Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Новосибирский государственный технический университет»

На правах рукописи

Марков Александр Владимирович

АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ И АНАЛИЗА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА UML И СЕТЕЙ ПЕТРИ

05.13.11 - Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание
учёной степени кандидата технических наук

Научный руководитель:
доктор технических наук,
профессор Воевода А.А.

Новосибирск – 2015

ВВЕДЕНИЕ

Актуальность темы. В настоящее время быстрая, экономичная, успешная работа различных предприятий зависит от качественных программных приложений, которые используют в экономической, управленческой и технической деятельности. Для создания качественного продукта пользуются различными техниками разработки ***программного обеспечения*** (ПО) от водопадной и итерационной моделей до других различных вариантов и их модификаций [2-6, 36]. В настоящее время применяются различные способы разработки ПО, но сложность применения затрудняет их использование в реальной жизни. Зачастую при проектировании ПО прибегают к CASE-технологиям, которые описаны F.P. Broooks, А.Л. Фуксманом, А.Н. Тереховым, А.М. Вендеровым, В.В. Липаевым и др. Одним из самых популярных средств визуального моделирования является унифицированный язык моделирования *UML* (Unified Modeling Language), предложенный группой разработчиков в *OMG* (Object Management Group) и представленный в работах G. Booch, J. Rumbaugh, I.H. Jacobson [95] и описанный в работах M. Fowler, K. Scott, И. Грэхэма, Д. Харела. Но, как известно, в данной структуре отсутствует формальная проверка созданных систем и отдельных диаграмм. Использование сетей Петри – математического аппарата, подробно описанного в трудах W. Reisig, M.H.T. Hack [111], James L. Peterson [87], S. Haddad [113], G.W. Brams, В.Е. Котова [54] и А.А. Лескина [56], И.А. Ломазовой [57], О.Л. Бандман [1], И.Б. Вербицкайте, В.А. Непомнящего, помогает решить проблему анализа многопоточных систем, параллельных вычислений (А.П. Ершов, Ч. Хоар, В.В. Корнеев, В.И. Воробьев, В.Э. Малышкин [58]) и проектируемых диаграмм (M, Westergaard [147], L. Baresi [100], L.Z. Zhu [150]).

При анализе сетей Петри разработчик зачастую сталкивается с трудностью, заключающейся в неограниченном росте количества состояний проектируемой системы. Несмотря на простоту структуры некоторых сетей Петри, их *пространство состояний*[1] может достигать значительных размеров, что приводит к одной из главных проблем при анализе автоматов и графов – *“взрыву” пространства*

[1] *Пространство состояний* – неупорядоченное множество всех состояний системы, включающее взаимосвязи между состояниями и информацию о них.

состояний[2]. Работы G.J. Holzmann [114, 116], S. Cristensen [107], L.M. Kristensen [122] посвящены решению данной проблемы. Несмотря на увеличение мощности ЭВМ, используемых при анализе, возрастает сложность структур и логики систем, поэтому решение данной проблемы не может сводиться только к увеличению мощности анализаторов. Стоит отметить, что при совместном использовании UML диаграмм и сетей Петри появляются трудности: отсутствии формализации правил преобразования и, следовательно, автоматической трансляции.

На кафедре автоматики с 2002 года ведётся работа по UML и сетям Петри, результатом которой стала методика совместного использования UML диаграмм, описывающих статические и динамические свойства, и сетей Петри, используемых для анализа полученных диаграмм. Разработанная методика применима к задачам проектирования программных приложений для персональных компьютеров, для автоматизированных систем, для системы управляемого светофора, взаимодействия пользователя с банкоматом и др. Разработаны алгоритмы и программные продукты, нацеленные на сокращение времени проектирования ПО за счет автоматизации проектирования и анализа – выявления логических ошибок (С.В. Коротиков [48-50, 52, 53], Д.О. Романников [89, 91]). Например, алгоритм автоматической трансляции диаграммы активности и диаграммы последовательности в сети Петри, приложение по преобразованию графического вида к матричной форме с последующим анализом.

Цель диссертационного исследования. Разработать методику проектирования программного обеспечения с использованием UML диаграмм и сетей Петри, позволяющую находить и устранять логические ошибки (зацикливание, тупиковые маркировки, мертвые переходы), в которой в отличие от более ранних версий не используется детализированная диаграмма прецедентов и диаграмма состояний, а для анализа отдельных сценариев используется диаграмма последовательности. Разрабатываемая методика должна быть применима для разработки систем широкого круга задач: производственные процессы, приложения для персональных компьютеров, систем транспортного регулирования, систем обработки информации и распределенных систем.

---

[2] *“Взрыв” пространства состояний* (англ, *state explosion problem*) – экспоненциальный рост количества исследуемых состояний, останавливающий процесс анализа по причине отсутствия требуемого количества памяти.

Для достижения поставленной цели решаются следующие **задачи**:

– составление пошаговой процедуры проектирования ПО с использованием UML диаграмм и сетей Петри для анализа поведенческих UML диаграмм;

– разработка способа проверки достижимости заданного состояния сети, представление правил и алгоритма для его реализации;

– разработка алгоритма преобразования UML диаграмм в сети Петри и способа выполнения автоматической трансляции UML диаграмм в сети Петри с использованием форматов .xmi и .cpn;

– разработка программного обеспечения для преобразования комбинации мест и переходов маркированных сетей Петри к матричной форме: составление матриц входной и выходной функций, получение составной матрицы и вектора начальной маркировки;

– разработка способа проектирования сетей Петри для задачи создания ПО в сфере робототехники, а именно робота-манипулятора; реализация рекурсивных функций в сетях Петри;

– разработка способов анализа сетей Петри, заключающихся в проверке частей пространства состояний.

Объект исследования. Средства проектирования и анализа программного обеспечения с целью их автоматизации. **Предметом исследования** является совместное использование унифицированного языка моделирования UML и сетей Петри при проектировании и анализе ПО.

Методы исследования. При выполнении задач диссертационного исследования применялись следующие методы: объектно-ориентированный анализ; математический аппарат сетей Петри; метод “плавающей” линии (*sweep-line method*); хеширование (*bit state hashing*).

При реализации предлагаемой методики использовались отраслевые и международные стандарты, CASE-технологии, современные инструментальные среды и пакеты моделирования: Rational Rose (среда проектирования UML диаграмм, способная преобразовывать проектируемые диаграммы в программный код основных языков программирования), Magic Draw (среда проектирования UML диаграмм с возможностью сохранять полученные диаграммы с расширени-

ем *.xmi*), CPN Tools (среда проектирования цветных сетей Петри – Colour Petri Nets Tools).

Научная новизна. В диссертационном исследовании предлагаются: формализация правил реализации автоматической трансляции UML диаграмм в сети Петри; новый способ проверки достижимости сетей Петри через реализацию инверсии самой сети и/или её графа состояний; разработка нового способа задания исходных данных о структуре проектируемой системы с использованием информации, закладываемой в *метке* (*маркере*)[3] сети Петри; проектирование регулирующих элементов на примере поддержания давления и температуры автоматизированных систем с использованием сетей Петри.

Личный вклад. Все основные результаты получены автором. А именно, разработана методика совместного использования UML диаграмм и сетей Петри, предложены правила инверсии сетей Петри, разработано приложение по преобразованию и анализу сетей в матричной форме. В совместной работе с Зимаевым И.В. получена взаимосвязь между статическими и динамическими диаграммами. При участии Романникова Д.О. разработаны алгоритмы для моделирования системы поддержания давления в трубопроводе. Также в совместных работах соавторам принадлежат постановка задачи, предварительное моделирование и обработка частных результатов.

Практическая ценность и внедрение. Полученные результаты подтвердили эффективность применения предлагаемой методики на этапе проектирования и отладки программного обеспечения.

Результаты диссертационной работы были использованы в проектировании системы подготовки железорудных окатышей на горно-обогатительном комбинате Акционерного Общества Соколовско-Сарбайского Горно-обогатительного Объединения (АО ССГПО, Республика Казахстан, г. Рудный), в проектировании ПО локальных подсистем АСУ ТП водоснабжения (г. Тюмень), в разработке интернет сайтов в ООО «Дабаз» (г. Новосибирск), Результаты диссертационного исследования используются в учебном процессе НГТУ на кафедре “Автоматика”.

[3] Метками представляют информацию, передаваемую в системе и/или системой.

Полученные результаты рекомендуется использовать при проектировании программных приложений для компьютеров в производственных сферах, для систем обработки информации и для распределенных систем, например, для поддержания температуры при обжиге окатышей, для поддержания регулируемых величин системы водоснабжения, для управления светофором, для работы банкомата, для передвижения манипулятора в ограниченном пространстве с препятствиями, а также для других программных приложений.

Кроме того, исследования были поддержаны внутренним грантом НГТУ фундаментальных и прикладных исследований «Использование UML-диаграмм и аппарата сетей Петри как формальных методик анализа архитектуры программного обеспечения», грантом на выполнение проекта, отобранного для финансирования в 2012 году в рамках реализуемой программы стратегического развития НГТУ по итогам конкурса НИОКР, определяющих формирование научно-технического задела по приоритетным направлениям развития науки. Направление 2.3.: "Информационные и цифровые технологии и системы". Работа выполнена при финансовой поддержке Минобрнауки России по государственному заданию №2014/138 тема проекта "Новые структуры, модели и алгоритмы для прорывных методов управления техническими системами на основе наукоемких результатов интеллектуальной деятельности".

На защиту выносятся следующие основные результаты и положения:

– методика проектирования ПО, включающая построение структурных UML диаграмм (прецедентов, классов и объектов) и диаграмм поведения (активности и последовательности, транслируемых в сети Петри для анализа и устранения логических ошибок (зацикливания, тупиковые маркировки, мертвые переходы);

– формализованные правила, позволяющие автоматизировать трансляцию UML диаграмм в цветные сети Петри;

– рекомендации по анализу систем с большим количеством состояний, заключающиеся в исследовании отдельных сценариев функционирования проектируемого ПО и частей пространства состояний;

– способ инверсии сетей Петри для проверки их основных свойств, а также анализ отдельных состояний и частей графа состояний;

– программа для представления сетей Петри в матричной форме: составление матриц входной и выходной функций, получение составной матрицы и вектора начальной маркировки;
– способ моделирования систем со сложной структурой при использовании сетей Петри с нагруженными метками, реализация рекурсии в сетях Петри.

Апробация работы. Основные результаты диссертационной работы были представлены на следующих конференциях и семинарах: международная русско-индийская конференция “The 2nd Russian-Indian Joint Workshop on Computational Intelligence and Modern Heurisics in Automation and Robotics” (Новосибирск, 10-13 сентября 2011 г.); одиннадцатая международная научно-техническая конференция “Актуальные проблемы электронного приборостроения” (Новосибирск, 2-4 октября 2012 г.); двенадцатая международная научно-техническая конференция “Актуальные проблемы электронного приборостроения” (Новосибирск, 2-4 октября 2014 г.); III международная конференция “Современные информационные технологии и ИТ-образование” (факультет вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 8-10 ноября 2013 г.); международная научная конференция “Математическое и компьютерное моделирование” (Омский государственный университет, 18-19 октября 2013 г.); 61-я международная молодёжная научно-техническая конференция “Молодёжь. Наука. Инновации” (Владивосток, МГУ им. адм. Г.И. Невельского, 20-21 ноября 2013 г.); школа молодых учёных САИТ-2011. Секция №2 “Информационные технологии в системах автоматического и автоматизированного управления” (Новосибирск, 12-16 сентября 2011г.); XIII Международная конференция “Информатика: проблемы, методология, технологии” (Воронеж, 7-8 февраля 2013 г.); международная научно-практическая конференция “Тенденции формирования науки нового времени” (Уфа, 18 октября 2014 г.); XIII международная научно-практическая конференция “Наука и современность” (Новосибирск, 15 ноября 2011 г.), международная заочная научно-практическая конференция “Наука и техника XXI века” (Новосибирск, 14 ноября 2011 г.); результаты диссертационной работы регулярно представлялись на конференциях и семинарах НГТУ (2011-2014 гг.).

Соответствие работы научной специальности. Диссертация соответствует п. 1 «Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирова-

ния», п. 8 «Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования», п. 7 «Модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем» паспорта специальности 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей».

Публикации. Основные положения и результаты диссертационной работы опубликованы в 36 работах, в том числе: 3 программы, зарегистрированные в Роспатент, стати в изданиях, рекомендованных ВАК РФ – 6, в сборниках научных трудов – 19, в материалах международных симпозиумов и конференций – 10.

Структура и объём диссертации. Диссертация состоит из введения, четырёх глав, заключения, библиографического списка использованной литературы и шести приложений. Работа изложена на 176 стр. машинописного текста: основное содержание на 138 стр. и включает 85 рисунков, 4 таблицы и список литературы из 150 наименований.

Во **введении** обоснована актуальность темы диссертационного исследования. Определены цели и задачи, сформулированы положения диссертационного исследования, которые вынесены на защиту, их научная новизна и практическая ценность.

В **первой главе** диссертации анализируются существующие способы и подходы к разработке ПО, принятые в индустрии программирования, различные парадигмы написания приложений, а также методы анализа проектируемого продукта как с точки зрения написания кода, так и управления процессом [2-6]. Из всего многообразия остановимся на моделирование систем с использованием UML диаграмм [15, 35, 55, 95, 100] и сетей Петри [36, 37, 54, 59, 87].

UML – инструмент моделирования, позволяющий преобразовывать сконструированные диаграммы в программные коды. Сети Петри разрабатывались для моделирования систем и содержат формальные правила, по которым исследуются их свойства. Поскольку унифицированный язык моделирования не обладает формальными методами проверки проектируемых диаграмм, поэтому применение связки UML диаграмм и сетей Петри выглядит весьма перспективно. Для их совместного использования обращают внимание на правила и методики, нацеленные на проектирование и проверку корректности построения различных систем [53, 93]. На этапе

анализа систем при помощи сетей Петри проектировщик может столкнуться с трудностями, связанными с большим количеством состояний, в которых может оказаться система, что может привести к “взрыву” пространства состояний. Чтобы избежать данных затруднений, используют методики, которые заключаются в удалении неиспользуемых состояний, а также сжатии информации о каждом состоянии системы.

Во **второй главе** предлагается методика проектирования ПО с использованием UML диаграмм и сетей Петри [34, 53, 93, 94, 100]. Поскольку в известных работах [53, 91] отсутствует формализация правил преобразования UML диаграмм в сети Петри для реализации автоматической трансляции, используется структура подобных форматов *.xmi* (UML диаграммы) и *.cpn* (сети Петри), что позволяет сократить время на ручное преобразование диаграмм, а также формализует правила по их преобразованию [40, 80, 150]. На данном примере проектирования систем со сложной структурой показаны параллелизм и гиперпоточность сетей Петри. Моделируемый пример представляет ограниченное пространство с препятствиями, по которому перемещается манипулятор. Преимуществом данного способа является хранение информации о замкнутом пространстве и манипуляторе в метках сети [60, 62, 63, 66, 79]. При создании программного обеспечения используются рекурсивные функции. На примере решения задачи переноса упорядоченного массива с ограничением выборки по одному элементу, нахождения значения числа Фибоначчи и нахождения факториала числа показана рекурсия в сетях Петри с использованием нагруженных меток [31, 73] в отличие от [57, 113], где рекурсия реализуется вложенными сетями Петри при вертикальной синхронизации системной сети и элементных. Предлагается один из способов исследования сетей Петри посредством отображения свободного языка в сжатой форме, что позволяет отследить последовательность срабатывания переходов, на примерах управляемого светофора и двухсимочного телефона [9, 14, 25].

