ПРОГРАММИРОВАНИЕ, 2016, № 5, с. 49–67

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

УДК 517.51+514.17

АВТОМАТИЧЕСКОЕ ПОСТРОЕНИЕ СИСТЕМ РАСПРЕДЕЛЕННЫХ КОМПОНЕНТОВ ПО МОДЕЛЯМ ВЛОЖЕННЫХ СЕТЕЙ ПЕТРИ *

© 2016 г. Л.В. Дворянский, И.А. Ломазова

Национальный исследовательский университет "Высшая школа экономики"

101000 Москва, ул. Мясницкая, 20

E-mail: leo@mathtech.ru, ilomazova@hse.ru

Поступила в редакцию 10.04.2016

Многоуровневые мультиагентные системы (МАС) с динамической структурой широко используются при решении важных прикладных задач в телекоммуникационных, транспортных, социальных и других системах. Поэтому обеспечение корректности таких систем является актуальной и значимой задачей. Одним из самых уязвимых для ошибок этапов разработки системы в рамках модельно-ориентированного подхода является этап реализации, на котором по разработанной модели строится программный код. В данной работе представлен алгоритм автоматической трансляции моделей МАС в виде вложенных сетей Петри в системы распределенных компонентов. Вложенные сети Петри являются расширением сетей Петри в рамках подхода "сети внутри сетей", когда фишки в сети Петри сами могут являться сетями Петри, обладать автономным поведением и взаимодействовать с другими фишками сети. Это позволяет естественным образом моделировать МАС с динамической структурой.

Представленная в работе трансляция сохраняет уровень распределенности и важные поведенческие свойства исходной модели (безопасность, живость, условная живость), а также обеспечивает справедливость исполнения целевой системы. Использование такой трансляции позволяет автоматизировать построение распределенных МАС по моделям вложенных сетей Петри. В качестве апробации трансляция вложенных сетей Петри в системы распределенных компонентов была реализована на основе компонентной технологии EJB.

1. ВВЕДЕНИЕ

В последнее время гетерогенные распределенные системы, состоящие из разнородных вычислительных устройств, находят все большее применение и обладают все большими суммарными вычислительными и информационными ресурсами. Такие системы позволяют распределять вычислительную нагрузку между вычислительными узлами, распределенными в физическом или информационном пространстве. Так, например, в августе 2014 года вычислительная мощность устройств, участвующих в специализированной распределенной системе электронной наличности BitCoin, превысила 3 зеттафлопс [1]. На тот же момент суммарная вычислительная мощность 500 самых мощных суперкомпьютеров была на 4 порядка меньше — 0,250 эксафлопс [2].

Другой причиной использования распределенных систем является распределенный характер решаемых задач. Примерами являются система сбора данных с сенсоров, распределенных в пространстве, или система взаимодействия и координации людей с портативными вычислительными устройствами в спасательных операциях [3, 4, 5]. Эти системы состоят из большого количества распределенных мобильных вычислительных устройств-агентов.

Распределенные мультиагентные системы (МАС) все сильнее интегрируются в нашу

*Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ.

жизнь и оказывают все большее влияние как на жизнь каждого человека в отдельности, так и на социум в целом. Как следствие, ошибки в таких системах могут привести к значительным финансовым потерям, нанести ущерб здоровью или привести к человеческим жертвам. Поэтому разработка эффективных методов построения корректных и безопасных распределенных МАС является актуальной задачей.

Однако разработка корректных распределенных МАС затруднена большой сложностью таких систем. В мультиагентных системах с распределенными компонентами координация работы и поддержание согласованности системы требуют специальных распределенных алгоритмов для синхронизации агентов, выделения ресурсов, реализации взаимного исключения, сохранения согласованности состояния и др. В распределенных МАС мобильные агенты могут как перемещаться внутри системы, так и менять свое внутреннее состояние. Это существенно усложняет архитектуру системы и анализ корректности ее поведения.

Для решения указанных проблем необходимы новые формализмы и методы разработки распределенных систем, основанные на принципах модульной декомпозиции и композициональных свойствах систем. В частности, о важности таких исследований пишет в [6] И. Сифакис, лауреат премии Тьюринга 2007 года. В настоящее время в этом направлении ведется большое количество исследований.

Одним из наиболее популярных формализмов для моделирования и анализа распределенных систем являются сети Петри [7]. Сети Петри имеют удобную графическую нотацию и позволяют выразить параллельность, недетерминированность, и причино-следственные связи между элементами системы.

Для композиционального моделирования распределенных мобильных агентов удобно использовать вложенные сети Петри [8, 9] — расширение сетей Петри в рамках подхода “сети-внутри-сетей” (*nets-within-nets*), когда фишки, с помощью которых задается маркировка сети, сами являются сетями Петри. Подход “сети-внутри-сетей” позволяет моделировать агенты в МАС сетевыми фишками, которые могут менять свое внутреннее состояние и перемещаться по позициям системной сети. Синхронизация переходов во вложенной сети Петри моделирует координацию агентов, а срабатывания переходов в системной сети могут “порождать”, “клонировать” и “убивать” агентов в системе.

Вложенные сети Петри могут использоваться для моделирования адаптивных распределенных систем [10, 11, 12, 13, 14], систем мобильных роботов [15], координации сенсорных сетей мобильных агентов [16], архитектуры инновационных космических систем [17], серийных и реконфигурируемых производственных систем [18, 19, 20].

Для вложенных сетей Петри разработаны методы анализа поведенческих свойств. Так, в работе [21] разработаны композициональные методы анализа свойств безопасности и живости. В [22] показано, как методы верификации, основанные на построении развертки модели (*unfolding*) могут применяться для проверки свойств вложенных сетей Петри. В работе [23] описана трансляция вложенных сетей Петри в раскрашенные сети Петри для анализа моделей с помощью инструментария CPNtools. В работе [24] представлена трансляция подкласса рекурсивных вложенных сетей Петри в язык PROMELA и показано, как можно верифицировать некоторые свойства этих сетей с помощью системы проверки модели SPIN.

Однако при использовании формальных моделей на практике очень важен этап построения корректной распределенной системы по уже верифицированной модели. При ручной реализации системы на основе модели в процессе разработки могут появиться ошибки, связанные с различным пониманием модели архитектором и разработчиком. Одна и та же модель может иметь различные непротиворечивые интерпретации. Также написание исполнимого кода для сложной распределенной МАС вручную требует высокой квалификации разработчиков и сложных процедур распределенного тестирования.

Альтернативой ручной разработке является автоматическая генерация исполнимого кода по модели. При автоматической кодогенерации достигается значительная экономия проектных ресурсов, качество целевого кода, возможность внесения изменений в модель после начала этапа реализации с последующей автоматической генерацией нового кода. Реализующая такую

генерацию программа, вообще говоря, тоже может содержать ошибки, но после отладки и при достаточно продолжительном использовании система кодогенерации становится достаточно надежной, чтобы получать исполнимый код с устойчивым качеством.

Реализации МАС посвящено достаточно большое количество исследований. Реализация МАС с использованием технологии CORBA рассматривается в работах [25, 26, 27]. В работе [28] описана реализация МАС с помощью технологий Java EE и EJB. Все эти работы показывают перспективность использования современных технологий (CORBA, EJB) для систем распределенных компонентов при разработке МАС.

Имеются также работы по автоматической генерации кода по заданной модели системы. Задача трансляции моделей параллельных объектно-ориентированных сетей Петри (*CO-OPN*) в распределенные компоненты Java рассматривалась в [29]. В этой работе для решения проблемы синхронизации конфликтов в распределенном исполнении сетей Петри используется подход, аналогичный стратегии PPA [39], основанный на использовании механизма транзакций для одновременного получения распределенных ресурсов-фишек. В случае неуспешного захвата ресурсов для срабатывания перехода вся транзакция, соответствующая переходу, отменяется, и все уже захваченные ресурсы освобождаются. Метод транзакций обеспечивает целостность системы, и переходы срабатывают в соответствии с правилами операционной семантики CO-OPN.

В работе [30] описано решение задачи кодогенерации для подкласса раскрашенных сетей Петри (*CPN*). Сложность кодогенерации для раскрашенных сетей Петри заключается в смешении потоков управления и потоков данных. Однако автоматическая кодогенерация может быть выполнена для раскрашенных сетей, в которых разделение процессов задано явно. В описанной в этой работе кодогенерации поток управления используется как каркас для синтезируемой системы.

В данной работе предлагается трансляция моделей вложенных сетей Петри в распределенные мультиагентные системы мобильных агентов. Представленная трансляция гарантирует сохранение уровня распределенности системы, справедливость исполнения переходов, поведенческую эквивалентность сгенерированной системы и исходной модели, а также сохранение важных свойств безопасности, живости, условной живости исходной модели.

Статья имеет следующую структуру. В разделе 2 дается пример и определение вложенных сетей Петри. В разделе 3 определяются свойства модели, которые нужно сохранить при трансляции модели в систему распределенных компонентов. В разделе 4 вводится промежуточный формализм — формальная модель систем распределенных компонент (ФМ СРК). В разделе 5 приводится алгоритм трансляции вложенных сетей Петри в ФМ СРК, в разделе 6 обосновывается корректность этой трансляции, а в разделе 7 описывается реализация алгоритма трансляции в системе EJBgenerator. В заключении обсуждаются полученные результаты и направления дальнейших исследований.

2. ВЛОЖЕННЫЕ СЕТИ ПЕТРИ

2.1. Иллюстрирующий пример

Для того, чтобы дать читателю предварительное представление о вложенных сетях Петри на рис. 1 приведен пример вложенной сети Петри, подробно описанный в [8]. Этот пример является моделью системы агентов, совместно использующих ограниченный набор ресурсов.

