* и *Loop_end*, которые помечаются спусковыми функциями *preGuardLoop* и *postGuardLoop*, осуществляющие проверку граничных условий цикла и статус цикла.
- каждая дуга, исходящая из перехода *Loop_end*, помечается функцией *EndLoop*(). Данная функция возвращает значение типа *PACKET* с удаленным значением текущего итератора цикла из списка *NOLIST*.

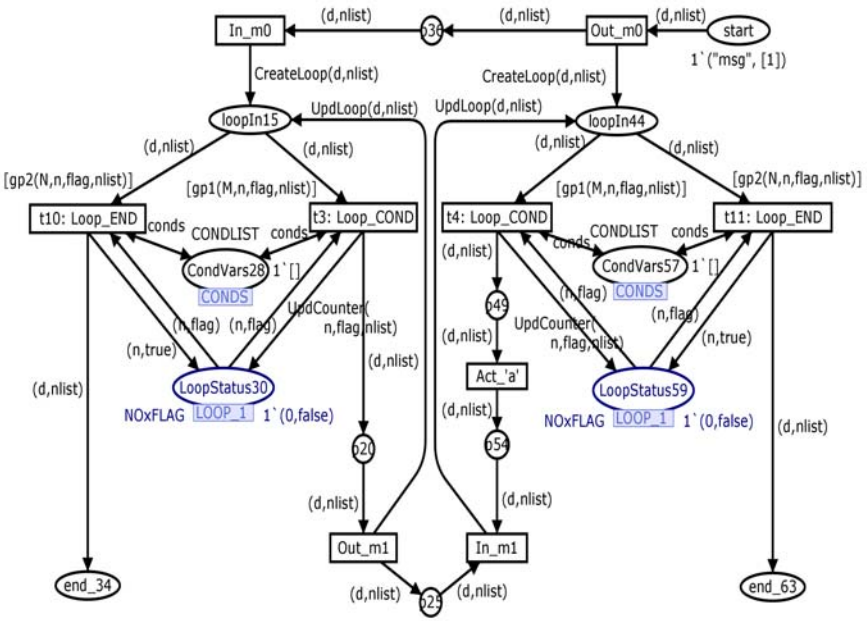

Рис. 18. Результат трансляции элемента циклического исполнения из рис. 16 с охранными условиями

На рис. 18 спусковыми функциями *gp1* и *gp2* для краткости обозначены функции *preGuardLoop* и *postGuardLoop*, описанные в приложении.

Поскольку элемент циклического исполнения в MSC на рис. 16 имеет вид *loop<n, m>*, то для каждого из переходов *Loop_BEGIN* и *Loop_END* сети на рис. 18 устанавливаются спусковые функции, выполняющие проверку граничных условий цикла, и создаются входные/выходные места *LoopStatus*, общие для данного цикла. Кроме этого, т.к. рассматриваемый циклический элемент MSC также снабжен охранным условием *COND*, то для каждого из переходов *Loop_BEGIN* и *Loop_END* создаются входные/выходные места *CondVars* по правилам, описанным для охранных условий в разделе 4.2. На рисунке спусковые функции для проверки сразу охранного и граничного условия обозначены через *preGuardLoop* и *postGuardLoop.* Установка охранного условия *COND* для цикла будет означать, что пока спусковая функция перехода *Loop_BEGIN* истинна, он будет доступен для исполнения. Если спусковая функция становится ложной, то будет доступен для исполнения только переход *Loop_END*.

Отметим, что на каждый элемент циклического исполнения приходится *2N* вспомогательных переходов в CPN, где *N* — число сущностей, входящих в элемент `loop`.

6. ПРЕОБРАЗОВАНИЕ ССЫЛОЧНЫХ ВЫРАЖЕНИЙ И HMSC-ДИАГРАММ В CPN

В данном разделе будет рассмотрена трансляция оставшихся структурных элементов стандарта MSC, к которым относятся ссылочные выражения и HMSC-диаграммы.

Эти элементы используются в качестве средства композиции MSC-диаграмм, и поэтому их присутствие в (H)MSC-спецификации задает иерархический характер представления описываемой системы. В правилах трансляции, представленных ниже, для простоты будет дано описание преобразования ссылочных выражений и HMSC-диаграмм в неиерархические раскрашенные сети Петри. Тем не менее, все описываемые построения можно естественным образом расширить на случай иерархических сетей Петри.

6.1. Ссылочные выражения MSC

Ссылки на MSC используются для удобного представления вложенности MSC-диаграмм внутри HMSC или MSC. На диаграмме они изображаются при помощи скругленного прямоугольника, содержащего текст ссылки. Каждая ссылка может содержать не только имя отдельной MSC-диаграммы, на которую она ссылается, но и ссылочное выражение, состоящее из комбинации имен нескольких диаграмм, связанных посредством различных видов композиции.

Ссылочное выражение может содержать следующие ключевые слова: `alt`, `par`, `seq`, `loop`, `opt` и `exc`. Пример ссылочного выражения: (*P* `alt` *Q*) `seq` *R*, где *P*, *Q*, *R* — ссылки на MSC с данными именами. Примерами ссылок на MSC внутри HMSC-диаграммы являются ссылки *A*, *B* и *C* на рис. 19.

Отметим, что ссылочное выражение, прикрепленное к нескольким сущностям диаграммы, согласно статическим ограничениям MSC обязано ссылаться на MSC-диаграмму, содержащую только те же самые сущности.