В третьей главе рассматривается способ анализа сетей Петри, основанный на выборе отдельных сценариев системы с последующим переходом в сеть Петри и анализом полученной сети. Моделирование сценариев осуществляется на основе UML диаграмм последовательности. Приводится способ анализа сетей Петри, который заключается в разбиении основного алгоритма системы на несколько алгоритмов, представленных в виде уровней иерархии. После чего полученная система преобразуется в иерархическую сеть. Если сеть Петри была спроектирована ранее, то

структуру данной сети разбивают на уровни иерархии. После разбиения сети отдельные подсети становятся отдельными сетями и анализируются отдельно от основной сети. Описывается способ, заключающийся в анализе отдельных фрагментов графа состояний системы. Проведенные исследования показали, что при анализе отдельных частей графа состояний все присущие свойства сохраняются, но необходима проверка достижимости выбранной для начала анализа маркировки [70, 74, 76]. Описывается анализ сети Петри с использованием матриц на примере системы “Интернет-магазин”. Предлагается приложение, преобразующее построенные сети Петри в пакете CPN Tools (version 3.4.0) в матричную форму [32, 69, 71, 75]. Предлагается инверсия, опираясь на которую проверяется достижимость выбранной маркировки, приводятся правила для инвертирования простых и ординарных сетей. Демонстрируется пример реализации инверсии на примере протокола передачи данных и предлагается реализация инверсии графа состояний, большим преимуществом которой является однозначность трансформации [77, 78, 81].

В **главе четыре** описывается процесс подготовки железорудных окатышей на горно-обогатительном комбинате по разработанной методике, включающей в себя моделирование на основе UML диаграмм прецедентов, классов, объектов, активности и её проверки через анализ пространства состояний соответствующей сети Петри [21, 48, 53, 93]. Моделируется регулятор поддержания температуры с использованием сетей Петри, который состоит из семи переходов, отвечающих за изменения температуры в определенном диапазон. Также методика применяется для проектирования ПО для АСУ ТП водонапорной станции [33]. Диаграмма активности, отвечающая за динамику работы системы, транслируется в соответствующую сеть Петри, в которой с использованием дискретных величин регулируется давление в трубопроводе. Приводится матричное представление сетей Петри на примере систем: управляемый светофор, двухсимочный телефон и взаимодействие пользователя с банкоматом [32, 64, 68, 69, 75].

В **заключении** перечислены основные результаты исследований. Разработана методика проектирования ПО (архитектуры и исполняемого поведения). Разработан алгоритм и правила реализации инверсии в сетях Петри для проверки достижимости выбранного состояния сети. Предложены алгоритмы для преобразования UML диаграмм в сети Петри (действие, выполнение условия, разделе-

ние/слияние), а также способ выполнения автоматической трансляции UML диаграммы активности в сети Петри. Разработано программное обеспечение для преобразования комбинации мест и переходов маркированных сетей Петри к матричной форме, предложенного Дж. Питерсоном. Разработан способ проектирования сетей Петри, при котором задание исходных данных о структуре проектируемой системы производится без использования мест и переходов, а при помощи информации, хранящейся в метках. Представлена реализация рекурсивных функций в сетях Петри. Разработаны способы анализа сетей Петри: анализ пространства состояний посредством моделирования сценариев работы системы, представление сетей Петри в иерархическом виде с целью анализа отдельных подсетей, анализ отдельно взятых частей пространства состояний при условии выполнения достижимости их начальной маркировки и отсутствия возвратных рёбер. Предложено представление работы регуляторов при использовании дискретных и дискретизированных аналоговых величин на основе сетей Петри.

В **приложениях** приведены акты внедрения диссертационного исследования (приложение А) и свидетельства о регистрации программ (приложение Б). Разрабатывается алгоритм автоматической трансляции UML диаграмм в сети Петри посредством схожих форматов *“.xmi”* и *“.cpn”*. Предлагаются уточнения правил для реализации инверсии сетей Петри, как проверки достижимости заданной маркировки пространства состояний для анализа отдельных частей графа состояний анализируемой сети. Описывается приложение, способное отображать сети Петри в виде матриц с их последующим анализом. Приводятся алгоритмы исследования пространства состояний: метод “плавающей” линии, хеширование.

1. НАПРАВЛЕНИЯ В РАЗВИТИИ ПРОЕКТИРОВАНИЯ, РАЗРАБОТКИ И АНАЛИЗА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Анализируются основные подходы к разработке программного обеспечения (ПО), различные парадигмы написания приложений, а также методы проектирования и анализа проектируемого программного продукта. Некоторые способы и подходы иллюстрируются на конкретных примерах [74, 79].

В *п. 1.1* из всего многообразия способов выбирается проектирование с использованием унифицированного язык моделирования UML (Unified Modeling Language), предназначенный для моделирования систем различной сложности с использованием диаграмм, и позволяющий преобразовывать сконструированные диаграммы в программные коды, и последующим анализом диаграмм на основе математического аппарата сетей Петри.

В *п. 1.2* приводится описание UML, структурных диаграмм, диаграмм поведения и диаграмм взаимодействия. *В разделе 1.3* дается описание сетей Петри, предназначенных для анализа моделируемых систем. Поскольку UML не обладает формальными методами проверки корректности проектирования диаграмм, а сети Петри нацелены на анализ проектируемых систем, их совместное использование видится целесообразным. *В разделе 1.4* анализируются методики совместного использование UML диаграмм и сетей Петри.

В *п. 1.5 и п. 1.6* демонстрируются способы анализа пространства состояния сетей Петри, которые заключаются в хранении информации о каждом состоянии, в удалении неиспользуемых состояний, а также сжатии информации о каждом состоянии системы. Приведённое алгоритмическое описание позволяет реализовать данные способы анализа.

В *п. 1.7* дается постановка задачи диссертационного исследования на основании анализа достоинств и недостатков проанализированных способов и методик проектирования программного обеспечения.

1.1. Подходы, методы, способы в разработке ПО

В настоящее время трудно представить успешную и высокопроизводительную работу во многих сферах человеческой деятельности без качественных программных продуктов. Постоянно растущие требования заказчиков приводят к повышению функциональности и мощности разрабатываемых систем, что в свою очередь влечёт увеличение материальных и временных затрат на создание программного обеспечения. Для минимизации вкладываемых ресурсов в разработку программных продуктов пользуются специальными методами анализа, синтеза, проектирования и реализации. Следствием растущей сложности проектируемых систем стало использование визуального программирования, в частности визуального моделирования.

В программной инженерии существуют разные способы для написания программного обеспечения [2, 3, 5, 89, 95]. Самые распространённые: *RUP* – итеративный способ разработки, включающий полный цикл разработки ПО; *Agile* – способ, который представляет набор принципов и ценностей, определяющих поведение команды разработчиков; *RAD* предполагает, что разработка ПО осуществляется с использованием инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки; *CMMI* (Capability Maturity Model Integration) содержит набор рекомендаций, инструкций, способствующих достижению целей, необходимых для полной реализации определённых областей деятельности; *MDA* (Model Driven Architecture), суть которого состоит в построении абстрактной метамодели управления и обмена метаданными (моделями) и задании способов ее трансформации в поддерживаемые технологии программирования; *MSF* (Microsoft Solutions Framework) опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения; *XP, SCRUM, DSDM, OpenUP* и др. При создании качественного программного обеспечения, минимизации рисков, постоянной верификации конечного продукта больше всего подходит итеративная модель разработки. Помимо способов написания программного обеспечения существует несколько общепринятых парадигм программирования.

Парадигмы программирования [4, 36, 100, 101]: структурное, императивное, декларативное, функциональное, объектно-ориентированное программирование, каждую из которых рассмотрим подробнее. *Структурное программирование* – мето-

дология, в основе которой лежит представление программы в виде иерархической структуры базовых блоков (последовательное исполнение, ветвление, цикл). *Императивное программирование* описывает процесс вычисления в виде инструкций, изменяющих состояние программы. В *декларативном программировании* программа описывает не процесс создания сущности, а саму сущность. При *функциональном программировании* процесс вычисления трактуется как вычисление значений функций в математическом понимании последних. Основными концепциями *объектно-ориентированного программирования* (ООП) являются понятия объектов и классов.

Наиболее популярны методы моделирования и анализа, которые могут гарантировать высокую производительность и безотказность работы разрабатываемых программных продуктов. К ним можно отнести: семейство стандартов IDEF[4], switch-технология, язык описаний и спецификаций SDL, язык MSC (диаграммы последовательностей и сообщений), ИВТ-сети, Workflow модели, алгоритмический язык ДРАКОН (Дружелюбный Русский Алгоритмический Язык, Который Обеспечивает Наглядность), графический язык UML [23, 35, 106], математический аппарат сетей Петри [24, 37, 44, 46, 59, 112], нормальный алгоритм Маркова, алгоритм Питерсона, алгоритм Лампорта.

Следует отдельно выделить методологии, базирующиеся на использовании диаграмм ***UML и сетей Петри***, предложенные L. Baresi, M. Pezze [100]. Сеть Петри – двудольный ориентированный граф, который состоит из вершин двух типов – мест и переходов, соединенных между собой дугами, вершины одного типа непосредственно не соединяются. В местах могут размещаться метки (маркеры), способные перемещаться по сети. Сети Петри используются для обеспечения автоматизированного процесса тестирования проектируемого программного обеспечения [130, 131]. Существует множество программных средств для проектирования сетей Петри, например, CPN-AMI, CPN Tools, GreatSPN, PEP, Petri Net Toolbox, WoPeD и др. Наиболее популярен пакет CPN Tools[5] [13, 38], который находится в свободном доступе, имеет интуитивно понятный интерфейс и возможность не только моделировать сети, но и

[4] Атисков, А.Ю. Разработка технологии и программной системы автоматизированной трансформации диаграмм функционального проектирования в диаграммы UML [Текст]: дис. … канд. техн. наук: 05.13.11 / А.Ю. Атисков. – Санкт-Петербург, 2011. – 128 с.

Программные средства IDEF: BPwin, Business Studio, System Architect, CA Erwin Data Modeler.

[5] Дистрибутив доступен по адресу: http://cpntools.org/download

анализировать их через генерацию пространства состояний. UML – язык графического описания систем, использование которого позволяет увеличить скорость разработки программного продукта и уменьшить количество синтаксических, семантических и других видов ошибок. UML включает 15 диаграмм: диаграмма классов, диаграмма компонентов [109], диаграмма композитной/составной структуры, диаграмма кооперации, диаграмма развёртывания, диаграмма объектов, диаграмма пакетов, диаграмма профилей, диаграмма деятельности, диаграмма состояний, диаграмма вариантов использования, диаграмма коммуникации, диаграмма обзора взаимодействия, диаграмма последовательности, диаграмма синхронизации. Охарактеризуем существующие CASE-средства (Computer-Aided Software Engineering) несколькими свойствами. Так, например, поддержкой всех стандартных типов диаграмм UML 2.0, возможностью генерации исходного кода на основе построенных диаграмм и *Reverse engineering* исходных кодов возможен в Sybase PowerDesigner, Enterprise Architect, Umbrello, Rational Rose, MagicDraw, Visual Paradigm. Поддержка основных баз данных Access, MS SQL, MySQL, Oracle, PostgeSQL доступен в Sybase PowerDesigner, MS Visio, Rational Rose, Visual Paradigm. В основном представленные пакеты имеют поддержку *XMI* и являются закрытыми платными продуктами. Для моделирования UML диаграмм выберем Rational Rose от компании IBM, Magic Draw от компании NoMagic. Данные пакеты имеют большой арсенал для моделирования диаграмм, связи их в единые проекты и преобразования полученных наборов в коды.

Одним из следующих этапов после проектирования ПО является его анализ [85], для реализации которого используются разные подходы и методы: метод *среза пучка (slice cluster[6])*, предложенный S. Islam, J. Krinke, D. Binkley и M. Harman, метод *гибридного спектрального среза (hybrid spectrum slice)*, предложенный X. Ju, S. Jiang, X. Chen, X. Wang, Y. Zhang, H. Cao, подход *minimal-MUMCUT*, позволяющий выполнять меньшее количество тестов для полного покрытия операторов управления, метод локализа-

[6] В основе данного метода используется понятие графа системы зависимости (system dependence graph) – ориентированный граф, отображающий соотношение элементов некоторой совокупности в соответствии с выбранным транзитивным отношением над ней. Например, пусть дано множество объектов M и отношение транзитивности $R = S \times S$, где $(a, b) \in R$, моделирующее зависимость «для вычисления a нужно сначала вычислить b», тогда граф зависимостей – это граф $G = (S, T)$, где $T \subseteq R$ и R является транзитивным замыканием T.

ции ошибок, описанный в работах X. Mao, Y. Lei, Z. Dai, Y. Qi, C. Wang, *ATEMES*, который позволяет генерировать набор тестов для функций, принимающих как простые, так и составные типы данных (работы C. Koong, C. Shih, P. Hsiung, H. Lai, C. Chang, W.C. Chu, N. Hsueh, C. Yang). Для проверки моделей программ языков C/C++ и PROMELA существуют следующие инструменты: CBMC, CPAchecker, LLBMC, SATABS, SPIN (язык PROMELA, автор G.J. Holzmann), DBRover, Dream, Ompca и др.

Объектно-ориентированный подход в написании программного обеспечения, по-видимому, является наиболее используемым. Первый шаг процесса объектно-ориентированного проектирования программного обеспечения заключается в создании структуры системы. На следующем этапе разработчик исследует динамические аспекты архитектуры и описывает алгоритм работы системы.

Моделирование программной системы начинается с описания структуры – множества составляющих её элементов (рисунок 1.1 [72, рис. 1]). После чего устанавливаются взаимосвязи между элементами. На следующем этапе описывается процесс взаимодействия элементов, а завершающим этапом является объединение знаний о структуре, связях и динамике элементов в одно целое – систему.

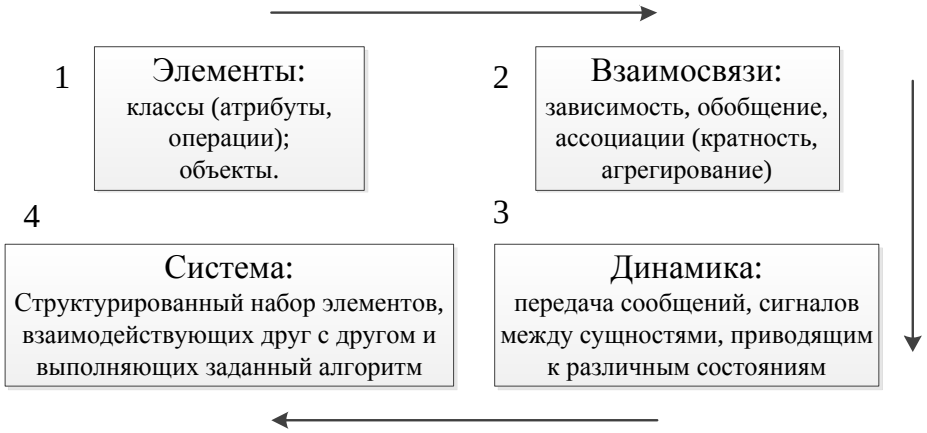


Рисунок 1.1 – Этапы процесса моделирования

Проектируемую модель необходимо проверить по ряду критериев. *Проверка корректности* (syntax checking) – операция, выполняемая лексическим анализатором компилятора или интерпретатора, и не позволяющая допускать синтаксические ошибки. *Верификацией* (verification) называют подтверждение смыслового соответствия модели входным требованиям с привлечением различных математических методов: синтаксических и семантических, в том числе и в автоматизированном режиме. *Аттестация* (validation), предметом которой является прагматическое качество модели – соответствие ожиданиям потребителя. Завершенное программное обеспечение тестируется при заданном наборе

входных параметров, для которых правильные значения результатов заранее известны. Основной *проблемой динамической проверки моделей системы* является *невозможность* в некоторых случаях *сформировать условия реального тестового окружения*.

Таким образом, разработка ПО является сложной, многогранной задачей, которую условно разбивают на несколько этапов: сбор требований к проекту, проектирование и анализ, программирование, тестирование, выпуск и поддержка. Существует множество графических средств, применяемых на этапе проектирования с использованием парадигмы ООП. Предпочтительным можно считать язык визуального моделирования UML, достоинством которого видится низкий порог вхождения, возможность привлекать к проектированию непосредственно заказчиков. К основным недостаткам относится отсутствие формальной проверки проектируемых диаграмм. На этапе проектирования разработчик может допустить ошибки, связанные с логикой работы системы (*зацикливание* $m' \to {}^*m'$, *мертвые переходы*, *тупиковые состояния* $m' \in V,\ m' = m_n,\ (m_n, t_n, m_{n+1}), t_n \notin T$ [7]), которые представляют особую трудность, поскольку могут быть выявлены уже в процессе функционирования программы. Для поиска данных ошибок выбираются сети Петри, которые разрабатывались специально с целью анализа исполняемого поведения с использованием пространства состояний. Таким образом, для разработки качественного программного обеспечения и соблюдения критериев надежности, функциональности предлагается совместно использовать UML диаграммы и сети Петри, что является продолжением работ, проводимых на кафедре автоматики НГТУ [30, 53, 93].