Вложенная сеть Петри NPN_1 состоит из основного компонента *SN*, называемого системной сетью, и элементных сетей *EN*, моделирующих поведение агентов. Системная сеть является сетью Петри высокого уровня, в разметке которой некоторые фишки сами являются сетями Петри. Фишки в позициях S_1 и S_4 системной сети — обычные атомарные “черные” фишки. В позициях S_2, S_3, S_6 системной сети находятся сетевые фишки — экземпляры элементной сети *EN* с, вообще говоря, различными собственными разметками. На рис. 1 от каждой сетевой фишки идет стрелка к соответствующей ей размеченной сети Петри.

Правила срабатываний переходов внутри сетевых фишек и в системной сети (кроме переходов помеченных метками l_i и $\bar{l}_i$) не отличаются от правил срабатывания переходов в классических

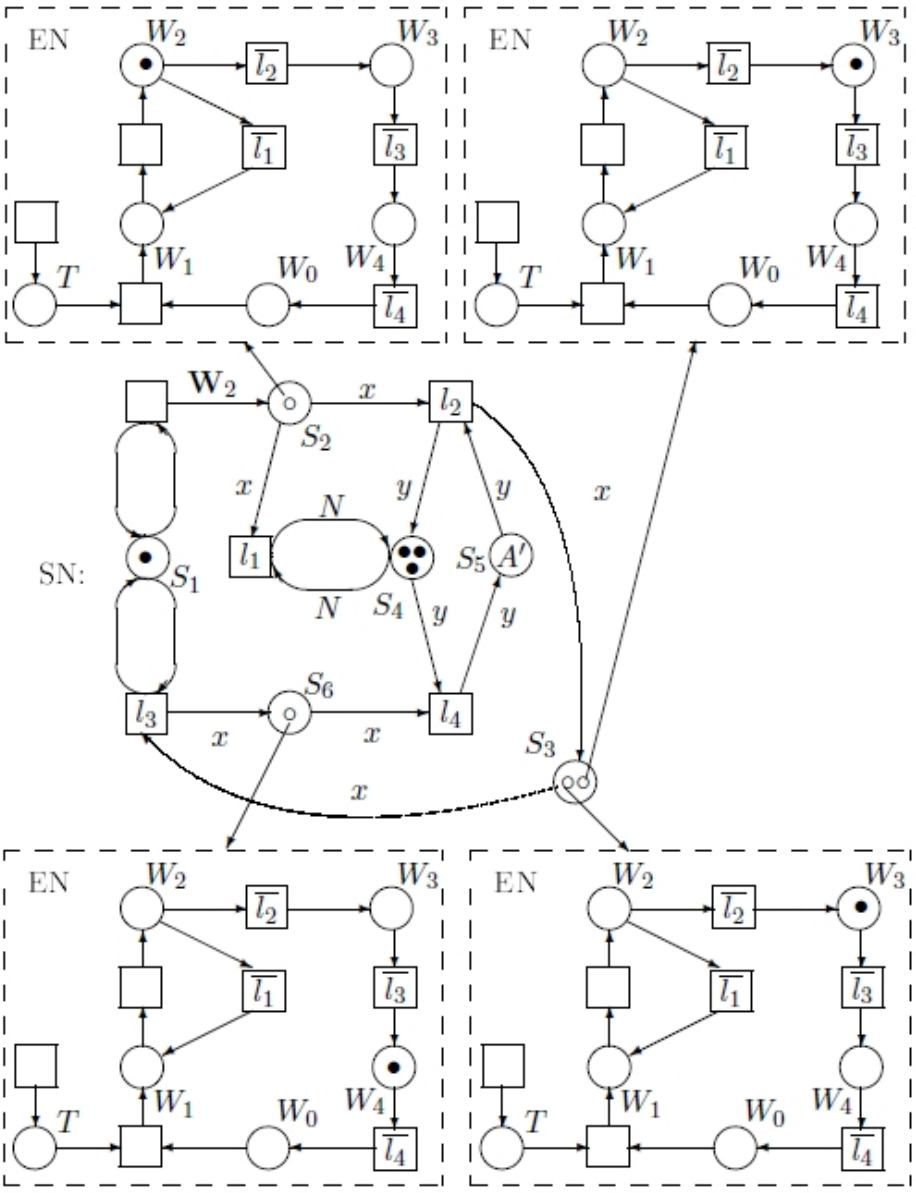


Рис. 1.
Пример вложенной сети Петри NPN_1.

сетях Петри. Метки l_i и $\bar{l}_i$ называются метками синхронизации, а переходы, помеченные ими, — переходами синхронизации. При этом метки l_i и $\bar{l}_i$ являются двойственными друг для друга. При срабатывании любого перехода синхронизации t_{SN} одновременно с ним срабатывают переходы во всех сетевых фишках, которые участвуют в срабатывании t_{SN}. В каждой сетевой фишке при этом срабатывает ровно один переход с меткой синхронизации, двойственный к метке перехода t_{SN} системной сети. В результате срабатывания синхронного перехода может измениться разметка как системной сети, так и задействованных в этом срабатывании сетевых фишек.

Другие примеры применения вложенных сетей Петри для моделирования мобильных агентов можно найти в [16, 17, 13, 18, 15, 14, 24, 20].

2.2. *Определения*

Приведенное далее определение раскрашенных сетей Петри почти не отличается от клас-

сического [31]. Вложенные сети Петри определяются как раскрашенные сети Петри над специальным универсумом. Поэтому определение раскрашенных сетей Петри параметризовано универсумом U возможных значений фишек. Также добавлены метки на переходах, необходимые для синхронизации переходов во вложенных сетях Петри.

Через $\mathbb{N}$ будем обозначать множество натуральных чисел, включая ноль.

Определим раскрашенные сети Петри над универсумом U. Каждая позиция раскрашенной сети типизирована подмножеством U. Дуги помечены выражениями *Expr* простого аддитивного языка над множеством переменных *Var* и множеством констант *Con*. Имеется фиксированная интерпретация $\mathcal{I}$ такая, что для любой согласованной по типам оценки $\nu : Var \to U$ значение $\mathcal{I}(e, \nu) \in \mathbb{N}^U$ выражения $e \in Expr$ определено. *Lab* — множество меток, которыми помечаются синхронизируемые переходы сети. Без ограничения общности можно считать, что для каждой метки $\lambda \in Lab$ имеется двойственная метка $\bar{\lambda}$.

Определение 1. Раскрашенная сеть *над универсумом U, это кортеж $(P, T, F, \upsilon, \rho, \Lambda)$, где*

- *P и T непересекающиеся конечные множества* позиций *и* переходов*;*
- *$F \subseteq (P \times T) \cup (T \times P)$ множество* дуг*;*
- *$\upsilon : P \to 2^U$ функция* типов позиций*, отображающая P в подмножества U;*
- *$\rho : F \to Expr$ функция* пометки дуг*;*
- *$\Lambda : T \to Lab$ частичная функция* пометки переходов.

Для заданной раскрашенной сети $N = (P, T, F, \upsilon, \rho, \Lambda)$ над универсумом U *разметка* определяется как функция $m : P \to \mathbb{N}^U$, такая что $m(p) \in \mathbb{N}^{\upsilon(p)}$ для $p \in P$. Пара (N, m) называется размеченной сетью Петри.

Пусть $N = (P, T, F, \upsilon, \rho, \Lambda)$ есть раскрашенная сеть Петри, m — некоторая разметка сети N. Переход $t \in T$ *активен* в разметке m тогда и только тогда, когда $\exists \nu \forall p \in P : (p, t) \in F \Rightarrow m(p) \geq \mathcal{I}(\rho(p, t), \nu)$. Активный переход t может *сработать*, порождая новую разметку m', такую что $m'(p) = m(p) - \mathcal{I}(\rho(p, t), \nu) + \mathcal{I}(\rho(t, p), \nu)$ для каждого $p \in P$ (обозначается $m \xrightarrow{t} m'$).

Размеченная раскрашенная сеть порождает систему переходов, представляющую поведение сети.

Определим *вложенные сети Петри* (*NP-сети*) как расширенные раскрашенные сети Петри над специальным универсумом. Этот универсум состоит из элементов некоторого конечного множества S (атомарные фишки) и размеченных сетей Петри (сетевые фишки). Мы рассматриваем только двухуровневые NP-сети, в которых сетевые фишки являются обыкновенными сетями Петри над конечным универсумом.

Итак, пусть S есть конечное множество атомарных объектов. Для раскрашенной сети Петри N через $\mathfrak{M}(N, S)$ обозначим множество всех размеченных сетей, получаемых из N добавлением разметок над универсумом S.

Пусть $N_1, \ldots, N_k$ — раскрашенные сети Петри над универсумом S. Определим универсум $\mathcal{U}(N_1, \ldots, N_k) = S \cup \mathfrak{M}(N_1, S) \cup \cdots \cup \mathfrak{M}(N_k, S)$, при этом элементы из S будем называть элементами типа S, а элементы из $\mathfrak{M}(N_i, S)$ — элементами типа $\mathfrak{M}(N_i, S)$. Через $\Omega(N_1, \ldots, N_k) = \{S, \mathfrak{M}(N_1, S), \ldots, \mathfrak{M}(N_k, S)\}$ обозначим множество типов элементов универсума $\mathcal{U}$.

Определение 2. *Пусть Lab — множество меток переходов, и пусть $N_1, \ldots, N_k$ — раскрашенные сети Петри над универсумом S, в которых некоторые переходы помечены метками из Lab.*

NP-сеть — это кортеж $NP = (N_1, \ldots, N_k, SN)$, где $N_1, \ldots, N_k$ называются элементными сетями, и SN называется системной сетью. Системная сеть $SN = (P_{SN}, T_{SN}, F_{SN}, \upsilon, \rho_{SN}, \Lambda)$ есть раскрашенная сеть Петри над универсумом $\mathcal{U} = \mathcal{U}(N_1, \ldots, N_k)$, где позиции типизированы типами из $\Omega = \Omega(N_1, \ldots, N_k)$, некоторые переходы помечены метками из Lab, и язык меток дуг Expr определен, как показано ниже. Для краткости, мы говорим, что позиция p, которой приписан тип $\mathfrak{M}(N, S)$, имеет тип N.