На первом этапе трансляции простое ссылочное выражение, содержащее только имя MSC-диаграммы, переводится в один узел типа *Ref* графа частичного порядка. Если ссылочное выражение содержит операторы композиции MSC, представленные ключевыми словами `alt`, `par`, `seq`, `loop`, `opt` и `exc`, то данное выражение транслируется в узлы графа частичного порядка по правилам для встроенных выражений с аналогичными типами композиции (см. раздел 5). Таким образом, граф частичного порядка ссылочного выражения после первого этапа трансляции может содержать только узлы типа *Ref* и вспомогательные узлы конструкций композиции.

Развертка ссылки на MSC в графе частичного порядка происходит на втором этапе по следующим правилам:

- элементы, содержащиеся в каждой ссылочной MSC-диаграмме, переводятся по правилам, изложенным на этапе 1 алгоритма трансляции;
- узел графа, имеющий в потомках узел *Ref*, соединяется дугой со стартовым узлом ссылочной MSC так, чтобы узел графа, из которого исходит дуга, и стартовый узел MSC принадлежали одной и той же сущности;
- конечный узел ссылочной MSC соединяется дугой с узлом графа, имеющим в предках узел *Ref* так, чтобы узлы ссылочной MSC и исходного графа принадлежали одной и той же сущности.

Пример использования ссылочного выражения в HMSC-диаграмме и фрагмент CPN для него представлен на рис. 19 и 20.

6.2. Элементы HMSC

Диаграмма HMSC представляет собой удобный способ задания отношений между различными MSC. По смыслу она схожа с графом потока управления и позволяет наглядно задавать разные способы композиции между ссылочными выражениями MSC.

Пример HMSC-диаграммы и соответствующей CPN изображен на рис. 19 и 20.

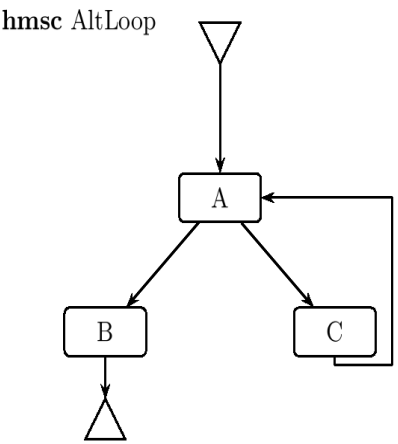

Рис. 19. Пример HMSC-диаграммы с альтернативой и циклом, содержащей ссылки на MSC-диаграммы *A*, *B* и *C*

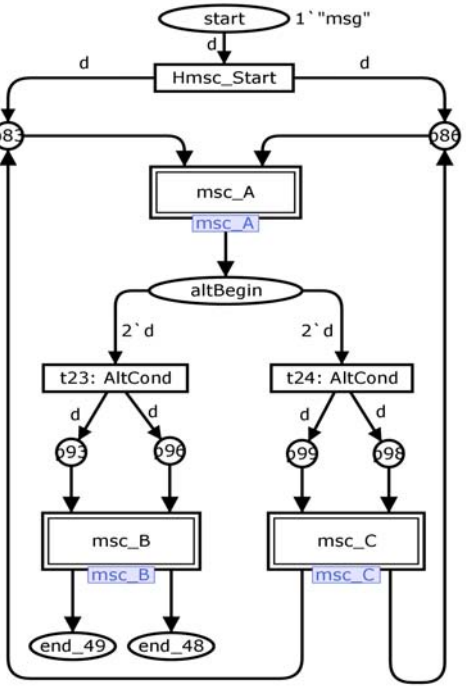

Рис. 20. Результат трансляции HMSC-диаграммы из рис. 19

Графически HMSC-диаграмма изображается в виде ориентированного графа, где каждый из узлов может быть одного из следующих типов:

- стартовый символ (изображается в виде треугольника с исходящей из него дугой). Каждая HMSC-диаграмма может содержать только один стартовый символ;
- конечный символ (изображается в виде треугольника с входящей в него дугой);
- ссылочное выражение MSC;
- условие MSC;
- связующий узел.

Построение графа частичного порядка для HMSC осуществляется по следующим правилам:

- начальная/конечная точка HMSC переводится в узел *Start* / *End* графа частичного порядка;
- связующая точка *connect* HMSC преобразуется в узел *Connect*;
- установочное и охранное условие переводятся в узлы *Cond* и *GuardCond* графа;
- начало параллельной композиции преобразуется в узел *HMSC_Par*;
- ссылочные выражения MSC переводятся в узлы типа *Ref*.

Преобразование условий, ссылочных выражений и всех видов композиции в CPN на третьей стадии трансляции происходит по правилам, описанным для данных элементов MSC в предыдущих разделах.

7. ОБРАБОТКА ГРАФА ЧАСТИЧНОГО ПОРЯДКА MSC

В данном разделе описывается обработка и оптимизация графа частичного порядка, полученного после первой стадии трансляции.

7.1. Обработка узлов и связей

Обработка графа частичного порядка на второй стадии трансляции включает в себя выполнение следующих действий:

- Удаление *connect* узлов HMSC-диаграммы. Данный тип узлов в HMSC не имеет определенной семантики и играет чисто

декоративную роль, которая заключается в более удобном и понятном визуальном размещении узлов HMSC-диаграммы.