1.2. Применение UML диаграмм

Приводятся достоинства и недостатки UML диаграмм. Диаграммы условно разбивают на три категории: структурные, поведенческие и диаграммы взаимодействия. Из всех диаграмм подробно рассматриваются наиболее используемые из них. Демонстрируется ***взаимосвязь*** и возможное ***слияние*** динамических и статических ***диаграмм*** на примере диспетчера процессов, а именно диаграммой классов и диаграммой активности.

---

[7] m – маркировка пространства состояний, V – множество достижимых маркировок, t – переход в сети Петри. Переход называется мёртвым, если не существует достижимая из начальной разметки маркировка $m' \notin \left[m_1\right\rangle$, при которой переход $t \in T$ может сработать $m \xrightarrow{t} m'$.

В состав UML входит формальный текстовый язык *Object Constraint Language*, с помощью которого описываются различные инварианты типов, пред- и постусловий [15, 35, 55]. Также существует возможность автоматического преобразования UML диаграмм в программный код, а изменения в программном коде – обратно в UML (например, *Rational Rhapsody*). Но, несмотря на все положительные качества, UML обладает рядом недостатков, которые заключаются в большом количестве избыточных или практически неиспользуемых диаграмм и конструкций, в возможности *рассогласования нагрузки*[8], в представлении одних систем более кратко и эффективно, чем других; и в отсутствии инструментов для проверки корректности построения проектируемых поведенческих диаграмм. Несмотря на это использование UML диаграмм в проектах с объектно-ориентированной парадигмой разработки является оправданным, так как позволяет ускорить этап проектирования и сократить временные ресурсы.

При моделировании систем используют структурные диаграммы, которые описывают статическую структуру моделируемой системы, поведенческие диаграммы, описывающие различные состояния элементов системы и их изменения в зависимости от различных условий и диаграммы взаимодействия, которые отображают взаимодействие элементов системы во времени. Приведем описание структурных и поведенческих UML диаграмм, которые чаще, чем остальные, используются при разработке ПО.

Диаграмма вариантов использования (*use case diagram*) применима для отображения отношений между актерами (актор) в системе и их последовательностью действий. На *диаграмме классов* (*class diagram*) отображают структуру системы – классы, атрибуты, операторы, а также отношения между классами. *Диаграмма активности* (*activity diagram*) и *диаграмма состояний* (*state machine diagram*)[9] относятся к диаграммам поведения, на которых проектируют поведение объектов класса при помощи различных представлений. На *диаграммах коммуникации (communication diagram) и последовательности (sequence diagram)* отображают взаимодействия объектов системы. Приведём графическое представление часто используемых элементов перечисленных диаграмм (таблица 1.1).

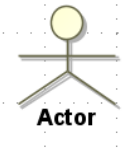
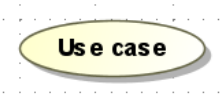
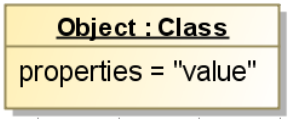
---

[8] *Рассогласование нагрузки* – неспособность системы воспринимать выходные данные другой, которые могут отличаться по типу, размерности и т.д.

[9] или *state chart*, для построения которой с последующей генерацией кода созданы следующие инструментальные средства: I-Logix Statemate, XJTek AnyState, StateSoft ViewControl, SCOPE, IAR Systems visualSTATE, The State Machine Compiler.

Взаимосвязь между структурными и поведенческими **диаграммами** заключается в описании одной и той же системы с разных сторон, а чёткая связь между этими диаграммами в языке UML отсутствует [4]. Архитектуру системы можно представить в виде множества всех элементов, а также множества связей между ними (в свою очередь любой элемент может рассматриваться как система элементов). При наличии связей между структурными и динамическими свойствами моделируемой системы разработчики получают дополнительную информацию о работе системы, которая отражает ее внутренние процессы, что крайне ценно для имитационного моделирования и верификации (рисунок 1.1).

Таблица 1.1 – Графическое представление часто используемых элементов UML диаграмм

Актёр (актор)	
Прецедент	
Класс	
Объект	
Действие	

Взаимосвязь диаграмм показана на примере диспетчера процессов [41], основная задача которого – распределение процессорного времени между запущенными приложениями. Соответственно, процесс и диспетчер будут являться ключевыми объектами в данном проекте. На рисунке 1.2 одновременно представлены диаграмма классов и диаграмма активности, которая является основным источником информации при трансляции полученного набора диаграмм в сеть Петри. Для отображения связи со структурой сеть Петри дополняется элементами, которые моделируют процессы работы с данными [72]. Из всего вышесказанного, следует, что синтез поведенческих и структурных диаграмм не только предоставляет более наглядное представление системы, но и может способствовать автоматической трансляции UML диаграмм в сеть Петри.

Таким образом, UML диаграммы разрабатывались для моделирования систем, включая статическое описание и её динамику работы. Также в UML присутствуют диаграммы, используя которые моделируются отдельные части системы, как физические объекты. Выбраны диаграммы, которые используются прак-

тически в каждой модели систем: диаграмма прецедентов, диаграмма классов, диаграмма объектов и диаграмма активности.

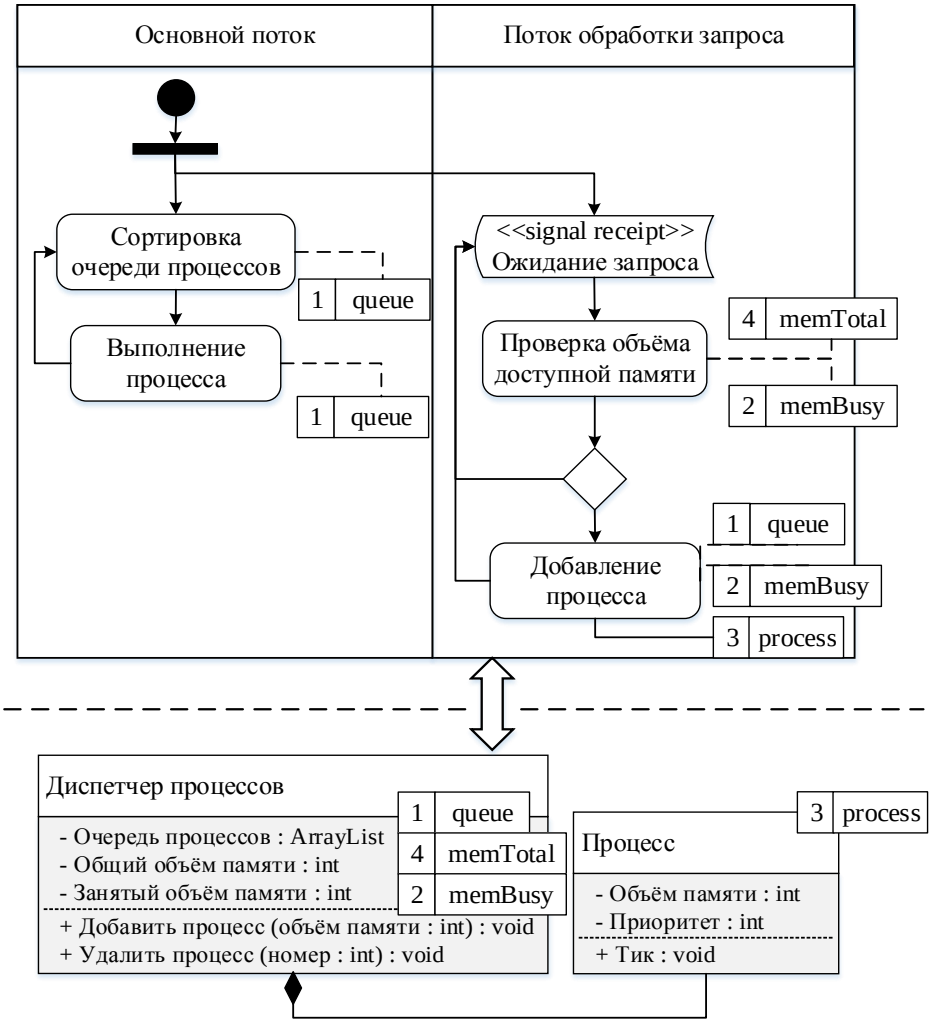


Рисунок 1.2 – Объединение структурной и динамической диаграмм диспетчера процессов

Так как в языке UML отсутствуют способы формальной проверки диаграммы активности, используем сети Петри и среду моделирования CPN Tools, использующая автоматизированный инструмент проверки пространства состояний.

1.3. Применение сетей Петри

Сеть Петри представляется четверкой $N = (P, T, F, m_I)$, где $P = \{p_1, p_2, \ldots, p_n\}$ – множество мест, $T = \{t_1, t_2, \ldots, t_m\}$ – множество переходов, таких что $P \cap T = \varnothing$, $F \subseteq P \times T \cup T \times P$ – отношение, а $m_I : P \to N$ – начальная *маркировка*[10] [54, 87, 118, 133, 135]. Графическое представление элементов сети Петри приводится в таблице 1.2.

[10] *Маркировка* – это размещение меток в местах сети Петри.

Для лучшего понимания элементов сети Петри и её графического представления на рисунке 1.3 представлена часть сети Петри протокола передачи данных (рисунок 3.23), которая содержит три места *P*, три перехода *T*, с начальной маркировкой m_I:

$P = \{ F, E, S1 \}$;
$T = \{ Receive\ Packet2, transmit\ packet2, send\ packet2 \}$;
$m_I = (0, 0, 1)$.

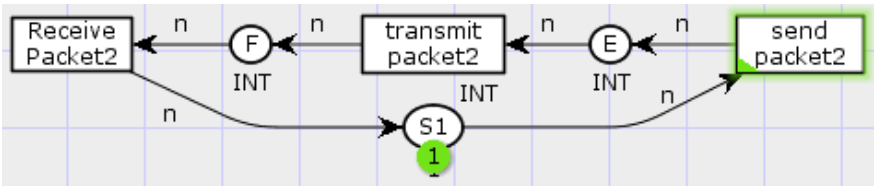
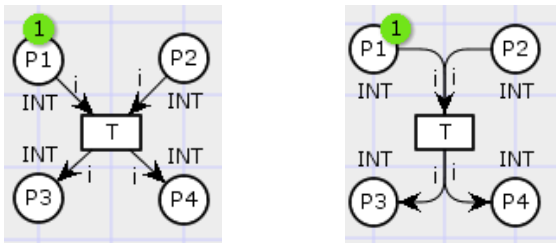


Рисунок 1.3 – Часть сети Петри протокола передачи данных

Взаимосвязи между вершинами сети (рисунок 1.4.а) на некоторых рисунках будем представлять, как показано на рисунке 1.4.б.



а) б)

Рисунок 1.4 – Представление взаимосвязей сети Петри: а) стандартное, б) компактное

Приведём правила срабатывания переходов:

– переход срабатывает при наличии метки в месте, входящем в этот переход, а количество меток должно быть не меньше, чем количество входящих дуг в переход;

– число меток, которые появляются в месте после срабатывания перехода, равно количеству дуг, входящих в данное место.

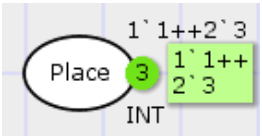
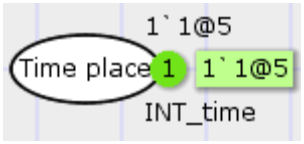
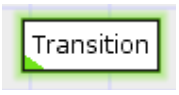
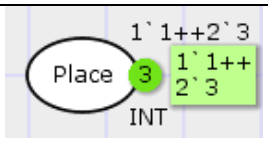
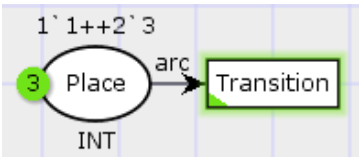
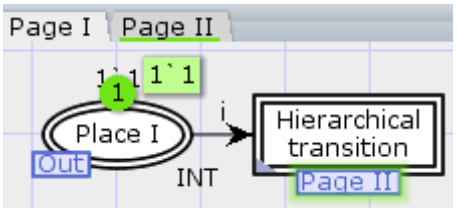
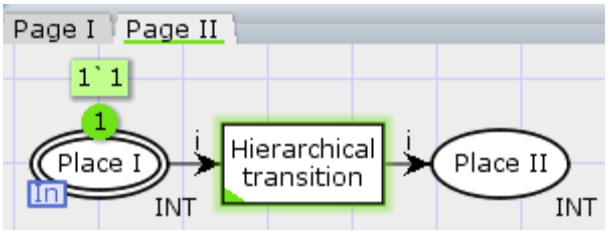
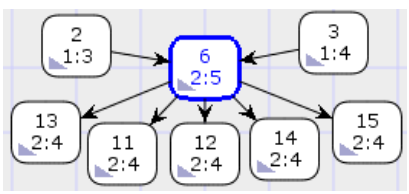
Иными словами, переход $t \in T$ разрешён в маркировке $m: P \to N$, если $m \geq \bullet t$. Если *t* разрешён в *m*, то происходит срабатывание перехода *t*, что приводит к маркировке m'. Записать это можно следующим образом: $m \left[t \right\rangle m'$ [11], где m' определяется как $m' = (m - \bullet t) + t \bullet$ [12]. Переменную *R* используют для обозначения количества достижимых маркировок, т.е. $R = \left| \left[m_I \right\rangle \right|$ (из начальной маркировки получаем возможные при выполнении цепочек срабатываний соответствующих переходов).

---

[11] $m \left[t \right\rangle m'$. Маркировку m' получают из m при срабатывании перехода t.

[12] $m' = (m - \bullet t) + t \bullet$. Маркировка m' получена из m, учитывая взаимосвязи (дуги и их количества) до перехода t и после него.

Таблица 1.2 – Элементы сетей Петри

Элемент	Графическое представление	Символьное обозначение	Пояснение
Место		$P = \{ Place \}$	P содержит три метки (одна – тип данных “1”, другие – тип данных “3”)
Место с элементом времени		$P = \{ Place \}$	P содержит одну метку с типом данных “1” и временной составляющей “5”
Переход		$T = \{ Transition \}$	Зеленое свечение у T – переход может сработать
Маркировка		$m = (3)$	m состоит из трех меток, находящихся в одном месте
Взаимосвязь место/переход		$F \subseteq Place \times \times Transition$	
Иерархическая сеть Петри	 		*Place I* является выходным местом страницы *Page I* и входным для страницы *Page II*. Переход *Hierarchical transition* является иерархическим и ведет на подсеть *Page II*. При срабатывании данного перехода метка перейдет в место *Place II*
Граф состояний		S_n	Состояние №6 имеет двух родителей и пять потомков

Решение большинства задач стало возможным благодаря интенсивному развитию сетей Петри[13] [42, 43, 96-99, 137-143], имеющих большое количество модификаций и разновидностей, основные из которых представлены на рисунке 1.5. В ***ординарных*** сетях Петри количество меток в местах не превышает одну, в ***простых*** сетях Петри в местах хранится любое количество меток. В ординарных и простых сетях все метки имеют одинаковый тип данных. ***Цветные*** сети Петри моделируются с использованием любого количества меток в местах с возможностью задания разных типов данных.