Пусть Con — множество констант, интерпретированных над $\mathcal{U}$, и Var — множество переменных типизированных Ω-типами. Тогда выражение языка Expr имеет вид $n_1 \cdot x_1 + \cdots + n_s \cdot x_s$, где $n_1, \ldots, n_s$ — натуральные коэффи-

циенты, $x_1, \ldots, x_s \in Con \cup Var$ и для каждого перехода $t \in T_{SN}$:

1. *Тип переменных и констант в выражении, приписанном дуге инцидентной позиции p, должен совпадать с типом p.*
2. *В выражениях на входных дугах перехода t нет констант, все коэффициенты при переменных равны 1, и множества переменных в выражениях на входных дугах одного и того же перехода не пересекаются.*
3. *Каждая переменная из выражения на любой выходной дуге перехода t должна встречаться в одном из выражений на входных дугах перехода t.*

Здесь первое условие обеспечивает правильное соответствие типов, второе исключает возможность проверить равенство внутренних разметок сетевых фишек. Без такого ограничения можно было бы проверять разметку сетевых фишек на пустоту, и формализм NP-сетей стал бы полным по Тьюрингу. Третье условие исключает бесконечное ветвление в системе переходов, представляющей поведение NP-сети.

Размеченная элементная сеть называется сетевой фишкой, а элемент множества S — атомарной фишкой.

Разметка NP-сети полностью определяется разметкой ее системной сети. Разметка в NP-сети сопоставляет каждой позиции системной сети некоторое мультимножество атомарных или сетевых фишек соответствующего типа.

Поведение NP-сети состоит из шагов трех видов.

Элементно-автономный шаг — это срабатывание непомеченного перехода в одной из сетевых фишек текущей разметки в соответствии с обычными правилами срабатывания для раскрашенных сетей Петри. Такие срабатывания изменяют только внутреннюю разметку одной из сетевых фишек.

Системно-автономный шаг — это срабатывание непомеченного перехода в системной сети в соответствии с правилами срабатывания для раскрашенных сетей Петри, как если бы сетевые фишки были просто раскрашенными фишками без внутренней разметки. Автономный шаг в системной сети может перемещать, копировать, генерировать или уничтожать фишки, задействованные в этом шаге, но не может изменить их внутренние разметки.

Шаг (вертикальной) синхронизации — это одновременное срабатывание перехода, помеченного меткой $\lambda \in Lab$, в системной сети вместе со срабатыванием переходов, помеченных $\bar{\lambda}$, во всех сетевых фишках задействованных (поглощаемых) срабатыванием этого перехода системной сети.

Пусть $M \xrightarrow{s} M'$ обозначает, что срабатывание шага s преобразует разметку M в разметку M'. Для последовательности шагов $\sigma = s_1, s_2, \ldots s_n$ мы пишем $m \xrightarrow{\sigma} m'$, когда $m = m_0 \xrightarrow{s_1} m_1 \xrightarrow{s_2} \ldots \xrightarrow{s_n} m_n = m'$. Разметка m достижима в NP-сети NP, если существует последовательность шагов σ такая, что $m_0 \xrightarrow{\sigma} m$, где m_0 — начальная разметка сети NP.

Поведение NP-сети NPN определяется системой переходов LTS_{NPN}, в которой состояниями являются достижимые в NPN разметки, переходы — шаги в NPN, а начальное состояние задается начальной разметкой сети.

Детальное описание NP-сетей можно найти в [9, 8]. Отметим, что в данной статье мы рассматриваем типизированный вариант NP-сетей, когда для каждой позиции определен её тип.

3. ОБЕСПЕЧЕНИЕ КОРРЕКТНОСТИ ГЕНЕРИРУЕМОЙ СИСТЕМЫ

Важным этапом разработки проектируемой системы в рамках модельно-ориентированой разработки является этап верификации модели. При этом применяются методы валидации и поведенческого анализа для выявления дефектов и несоответствия спецификациям модели.

Одной из основных проблем синтеза исполняемой системы по заданной модели является сохранение в результирующей программной системе поведенческих свойств уже проверенной модели. Необходимо, чтобы результирующая система на каждом шаге вела себя так же, как и исходная модель. Это означает, что система должна исполнять все действия, представленные в модели, и реализация модельного действия в системе должна иметь тот же наблюдаемый эффект, т.е. переводить систему в состояние, подобное состоянию модели после срабатывания соответствующего действия.

При этом допускается, что одному модельному действию соответствует несколько промежуточных шагов, выполняемых в нескольких компонентах программной системы, прежде чем она достигнет состояния соответствующего состоянию исходной модели. Состояния программной системы, в которых нет невыполненых промежуточных действий, называются *стабильными*. Для обоснования корректности реализации между состояниями модели и стабильными состояниями системы определяются отношение поведенческой эквивалентности или предпорядка [32, 33].

Выбор отношения эквивалентности/предпорядка определяется необходимостью сохранения в системе тех или иных поведенческих свойств модели. При правильном выборе эквивалентности две системы переходов, представляющие поведение модели и системы, одинаково хорошо подходят для выполнения практических задач, если они эквивалентны относительно выбранного отношения. Если между системами переходов LTS_1 и LTS_2 выполняется отношение поведенческого предпорядка $LTS_1 \sqsubseteq_R LTS_2$, то система LTS_2 ведет себя "не хуже" системы LTS_1. При этом выбранное отношение эквивалентности/предпорядка должно сохранять важные для выбранных задач поведенческие свойства. Подробное описание и классификацию таких отношений можно найти в работах [34, 35]. Приведенное словесное описание требования соответствует отношению предпорядка слабой симуляции с учетом стабильности $\mathcal{O}_{model} \sqsubseteq_{W^s} \mathcal{O}_{system}$.

Важными и наиболее часто проверямыми свойствами дискретных динамических систем являются свойства живости и безопасности. *Безопасность (safety)* системы относительно некоторого заданного нежелательного свойства ϕ означает, что во всех достижимых состояниях ϕ не выполняется. Примером нарушения свойства безопасности может служить одновременное нахождение нескольких процессов в критической секции.

Свойство *живости (liveness)* системы гарантирует выполнение в будущем некоторого "хорошего" свойства, т.е. живость системы относительно свойства ϕ означает, что в любом максимальном исполнении встречается состояние, удовлетворяющее ϕ. Исполнение называется максимальным, если оно не может быть продолжено, т.е. оно бесконечно или завершается тупиковым состоянием (deadlock). Примером условия живости является требование к участникам сетевого протокола прислать подтверждение об успешном завершении на сервер.

Условная живость (conditional liveness) [36, 37] системы гарантирует выполнение в будущем некоторого "хорошего" свойства ϕ, если до этого встретилось состояние удовлетворяющее условию ψ.

Необходимо, чтобы при трансляции модели в исполняемую систему были сохранены свойства безопасности, живости и условной живости исходной модели.

Еще одним важным аспектом при разработке распределенных систем является обеспечение равноправия компонентов системы. В литературе описаны три вида равноправия или справедливости (fairness) для процессов в распределенных программных системах: безусловная, слабая и сильная. В данной работе используется сильная справедливость. В системе переходов LTS исполнение $\rho = m_0 \xrightarrow{s_1} m_1 \xrightarrow{s_2} \ldots$ справедливо для действия s при выполнении одного из следующих условий

- исполнение ρ имеет конечную длину;
- если исполнение ρ содержит бесконечное число состояний, в которых действие s может выполниться, то s выполняется в ρ бесконечное число раз.

Справедливость всех исполнений LTS не позволяет одним процессам, конкурирующим за ресурсы, постоянно захватывать ресурсы в ущерб другим процессам.

В [38] показано, что для многих систем свойство живости и условной живости может быть выполнено лишь при условии справедливости исполнений. Если при анализе модели мы предполагаем выполнение условия справедливости, а в синтезированной системе эта справедливось не обеспечена, то трансляция не сохраняет свойства живости, верные для модели.

Справедливость исполнения может быть обеспечена как на уровне модели, так и в виде дополнительных требований (спецификаций). Формализм вложенных сетей Петри не имеет средств

для того, чтобы обеспечить выполнение справедливости. Поэтому справедливость исполнений должна быть обеспечена применением справедливых стратегий распределения ресурсов в результирующей системе.

Таким образом, необходимо, чтобы трансляция модели в виде NP-сети в исполняемую систему распределенных компонентов обеспечивала такое подобие исходное модели и целевой системы, которое гарантирует сохранение свойств безопасности, живости и условной живости. Кроме этого, в целевой системе должна обеспечиваться справедливость всех исполнений.

Чтобы формально обосновать корректность трансляции моделей NP-сетей в системы распределенных компонентов, необходимо определить формальную модель системы распределенных компонентов (ФМ СРК). При этом ФМ СРК, хотя и являются формальными моделями, но имеют низкий уровень абстракции, близкий к целевому коду, и непосредственно повторяют структуру целевой системы компонентов так, что отображение ФМ СРК в компоненты целевой платформы не представляет сложности. В качестве апробации представленного в данной работе метода был разработан прототип транслятора NP-сетей в код на платформе EJB (см. раздел 7). Конечно, перевод ФМ СРК в код целевой платформы также желательно строго обосновать. Это однако выходит за рамки данной статьи и является предметом последующего исследования.

4. ОПРЕДЕЛЕНИЕ ФОРМАЛЬНОЙ МОДЕЛИ СИСТЕМ РАСПРЕДЕЛЕННЫХ КОМПОНЕНТОВ

Система распределенных компонентов (СРК) состоит из контейнеров компонентов, которые содержат программные компоненты. Контейнеры обеспечивают важные для распределенной системы сервисы, такие как сохраняемость (persistence), коммуникация, распределение нагрузки и др. Программные компоненты взаимодействуют друг с другом и с внешней средой посредством передачи сообщений и удаленных вызовов процедур. Каждый программный компонент может исполняться с помощью нескольких нитей.