- Пометка узлов HMSC-диаграммы, представляющих собой начало параллельной и альтернативной композиции. Узлы, имеющие более одного потомка и не являющиеся узлами *HMSC_Par*, помечаются типом *HMSC_Alt*.
- Обработка ссылок на MSC. Для каждой ссылки на MSC строится MSC-диаграмма, в которой отмечаются точки начала и конца диаграммы (стартовые и конечные узлы). По данным точкам происходит связывание данной диаграммы с исходной HMSC/MSC-диаграммой (см. раздел 6.1).
- Удаление лишних связей. Для множеств входящих и исходящих узлов для текущего узла проверяется, не содержат ли эти множества узлы, не соответствующие отношению доминирования узлов, установленных статическими ограничениями MSC. Найденные узлы удаляются из графа.

7.2. Свертка дублирующихся фрагментов в цикл

Необходимость данного вида оптимизации вызвана тем, что зачастую многие MSC, описывающие сложные системы и протоколы, имеют довольно большой размер, причем в большинстве таких диаграмм используется большое количество событий из базового набора MSC, которые являются статическими. При трансляции таких MSC-диаграмм бывает, что строящаяся сеть Петри достигает огромных размеров. Для решения этой проблемы был разработан алгоритм уменьшения общего объема генерируемой CPN.

Идея алгоритма заключается в следующем. Для построенного на первой стадии графа частичного порядка MSC-диаграммы осуществляется поиск нескольких одинаковых подряд идущих подмножеств узлов, таких, что множество статических трасс одного подмножества узлов совпадает с множеством статических трасс другого. Найденные подмножества узлов заменяются на конструкцию с циклом типа *loop*<*n*>, описанную ранее. Т.к. конструкция *loop*<*n*> есть не что иное, как слабая последовательная композиция, примененная к *n* подмножествам событий из тела цикла, то набор трасс, соответствующий неоптимизированной MSC-диаграмме, будет однозначно совпадать с набором трасс MSC-диаграммы, полученной в результате оптимизации. После завершения оптимизации графа частичного порядка выполняется третий этап алгоритма, где модифицированный граф транслируется в CPN по правилам, описанным ранее.

Отметим, что, поскольку данный вид оптимизации модифицирует CPN, полученную на выходе транслятора, то в результате такой оптимизации информация о некоторых событиях MSC-диаграммы может быть утеряна.

Ниже описаны основные шаги алгоритма поиска повторяющихся фрагментов:

1. Каждому типу событий из базового набора элементов MSC-диаграммы приписывается кодовый номер. Остальные типы событий помечаются пустым символом (например, «#»). Для каждой сущности MSC-диаграммы создается упорядоченный массив кодов событий, в соответствии с порядком событий, установленным для этой сущности. Например, в диаграмме на рис. 21 сущности *i1* может соответствовать следующий массив кодов событий: «11113», где 1 — код события отправки сообщения, 3 — код локального события.
2. Для каждой сущности в порядке их следования в MSC-диаграмме выполняем следующие действия.
   2.1. Фиксируем сущность *I* и ищем в массиве кодов этой сущности максимальную область с одинаковыми подряд идущими подмножествами кодов (фрагментами). Например, в массиве кодов «11113» будут найдены максимальные области (11, 11) и (1,1,1,1) с двумя и четырьмя подмножествами кодов соответственно. Если такие области не найдены, то переходим на пункт 2, выбирая следующую сущность.
   2.2. Для каждой найденной области выполняем следующие действия.
      2.2.1. Ищем в массиве кодов следующей сущности *J* кодовые области, совпадающие с исходной областью по количеству фрагментов. В примере на рис. 21 у сущности *i2* будут найдены кодовые области (2, 2), (21, 21) и (1, 1), совпадающие по числу фрагментов с областью (11, 11). Если такие области не найдены, то переходим к пункту 2.2, выбирая следующую возможную максимальную область.
      2.2.2. Формируем массив связей для тех сущностей, которые обмениваются сообщениями с текущей сущностью *I*. Массив связей похож на обычную кодовую область, за исключением того, что он содержит коды событий сообщений, которые обязаны встретиться в сущности для корректного сопоставления событий приемки/отправки сообщений. Например, для области (11, 11) сущности *i1* на рис. 21 будут созданы массивы связей для сущностей *i2* и *i3* — (2, 22) и (2,). Так, массив связей для *i2* (2, 22) означает, что существует

событие в первом фрагменте кодовой области *i1*, которое было отправлено и должно быть принято в *i2*, и существуют два события во втором фрагменте кодовой области *i1*, которые были отправлены и должны быть приняты в *i2*.

2.2.3. В каждой найденной кодовой области сущности *J* выполняем проверку связей. Для этого каждый массив связей, сформированный в 2.2.2, сравниваем с найденной кодовой областью. Сравнение происходит таким образом, чтобы определить, принадлежат ли события отправки/приема сообщений из разных областей к одному и тому же сообщению. Если для всех событий массива связей найдено соответствующее событие из зафиксированной кодовой области сущности *I*, то проверка завершилась успешно, переходим на шаг 2.2.4, иначе — на шаг 2.2, для поиска других кодовых областей.

2.2.4. Сохраняем все события, соответствующие найденной кодовой области в списке с именем «*Result*_<вид исходной кодовой области>».

2.2.5. Переходим на пункт 2.2.1 для выбора следующей сущности.