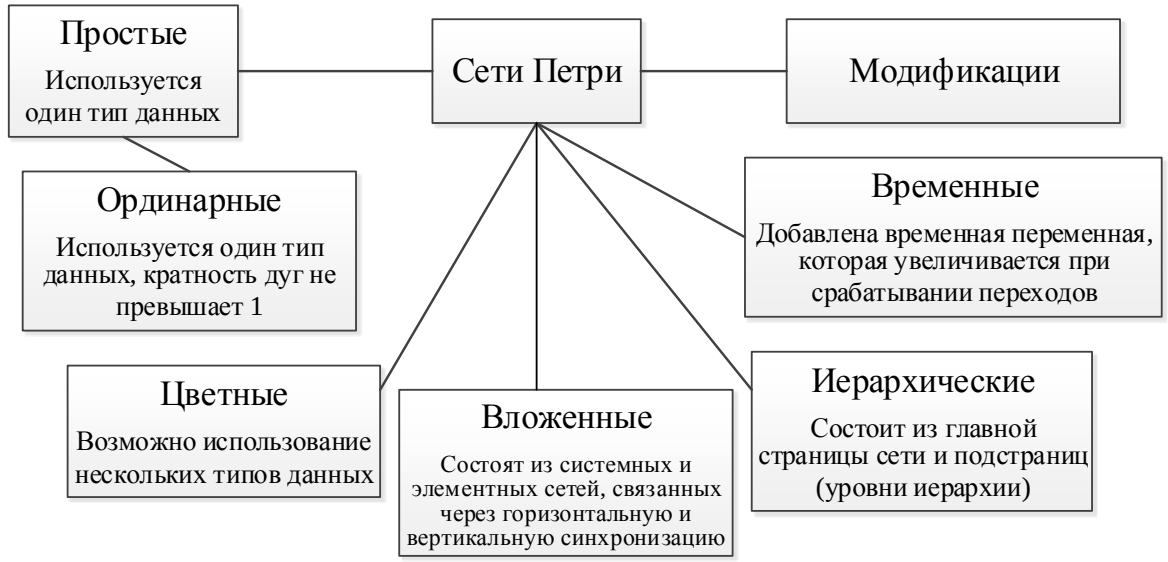


Рисунок 1.5 – Расширения сетей Петри

Приведем свойства сетей Петри, одним из которых является свойство ***ограниченности***. Сеть Петри (P, T, F, m_I) k-ограниченна, если и только если все маркировки $m \in [m_I\rangle$ и для всех $p \in P: m(p) \le k$. Частным случаем ограниченности является ***безопасность***. Сеть Петри (P, T, F, m_I) безопасна, если и только если все маркировки $m \in [m_I\rangle$ и для всех $p \in P: m(p) \le 1$. Также существуют проблемы ***достижимости***[14] и ***покрываемости***[15], заключающиеся в

[13] Hack, M.H.T. Decidability for Petri nets [Text] / M.H.T. Hack. – Massachusetts. Cambridge. – 1976. – 194 p.

[14] *Достижимость* – получение заданных состояний из начального состояния $m \in [m_I\rangle$.

[15] *Покрываемость* – получение заданных состояний из других состояний $m'' \in [m'\rangle$.

возможности полного анализа пространства состояний системы и проверки принадлежности выбранных маркировок ко всему пространству состояний.

Стоит отметить, что сети Петри используются для моделирования систем, содержащих взаимодействующие параллельные потоки обработки данных. Можно предположить, что системы состоят из отдельных взаимодействующих компонент, каждая из которых может быть системой, но её поведение описывается независимо от других компонент системы, за исключением точно определенных взаимодействий с другими компонентами.

При проектировании ПО с использованием сетей Петри, зачастую возникает необходимость моделирования систем сложной структуры. Использование цветных и иерархических сетей способствует более компактному и наглядному представлению моделируемой системы. Но моделирование на основе сетей (рисунок 1.5) не всегда приводит к желаемому результату. Для этого необходим способ, заключающийся в интерпретировании информации не через комбинацию мест и переходов, а через информацию, хранящуюся в метке с использованием составных типов данных, так называемых нагруженных меток. Также при разработке ПО используют рекурсивные функции, моделирование которых рассматривается в [31, 57, 73, 113].

Таким образом, приведено определение математического аппарата сетей Петри и описаны основные правила их работы. Перечислены разновидности сетей: простые, цветные, иерархические, временные, их модификации и т.д. Описано свойство ограниченности и достижимости сетей. Также стоит отметить, что существует возможность анализа довольно сложных систем с использованием сетей Петри: определять зацикливающиеся и неиспользуемые сценарии работы, выявлять возможное обращение к несуществующему элементу массива и т.д. Совместное использование UML диаграмм и сетей Петри способствует повышению качества создаваемого ПО при сокращении времени на этап проектирования.

1.4. Проектирование программного обеспечения на основе UML диаграмм и сетей Петри

Совместное использование UML диаграмм и сетей Петри рассматривается в работах [48, 108, 110, 120, 121], в которых приводятся правила и подробные инструкции для преобразования диаграмм в сети Петри. Создание программных

продуктов на основе совместного использования UML диаграмм и сетей Петри [51, 90, 126-129, 132] можно разделить на несколько этапов: выбор и разработка необходимых UML диаграмм, трансляция полученных диаграмм в сети Петри, анализ сетей Петри и внесение необходимых изменений в UML диаграммы в соответствии с результатами проверки. ***Анализируется методика***, представленная в работе [53] ***и её*** последующее ***развитие*** [93], предлагается модификация, которая заключается в уменьшении набора используемых диаграмм, что расширяет область применения методики до систем локальной автоматики.

Одной из первых работ, в которой предлагается совместное использование UML диаграмм и сетей Петри для разработки ПО, была [48]. В данной работе моделируется диаграмма активности процесса контроля и управления, ожидания сообщения от вновь подключенного блока релейной защиты и обновление списка блока релейной защиты для диспетчерского центра системы дистанционного контроля и управления. Полученные диаграммы состояний преобразуются в цветную иерархическую сеть Петри по предложенным правилам преобразования (последовательность действий, разделение/слияние, выполнение условия). Анализ полученной сети позволяет говорить об отсутствии логических ошибок: зацикливаний, мёртвых переходов, накоплении неограниченного числа меток.

В работе [108] сети Петри рассматривается преобразование действий и передачи сообщений между параллельными потоками диаграммы последовательности в сети Петри. Предлагается добавление временной составляющей (затраченное время на передачу сообщений) к диаграммам последовательности. Работа [110] посвящена проектированию диаграммы прецедентов и диаграммы последовательности, которые преобразуются в соответствующие сети Петри для моделирования проектируемой системы, что продемонстрировано на примере взаимодействия пользователя с банкоматом. В работе [120] также говорится, что динамические модели UML, следовательно, и диаграммы последовательности не имеют достаточно формальной семантики, из-за чего трудно построить автоматизированные инструменты для их анализа. По этой причине предлагается использовать сети Петри для анализа проектируемых диаграмм последовательности с описанием выполнения. В [126] выделяются основные принципы программной инженерии, сети Петри выбираются в виде формализма проверки UML диаграмм. Рассматривается преобразование диаграммы состояний и диаграммы активности в стохастическую сеть Петри по предложенным правилам. В

[127] сети Петри используются при построении диаграммы состояний из диаграммы коммуникации, т.е. на основании диаграмм коммуникаций строятся соответствующие сети Петри, которые в последствие объединяются в одну сеть, для которой строится граф состояний. Из полученной сети Петри получают диаграмму состояний, при необходимости разделяющуюся в соответствии с каждой диаграммой коммуникации. В [128] описывается моделирование системы при помощи языка OWL-DL с последующим преобразованием в UML диаграммы и сеть Петри. Для проектирования сетей Петри используется пакет CPN Tools, показана возможность анализа сетей Петри с использованием генерации и исследования пространства состояний. В приложении приводится описание элементов UML диаграмм, OML и CPN в виде сводной таблицы: начальный и конечный узел, действие, условие, разделение и слияние. В работе [132] рассматривается использование UML диаграммы активности при автоматизации документооборота в крупных компаниях и государственных учреждениях. Сети Петри используются для проверки проектируемых диаграмм деятельности, но, к сожалению, не описано преобразование диаграммы активности в сеть Петри.

В диссертационной работе [53] предлагается методика, состоящая из двух этапов совместного использования UML диаграмм и сетей Петри и её графическое представление. На первом этапе собираются требования, предъявляемые к системе, и представляются на *диаграмме прецедентов* (рисунок 1.6 [79]).

Следующим шагом является построение *диаграммы активности* для вариантов использования, требующих не тривиальных подходов к проектированию и реализации. После чего на *диаграмме процессов* отражаются процессы и потоки будущей системы, ответственные за выполнение прецедентов. Следующим шагом является построение *диаграммы активности*, на которой приводится выполнение действий и их взаимодействия. Полученную диаграмму активности преобразуют в *сеть Петри*, в которой с использованием различных начальных данных выполняют *ручное моделирование*. При обнаружении ошибок вносят корректировки в диаграмму активности, после исправленная диаграмма вновь преобразуется в сеть Петри для проверки. На следующем шаге выполняют *генерацию и исследование пространства состояний* модели. Данную операцию в автоматическом режиме выполняют в программном пакете CPN Tools, а от ручного моделирования диаграммы активности лучше отказаться.

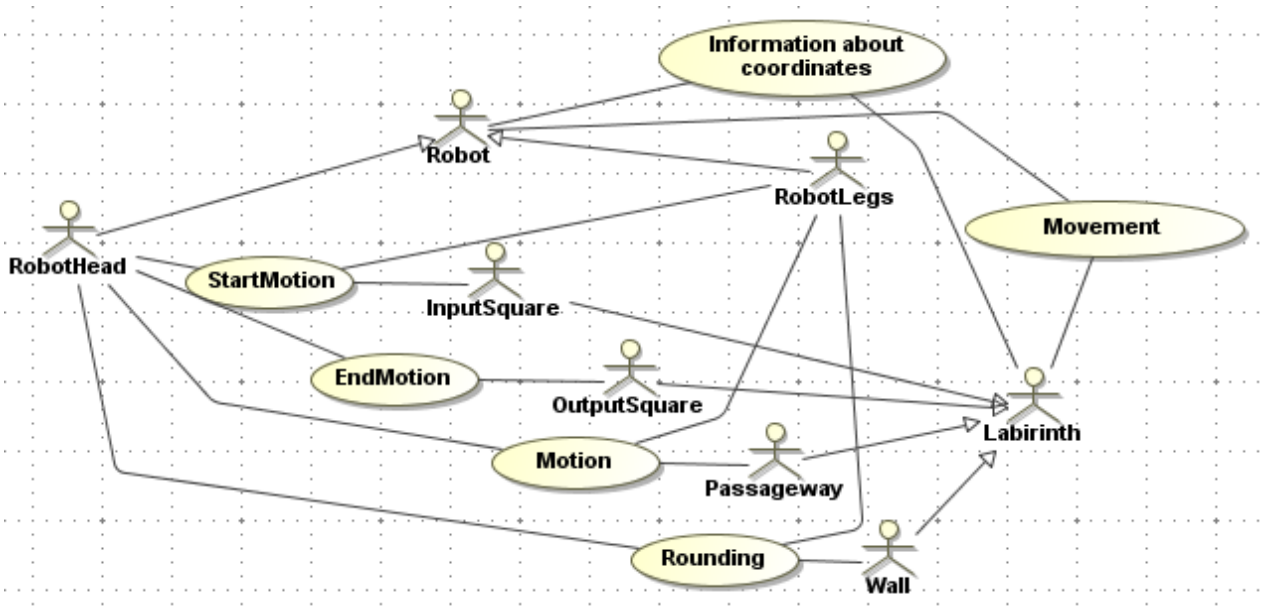


Рисунок 1.6 – Диаграмма прецедентов перемещения манипулятора в ограниченном пространстве с препятствиями[16]

После переходят ко второму этапу методики, а при нахождении ошибок возвращаются к диаграмме активности для её корректировки. На первом шаге второго этапа проектируется *диаграмма классов*, опираясь на процессы и потоки, отображенные на диаграмме процессов. Для каждого из классов создаётся *диаграмма состояний*, в которых класс может оказаться при выполнении программы. Взаимодействие между классами отображается на *диаграмме активности*. Диаграмма состояний каждого класса преобразуется в иерархическую сеть Петри. *Связь между* всеми *страницами* иерархической сети представляют с использованием страницы, проектируемой на основе полученной диаграммы активности. После производят генерацию и исследование пространства состояний полученной сети, а в случае обнаружения ошибок и неточностей, вносят корректировки в сеть Петри и соответствующую UML диаграмму. При положительных результатах, полученных при анализе пространства состояний, приступают к *автоматической генерации* набора UML диаграмм *в программный код*. ***Достоинством*** данной методики является четкая последовательность действий, выполняя которые проектируют ПО с последующим получением программного кода. Описание структуры и динамики функционирования выполняется при проектировании всех предложенных диаграмм. К ***недостаткам*** можно отнести выполнение методики в несколько этапов – отдельно для описания струк-

[16] Отношение ассоциации у акторов и прецедентов в программной среде Magic Draw выполняется прямыми линиями

туры и отдельно для описания поведения. Предлагается ручное моделирование сети Петри, что всегда уместно при её проектировании. Выносить данное действие в отдельный пункт нет необходимости. Стоит отметить, что данная методика специализирована для разработки ПО центров дистанционного контроля и управления, следовательно, неизвестно приложима ли она для другого рода задач. Также в данной работе анализируются сети Петри с количеством состояний порядка 10^4.

В диссертационной работе [93] предлагается развитие представленной выше методики и её графическое представление в виде взаимодействующих блоков. Данную методику реализуют в один этап: как и в более ранней версии разработчики собирают требования, предъявляемые к системе, и строят *диаграмму прецедентов*, после чего приступают к моделированию *диаграммы классов*, например, на рисунке 1.7 представлена диаграмма классов управляемого светофора [61].

Рисунок 1.7 – Основная часть диаграммы классов управляемого светофора

После приступают к составлению *диаграммы объектов*, на которой определяют начальные значения объектов системы. Далее используют *диаграмму состояний или*

активности для описания динамических свойств системы. На следующем шаге полученную поведенческую диаграмму преобразуют в эквивалентную *сеть Петри*. Анализ сети Петри, как и в более ранней версии методики, реализуется на основе *генерации и исследования пространства состояний*. При нахождении ошибок и неточностей во время анализа необходимо добавить *корректировки* в построенные UML диаграммы. При отсутствии ошибок составленный набор диаграмм готов к *генерации программного кода*. ***Достоинством*** данной методики является сокращение числа этапов и шагов по отношению к более раннему варианту. Диаграмму состояний или активности используют только для детального рассмотрения диаграммы классов, а детальное рассмотрение диаграммы прецедентов опускают. Стоит отметить, что данный вариант методики подходит для более широко круга задач и в частности для систем локальной автоматики. К ***недостаткам*** можно отнести отсутствие детального рассмотрения диаграммы объектов. Использование диаграммы состояний при проектировании динамических свойств системы можно опустить, так как интересующие состояния можно проанализировать на сгенерированном пространстве состояний системы. В данной работе с использованием предложенной методики анализируются сети Петри с количеством состояний порядка 10^5.

Отметим, что методику проектирования [53] можно реализовать в один этап для сокращения итераций моделирования диаграмм, изменить последовательность построения и проверки диаграмм, а также отказаться от использования диаграммы состояния [53, 93] и заменить выявление прецедентов, которые необходимо проверить более детально на выявление соответствующих классов системы. В способах проектирования можно увидеть использование диаграммы прецедентов – для сбора информации о системе, проектирование диаграммы классов и диаграммы объектов – для формирования сущностей, а также проверку динамики работы системы на основе диаграммы активности и/или диаграммы состояний.

Трансляция диаграмм в сеть Петри осуществляется следующим образом: исходная поведенческая диаграмма (как правило, это диаграмма состояний или диаграмма активности) согласно набору правил преобразуется в сеть Петри. Если при работе сети возникли ошибки (например, тупиковые ветви) – значит, при проектировании архитектуры тоже допущены ошибки, а результаты моделирования сети Петри позволяют их локализовать.