В данном разделе приводится определение формальной модели системы распределенных компонентов (ФМ СРК).

ФМ СРК будет использоваться в качестве уточнения (refinement) формализма NP-сетей. Например, реализация шага вертикальной синхронизации происходит атомарно в операционной семантике NP-сетей, но реализация такого шага в целевом коде системы требует синхронизации нескольких конкурирующих распределенных узлов. При этом в зависимости от способа реализации синхронизации результирующая система может как сохранять, так и не сохранять желаемые поведенческие свойства модели. В ФМ СРК переходы вертикальной синхронизации NP-сети разворачиваются в наборы виртуальных переходов, что позволяет применить стратегию **ETO** [39] для синхронизации распределенных компонентов. ФМ СРК позволяет задать способ распределения поведения и состояния элементов NP-сетей по целевым программным компонентам, сохраняющим уровень распределенности исходной модели. В формализме NP-сетей такое распределение никак не определено.

Распределенная программная реализация срабатывания переходов сетей Петри требует разрешения конфликтов между переходами. При централизованной реализации разрешения конфликтов на одном вычислительном устройстве выбор между переходами происходит в управляющем процессе, которому доступен весь набор активных конфликтующих переходов. Но при централизованном управлении в распределенной системе управляющий процесс становится узким местом всей системы, и система теряет все преимущества распределенного исполнения.

В работе [39] были рассмотрены возможные стратегии программной реализации распределенного разрешения конфликтов между переходами сетей Петри. Наименьшей коммуникационной сложностью обладают стратегии **PPA** и **PTA**, но, к сожалению, данные стратегии не обеспечивают сохранение свойств живости. Свойства живости сохраняет стратегия **ETO**, которая состоит в опросе ресурсов-фишек в зафиксированном ациклическом порядке и ожидании освобождения необходимых ресурсов. Для повышения эффективности стратегия **ETO** учи-

тывает кратность входящих дуг переходов. Важно также, что при реализации очереди запросов на выделение ресурсов, стратегия **ETO** является справедливой относительно исполнения конфликтующих переходов.

Шаг синхронизации NP-сети включает срабатывание переходов в задействованных в этом шаге сетевых фишках и перехода системной сети. Сетевые фишки в системной сети могут перемещаться между позициями. Поэтому применение стратегии **ETO** для разрешения конфликтов переходов синхронизации напрямую невозможно.

Для решения данной проблемы для каждого перехода с меткой синхронизации строится компонент-монитор, который содержит в себе множество *виртуальных переходов*. Переход синхронизации системной сети может входить в несколько разных шагов вертикальной синхронизации. Каждый такой шаг задает комбинацию переходов элементных сетей, которые срабатывают одновременно с переходом системной сети. Для каждой уникальной комбинации переходов элементных сетей, которые могут сработать одновременно с заданным переходом системной сети, создается виртуальный переход, который является слиянием перехода системной сети и множества переходов элементных сетей.

На рис. 2 в серой зоне изображены виртуальные переходы для перехода вертикальной синхронизации t_1 в SN. В данной разметке сети возможны четыре шага вертикальной синхронизации — в синхронном срабатывании с t_1 может быть задействована фишка α_1 или фишка α_2 и один из переходов t_1, t_2 в фишке α_3. Для каждого шага создается виртуальный переход.

Стратегия **ETO** применяется к объединению автономных переходов системной сети, автономных переходов всех сетевых фишек и виртуальных переходов. Если в результате выполнения очередного шага во входных позициях перехода вертикальной синхронизации появляются или исчезают сетевые фишки, то для данного перехода множество виртуальных переходов переопределяется. Виртуальные переходы, которые содержали переходы из исчезнувших фишек, удаляются. Для появившихся фишек добавляются новые виртуальные переходы. После обновления множества переходов пересчитывается порядок опроса позиций в соответствии со стратегией **ETO**.

Полезные действия, которые совершаются при выполнении переходов NP-сети, определяются во внешних аннотациях, разрабатываются вручную и не являются частью формализма. Полезные действия, выполняющиеся в построенной ФМ СРК, соответствуют полезным действиям исходной NP-сети. При этом предполагается, что любые такие внешние действия выполняются за конечное время. Такое допущение реалистично, т.к. основной распределенный поток управления системой описывается с помощью NP-сети, а полезными действиями являются, например, вызовы внешних web-сервисов или передача сигнала по шине данных. Внешние ошибки не рассматриваются в данной работе, и в ФМ СРК не используются стратегии разрешения внешних дефектов кода (тупики, зависания, дивергенции) в полезных действиях. Таким образом при условии завершимости выполнения внешнего полезного кода за конечное время гарантируется корректность поведения всей целевой системы.

Определение 3. Формальная модель системы распределенных компонентов (ФМ СРК) — *это набор* $SDC = (C_T, C_P, Atoms, NetTokens, inscript)$, *где* C_P, C_T — *множества компонентов позиций (p-компоненты) и переходов (t-компоненты), соответственно.* $Atoms$ — *множество всех возможных значений атомарных фишек и* $NetTokens$ — *множество всех ссылок на компоненты сетевых фишек (α-компоненты). Каждый t-компонент состоит из контейнеров* $(OutArcs_t, InArcs_t)$, *где* $OutArcs_t$ — *неупорядоченный контейнер ссылок на p-компоненты, соответствующие исходящим дугам, а* $InArcs_t$ — *упорядоченный контейнер ссылок на p-компоненты, соответствующие входящим дугам. Функция* $inscript : OutArcs_t \cup InArcs_t \to Expr$ *ставит в соответствие дугам подписи на языке* $Expr$. *Каждый t-компонент* c_t *с меткой синхронизации* λ *содержит контейнер виртуальных переходов* $c_t : VirtTrans_t$.

Конфигурацией ФМ СРК SDC называется кортеж $C_{SDC} = (A, C_\alpha, NT, VT)$, где

1. $A = \{A_{p_1}, \cdots, A_{p_N}\}$ — набор множеств зна-

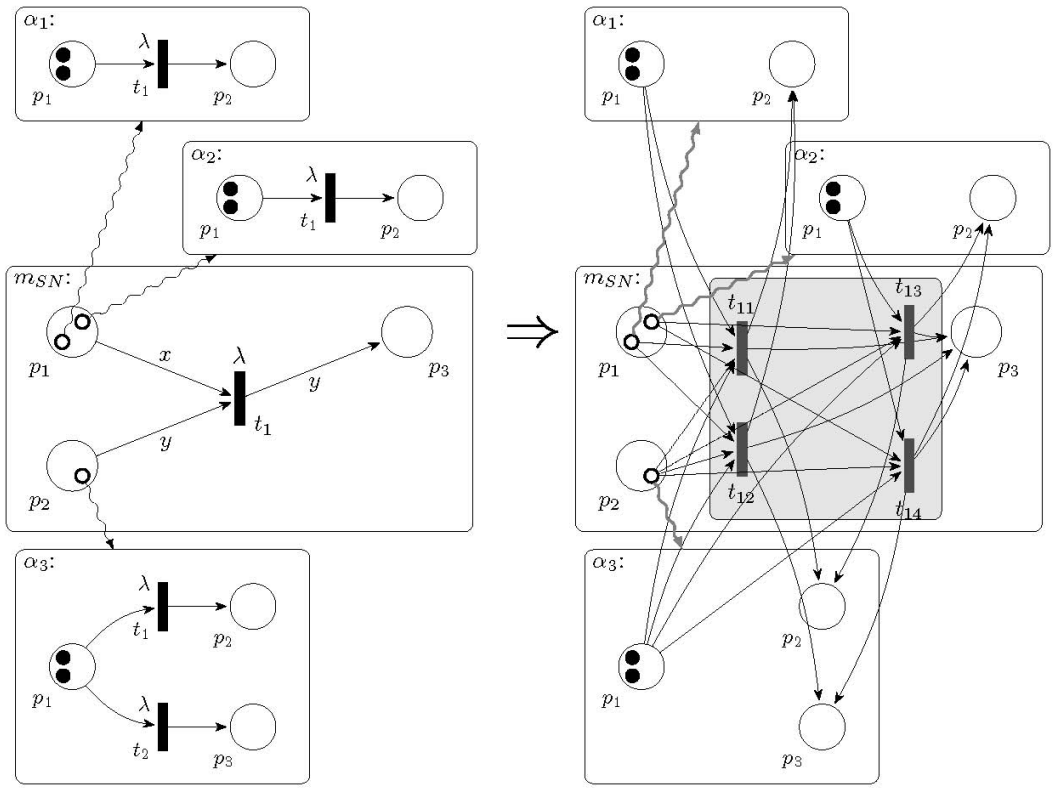


Рис. 2.
Создание виртуальных переходов вертикальной синхронизации.

чений атомарных фишек в p_i-компонентах;

2. $C_\alpha = \{c_{\alpha_1}, \cdots, c_{\alpha_M}\}$ — множество α-компонентов. Каждый компонент сетевой фишки (α-компонент) c_α — это помеченная раскрашенная сеть Петри над универсумом $Atoms$;

3. $NT = \{NT_{p_1}, \cdots, NT_{p_N}\}$ — набор множеств ссылок на α-компоненты в p_i-компонентах;

4. $VirtTrans$ — множество наборов $\{(T_t^{virt}, InArcs_t^{virt}, OutArcs_t^{virt})$ для каждого контейнера $VirtTrans_t$ t-компонента c_t где контейнер T_t^{virt} содержит виртуальные переходы порожденные для t, а входящие и исходящие дуги виртуальных переходов хранятся в контейнерах $InArcs^{virt} : \left(C_P \cup \bigcup\limits_{c_\alpha \in C_\alpha} P_\alpha\right) \times T^{virt}$ и $OutArcs^{virt} : T^{virt} \times \left(C_P \cup \bigcup\limits_{c_\alpha \in C_\alpha} P_\alpha\right)$. Дуги виртуального перехода соединяют виртуальные переходы с p-компонентами системной сети и внутренними позициями α-компонент, которые участвуют в срабатывании данного виртуального перехода.