3. Выбираем в качестве результата среди всех списков *Result* список с максимальным числом событий.
4. Выбираем среди полученных событий набор, соответствующий одной итерации цикла и обрабатываем их так, как если бы они содержались в элементе циклического исполнения loop<n>. Остальные события списка *Result* удаляются из MSC-диаграммы.
5. Переходим на пункт 1 для повторного поиска повторяющихся фрагментов до тех пор, пока результирующий список *Result* не станет пустым.

Рассмотрим пример свертки дублирующихся фрагментов. На рис. 21 представлена исходная диаграмма. На рис. 22 изображена та же диаграмма, но со сверткой дублированного фрагмента в цикл. Обе диаграммы имеют эквивалентное множество возможных статических трасс. CPN, которая получается в результате трансляции диаграммы из рис. 22, изображена на рис. 23.

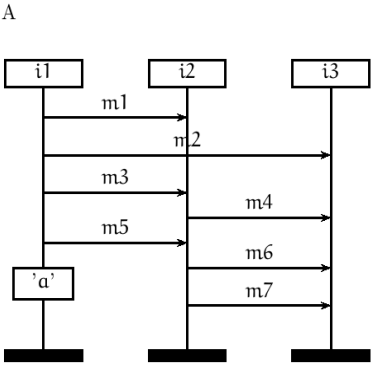

Рис. 21. MSC-диаграмма с дублирующимися фрагментами

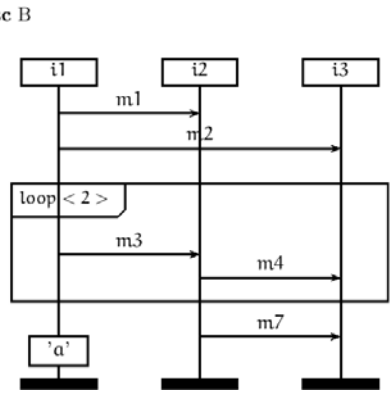

Рис. 22. MSC-диаграмма из рис. 21 с оптимизацией

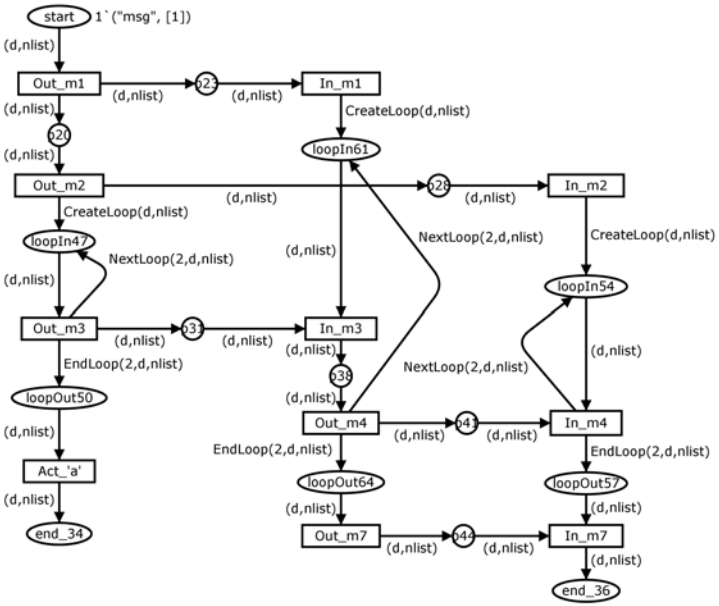

Рис. 23. Результат трансляции MSC-диаграммы из рис. 21 с применением свертки дублирующихся фрагментов

Отметим, что описанный в данном разделе алгоритм может иметь большое время работы, поскольку задача свертки дублирующихся фрагментов MSC-диаграммы в цикл схожа с упрощенной задачей поиска подграфа в ориентированном графе.

8. ПРИМЕР ТРАНСЛЯЦИИ MSC-СПЕЦИФИКАЦИИ В CPN

Для демонстрации работы алгоритмов трансляции рассмотрим немного видоизмененный пример MSC-спецификации, приведенной в [8]. На рис. 24 изображена HMSC-диаграмма *AccessControl*, описывающая абстрактную систему идентификации и доступа пользователя к объекту.