Трансляция UML диаграмм в раскрашенную иерархическую сеть Петри осуществляется по *формальным правилам преобразования*, предложенных в [16, 40, 48, 110, 121] и рассмотренных более детально в диссертационной работе [53]:

– состояние ожидания трансформируется в место, а состояние действия преобразуется: в переход, который начинает действие; место, отражающее состояние выполнения действия; переход, завершающий выполнение действия;

– разделение и слияния параллельных потоков управления UML диаграмм преобразуется в соответствующий эквивалент сети Петри;

– ветвление на диаграмме деятельности преобразуется в эквивалент сети Петри;

– условия ограничения выполняются при использовании условий ограничения переходов;

– критическая секция (доступ к критическому ресурсу, реализуемый критической секцией) преобразуется в место с одной меткой в начальной маркировке;

– семафор, используемый для моделирования занятости конкретного ресурса из пула ресурсов, преобразуется в место с тем же количеством меток, которые различаются друг от друга цветами в начальной маркировке;

– дорожка (swimlane) – область на диаграмме деятельности, содержащая элементы модели, которые выполняет отдельная подсистема, отражающаяся в виде отдельной подсети (страницы);

– к критическим по времени исполнения секциям добавляется место-счетчик.

При формальном преобразовании UML диаграммы состояния, описывающей поведение классов, в сеть Петри одним из недостатков является отсутствие возможности моделирования поведения нескольких объектов одновременно. Для решения этой проблемы в [93] были предложены *изменения в правилах* формирования сети Петри на основе UML диаграмм:

– в переходы сети добавляются защитные условия, определяющие метку (т.е. объект), способную совершить переход (т.е. вызвать метод);

– для вызова методов используются дополнительные места, соединенные с переходом, обозначающим срабатывание метода;

– для моделирования статических методов используется схема, как и для моделирования обычных методов.

Итак, методика проектирования ПО с использованием UML диаграмм и сетей Петри, предложенная в [53], состояла из двух этапов. Развитие методики [93] заключалось в сокращении её до одного этапа и детализации некоторых диаграмм. Тем не менее, в методике использовались диаграммы, наличие которых необязательно, что влияло на временные ресурсы при проектировании. По этой причине модифицируем методику: используем диаграмму прецедентов, диаграмму классов и диаграмму объектов, которые при необходимости могут быть модифицированы после моделирования диаграммы активности, диаграммы последовательности и анализа их в сетях Петри. В одной из первых работ [94] на основе наработок совместного использования UML диаграмм и сетей Петри выполняется проектирование ПО на примере современной станции управления лифтом, для которой разрабатывается расширение для пакета MATHCAD при решении задачи полиномиального синтеза. Ключевым моментом методики является отказ от моделирования диаграммы состояний, которую заменит исследование пространства состояний сети Петри, транслированной из диаграммы активности. Поскольку в перечисленных работах упоминалось о необходимости автоматической трансляции UML диаграмм в сети Петри, предлагается способ выполнения данной процедуры, для чего основные правила преобразования представляются в алгоритмическом виде. В последней части раздела приведены правила преобразования UML диаграмм в сети Петри.

1.5. Построение пространства состояний сетей Петри с сохранением информации о всех состояниях

Для поиска логических ошибок в разрабатываемом ПО используются сети Петри, нацеленные на анализ проектируемых систем [10, 11, 54, 87] на основе ручного моделирования или исследования пространства состояний [10, 29, 92, 103, 125, 134]. В данном разделе приводится определение графа состояний [70, 74, 76] и *системы меченных переходов*[17], а также приводится ***алгоритм*** для ***исследования графа состояний*** с сохранением информации о всех состояний.

---

[17] *Labelled Transition System (LTS,* англ., система меченных переходов) – четверка $LTS = (S, T, \Delta, s_I)$, где $S \neq \emptyset$ – множество состояний, T – множество переходов, $\Delta \subseteq S \times T \times S$ – отношение переходов, определяющее потомков состояний, $s_I \in S$ – начальное состояние.

Рассмотрим способ анализа сетей Петри, заключающийся в построении *графа состояний*, который представляется четверкой (V, E, src, trg), где V – множество вершин, E – множество рёбер между вершинами, а $src, trg : E \to V$ отображает в каждом ребре вершину, из которого оно получено, и вершину, к которому приводит выполнение срабатывания перехода, соответственно. *Корневой граф* – это множество (V, E, src, trg, r), такое что (V, E, src, trg) – граф, а $r \in V$ – *корень*[18].

Описание графа состояний:

– $V = \{m_I\}$ – множество достижимых маркировок;

– $E = \{(m, t, m') \in V \times T \times V \mid m[t\rangle m'\}$ – множество переходов из одной достижимой маркировке к другим маркировкам;

– src означает, что $src\,(m, t, m') = m$;

– trg означает, что $trg\,(m, t, m') = m'$;

– $r = m_I$ – корень представлен, как начальная маркировка.

Пусть $m, m' \in V$ два состояния, а $t \in T$ – переход. Если $(m, t, m') \in E$, тогда срабатывание t для m приведёт к m' или кратко $m \xrightarrow{t} m'$. Последовательность состояний s_i и переходов t_i может быть записана как последовательность вида:

$$m_1 \xrightarrow{t_1} m_2 \xrightarrow{t_2} m_3 \xrightarrow{t_3} \ldots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} m_{n+1}. \qquad (1.1)$$

Данную последовательность представим кратко: $m_i \xrightarrow{t_i} m_{i+1}$ для $1 \le i \le n$. Также существует отношение $m \to *m'$ ($\to *$ означает, что состояние m' получено из состояния m при последовательном срабатывании определенных переходов), если и только если при текущей последовательности (1.1), $n \ge 1$, с $m = m_1$ и $m' = m_{n+1}$. Состояние m' получают из m, если и только если $m \to *m'$, и указывает на множество состояний достижимых из s:

$$reach\,(m) = \{m' \in V \mid m \to *m'\}. \qquad (1.2)$$

---

[18] *Корень* – это часть ребра такая, что $r\,(m, t, m') = (t, m')$.

Стандартный способ построения ***графа состояний*** представим в виде алгоритма 1.1.

Алгоритм 1.1 – Построение графа состояний[19]

$Require: (P,\ T,\ F,\ m_I)$ // дана сеть Петри,

$Ensure: (V, E)$ // необходимо исследовать соответствующий граф состояний,

1: $V := \{ m_I \}$ // переменной V (множество посещённых состояний) присваивается множество с единственной переменной – начальное состояние m_I,

2: $W := \{ m_I \}$ // переменной W (множество не посещённых состояний) присваивается множество с единственной переменной – начальное состояние m_I,

3: $E := \emptyset$ // переменной E (множество рёбер) присваивается значение $\emptyset$,

4: *while* $W \neq \emptyset$ *do* // пока множество W не станет пустым, выполняем следующие действия,

5: *Select an* $m \in W$ // выбираем состояния m из множества W,

6: $W := W \setminus \{ m \}$ // убираем из множества W состояние m,

7: *for all* t, m' *such that* $m \xrightarrow{t} m'$ *do* // для всех t, m', удовлетворяющих $m \xrightarrow{t} m'$, выполняем следующие действия,

8: $E := E \cup \{(m, t, m')\}$ // добавляем новое ребро во множество,

9: *if* $m' \notin V$ *then* // если найдено новое состояние, не входящее в множество V, тогда,

10: $V := V \cup \{ m' \}$ // добавляем его к этому множеству,

11: $W := W \cup \{ m' \}$ // и к множеству W,

12: *return* (V, E) // возвращаем обновленные значения переменных V и E,

13: *end while* // конец цикла.

Основной проблемой при анализе систем является возможность “взрыва” пространства состояний, которая заключается в экспоненциальном росте количества состояний $\exp(|P|) = |V|$, что приводит к досрочному завершению анализа из-за недостатка нужных объёмов оперативной памяти. Следовательно, без оценки свойств возможно останется значительная часть пространства состояний, в которой могут присутствовать зацикливания и тупиковые маркировки, что свидетельствует о некорректном построении системы. Конечно, для хранения информации об исследованных

[19] Алгоритм представлен в виде псевдокода (M. Westergaard, [147]), к которому добавлены комментарии для лучшего понимания (А.В. Марков [82]).

состояниях можно использовать постоянную память машины, но это приведет к значительному росту времени, требуемого для анализа [103]. Для анализа подобных систем используются разные способы анализа пространства состояний.

1.6. Способы анализа пространства состояний

Приведём распространённые ***способы анализа пространства состояний***, использующие иные алгоритмы по отношению к традиционному – хранение исследованных состояний в оперативной памяти. Это *sweep-line method*[20] [105, 122] и *bit-state hashing*[21] [107, 114-116, 148] и их возможные модификации [147]. Данные способы нацелены на исследования пространства состояний при условии, что вычислительных ресурсов машины недостаточно.

Метод "плавающей" линии (S. Christensen [105]) показан на рисунке 1.8, где s_0 используется для обозначения начального состояния системы. Две серые области – области состояний, удерживаемых в памяти. Часть из них вычислена (светло-серый цвет), другая вычисляется (тёмно-серый). Sweep-line метод (алгоритм П. 6.1) использует характеристику *прогрессивной оценки*, где прогрессивная оценка[22] позволяет упорядочить $(V, \sqsubseteq)$ состояния системы и прогрессивные значения φ, назначая прогрессивное значение $\varphi(m) \in V$ каждому состоянию m. Пунктиром изображена линия, которая соответствует текущей прогрессивной оценке. Существует большая вероятность, что состояния строго левее линии не будут учувствовать при анализе системы и поэтому удаляются. По мере перемещения пунктирной линии по графу состояний, происходит вычисление достижимых состояний и удаление изученных. Каждому состоянию для данного способа исследования графа состояний присваивается соответствующая прогрессивная оценка.

К достоинствам данного способа можно отнести тот факт, что для анализа даже весьма сложных систем, используется определённое количество ресурсов анализатора, а именно оперативной памяти. Но стоит отметить, что sweep-line

---

[20] Метод "плавающей" линии – sweep-linemethod (англ.). Дословный перевод – метод линии-движения. Но более точно излагает суть перевод, предложенный авторами.

[21] Хеширование битового состояния – bit-state hashing (англ.).

[22] Прогрессивная оценка – мера для определения глубины исследования графа состояний.

метод отлично подходит только к *монотонным системам*[23]. При использовании данного способа не к монотонным системам стоит учитывать, что некоторые из состояний системы могут остаться не исследованными.

Рисунок 1.8 – Sweep-line метод [105]

Дальнейшее развитие посредством возвратных рёбер метод “плавающей” линии получил в работе [123]. При обнаружении состояний, потомками которых могут являться уже ранее проанализированные состояния, “плавающая линия” отодвигается до соответствующего потомка, но состояния с такой же прогрессивной оценкой не восстанавливаются, а дальнейшее исследование продолжается с найденного потомка, который сохраняется в памяти до окончания исследования (рисунок 1.9).

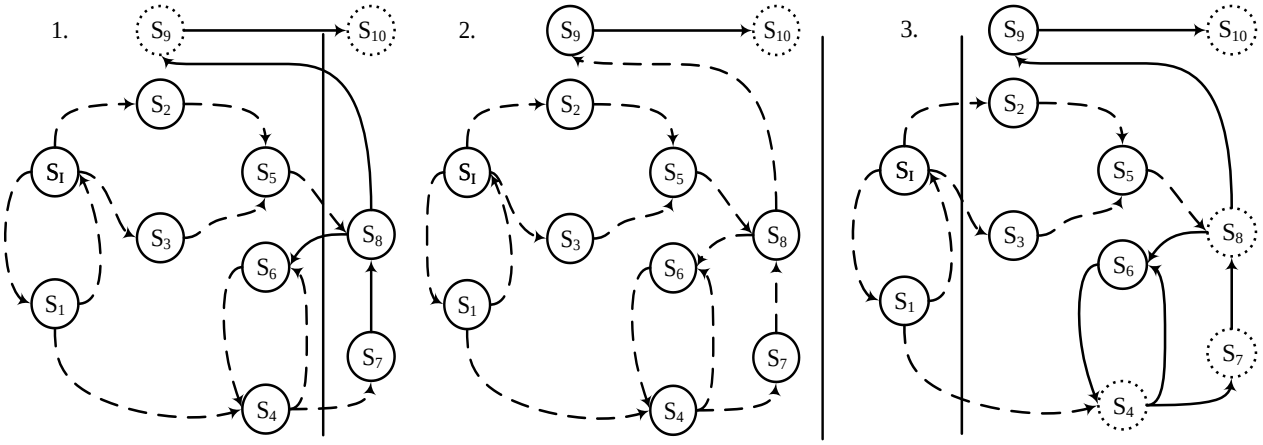


Рисунок 1.9 – Обобщённый sweep-line метод [120]

Для наглядности представим sweep-line метод с использованием возвратных рёбер (алгоритм П. 6.2 [120]) для системы меченных переходов $S = (V, T, E, m_I)$. Упо-

[23] *Монотонная система* – система, при исследовании пространства состояний которой новые состояния находятся из уже посещенных и взаимосвязи между состояниями направлены только от уже исследованных к новым.

рядоченное множество $(m, t, m') \in \Delta$ – возвратное ребро, необходимое для вычисления прогрессивной оценке $\varphi : V \to V$, если и только если $\varphi(m) \supset \varphi(m')$.

Данный способ эффективнее справляется с анализом систем, чем sweep-line метод, а область его применения значительно расширяется за границы монотонных систем. Стоит отметить, что более двух раз состояние не будет посещено при анализе всех состояний системы. К примеру, воспользуемся данным способом для анализа графа состояний (рисунок 1.10) взаимодействия пользователя с банкоматом, описанного в работе [74], количество состояний, которого потребуется запомнить до окончания исследования графа состояний не велико.

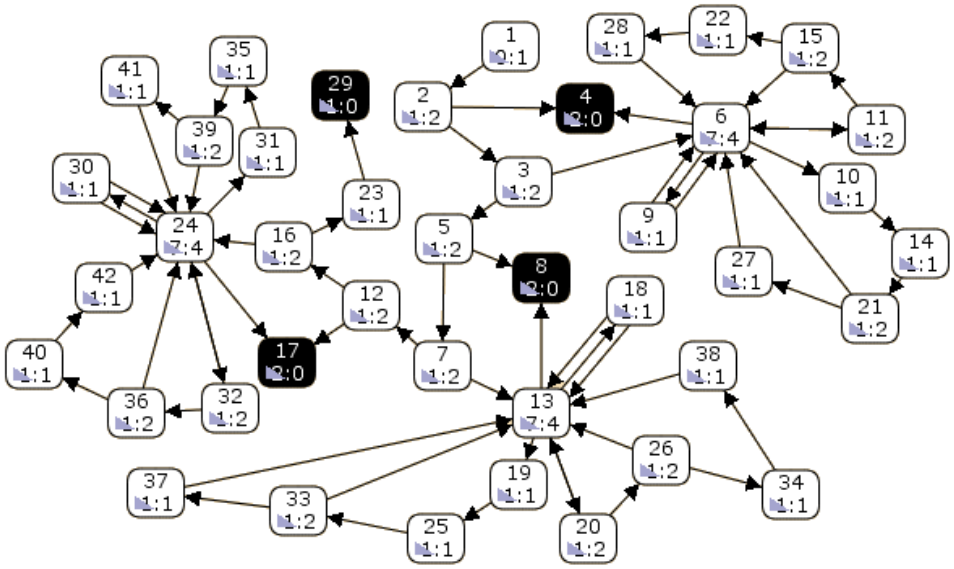


Рисунок 1.10 – Граф состояний сети взаимодействия банкомата и пользователя

Так в первой части графа состояний (рисунок 1.11), состояния №4, №6 необходимо запомнить, поскольку к ним обращаются состояния с большей прогрессивной оценкой.