В ФМ СРК SDC возможны исполнения шагов внутренних переходов в α-компонентах, шагов переходов реализованных компонентами системных переходов и шагов виртуальных переходов. Последние соответствуют отдельным означиваниям в шагах вертикальной синхронизации исходной NP-сети.

Внутренние переходы α-компонентов срабатывают согласно правилам срабатываниям переходов раскрашенной сети Петри: из контейнеров соответствующих входным позициям извлекаются атомарные фишки и помещаются в контейнеры, соответствующие выходным позициям. Шаги, соответствующие срабатыванию внутренних переходов компонентов сетевых фишек, назовем элементно-автономными по аналогии с шагами NP-сети.

Компонент перехода, соответствующий системным автономным переходам, срабатывает,

когда в p-компонентах определенных $InArcs_t$, имеется достаточно значений атомарных фишек в контейнере $Atoms_p$ и достаточно α-компонент в контейнере $NetTokens_p$. При срабатывании t-компонент извлекает из контейнеров входных позиций значения и ссылки на α-компоненты и помещает их в контейнеры выходных позиций в соответствии. Каждое выполнение шага в SDC состоит из нескольких промежуточных шагов: связывание и блокировка ресурсов t^-, выполнение внешних действий τ, освобождение ресурсов t^+. Наглядно соответствие между срабатыванием перехода в NP-сети и ФМ СРК изображено на рис. 3. На шаге "связывание и блокировка ресурсов" происходит извлечение ресурсов (значения атомарных фишек и ссылки на α-компоненты) из контейнеров входных позиций. В реализации (но не формализме) при этом происходит связывание этих ресурсов с локальными переменными для использования в полезных действиях, а также блокировка доступа к ресурсам. На шаге "внешние действия" происходит выполнение полезных действий, которые определяются внешними спецификациями. Так как данные действия определяются вне формализма и, по заданным условиям, должны заканчиваться за конечное время, данный шаг моделируется в системе переходов ненаблюдаемым действием τ. Данный шаг используется в конкретной реализации для инструментирования целевого кода полезными действиями. На шаге "освобождение ресурсов" ресурсы помещаются в контейнеры выходных позиций. В реализации при этом дополнительно происходит освобождение от блокировки. В системе переходов данный шаг помечается меткой t^+.

Срабатывание виртуальных переходов полностью повторяет срабатывание компонентов системных автономных переходов, за исключением того, что фишки могут извлекаться и перемещаться не только в p-компонентах системной сети, но и во внутренних позициях задействованных α-компонентов. Исполнение также разбито на 3 подшага. На первом шаге t^- извлекаются все ресурсы из контейнеров входных позиций как в p-компонентах, так и в α-компонентах. На втором шаге выполняются внешние полезные действия. На заключительном t^+ шаге ресурсы освобождаются и помещаются в выходные p-компоненты и α-компонентов.

5. ТРАНСЛЯЦИЯ *NP*-СЕТЕЙ В ФМ СРК

Для определения трансляции необходимо задать отображение элементов NP-сети на физические элементы систем распределенных компонентов (см. рис. 4). Каждый элемент NP-сети (позиция, переход, фишка), в зависимости от выбранной схемы отображения, может быть представлен с помощью одного или множества элементов СРК. Выбранная схема отображения определяет уровень распределенности, гибкости и масштабируемости результирующей СРК.

В данном разделе описывается схема трансляции двухуровневых вложенных сетей Петри в ФМ СРК. По каждой позиции системной сети строится p-компонент, который содержит в себе значения атомарных фишек и ссылки на α-компоненты. Данные, относящиеся к значениям атомарных фишек, представлены в виде объектов с сохраняемым (persistent) состоянием. По каждому переходу системной сети строится t-компонент, который имеет ссылки на p-компоненты для входных и выходных позиций перехода в модели. Если переход системной сети имеет метку вертикальной синхронизации, то для него создается множество виртуальных переходов, соответствующее возможным срабатываниям перехода. По каждой сетевой фишке строится α-компонент, который представляет поведение и текущее состояние сетевой фишки. Если позиция системной сети в NP-сети содержит сетевые фишки, то соответствующий ей p-компонент хранит ссылки на α-компоненты соответствующих сетевых фишек.

Активными элементами результирующей ФМ СРК являются t-компоненты, полученные из переходов системной сети, а также α-компоненты. t-компоненты запрашивают ресурсы у p-компонентов, соответствующих входным позициям и в результате срабатывания помещают ресурсы-фишки в выходные p-компоненты. Данные, соответствующие значениям атомарным фишкам, передаются между p-компонентами по значению. При срабатывании t-компонента, перемещающего сетевые фишки, в выходные p-компоненты передаются ссылки на соответствующие α-компоненты.

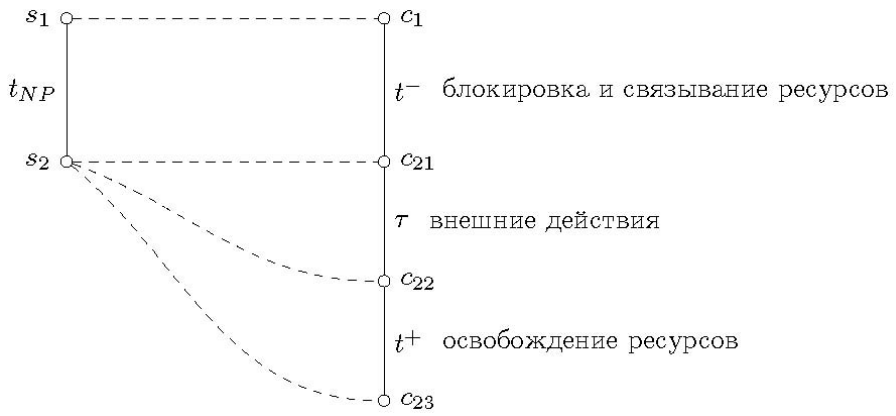
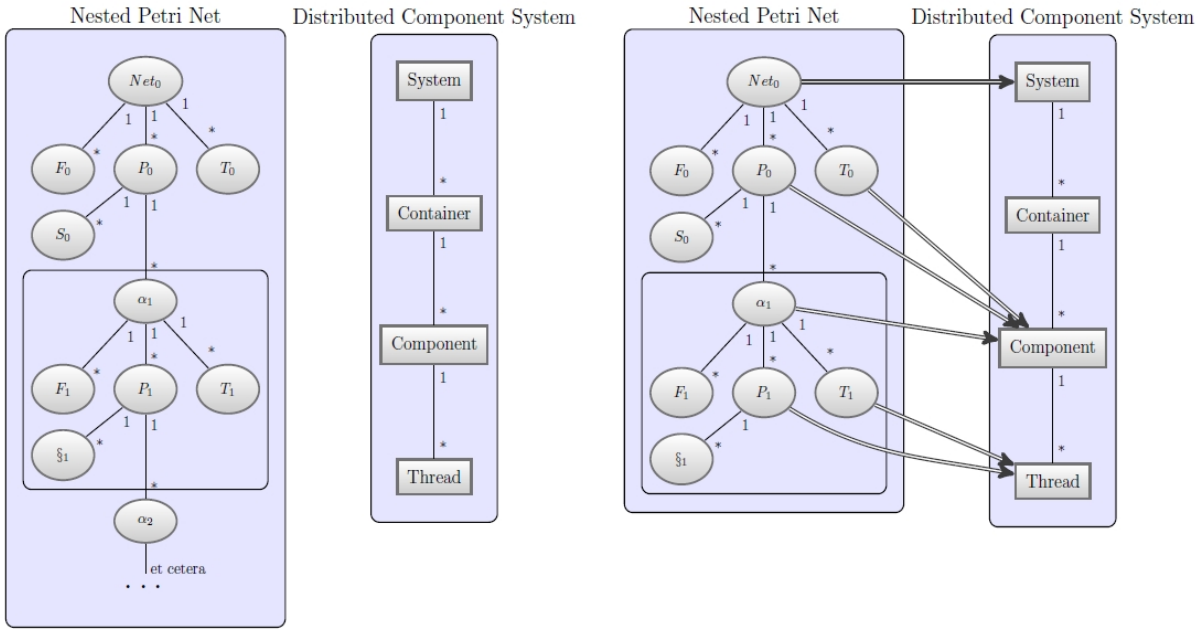


Рис. 3.
Соответствие между шагом NP-сети и шагом ФМ СРК



(*а*) Структура NP-сетей и СРК. (*б*) Отображение NP-сети на СРК.

Рис. 4.
Схема отображения NP-сети на СРК.

Опишем алгоритм трансляции NP-сети в ФМ СРК. Для заданной NP-сети $NPN = (N_1, \ldots, N_k, SN)$ с начальной разметкой M_0 результатом трансляции будет ФМ СРК $SDC = (C_T, C_P, Atoms, NetTokens, inscript)$ и начальная конфигурация $C_{SDC} = (A, C_\alpha, NT, VT)$, которая удовлетворяет требованиям сохранения поведенческих свойств и справедливости.

Одновременно с построением системы распределенных компонентов будут определены две биекции: функция $tr_P : P_{SN} \to C_P$, отображающая позиции NP-сети в p-компоненты, и функция $tr_T : T_{SN} \cup T_{N1} \cup \cdots \cup T_{Nk} \to C_T$, отображающая переходы NP-сети в t-компоненты.

Определяемая трансляция состоит из следующих шагов.