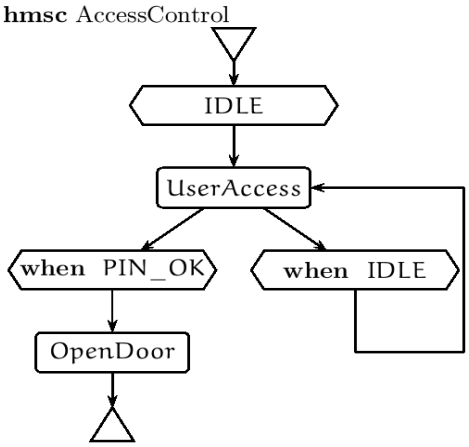

Рис. 24. HMSC-диаграмма *AccessControl* с общим описанием системы

Изначально, пользователь (сущность *User*) и система (сущность *System*) находятся в исходном состоянии ожидания *IDLE*. Затем пользователь может начать процесс получения доступа к объекту. Началу пользовательской активности соответствует переход к ссылочному выражению *UserAccess* в HMSC-диаграмме, что означает исполнение всех событий диаграммы *UserAccess*. В этом случае, если система и пользователь находятся в состоянии ожидания (охранное условие «when *IDLE*»), тогда пользователь отправляет свои идентификационные данные системе. Система выполняет проверку данных и отправляет результат проверки пользователю. Если результат проверки — *IllegalPIN*, т.е. проверка прошла неуспешно, то система и пользователь переходят в изначальное состояние ожидания; если результат проверки — *LegalPIN*, то устанавливается состояние *PIN_OK*, символизирующее успешную проверку данных пользователя. После исполнения событий в ссылочном выражении *UserAccess*, в HMSC-диаграмме выполняется проверка статуса доступа пользователя. Если доступ пользователю разрешен (охранное условие «when *PIN_OK*»), то в HMSC-диаграмме выполняется переход к ссылочному выражению *OpenDoor*, символизирующему получение доступа пользователя к объекту. В противном случае («when *IDLE*») пользователь получает отказ в доступе, но может повторить попытку получить доступ (в HMSC-диаграмме выполняется обратный переход к ссылке *UserAccess*).

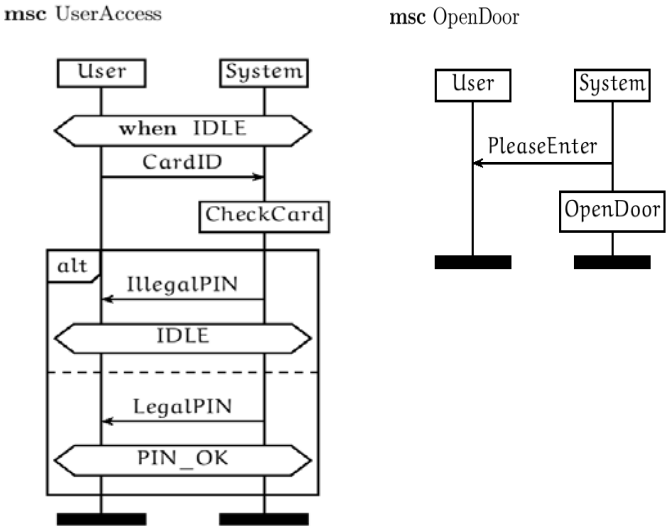

Рис. 25. MSC-диаграммы *UserAccess* и *OpenDoor* с детальным описанием системы

Результат трансляции HMSC-диаграммы *AccessControl* приведен на рис. 26 и 27.

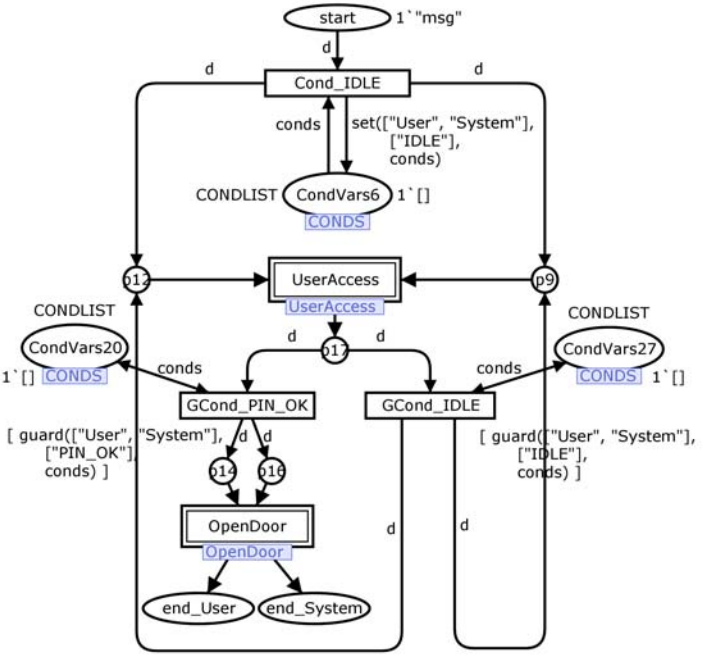

Рис. 26. Результат трансляции HMSC-диаграммы *AccessControl*

В результате анализа раскрашенной сети Петри из рис. 26 при помощи системы CPNTools получили, что маркировка сети, соответствующая размещению фишек в местах *end_User* и *end_System*, является одновременно домашней и мертвой. Это означает, что из любого состояния системы можно прийти к ее успешному завершению, т.е. получению пользователем доступа. При этом в исследуемой сети нет неограниченных мест, и невозможно одновременное пребывание в местах сети более одной фишки. Следовательно, данная сеть безопасна, и в исследуемой системе не может быть неограниченного накопления запросов пользователя к системе.