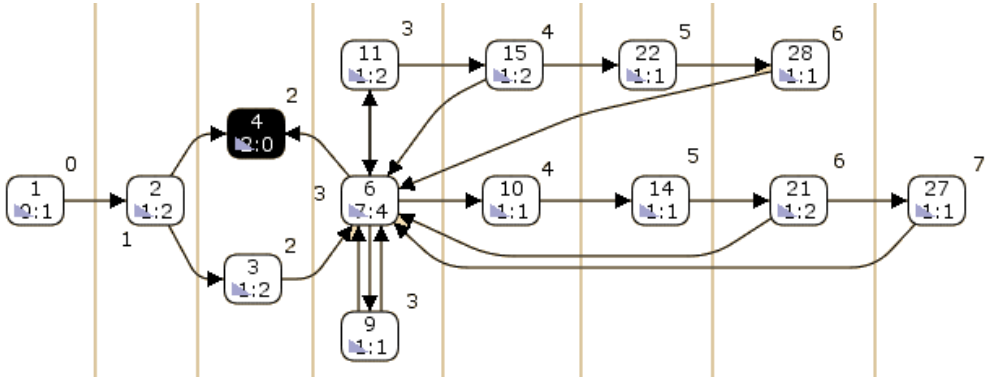


Рисунок 1.11 – Первая часть графа состояний взаимодействия пользователя с банкоматом

Но нужно помнить, что существуют системы, у которых возвратные рёбра присущи, практически, каждой маркировке графа состояний, что может привести к постоянному хранению большинства состояний, т.е. обобщённый sweep-line метод будет хранить состояния, как и при традиционном анализе графа состояний.

Побитовое хеширование состояний или Bit-state hashing (G.J. Holzmann [115]) – способ анализа пространства состояний, который заключается в сжатии информации о каждом отдельном состоянии через использование хеш-функций.

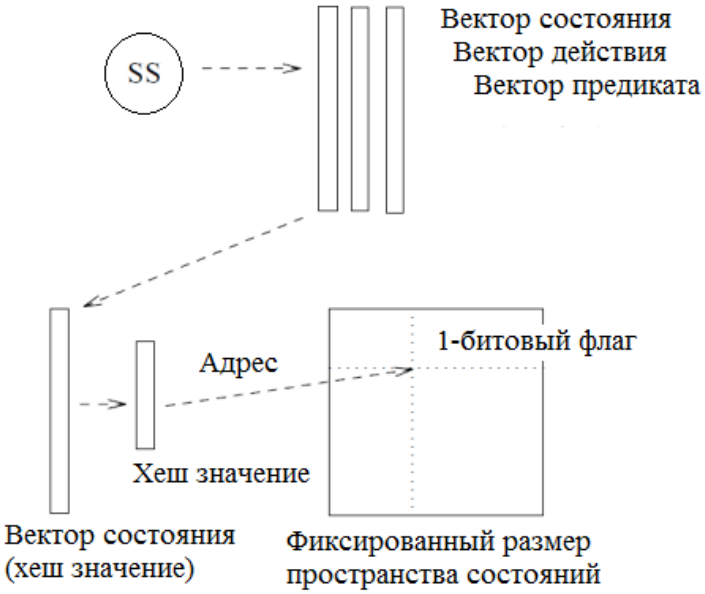


Рисунок 1.12 – Преобразование вектора состояния в однобитовый флаг[24] [115]

В данном способе получают вектор состояний от каждой маркировки системы, т.е. информация преобразуется в вектор состояния *State Vector*, вектор действия *Action Vector* и вектор предиката *Predicate Vector*, после чего вектор состояний хешируется для получения хеш-суммы[25]. Полученное значение преобразуют в адрес битового пространства состояний фиксированного размера (рисунок 1.12). Каждый вектор состояния используется для вычисления хеш-ключа, который используется для индексации в массиве состояний.

Отметим, что при появлении хеш конфликтов (наличие состояний в виде сжатой информации, имеющих одинаковую хеш-сумму), метод продолжает работать без ошибок, но стоит помнить, что в данном случае полную покрываемость пространства состояний получить не удастся. Коллизия у анализатора восприни-

[24]Битовый флаг состоит из последовательности доступных чисел 1, 2, 4, 8, 16, 32 и т.д., позволяющих хранить в одном параметре несколько опций, которые имеют состояние включено или выключено.

[25] Контрольная сумма (хеш-сумма) – значение, которое вычисляют по набору данных посредством применения определенного алгоритма, используемого для проверки целостности данных (данные не изменяются при выполнении любой операции над ними, будь то передача, хранение или представление).

мается, как уже обработанное ранее состояние и при исследовании игнорируется. Для решения данной проблемы используют повторение проверки с альтернативными хеш-функциями или альтернативной стратегией поиска [114]. Это приведёт к восстановлению пропущенных состояний и увеличению покрываемости.

Большим преимуществом данного способа является возможность хранения информации о состоянии всего в одном бите, что является существенной экономией ресурсов при анализе систем машинами с небольшими вычислительными мощностями. Но также существует вероятность получения коллизий, что плохо отразится на исследовании пространства состояний и, тем самым, снизит корректность анализа системы.

Метод возвратных рёбер или ComBack method (M. Westergaard [147]) – способ анализа пространства состояний с использованием различных хеш-функций, по которым состояние преобразуется с получением дополнительных хеш-сумм и информацией о предыдущих рёбрах.

Для анализа пространства состояний на основе ComBack метода используют комбинацию состояний и переходов между ними, т.е. если $m \xrightarrow{t} m'$ и $m \xrightarrow{t} m''$, тогда $m' = m''$, что справедливо и для переходов различных сетей Петри. Также используют обозначение $m \to *m'$, если и только если существует выполнение последовательности (1.1), с $m = m_1$ и $m' = m_n$. Состояние m' достижимо из m, если и только если $m \to *m'$ (1.2), что обозначает множество достижимых состояний из m. Пространство состояний системы может быть представлено в виде направленного графа (V, E), где $V = reach(m_I)$ – множество узлов и $E = \{ (m, t, m') \in \Delta \mid m, m' \in V \}$ – множество рёбер.

При реализации данного способа создают две таблицы, в одной из них хранят информацию о хеш-суммах и состояниях, соответствующих этим хеш-суммам, а в другой – информацию о родителе этого состояния и переходе, при срабатывании которого было получено данное состояние. ComBack метод, как модификация способа с использованием хеш-функций, заключается в следующем:

– каждому посещенному состоянию m присваивается номер $N(m)$;

– в таблице состояний хранят информацию обо всех хеш-суммах и соответствующих номерах состояний;

– в таблице возвратных рёбер хранят информацию о номере посещённого состояния $N(m)$, а также информацию об его родителях, состоящей из перехода t и номера состояния $N(m)$ посещенного состояния m', такого, что $m' \xrightarrow{t} m$.

Представим ComBack метод в виде алгоритма П. 6.3. Предложенный способ значительно уменьшает возможность возникновения коллизий при сжатии информации о состояниях и является отличной альтернативой хешированию состояний. Из проанализированных выше способов данный является лучшей альтернативой при наличии небольших мощностей у анализатора и условии полного исследования пространства состояний.

Итак, при необходимости используются методики анализа пространства состояний, которые заключаются в удалении уже посещённых ранее состояний – метод “плавающей” линии, а также в сжатии информации, присущей каждому состоянию – так называемое, хеширование. Способ двойного хеширования реализован в пакете SPIN [107], метод “плавающей” линии используют для генерации пространства состояний в пакете ASAP [147]. К достоинствам этих способов можно отнести сокращение ресурсов, требуемых на анализ систем со значительными размерами пространства состояний, но существенным недостатком является либо частичная покрываемость всех исследуемых состояний, либо повторное посещение уже изученных ранее состояний, поэтому необходимы методики, нацеленные на анализ полного пространства состояний и/или его отдельных частей.

1.7. Постановка задачи диссертационного исследования

На основе анализа способов проектирования ПО выбран язык визуального моделирования UML, а также математический аппарат сетей Петри. Выявлено, что совместное использование UML диаграмм и сетей Петри позволяет проектировать ПО с его последующим анализом. После чего выполняют генерацию программного кода. Существующие методологии проектирования ПО при совместном использовании UML диаграмм и сетей Петри включают проектирование излишних динамических диаграмм, а также выполнение лишних итераций, что увеличивает время проектирования. Следовательно, необходимо улучшение данных методик.

На основе сравнения способов анализа пространства состояний сетей Петри: сохранение информации о всех состояниях, метод “плавающей” линии, побито-

вое хеширование состояний, отметим метод возвратных рёбер, который является лучшей альтернативой при наличии небольших мощностей у анализатора и условии полного исследования пространства состояний.

На основании проведенных анализов и исследований сформулируем цель диссертационного исследования. Разработать ***методику*** проектирования программного обеспечения с использованием UML диаграмм и сетей Петри, позволяющую находить и устранять логические ошибки (зацикливание, тупиковые маркировки, мертвые переходы), в которой в отличие от более ранних версий не используется детализированная диаграмма прецедентов и диаграмма состояний, а для анализа отдельных сценариев используется диаграмма последовательности. Разрабатываемая методика должна быть применима для разработки систем широкого круга задач: производственные процессы, приложения для пользовательских компьютеров, систем транспортного регулирования, систем обработки информации и распределенных систем. Для чего решаются следующие задачи:

– составление пошаговой процедуры проектирования ПО с использованием UML диаграмм и сетей Петри для анализа поведенческих UML диаграмм;

– разработка способа проверки достижимости заданного состояния сети, представление правил и алгоритма для его реализации;

– разработка алгоритма преобразования UML диаграмм в сети Петри и способа выполнения автоматической трансляции UML диаграмм в сети Петри с использованием форматов .xmi и .cpn;

– разработка программного обеспечения для преобразования комбинации мест и переходов маркированных сетей Петри к матричной форме: составление матриц входной и выходной функций, получение составной матрицы и вектора начальной маркировки;

– разработка способа проектирования сетей Петри для задачи создания ПО в сфере робототехники, а именно робота-манипулятора; реализация рекурсивных функций в сетях Петри;

– разработка способов анализа сетей Петри, заключающихся в проверке частей пространства состояний.

Кроме того, разрабатываемую методику необходимо апробировать на задачах перемещения манипулятора в ограниченном пространстве, управляемого светофора и взаимодействия пользователя с банкоматом.

2. МЕТОДИКА ПРОЕКТИРОВАНИЯ И АНАЛИЗА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. РЕШЕНИЕ ЗАДАЧ, СВЯЗАННЫХ С ПРОЕКТИРОВАНИЕМ СЕТЕЙ ПЕТРИ СЛОЖНОЙ СТРУКТУРЫ

Во второй главе предлагается способ построения сетей, при котором представление структуры системы реализуется не через комбинацию мест и переходов, а через использование информации, хранящейся в метках. Стоит отметить, что метки, перемещающиеся по сети, нагружены дополнительным свойством, по которому можно отследить историю передвижения [79]. В *п. 2.2* на примерах переноса упорядоченного массива с ограничением выборки по одному элементу [31], нахождения значения числа Фибоначчи и нахождения факториала числа [73] предлагается реализация рекурсии в сетях Петри посредством нагруженных меток.

В *п. 2.3* для формализации основных правил преобразования UML диаграмм в сеть Петри (действие, выполнение условия, разделение/слияние) используется их алгоритмическое описание. Для автоматического преобразования UML диаграмм в сеть Петри предлагается использовать подобную структуру двух форматов *.xmi* (UML диаграммы) и *.cpn* (сети Петри), что позволяет сократить время на ручное преобразование диаграмм, а также помогает формализовать правила по их преобразованию. В *п. 2.4* даётся усовершенствованная методика совместного использования UML диаграмм и сетей Петри. Для описания общей логики работы системы используется диаграмма прецедентов, для описания статических свойств – диаграмма классов и объектов, а для моделирования динамических свойства используется диаграмма активности, которая анализируется на основе сетей Петри.

В *п. 2.5* предлагается один из способов анализа систем, основанный на свободном языке сетей Петри, а для его компактного представления используется сжатая форма, что демонстрируется на примере управляемого светофора и двухсимочного телефона.

2.1. Сети Петри с нагруженными метками: перемещение манипулятора в пространстве с препятствиями

Существуют задачи, например, перемещения манипулятора в ограниченном пространстве с препятствиями, проектирование которых с использованием известных модификаций сетей Петри затруднительно. Решение таких задач возможно способами построения сетей Петри, которые использовались в работах [53, 93], но проектирование сети займет значительное время, а сама сеть будет довольно большой и малоудобной для восприятия. Для решения данной задачи используются ***нагруженные метки*** (метка с дополнительной информацией об истории её передвижения или о структуре системы), схожие с предложенным независимо способом интерпретации вложенных сетей Петри[26]. Способ с использованием нагруженных меток предполагает хранение информации об ***истории их передвижения*** или о частичном представлении структуры системы, что может быть полезно при отслеживании изменения информации, хранящейся в метках, и при проектировании систем, чья статическая структура довольно громоздка. Особенностью данного способа является демонстрация свойств сетей Петри [12, 26, 39], а именно гиперпоточноить и параллелизм[27] (В.Э. Малышкин [58]).

Рассмотрим детально задачу перемещения манипулятора до выбранного места в ограниченном пространстве с препятствиями [88], например, робот-манипулятор на сборочном конвейере. Для таких систем создается программное обеспечение по алгоритмам, разработанным проектировщиками, но возможно альтернативное решение – самообучение, нахождение лучшего пути самим роботом-манипулятором. Предложим решение данной задачи и продемонстрируем его на иллюстративном примере – перемещение части звена манипулятора в плоскости (ограниченное пространство представим в виде лабиринта).

Представим лабиринт в виде двухмерного массива $x\,[i][j]$, каждый элемент которого имеет значение 1 (препятствие) или 0 (свободное пространство). Элемент

---

[26] Дворянский, Л.В. Имитационное моделирование и верификация вложенных сетей Петри с использованием CPN Tools [Текст] / Л.В. Дворянский, И.В. Ломазова. – Модел. и анализ информ. систем. – 2012. – Т.19, №5. – С. 115–130.

[27] При моделировании и программировании необходима синхронизация компонент, а их согласование действий во времени, зачастую, сложная операция. Например, возможно появление, так называемых, взаимных блокировок (дедлок) и зацикливаний.

звена за один шаг передвигается на одну клетку, после чего обращается к массиву, который описывает лабиринт, и из него получает информацию о своих координатах и возможных передвижениях, а в метке указываются координаты местоположения. Таким образом, идет распараллеливание на основе деления на копии части звена манипулятора, которые двигаются в доступных направлениях. Если поле уже было пройдено одной из копий части звена манипулятора, другая копия упрется в это поле и пройти дальше не сможет. При попадании элемента звена на поле, из которого нет возможных вариантов перемещения, заданное поле по данной траектории движения найти не удалось, а история перемещений, которая получена при движении до данного поля, сохраняется, чтобы этот путь не был пройден снова.