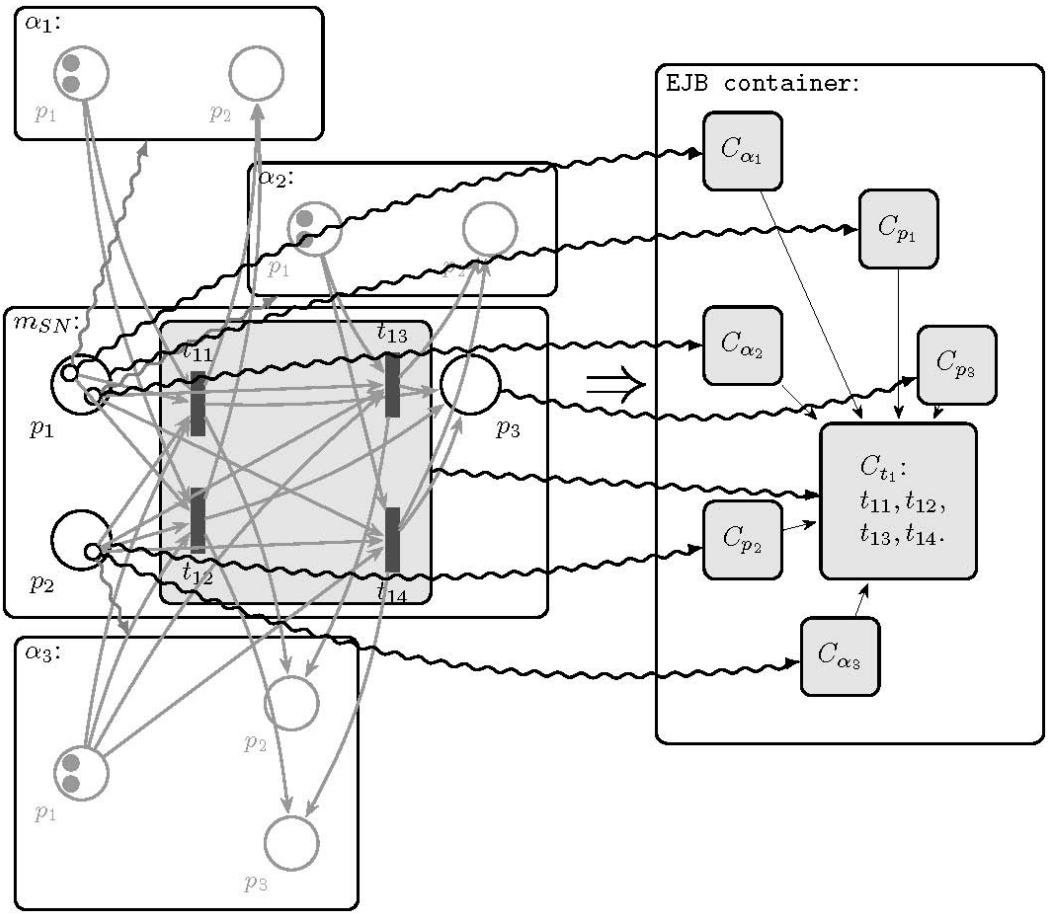


Рис. 5.
Трансляция виртуальных переходов в t-компоненты.

Шаг 1: создание p-компонентов для позиций системной сети *SN*. Для каждой позиции $p \in P_{SN}$ в системной сети SN строится новый p-компонент c_p. Полагаем $tr_P(p) = c_p$.

Тип контейнера A_p для значений совпадает с типом позиции p. Если p типизирована элементной сетью N в SN, то компонент содержит контейнер $c_p : NetTokens$, который содержит ссылки на α-компоненты типа E.

Шаг 2: создание компонентов для системно-автономных переходов. Для каждого непомеченного перехода $t \in T_{SN}$ строим новый t-компонент c_t и задаем $tr_T(t) = c_t$. Порядок элементов в $c_t : InArcs$ определяется стратегией **ETO**.

Для каждой входной дуги $f = (p, t) \in F_{SN}$ (выходной дуги $f = (t, p) \in F_{SN}$) в контейнер $c_t : InArcs$ (в $c_t : OutArcs$, соответственно) добавляем ссылку на компонент c_p. Ссылка имеет атрибут-контейнер $inscript$, который хранит значение $\rho_{SN}(f)$.

Шаг 3: создание компонентов для сетевых фишек. Для каждой сетевой фишки α в позиции системной сети $p \in P_{SN}$ создаем компонент c_α, который располагается в том же контейнере компонентов, что и компонент c_p, и задаем $tr_\alpha(\alpha) = c_\alpha$.

Элементные сети являются раскрашенными сетями Петри над универсумом атомарных фишек, поэтому для реализации их поведения можно применить стандартные способы кодогенерации для классических сетей Петри [40]. Однако переходы синхронизации элементных сетей будут обрабатываться в компонентах-мониторах для переходов синхронизации системной сети. Поэтому код

для исполнения таких переходов не генерируется. Таким образом, компонент c_α содержит код для исполнения элементной сети и контейнер $c_\alpha : Atoms$ для значений атомарных фишек. В контейнер позиции $c_p : : NetTokens$ добавляется ссылка на c_α.

Шаг 4: создание компонентов-мониторов для переходов синхронизации. Для каждого перехода $t \in T_{SN}$ с меткой синхронизации λ строим новый t-компонент c_t, который содержит контейнер $c_t : VirtTrans$. Далее мы рассматриваем все уникальные означивания переменных на входных дугах сетевыми фишками из начальной маркировки. Для каждого означивания ν и для каждой возможной комбинации переходов синхронизации в сетевых фишках, входящих в ν, строим виртуальный переход v. В реализации виртуальный переход не является отдельным компонентом, а лишь структурой данных, которая соответствует исполнению шага синхронизации NP-сети при заданном означивании и выбранных внутренних переходах сетевых фишек.

Шаг 5: применение стратегии разрешения конфликтов ETO. После создания всех виртуальных переходов на Шаге 4 ко всем автономным переходам системной сети и сетевых фишек, а также к виртуальным переходам применяется стратегия **ETO**. Применение стратегии состоит в нахождении конфликтующих блоков переходов и задании ациклического порядка запроса ресурсов [39] в упорядоченных контейнерах $InArcs$.

Так как стратегия решения конфликтов применяется к виртуальным переходам, справедливость обеспечивается на уровне означиваний переходов синхронизации. То есть справедливый выбор осуществляется не между конфликтующими переходами, а между конфликтующими возможными означиваниями переходов синхронизации.

6. КОРРЕКТНОСТЬ ТРАНСЛЯЦИИ *NP*-СЕТЕЙ В ФМ СРК

Для обоснования корректности описанной трансляции построим отношение поведенческого предпорядка между исходной моделью NPN и результирующей ФМ СРК.

Для сравнения поведения пометим все автономные переходы в системной сети и элементных сетях NP-сети NPN индивидуальными метками из множества Λ. Будем считать, что помимо меток автономных переходов множество Λ включает также множество Lab меток синхронизации. Таким образом, каждый шаг NP-сети NPN помечен символом из Λ.

Поведение ФМ СРК также может быть представлено системой переходов, в которой состояниями являются достижимые конфигурации ФМ СРК, а действия выполнения кода t-компонентов или виртуальных переходов помечены соответствующими метками из Λ. При этом все внешние действия, как было указано выше, помечены специальным символом невидимого действия τ.

Пусть NPN есть NP-сеть с переходами, помеченными символами из Λ, как описано выше, и пусть ФМ СРК SDC получена из NPN в результате описанной выше трансляции. Через $\mathfrak{M}(NPN)$ и $\mathfrak{M}(SDC)$ обозначим множества всех достижимых разметок NPN и всех достижимых конфигураций SDC соответственно.

Отношение $\mathcal{R}_\mathcal{W} \subseteq \mathfrak{M}(NPN) \times \mathfrak{M}(SDC)$ определим как отношение, удовлетворяющее следующим условиям:

1. $(M_0, C_0) \in \mathcal{R}_\mathcal{W}$, где M_0 — начальная разметка сети NPN, а C_0 — начальная конфигурация SDC;

2. $(M, C) \in \mathcal{R}_\mathcal{W}$, если и только если для каждой позиции $p \in SN$

   - для каждой атомарной фишки $a \in M(p)$ p-компонент $c_p = tr_P(p)$ в $c_p : : Atoms$ содержит данные, соответствующие значению a;
   - для каждой сетевой фишки $\alpha \in M(p)$ p-компонент $c_p = tr_P(p)$ содержит ссылку на α-компонент $tr_\alpha(\alpha)$ в контейнере $c_p : : NetTokens$;

3. Если $(M, C) \in \mathcal{R}_\mathcal{W}$ и C' достижима из C в результате срабатывания нескольких ненаблюдаемых шагов, то $(M, C') \in \mathcal{R}_\mathcal{W}$.

Теорема 1. *Для NP-сети NPN и ФМ СРК SDC, полученной из NPN в результате описанной выше трансляции, отношение* $\mathcal{R}_{\mathcal{W}} \subseteq \subseteq \mathfrak{M}(NPN) \times \mathfrak{M}(SDC)$ *является предпорядком слабой симуляции, т.е. если для состояния* M *в* $\mathfrak{M}(NPN)$ *и конфигурации* C *в* $\mathfrak{M}(SDC)$ *выполняется* $(M, C) \in R_W$ *и* $M \xrightarrow{t} M'$, *то существует конфигурация* C' *в SDC такая, что* $C \overset{t}{\Rightarrow} C'$ *и* $(M', C') \in R_W$, *где* $C \overset{t}{\Rightarrow} C'$ *означает, что конфигурация* C' *достижима из* C *путем исполнения действия с пометкой* t *и некоторого числа невидимых действий.*

Мы не будем приводить полное доказательство этой теоремы в силу его громоздкости. Опишем содержательно идею доказательства. Нужно показать, что если $(M, C) \in \mathcal{R}_{\mathcal{W}}$, то любой шаг $M \xrightarrow{t} M'$ в NPN может быть просимулирован действием с той же самой видимой меткой в конфигурации C и, возможно, несколькими невидимыми действиями τ в SDC.