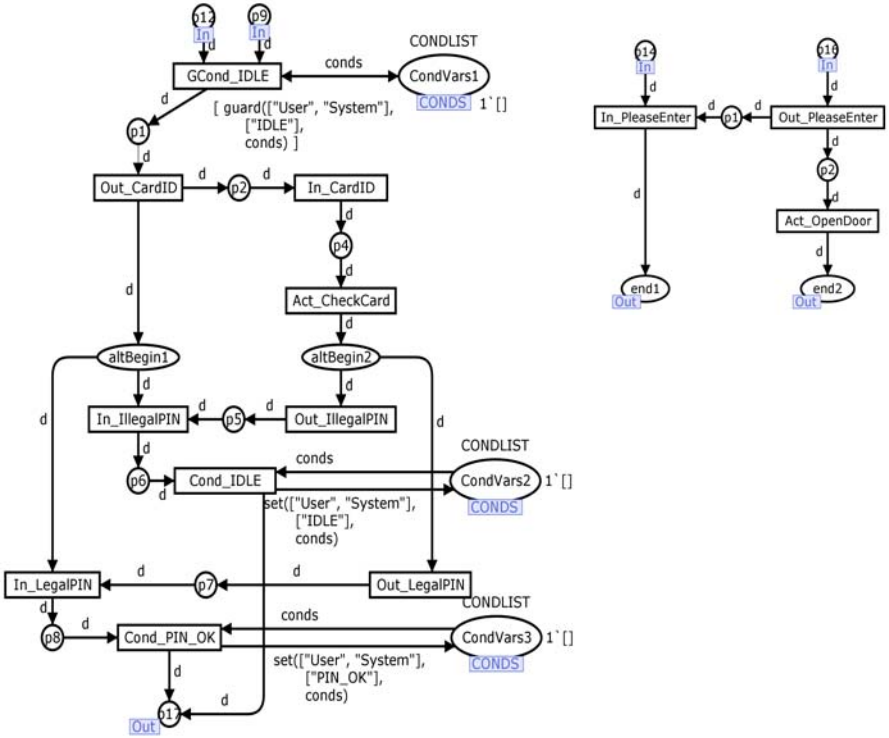

Рис. 27. Результат трансляции MSC-диаграмм *UserAccess* и *OpenDoor*

9. ЗАКЛЮЧЕНИЕ

В настоящей работе описаны алгоритмы трансляции конструкций диаграмм MSC и HMSC в раскрашенные сети Петри. Алгоритмы представлены для всех элементов из основного стандарта (H)MSC, кроме расширений в виде интерпретируемых данных и времени. Каждая конструкция (H)MSC-диаграммы, за исключением элемента альтернативного исполнения со свойством нелокального выбора, моделируется в CPN без ограничений. На основе представленных алгоритмов трансляции был реализован транслятор, работа которого была протестирована на различных примерах с выполнением анализа некоторых свойств результирующих сетей с помощью системы CPNTools.

Отметим, что на практике помимо (H)MSC-диаграмм используются также другие языки сценарных спецификаций: UML Sequence Diagrams (UML SD) и LSC-диаграммы. Диаграммы UML SD являются аналогом MSC-диаграмм для стандарта моделирования UML 2.0 [6, 7]. Они имеют похожее на MSC графическое представление и набор элементов. Основное различие между ними заключается в наличии более выразительного способа сценарной композиции (HMSC) для стандарта MSC. LSC-диаграммы (Live Sequence Charts) [2] являются расширением к стандартному языку MSC и обладают дополнительными возможностями для описания полных спецификаций систем, такими как наличие предусловий для набора событий, возможность и обязательность событий, механизм наложения сценариев с общим предусловием. Тем не менее, композиция набора LSC-диаграмм имеет сложную семантику, в отличие от простой семантики HMSC-диаграмм.

Проблема анализа сценарных спецификаций распределенных систем исследуется рядом авторов. Обзор литературы по этой проблеме до 2006 года приведен в [22]. Авторами данной работы было выполнено сравнение известных подходов по переводу сценарных моделей в конечно-автоматные модели и сети Петри. Отметим, что среди рассмотренных подходов, в которых используются сети Петри в качестве целевой модели, не рассмотрены HMSC-диаграммы как механизм композиции сценариев, а также не рассмотрена проблема локального выбора в MSC [1].

Цикл статей [11, 20, 21] посвящен трансляции MSC-диаграмм в ординарные сети Петри. В работах описаны алгоритмы трансляции ограниченного числа конструкций из стандарта MSC. По сравнению с нашим подходом, в этих работах присутствуют следующие структурные ограничения на MSC-диаграммы: рассматривается строгая последовательная композиция структурных элементов, элемент «условие» не несет конкретной семантики.

В работах [4, 17] рассматривается трансляция диаграмм UML SD в раскрашенные сети Петри. В отличие от нашей работы, эти статьи описывают правила трансляции ограниченного набора элементов диаграммы, который включает в себя сообщения и несколько видов композиции. Также имеются структурные ограничения на элементы обмена сообщениями (рассматриваются только синхронные сообщения) и интерпретацию условий в диаграмме.

Работы [3] и [19] вводят в рассматриваемые сценарные спецификации интерпретацию данных и/или времени. Обе группы используют модификации диаграмм MSC или UML SD для составления спецификаций с динамическими конструкциями. Так, в работе [3] вместо MSC-диаграмм

используются так называемые *«оклеты»* (*oclets*), описывающие частично-упорядоченный набор действий при помощи помеченной сети Петри, и которые предназначены как для более удобной абстракции всей системы, так и для простого способа представления данных в оклетах. В работе [19] стандарт диаграмм UML SD дополняется аннотациями из расширения *MARTE* для UML, позволяющего задавать временные характеристики системы, а также описывать нефункциональные свойства системы. В настоящей работе, в отличие от [3] и [19], рассматриваются только стандартные MSC-диаграммы без дополнительных модификаций.

На основании приведенного обзора литературы можно заключить, что основными преимуществами подхода, описанного в данной работе, являются широкий охват конструкций стандарта MSC с минимумом ограничений, динамическая интерпретация элементов типа «условие», поддержка HMSC-диаграмм в качестве механизма композиции основных MSC, а также разрешение проблемы локального выбора для элементов с альтернативами.