Сеть Петри (рисунок 2.1) для нахождения выбранного поля элементом звена манипулятора в ограниченном пространстве с препятствиями имеет девять мест *P*, восемь переходов *T* и начальную маркировку m_I:

$$P = \{\, Structure\ of\ the\ labyrinth,\ Drawing\ of\ the\ passed\ way,\ Movement\ of\ Robot,\ Up,\ Down,\ Left,\ Right,\ Robot,\ Final\ position \};$$

$$T = \{\, Drawing\ of\ the\ passed\ way,\ Direction\ up,\ Direction\ down,\ Direction\ left,\ Direction\ right,\ R1,\ R2,\ Final\ position\ Final\ position \};$$

$$m_I = (\,24,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0\,).$$

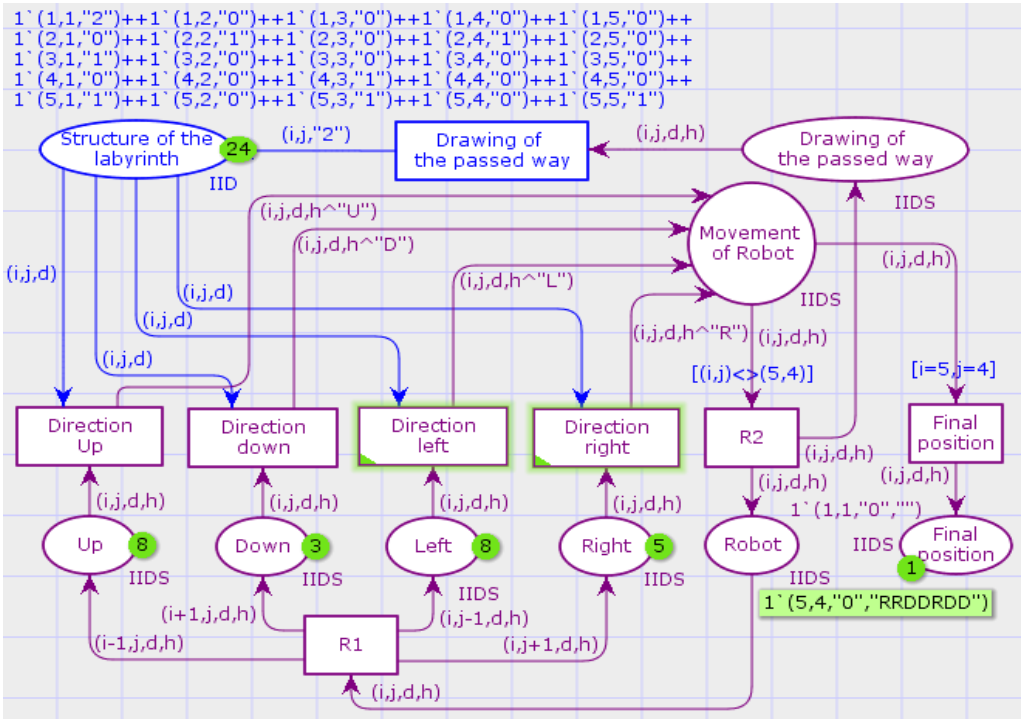


Рисунок 2.1 – Сеть Петри перемещения манипулятора в ограниченном пространстве с препятствиями

Начало моделирования сети реализуется через срабатывание перехода *R1*, после чего места *Up, Down, Left, Right* получают по одной метке. При срабатывании переходов *Direction up, Direction down, Direction left, Direction right* координаты робота-манипулятора изменяются. Переход *Comparison n* сработает в том случае, если координата следующего перемещения в структуре массива свободна (присутствует фишка с символьным элементом равным "0"). Представление лабиринта задано в месте *Structure of the labyrinth* и состоит из 24 меток с использованием разных типов данных, которые представляют координаты лабиринта. *Structure of the labyrinth* показывает структуру лабиринта: "1" – препятствие, "0" – свободное пространство. Метке в месте *Robot* задаются начальные координаты, после срабатывания перехода *Comparison n* метка возвращается в место *Robot* и в место *Structure of the labyrinth* с измененным символьным типом данных на "2" – пройденный путь, для того чтобы копии метки не пересекали пути, пройденные другими копиями. Переход *Final position* сработает только в том случае, если к нему обратится метка с координатами выбранного заранее поля. В месте *Final Position* находится метка, хранящая в себе координату местоположения и историю передвижений до заданного поля. История накапливается при срабатывании перехода *Direction …* .

Достоинством данного алгоритма является возможность изменения и задания различных структур лабиринта без изменения структуры сети, при этом структура лабиринта может быть сколь угодно большой. К недостаткам относится возможность нахождения не самого кратчайшего пути до заданной координаты. При делении метки на "перекрестках" в лабиринте удалось наглядно продемонстрировать свойство параллелизма сетей Петри и показать возможность получения сетей Петри по словесному описанию задания [60, 66, 67, 145, 146].

Моделирование перемещения однозвенного манипулятора в ограниченном пространстве с препятствиями – это следующий этап развития системы и алгоритма. Разработаем алгоритм для однозвенного манипулятора – конструкции длиной в две клетки и усложним лабиринт. Данный манипулятор, представленный чёрным цветом, передвигается по лабиринту – пространство, ограниченное светлым цветом (рисунок 2.2). На рисунке 2.3 представлено восемь вариантов положения робота[28].

В данном случае моделирование сети начинается при срабатывании перехода *InMovement*. Метка из места *Robot* размножается на 10 частей – возможные передви-

[28] Конструкция манипулятора может увеличиваться на половину клетки [62, 63]

жения и повороты: *TurnLeft, TurnRight, Down, Up, Left, Right, DownLeft, DownRight, UpLeft, UpRight*. В свою очередь каждое передвижение и поворот меняет координаты робота в зависимости от его положения в лабиринте, для чего созданы подстраницы, показывающие изменения координат звеньев робота при передвижении или повороте в зависимости от положения самого робота. Представление структуры лабиринта также задаётся в одном месте *Structure of the labirinth* и состоит из 49 меток с определенным набором типов данных (*i* – значение по горизонтали, *j* – значение по вертикали, *d* – информация о структуре клетки лабиринта). Как видно из рисунка 2.4 манипулятор имеет три положения, при которых будет найден выход из лабиринта. В переходах *Final position 01…03* заложены условия их срабатывания: $[(i1, j1) = (7, 7), (i2, j2) = (6, 7)] \vee [(i1, j1) = (7, 7), (i2, j2) = (6, 6)] \vee [(i1, j1) = (7, 7), (i2, j2) = (7, 6)]$; если одно из них будет выполнено, то это означает, что манипулятор находится в заданном месте.

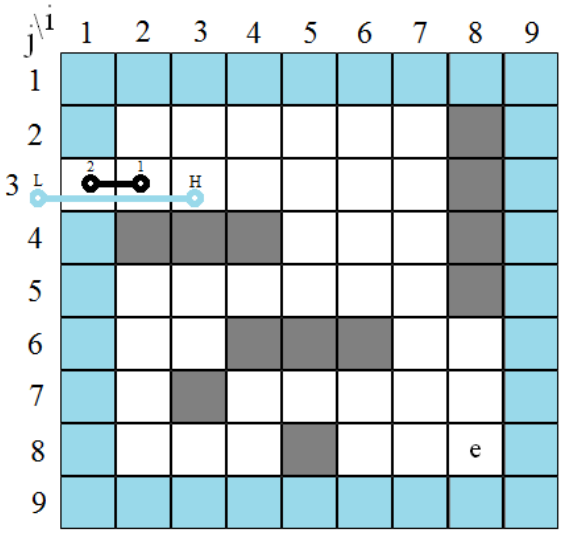


Рисунок 2.2 – Модифицированное замкнутое пространство

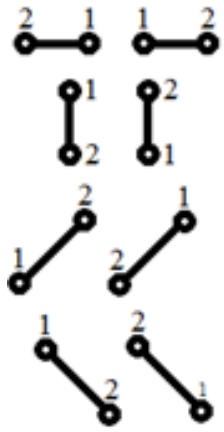


Рисунок 2.3 – Варианты положения манипулятора

Метка в месте *Robot* имеет составное множество цветов, т.е. составное множество типов данных (рисунок 2.5). Первые две цифры соответствуют координате первого элемента конструкции робота *(i, j)* – переменные типа *integer*, третья и четвёртая цифра иллюстрируют положение второго элемента в лабиринте. По последним двум типам данных можно проследить историю передвижений первого и второго элемента, соответственно – переменные типа *string*. Алгоритм перемещения однозвенного манипулятора, как и алгоритм перемещения элемента звена манипулятора имеет следующие достоинства: при изменении структуры лабиринта структура сети не меняется, структура лабиринта может быть сколь угодно большой, а все свойства закладываются в метке.

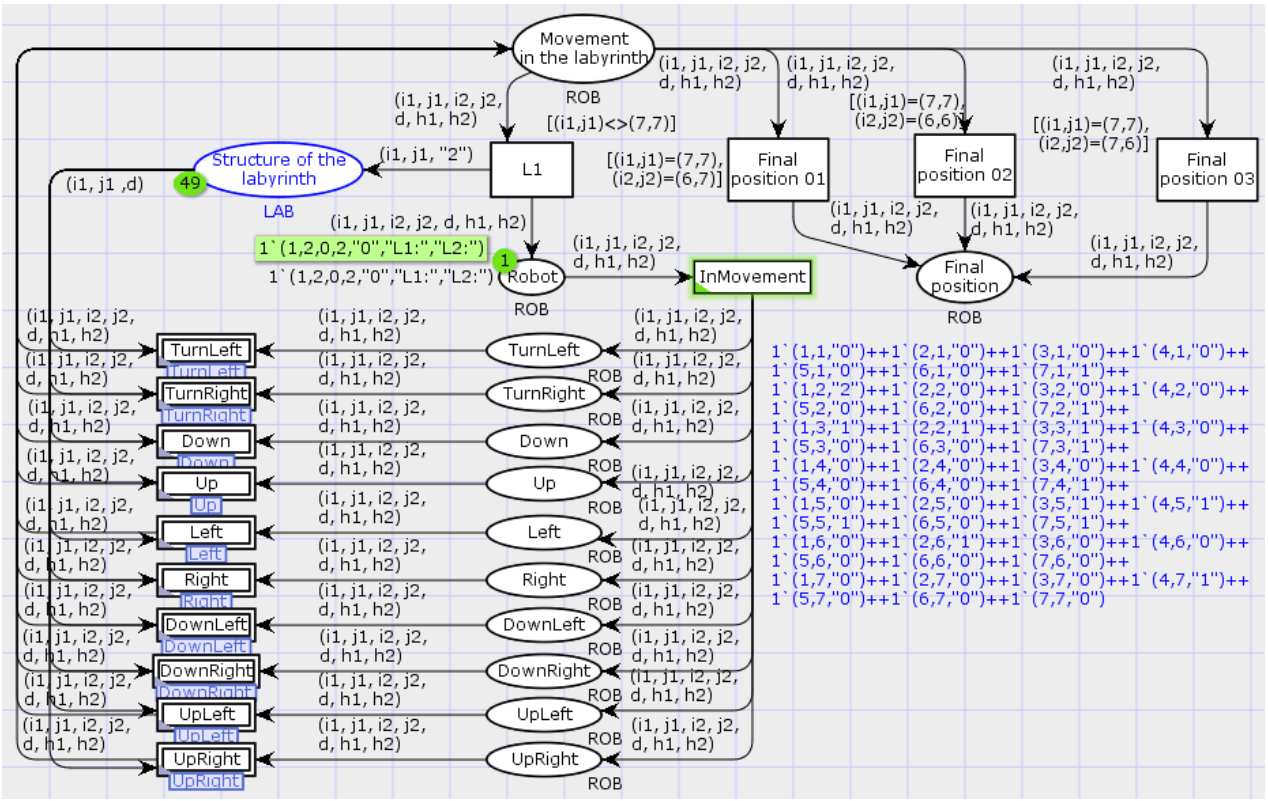


Рисунок 2.4 – Главная страница сети Петри перемещения манипулятора в ограниченном пространстве с препятствиями

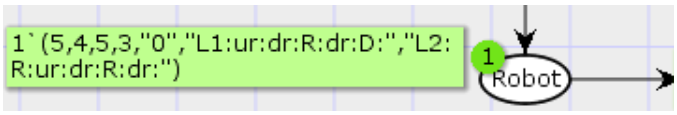


Рисунок 2.5 – Состояние манипулятора после пяти перемещений в замкнутом пространстве

Тем не менее, система претерпела значительные изменения по сравнению с более ранним вариантом, которые заключаются в модификации конструкции манипулятора: однозвенный манипулятор длиной в две клетки. Данные изменения повлияли на добавление целочисленных и символьных типов данных для координирования звена и отображения истории его перемещений.

Модифицируем манипулятор, увеличив его длину [67, 79]. Манипулятор представляет конструкцию из двух связных элементов длиной четыре клетки. Движение звена будет ориентировано относительно начального элемента. Во избежание блокировки про-

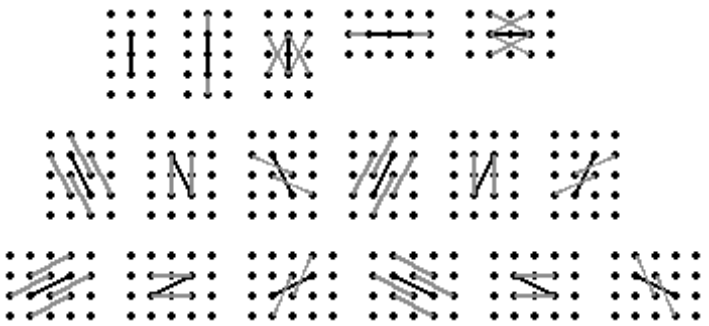


Рисунок 2.6 – Возможные передвижения и повороты манипулятора. Черная линия – начальное положение, серая – возможное передвижение

граммы, во время перебора возможных перемещений робота, вокруг лабиринта добавлены клетки с типом данных *"Wall"* (рисунок 2.2). Манипулятор может находиться в 16 положениях (рисунок 2.6 [65]) и перемещается по следующим правилам: каждый элемент манипулятора за один такт имеет возможность переместиться на одну клетку, передвижение начального/конечного элемента влево/вправо одновременно запрещается[29].

Проектируемая сеть Петри (рисунок 2.7, 2.8) перемещения однозвенного манипулятора заданной длины состоит из 16 мест P, переходов T и начальной маркировки m_I:

$$P = \{Final\ \ postion,\ Structure\ of\ \ the\ labirinth\ \ for\ link\ 2,\ After\ inspection,$$
$$Structure\ of\ \ the\ labirinth\ \ for\ link\ 1,\ Robot,\ Left1,\ Left2,\ Left3,\ Right1,$$
$$Right2,\ Right3,\ Movement,\ Data,\ Counter,\ a3,\ Inspection\};$$
$$T = \{Contrast,\ Final\ \ position,\ Movement,\ Inspection,\ InMovement,\ t2$$
$$Contrast\ \ d3\ to\ labirinth,\ Contrast\ \ d4\ to\ labirinth\};$$
$$m_I = (0,\ 81,\ 0,\ 81,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0).$$

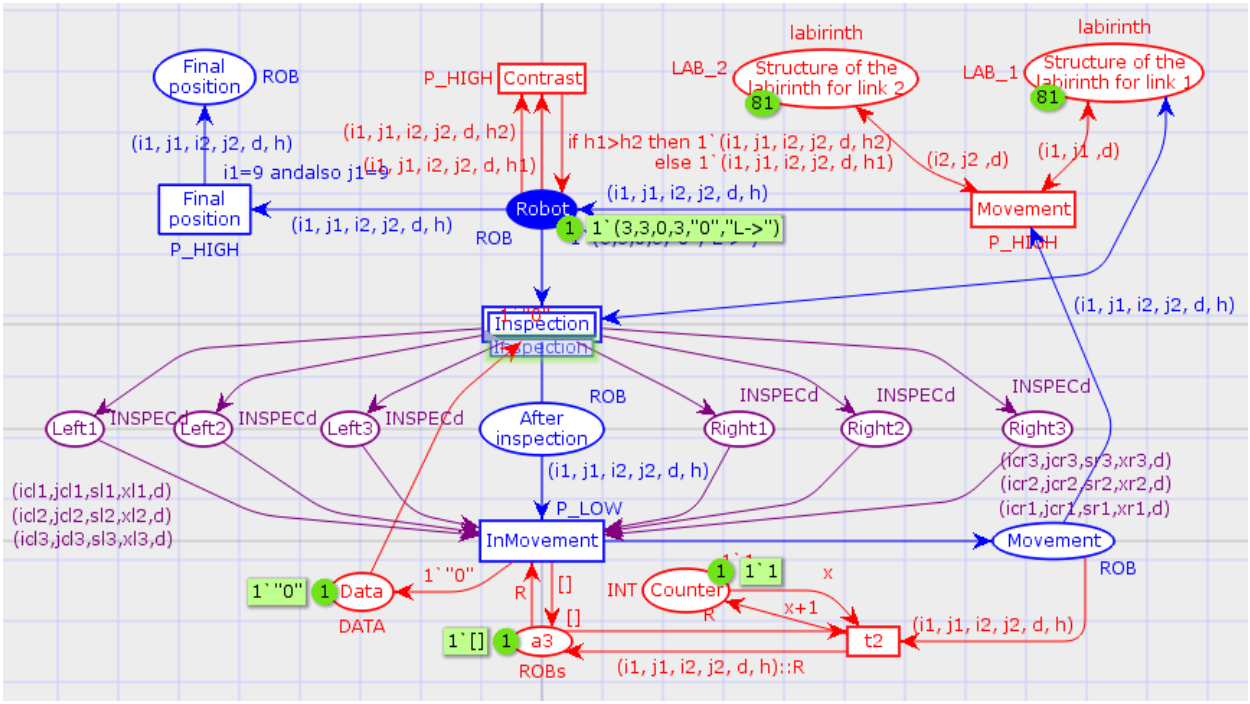


Рисунок 2.7 – Сеть Петри перемещения манипулятора
в замкнутом пространстве

Место *Robot* содержит одну метку с множеством типов данных (рисунок 2.7). Первые два типа данных ($i1$, $j1$) показывают координаты начального элемен-

[29] длина конструкции меняется в пределах 0.2 части клетки, изменения положения звена разрешено только вверх, вправо, влево, вниз, движение по наклонной, например, перемещение начального элемента влево и конечного элементов вверх, запрещено.