Рассмотрим интерпретацию шага $M \xrightarrow{t} M'$ сети NPN в SDC. Исполнение t-компонента c_t состоит из трех этапов. Первый этап — это захват ресурсов из входных p-компонент, второй — выполнение вспомогательных действий, третий — добавление ресурсов в выходные p-компоненты. По определению трансляции только одно действие в трехэтапном исполнении t-компонента c_t помечено меткой перехода t. Все остальные являются ненаблюдаемыми. Нетрудно проверить, что если исполнение t-компонента c_t переводит конфигурацию C в конфигурацию C' в SDC, то $(M', C') \in \mathcal{R}_{\mathcal{W}}$, т.е. исполнение t-компонента c_t слабо симулирует срабатывание перехода t в NPN.

В работе [37] было показано, что отношение слабой симуляции сохраняет живость и условную живость при условии завершимости симуляции каждого перехода. Завершимость исполнения t-компонентов обеспечивается предположением о конечной завершимости внешних действий и стратегией разрешения конфликтов. Таким образом, описанная трансляция NP-сетей в ФМ СРК сохраняет свойства живости и условной живости исходной модели.

Сохранение свойств безопасности стабильных состояний системы распределенных компонентов обеспечивается взаимно-однозначным соответствием между состояниями NP-сети и стабильными состояниями соответствующей SDC. Стабильные состояния SDC можно рассматривать как коды состояний NP-сети.

И, наконец, свойство справедливости исполнения ФМ СРК обеспечивается справедливостью стратегии **ETO**.

7. РЕАЛИЗАЦИЯ

Описанная выше трансляция была реализована в инструменте EJBgenerator [41], который получает на вход модель NP-сети в формате npnets(ecore) [42].

Представленная в данной работе трансляция задает способ построения ФМ СРК по NP-сети, но не указывает способ привязки срабатываний переходов к полезным действиям результирующей системы. Выбор способа привязки осуществлятся в зависимости от прикладной области целевой системы. В web-системах необходима привязка срабатываний переходов к вызовам web-сервисов, http-запросам и запросам к базам данных. В корпоративных системах срабатывания переходов привязываются к запросам API внутренних систем журналирования, управления бизнес процессами и поддержки совместной работы.

В инструменте EJBgenerator привязка полезных действий к срабатываниям переходов реализована с помощью внешних спецификаций, которые являются простыми текстовыми файлами, содержащими три секции — пакеты, инициализация, действие. Первая секция содержит список подключаемых Java пакетов, необходимых для исполнения кода перехода. Вторая секция содержит Java код, выполняемый при инициализации компонента перехода. Третья секция содержит Java код, реализующий полезные действия, который будет выполняться при каждом срабатывании перехода в контексте компонента перехода.

Так как Java код t-компонентов выполняется в контексте EJB окружения, ему доступны все сервисы систем EJB, J2SE и J2EE. Данный способ привязки позволяет разделить разработку модели системы и ее интеграцию в целевой контекст. Это облегчает переиспользование системы в других окружениях, интеграцию системы при изменении интерфейсов внешних сервисов, моди-

фикацию самой системы без изменения внешних интерфейсов. Результатом работы транслятора EJBgenerator является набор EJB компонентов, инструментированный кодом из спецификаций, готовых для развертывания в EJB контейнере.

Реализация была опробована с помощью инструмента EJBgentrator на примере распределенной системы управления складом (см. рис. 6а)), подробно описанной в [8]. На рис. 6б) и 6в) изображен набор EJB компонентов, полученный с помощью инструмента EJBgenerator, и результат запуска системы в EJB контейнере Glassfish 4.0.

Размер генерируемой системы распределенных компонентов может быть существенно больше размера модели из-за большого числа виртуальных переходов. Следующая формула дает оценку числа виртуальных переходов, создаваемых в компоненте-мониторе для перехода t:

$$\prod_{p \in {}^\bullet t} \binom{|m(p)|}{k} |T_{E_p}|_\lambda,$$

где позиция p типизирована типом E_p, а k — кратность дуги $(p,t) \in F_{SN}$, связывающей переход t с входной позицией p. Здесь $|T_{E_p}|_\lambda$ — число переходов с меткой λ в элементной сети E_p, $\binom{|m(p)|}{k}$ соответствует возможным уникальным означиваниям входных переменных без учета имени переменных.

С практической стороны можно заметить, что в реальных моделях систем кратность дуг k обычно бывает небольшой, и размер $|T_{E_p}|_\lambda$ множества переходов, использующих одну метку синхронизации, также невелик.

Число порождаемых виртуальных переходов можно уменьшить, если в исходной NP-сети не переиспользовать одну метку синхронизации для синхронизации разных переходов.

В отдельных случаях использование одной метки для синхронизации разных пар переходов не приводит к нарушению функционирования системы. Однако такое использование метки синхронизации приведет к увеличению числа комбинаций переходов синхронизации, которые никогда не срабатывают. Автоматическое обнаружение таких разделяемых множеств синхронизации основывается на анализе поведенческих свойств системы, и, как правило, имеет экспоненциальную сложность. Поэтому для уменьшения числа порождаемых виртуальных переходов рекомендуется вручную ввести различные метки для различных синхронизаций.

8. ЗАКЛЮЧЕНИЕ

В данной работе представлен и обоснован алгоритм трансляции вложенных сетей Петри в системы распределенных компонентов, лежащий в основе автоматической генерации кода. Описан подход к разрешению конфликтов между переходами, в которых задействованы несколько компонентов системы. Обоснована возможность применения трансляции для синтеза корректных систем распределенных компонентов по корректным моделям.

Предложенная трансляция может быть расширена на многоуровневые вложенные сети Петри. Для этого необходимо выбрать схему отображения элементов NP-сети на физические элементы СРК и расширить механизм виртуальных переходов на все уровни NP-сети, в которых присутствуют сетевые фишки. Так как в многоуровневых сетях шаги синхронизации задействуют только смежные уровни, оценка числа порождаемых виртуальных переходов останется та же. Выбор схемы отображения элементов NP-сети не накладывает дополнительных ограничений на трансляцию и должен основываться лишь на специфике прикладной области и требованиях к уровню производительности, гибкости и масштабируемости системы.

Разрешение конфликтов переходов синхронизации ведет к экспоненциальному росту числа виртуальных переходов. Это обусловлено тем, что шаги вертикальной синхронизации могут задействовать одновременно несколько компонентов вложенной сети Петри, и при этом компоненты могут появляться и исчезать в ходе исполнения шага синхронизации. При использовании стратегий **PPA** и **PPW** [39] нет необходимости порождать виртуальные переходы, но стратегии **PPA** и **PPW** не обеспечивают сохранение свойств живости и условной живости в синтезируемой системе.

В общем случае, даже если сетевые фишки во входной позиции обладают одинаковым функционалом, число порождаемых виртуальных переходов не может быть уменьшено путем созда-

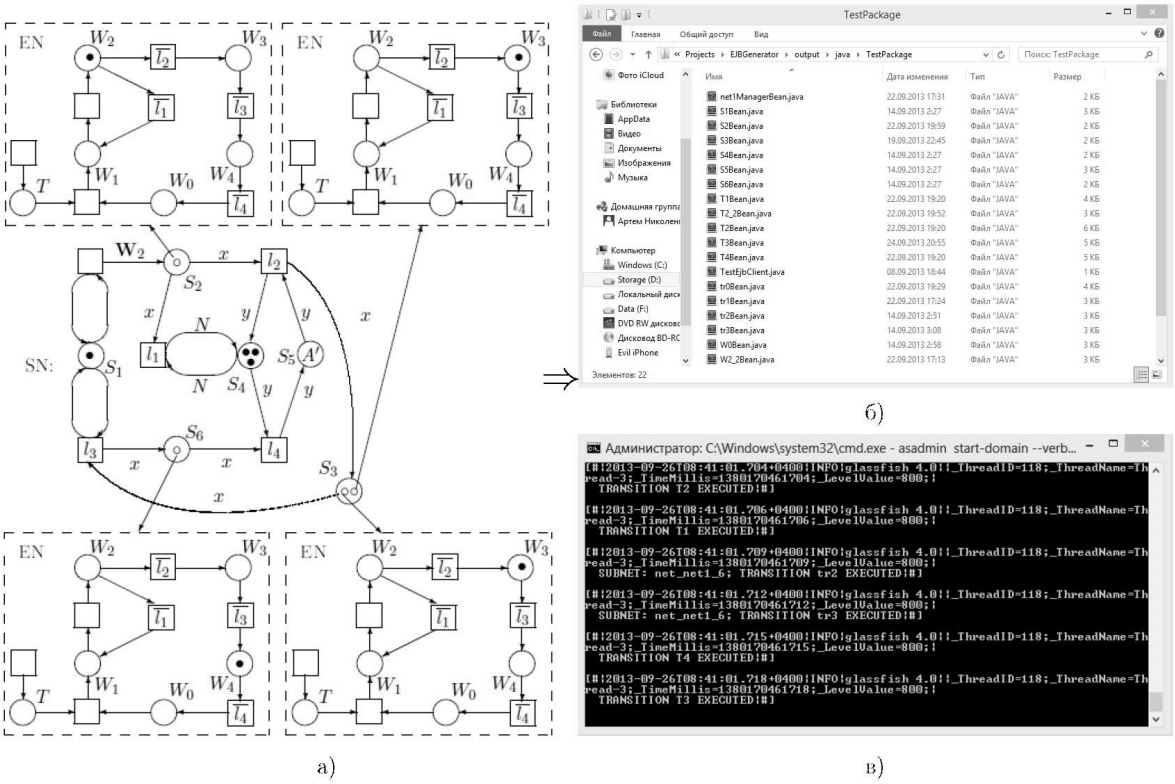


Рис. 6.
Пример использования инструмента EJBgenerator для генерации компонентов EJB по NP-сети.