В планы дальнейшего исследования входит работа по реализации механизма интерпретируемых данных, описанных в расширении стандарта MSC, а также применение транслятора на практически значимых примерах (телекоммуникационные протоколы, система тестирования *NSUts*). Кроме того, планируется проведение верификации (H)MSC-спецификаций методом проверки моделей при помощи системы, разрабатываемой в лаборатории теоретического программирования ИСИ СО РАН.

СПИСОК ЛИТЕРАТУРЫ

1. Abdallah R., Gotlieb A., Helouet L., Jard C. Scenario Realizability with Constraint Optimization / FASE 2013 // LNCS. — 2013. — V. 7793. — P. 194-209.
2. Damm W., Harel D. LSCs: Breathing Life into Message Sequence Charts // Formal Methods in System Design. — 2001. — V. 19. — I. 1. — P. 45-80.
3. Fahland D., Prufer R. Data and Abstraction for Scenario-Based Modeling with Petri Nets / Application and Theory of Petri Nets // LNCS. — 2012. — V. 7347. — P. 168-187.
4. Fernandes J.M., Tjell S., Jorgensen J.B., Ribeiro O.R. Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net // 6th Intl Workshop on Scenarios and State Machines (SCESM 2007): ICSE 2007. — Minneapolis, USA, 2007.

5. Haugen O. MSC-2000 interaction diagrams for the new millennium // Computer Networks. — 2001. — V. 35(6). — P. 721-732.
6. Haugen O. From MSC-2000 to UML 2.0 – The Future of Sequence Diagrams / SDL 2001: Meeting UML // LNCS. — 2013. — V. 2078. — P. 38-51.
7. Haugen O. Comparing UML 2.0 Interactions and MSC-2000 / SAM 2004 // LNCS. — 2004. — V. 3319. — P. 65-79.
8. ITU-T Recommendation Z.120 (02/2011): Message Sequence Chart (MSC). — Geneva, 2011. — 146 p.
9. ITU-T Recommendation Z.120 (04/1998) Annex B: Algebraic semantics of Message Sequence Charts. — Geneva, 1998. — 84 p.
10. Jensen K., Kristensen L.M. Coloured Petri Nets. — Springer Verlag, Berlin Heidelberg, 2009. — 384 p.
11. Kryvyi, S.; Matvyeyeva, L. Algorithm of Translation of MSC-specified System into Petri Net // Fundamenta Informaticae – Special Issue on Concurrency Spec. and Programming. — 2007. — V. 79. — I. 3-4. — P. 431-445.
12. Krüger I.H. Distributed System Design with Message Sequence Charts: Ph.D. Thesis. — Technische Universität München, 2000. — 370 p.
13. Mauw S., Reniers M.A., Willemse T.A.C. Message Sequence Charts In The Software Engineering Process // Computing Science Reports 00/12. — Eindhoven, 2000.
14. Muscholl A., Peled D. Deciding Properties of Message Sequence Charts / Scenarios: Models, Transformations and Tools // LNCS. — 2005. — V. 3466. — P. 43-65.
15. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modelling Language (3rd Edition). — Addisson-Wesley, 2003.— 224 p.
16. Reniers M.A. Message Sequence Chart: Syntax and Semantics: PhD Thesis. — Eindhoven University of Technology, 1999. — 216 p.
17. Ribeiro O.R., Fernandes J.M. Some Rules to Transform Sequence Diagrams into Coloured Petri Nets // 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006). — Aarhus, Denmark, 2006. — P. 237-256.
18. Mauw S., Bos V. Drawing Message Sequence Charts with LaTeX // Tugboat. — 2001. — V. 22. — P. 87-92.
19. Yang N., Yu H., Sun H., Qian Z. Modeling UML sequence diagrams using extended Petri nets // Telecommunication Systems. — 2012. — V. 51. — I. 2-3. — P. 147-158.

20. Крывый С.Л., Чугаенко А.В., Матвеева Л.Е. Исследование свойств документов MSC с помощью преобразования их в сети Петри // Кибернетика и системный анализ: Международный научно-теоретический журнал. — 2009. — N 6. — С. 165-171.
21. Чугаенко А.В., Крывый С.Л. Об алгоритме перевода документов MSC в сети Петри // Проблемы программирования — 2008. — N 2-3. — С. 587-594.
22. Liang H., Dingel J., Diskin Z. A Comparative Survey of Scenario-Based to State-Based Model Synthesis Approaches / SCESM 2006 // ACM. — — NY, 2006. — P. 5–12.

ПРИЛОЖЕНИЕ

Список используемых функций языка CPN ML

- функция создания нового счетчика текущего цикла и добавление его в список счетчиков nlist:

```
fun CreateLoop( d, nlist ) = ( n, [1] ^^ nlist );
```

- функция увеличения текущего счетчика цикла на единицу при условии, что значение счетчика не превышает значения n:

```
fun NextLoop( n, d, nlist ) =
    if ( List.hd nlist < n )
    then 1`( d, [List.hd nlist + 1] ^^ List.tl nlist )
    else empty;
```

- функция удаления текущего счетчика цикла (завершение цикла) при условии, что значение счетчика совпадает со значением n:

```
fun EndLoop( n, d, nlist ) =
    if ( List.hd nlist = n )
    then 1`( d, List.tl nlist )
    else empty;
```

- функция для определения принадлежности элемента a списку:

```
fun mem [] a = false
      | mem (x::xs) a = a=x orelse mem xs a;
```

- функция для определения принадлежности одного списка другому:

```
    fun contains _ [] = true
            | contains [] (x::xs) = false
            | contains ys (x::xs) = (mem ys x) andalso (contains ys xs);
```

- функция для получения списка текущих значений меток множества сущностей glist (list — текущее множество установленных меток диаграммы):

```
    fun getCondList( glist, list ) =
if list = [] then []
    else
        let
            val head = List.hd( list )  : COND
            val tail = List.tl list
            val inst = #1( head )
            val conds = #2( head )
        in
            if contains glist inst andalso List.length glist = List.length
inst
            then conds
            else getCondList(glist, tail)
    end;
```

- функция для удаления в заданном множестве сущностей glist списка текущих значений меток (list — текущее множество установленных меток диаграммы):

```
     fun removeCond( glist, list ) =
if list = [] then []
    else
        let
            val head = List.hd( list )  : COND
            val tail = List.tl list
            val inst = #1( head )
        in
            if contains glist inst andalso List.length glist = List.length
inst
            then removeCond(glist, tail)
```

```
            else head::( removeCond(glist, tail) )
    end;
```

- функция для определения установочного условия (множеству сущностей instlist устанавливается список меток namelist; condlist — текущее множество установленных меток диаграммы):

```
    fun set( instlist, namelist, condlist ) =
 if instlist = [] then condlist
    else
        let
            val newlist = removeCond( instlist, condlist )
        in
            ( instlist, namelist )::newlist
    end;
```

- функция для проверки охранного условия (возвращает значение *true*, если множество сущностей instlist содержит список меток namelist; condlist — текущее множество установленных меток диаграммы):

```
    fun guard( instlist, namelist, condlist ) =
 if instlist=[] then true
    else
        let
            val newlist = getCondList( instlist, condlist )
        in
            contains newlist namelist
    end;
```

- функция, являющаяся отрицанием возвращаемого значения вышеописанной функции guard:

```
    fun eguard( instlist, namelist, condlist ) =
        not( guard( instlist, namelist, condlist) );
```

- функция для увеличения общего счетчика итераций n для циклов вида *loop<n,m>* (устанавливает новое значение счетчика, если значение flag, отражающее факт выполненной фиксации верхней границы цикла, ложно):

```
     # functions for complex Loop
     fun UpdCounter( n, flag, nlist ) =
if flag = false
```

```
    then if (List.hd nlist + 1 > n)
        then 1`( n+1, flag )
        else 1`( n, flag )
    else 1`( n, flag );
```

- функция увеличения текущего счетчика сущности для циклов вида *loop<n,m>*:

```
     fun UpdLoop( d, nlist ) =
        1`( d, [List.hd nlist + 1] ^^ List.tl nlist );
```

- функция удаления текущего счетчика сущности для циклов вида *loop<n,m>*:

```
     fun DeleteLoop( d, nlist ) =
        1`( d, List.tl nlist );
```

- спусковая функция для циклов вида *loop<n,m>*, которая в зависимости от возвращаемого логического значения определяет возможность начала новой итерации в цикле:

```
     fun PreLoop( k, n, flag, nlist ) =
if (List.hd nlist < k orelse k = 0)
    then if flag = true
         then if (List.hd nlist < n)
              then true
              else false
         else true
    else false;
```

- спусковая функция для циклов вида *loop<n,m>*, содержащих охранное условие:

```
     fun PreGuardLoop( k, n, flag, nlist, il, nl, cl ) =
guard( il, nl, cl ) andalso PreLoop( k, n, flag, nlist );
```

- спусковая функция для циклов вида *loop<n,m>*, которая в зависимости от возвращаемого логического значения определяет возможность завершения цикла:

```
     fun PostLoop( k, n, flag, nlist ) =
if (List.hd nlist >= n andalso (List.hd nlist > k orelse k = 0))
    then if flag = true
```

```
        then if (List.hd nlist = n)
             then true
             else false
        else true
    else false;
```

- спусковая функция для циклов вида *loop<n,m>*, содержащих охранное условие:

```
     fun PostGuardLoop( k, n, flag, nlist, il, nl, cl ) =
eguard( il, nl, cl ) orelse PostLoop( k, n, flag, nlist );
```

- функции на дугах, определяющие охранные условия для элемента параллельной композиции:

```
     fun func( d, instlist, namelist, condlist ) =
if guard( instlist, namelist, condlist )
        then 1`d
        else empty;


     fun nfunc( d, instlist, namelist, condlist ) =
if not( guard( instlist, namelist, condlist ) )
        then 1`d
        else empty;

     fun efunc( d, instlist, namelist, condlist ) =
if eguard( instlist, namelist, condlist )
        then 1`d
        else empty;

     fun enfunc( d, instlist, namelist, condlist ) =
if not( eguard( instlist, namelist, condlist ) )
        then 1`d
        else empty;
```

С. А. Черненок, В. А. Непомнящий

АНАЛИЗ MSC-ДИАГРАММ РАСПРЕДЕЛЕННЫХ СИСТЕМ С ПОМОЩЬЮ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ

Препринт
171

Рукопись поступила в редакцию 26.09.2013
Редактор Т.М. Бульонкова
Рецензент Т.Г. Чурина

Подписано в печать 24.10.2013
Формат бумаги 60 × 84 1/16 Объем 3.87 уч.-изд.л., 3.54 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г. Бердск, ул. О. Кошевого, 6, оф. 2, тел. (383-41) 2-12-42