ния одного перехода для класса всех сетевых фишек одного типа, так как в каждой фишке активность перехода синхронизации определяется ее индивидуальным внутренним состоянием. Однако интересно исследовать возможность уменьшить число виртуальных переходов с помощью выявления “мертвых” комбинаций переходов синхронизации или путем редукции числа виртуальных переходов для подклассов вложенных сетей Петри, в которых можно проверять эквивалентность разметок сетевых фишек.

СПИСОК ЛИТЕРАТУРЫ

1. Bitcoin charts. Financial and technical data related to the Bitcoin network. http://bitcoincharts.com
2. Top500. List of the 500 most powerful commercially available computer systems. http://www.top500.org/
3. *Budinská I., Kasanický T., Zelenka J.* Distributed Multi-agent System for Area Coverage Tasks: Architecture and Development. In: Emergent Trends in Robotics and Intelligent Systems. Advances in Intelligent Systems and Computing, 2015, vol. 316, pp. 237–245. 2015.
4. *Domnori E., Cabri G., Leonardi L.* Multi-agent approach for disaster management. In: Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011, pp. 311–316.
5. *Bartels R., Herskovic V., Monares A., Ochoa S.F., Pino J.A., Borges M.R.S.* A Simple and Portable Command Post to Coordinate Search and Rescue Activities in Disaster Relief Efforts. In: Collaboration and Technology. Lecture Notes in Computer Science, Springer, 2010, vol. 6257, pp. 337–344.
6. *Attie P., Baranov E., Bliudze S., Jaber M., Sifakis J.* A General Framework for Architecture Composability. In: Software Engineering and Formal Methods. Lecture Notes in Computer Science, Springer, 2014, vol. 8702, pp. 128–143.
7. *Reisig W.* Understanding Petri Nets. Springer. 2013.
8. *Ломазова И.А.* Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой. Научный мир, 2004. 208 с.

9. *Lomazova I.A.* Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems. Fundamenta Informaticae, 2000, vol. 43, no. 1–4, pp. 195–214.

10. *van Hee K., Lomazova I.A., Oanea O., Serebrenik A., Sidorova N., Voorhoeve M.* Nested Nets for Adaptive Systems. In: 27th Int. Conf. on Application and Theory of Petri Nets and Other Models of Concurrency. Lecture Notes in Computer Science, Springer, 2006. vol. 4024, pp. 241–260.

11. *van Hee K., Oanea O., Serebrenik A., Sidorova N., Voorhoeve M., Lomazova I.A.* Checking Properties of Adaptive Workflow Nets. Fundamenta Informaticae, 2007, vol. 79, no. 3, pp. 347–362.

12. *Lomazova I.A.* Modeling dynamic objects in distributed systems with nested Petri nets. Fundamenta Informaticae, 2002, vol. 51, no. 1–2, pp. 121–133.

13. *Lomazova I.A.* Nested Petri nets for adaptive process modeling. In: Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday. Lecture Notes in Computer Science, Springer, 2008, vol. 4800, pp. 413–426.

14. *Tarabuta O.* Use of "Petri nets system" concept in modeling dynamics with increased complexity. In: 15th Int. IEEE Conf. on System Theory, Control, and Computing, 2011, pp. 1–6.

15. *López-Mellado E., Almeyda-Canepa H.* A three-level net formalism for the modeling of multiple mobile robot systems. International Journal of Computer Integrated Manufacturing, 2005, vol. 18, no. 2–3, pp. 137–146.

16. *Chang L., He X., Lian J., Shatz S.* Applying a Nested Petri Net Modeling Paradigm to Coordination of Sensor Networks with Mobile Agents. In: Proc. of Workshop on Petri Nets and Distributed Systems 2008, Xian (2008), pp. 132–145.

17. *Cristini F., Tessier C.* Nets-within-Nets to model innovative space system architectures. In: Application and Theory of Petri Nets. Lecture Notes in Computer Science, Springer, 2012, vol. 7347, pp. 348–367.

18. *López-Mellado E., Villanueva-Paredes N., Almeyda-Canepa H.* Modelling of batch production systems using Petri nets with dynamic tokens. Mathematics and Computers in Simulation, 2005, vol. 67. no. 6, pp. 541–558.

19. *Kahloul L., Djouani K., Chaoui A.* Formal Study of Reconfigurable Manufacturing Systems: A High Level Petri Nets Based Approach In: Industrial Applications of Holonic and Multi-Agent Systems. Lecture Notes in Computer Science, Springer, 2013, vol. 8062, pp. 106–117.

20. *Zhang L., Rodrigues B.* Nested coloured timed Petri nets for production configuration of product families. International Journal of Production Research, 2010, vol. 48, no. 6, pp. 1805–1833.

21. *Dworzanski L.W., Lomazova I.A.* On Compositionality of Boundedness and Liveness for Nested Petri Nets. Fundamenta Informaticae, 2012, vol. 120, no. 3–4, pp. 243–257.

22. *Frumin D. Lomazova I.* Branching Processes of Conservative Nested Petri Nets. In: VPT 2014. 2nd Int. Workshop on Verification and Program Transformation Vol. 28: EPiC Series. EasyChair, 2014, pp. 19–35.

23. *Dworzański L.W., Lomazova I.A.* CPN tools-assisted simulation and verification of nested Petri nets. Automatic Control and Computer Sciences, 47(7), Springer, 2013, pp. 393–402.

24. *Venero M.L.F.* Verifying Cross-Organizational Workflows Over Multi-Agent Based Environments, In: Enterprise and Organizational Modeling and Simulation, Lecture Notes in Business Information Processing, Springer, 2014, vol. 191, pp. 38–58.

25. *Raje R.R., Zhu D., Mukhopadhyay S., Tang L., Palakal M., Mostafa J.* COBioSIFTER &Ndash; A CORBA-Based Distributed Multi-Agent Biological Information Management System. Cluster Computing, 2004, vol. 7, no. 4, pp. 373–389.

26. *Peng M., He Y., Liu J.* The application of multi-agent and CORBA in mobile-GPS system. In: Int. Conf. on Computer and Information Technology, IEEE Computer Society, 2004, pp. 503–508.

27. *Cheng T., Guan Z., Liu L., Wu B., Yang S.* A CORBA-based multi-agent system integration framework. In: 9th IEEE Int. Conf. on Engineering Complex Computer Systems, 2004. Proceedings, pp. 191–198.

28. *Ivanovic M., Vidakovic M., Mitrovic D., Budimac Z.* Evolution of Extensible Java EE-Based Agent Framework. In: Agent and Multi-Agent Systems. Technologies and Applications. Lecture Notes in Computer Science, Springer, 2012, vol. 7327, pp. 444–453.

29. *Chachkov S., Buchs D.* From Formal Specifications to Ready-to-Use Software Components: The Concurrent Object Oriented Petri Net Approach. In: 2nd Int. Conf. on Application of Concurrency to System Design (ACSD 2001), 2001, pp. 99.

30. *Kristensen L.M., Westergaard M.* Automatic Structure-Based Code Generation from Coloured Petri Nets: A Proof of Concept. In: Formal Methods for Industrial Critical Systems. Lecture Notes in Computer Science, Springer, 2010, vol. 6371, pp. 215–230.

31. *Jensen K., Kristensen L.M.* Coloured Petri Nets — Modelling and Validation of Concurrent Systems. Springer, 2009.

32. *Bourdonov I.B., Kossatchev A.S., Kuliamin V.V.* Formalization of test experiments. Programming and Computer Software, 2007, vol. 33, pp. 239–260.

33. *Bourdonov I.B., Kossatchev, A.S.* Formalization of a test experiment-II. Programming and Computer Software, 2013, vol. 39, no. 4, pp. 163–181.

34. *van Glabbeek R.J.* The linear time — branching time spectrum. In: CONCUR '90 Theories of Concurrency: Unification and Extension. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1990, vol. 458, pp. 278–297.

35. *van Glabbeek R.J.* The linear time — Branching time spectrum II. In: CONCUR'93. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1993, vol. 715, pp. 66–81.

36. *Levy P.B.* Infinite trace equivalence. Annals of Pure and Applied Logic, 2008, vol. 151, no. 2–3, pp. 170–198.

37. *van Glabbeek R.J., Voorhoeve M.* Liveness, Fairness and Impossible Futures. In: CONCUR 2006 — Concurrency Theory. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, vol. 4137, pp. 126–141.

38. *Fehnker A., van Glabbeek R.J., Höfner P., McIver A., Portmann M., Tan W.L.* A Process Algebra for Wireless Mesh Networks used for Modelling, Verifying and Analysing AODV, arXiv preprint arXiv:1312.7645, 2013.

39. *Taubner D.* On the implementation of Petri nets. In: Advances in Petri Nets 1988. Lecture Notes in Computer Science, Springer, 1988, vol. 340, pp. 418–439.

40. *Philippi S.* Automatic code generation from high-level Petri-Nets for model driven systems engineering . Journal of Systems and Software, 2006, vol. 79, no. 10, pp. 1444 –1455.

41. EJB Generator: NP-nets to EJB distributed components system. https://github.com/Evil-Crab/EJBGen

42. *Dworzanski, L., Frumin D.* NPNtool: Modelling and Analysis Toolset for Nested Petri Nets. In: Proc. of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering, 2013, pp. 9–14.

43. *Venero M.L.F., da Silva F.S.C.* On the Use of SPIN for Studying the Behavior of Nested Petri Nets. In: Formal Methods: Foundations and Applications. Lecture Notes in Computer Science, Springer, 2013, vol. 8195, pp. 83–98.