та, третий и четвёртый (*i*2, *j*2) – местоположение конечного элемента манипулятора, пятый (*d*) необходим для движения по лабиринту, а в последнем (*h*) хранится вся история движения робота.

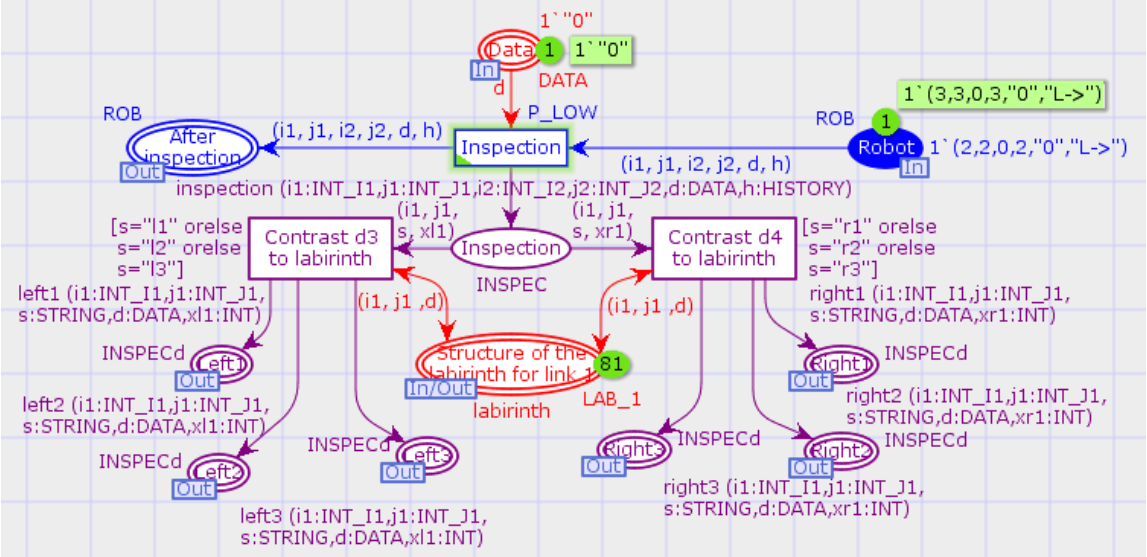


Рисунок 2.8 – Подсеть сети Петри
перемещения манипулятора в замкнутом пространстве

Первой процедурой является проверка возможных передвижений манипулятором из места *Robot* при срабатывании перехода *Inspection*. Опытным путём стало известно, что для манипулятора длиной в четыре клетки нужно проверять тип данных шести координат возможного перемещения, которые отображаются в местах *Left1*, *Left2*, *Left3*, *Right1*, *Right2*, *Right3*. После проверяется, попали ли начальный и конечный элементы манипулятора на поле с препятствием. При нахождении манипулятора на свободных координатах происходит его передвижение по сети. Если происходит наложение на препятствия лабиринта, то данная копия манипулятора удаляется с использованием перехода *t*2 и места *a*3. После каждого перемещения (срабатывание перехода *Movement*) проверяется: нашел ли манипулятор заданное поле в лабиринте (возможность срабатывания перехода *Final position*). Также возможно присутствие в системе звеньев с одинаковыми координатами начального и конечного элементов, что означает возможность нахождения одного поля в лабиринте разными путями.

Во время проектирования сети использовался встроенный в пакет CPN Tools язык *ML*. Для удобства все основные условия, а также массив данных, через который была задана структура лабиринта, создавались в текстовом редакторе с расширением *“.txt”*. Представление лабиринта задано массивом, состоящим из нескольких типов данных (*i*, *j*, *d*). Тип данных *i* является горизонтальной ко-

ординатой, *j* – вертикальной координатной, *d* показывает, каким полем является данная координата: свободным пространством (0) или препятствием (1).

Анализ пространства состояний полученной сети осуществлялся через автоматическую генерацию отчета о пространстве состояний, которое вычислено частично и содержит 21 687 узлов и 56 643 дуги, в сети отсутствуют мёртвые переходы. При усложнении конструкции ограниченного пространства количество состояний значительно возрастает и превышает 10^6, что на порядок превышает количество состояний, анализируемых в работах [53, 93] при использовании предложенных методик. Программное моделирование, с использованием генерации и исследования пространства состояний, и ручное моделирование показали соблюдение условий корректной работы алгоритма – огибание роботом препятствий. При необходимости манипулятор делится на доступное количество копий, движение которых по лабиринту в полной мере демонстрирует свойство параллелизма сетей Петри.

Таким образом, предложен способ проектирования систем со сложной структурой на примере задачи перемещения манипулятора до определённого поля в ограниченном пространстве с препятствиями на основе сетей Петри с нагруженными метками. Сложной структурой в данном случае является ограниченное пространство (лабиринт), которое моделируется с использованием набора цветных меток в одном из мест сети. Манипулятор в данном примере представляет собой конструкцию в виде звена определённой длины. В нагруженных метках хранится информация о текущих координатах, а также история о движения манипулятора в замкнутом пространстве. Данный способ позволяет изменять структуру системы без изменения структуры сети, а информация, хранимая в метках, может изменяться в зависимости от изменения логики функционирования самой системы.

2.2. Сети Петри: реализация рекурсивных функций

При создании программных продуктов не редко используют ***рекурсивные функции***[30] **(Н. Вирт** [6]). В данном разделе отрабатывается возможность исполь-

[30] *Рекурсия* – это процесс повторения элементов самоподобным образом. В программировании рекурсией называют вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия).

зования рекурсивных функций в сетях Петри (S. Haddad [113], И.А. Ломазова [57])[31] на примерах переноса упорядоченного массива с ограничением выборки по одному элементу [31], нахождения выбранного числа из ряда Фибоначчи и вычисления факториала числа [73].

Пример рекурсивных функций моделируемых в сетях Петри разберем на задаче переноса упорядоченного массива с ограничением выборки по одному элементу на примере “Ханойская башня”. Даны три стержня, на один из которых помещено определенное количество дисков, причем диски отличаются размером и лежат строго меньший на большем. Необходимо перенести пирамиду из определенного количества дисков с одного стержня на другой. Разрешено за один раз перенести только один диск, также нельзя класть больший диск на меньший.

Для реализации задачи ханойская башня, воспользуемся цветной сетью Петри, представленной на рисунке 2.9. Места *Tower n* символизируют башни, а метки – диски. Тип данных каждой метки представляет список из целочисленных значений, определяющий количество колец на башне. Например, место *Tower 1* (в задаче первая башня) содержит следующую метку 1`[1, 2, 3][32]. Элемент списка каждой метки символизирует радиус диска, метка с типом данных *“0”* указывает на отсутствие дисков на стержне. При срабатывании переходов первый элемент списка изымается из места и добавляется в начало списка соответствующего места.

Изначально диски находятся на башне *Tower I*. После первого перемещения (срабатывание перехода *From I to II* или *From I to III*) на башне *Tower II/Tower III*

---

[31] Для реализации рекурсии используются вложенные сети Петри $NPN = \left(Atom, Lab, SN, \left(EN_1, ..., EN_k\right), \Lambda\right)$, где $Atom = Var \cup Con$ – множество меток имен переменных и имен констант, $Lab = Lab_v \cup Lab_h$ – множество меток, использующихся для вертикальной и горизонтальной синхронизации срабатывания переходов, $\left(EN_1, ..., EN_k\right)\left(k \geq 1\right)$ – конечный набор обыкновенных (элементных в *NPN*) сетей Петри, Λ – частичная функция пометки переходов, *SN* – сеть высокого уровня, которую представляют набором: $SN = \left(N, L, U, W, M_0\right)$, где $N = \left(P, T, F\right)$ – сеть, $L = Expr\left(Atom\right)$ – язык выражений для представления переменных и констант, *U* – модель языка *L*, *W* – функция, сопоставляющая каждой дуге некоторое выражение, M_0 – начальная разметка сети.

[32] При анализе сети посредством исследования пространства состояний и ручном моделировании выявлена ошибка: множество целочисленных значений может оказаться пустым, так как все элементы из этого множества будут изъяты, следовательно, метка должна содержать следующий тип данных: 1`[1, 2, 3, 0]

появляется диск с радиусом 1. Для каждого из переходов *From n to n* добавляется условие, которое не позволяет переместить диск с большим радиусом на меньший. Для информации, хранящейся в метке, перемещение дисков соответствует добавлению/удалению количества целочисленных значений в списке. При смене начальных условий (нахождение дисков на другом стержне), работа сети выполняется в том же режиме. Места *Tower I* и *Tower II*, выделенные свечением, заменяют соответствующие элементы в сети.

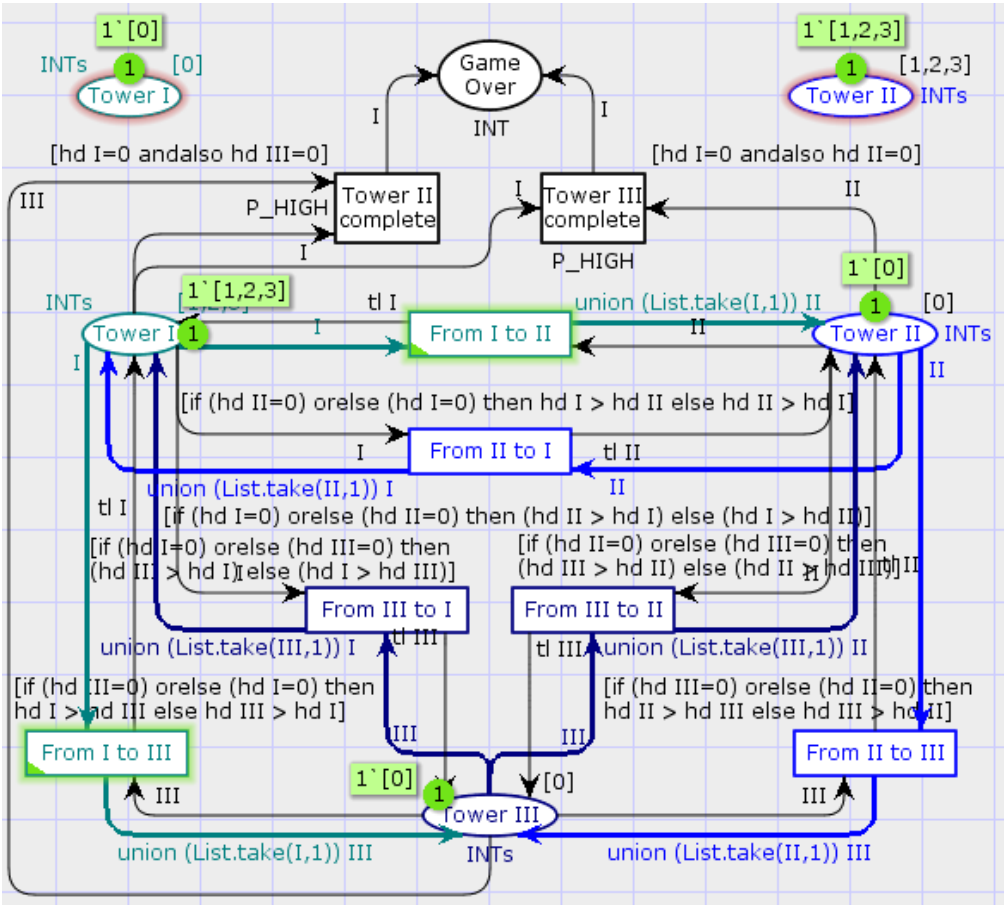


Рисунок 2.9 – Реализация рекурсивной функции – перемещение упорядоченного массива с ограничением выборки по одному элементу на основе сетей Петри

Анализ пространства состояний полученной сети осуществлялся через автоматическую генерацию отчета, который показал, что в сети нет тупиковых состояний, а граф состояний состоит из 27 узлов и 78 дуг (рисунок 2.10). При анализе сети для задачи из 6 дисков и 8 дисков, пространство состояний содержит 731 узел и 2 182 дуг, 6 563 узла и 19 678 дуг, соответственно.

Продемонстрируем выполнение рекурсивных функций с использованием сетей Петри не примере нахождения **факториала числа**, нахождения значения

заданного **числа Фибоначчи**. Числа Фибоначчи – элементы числовой последовательности, в которой каждое последующее число равно сумме двух предыдущих чисел. Пример: *0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …* . Факториал числа X – произведение всех натуральных чисел от одного до X включительно: $X!=1\cdot 2\cdot \ldots \cdot X=\prod_{i=1}^{X} i$, например, $5!=5\cdot 4\cdot 3\cdot 2\cdot 1=120$.

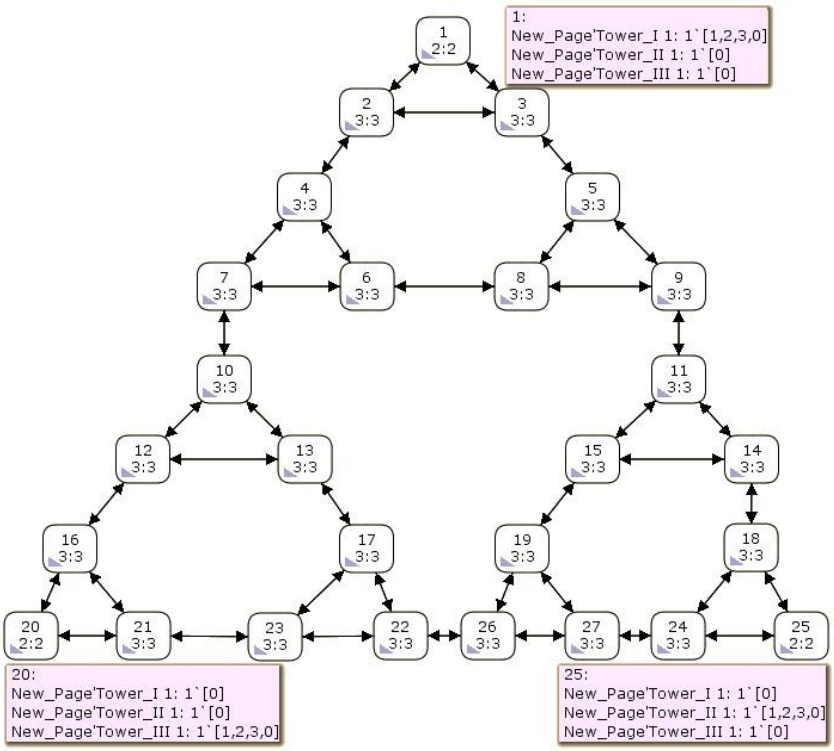


Рисунок 2.10 – Граф состояний сети Петри,
представленной на рисунке 2.9

На рисунке 2.11 представлена сеть Петри для рекурсивной задачи нахождения заданного числа ряда Фибоначчи, в данном случае 20-го числа из ряда.

Место *x_i* содержит фишку, в которой представлены два типа данных: первый – это значение *n*-го числа ряда, второй – номер числа, а место *x_i1* значение предыдущего числа ряда Фибоначчи. Перед выполнением рекурсивной процедуры происходит проверка нахождения нужного числа с помощью перехода *Complete1* и *Complete2*. Рекурсивную функцию выполняет переход *Recursion*, который также пополняет список полученных значений в месте *X i-2*.

На рисунке 2.11 присутствуют места *Complete, x_i, x_i1* и *X i-2*, которые отмечены небольшим свечением по контуру. Данные места соответствуют местам сети решенной задачи, остальные элементы при найденном решении остаются неизменными. Место *Complete* содержит фишку, в которой представлены два ти-

па данных. Первый – это значение *n*-го числа ряда Фибоначчи, а второй указывает на номер числа Фибоначчи. Список в месте *X i-2*, содержит все значения чисел от 1 до 20 числа ряда Фибоначчи.

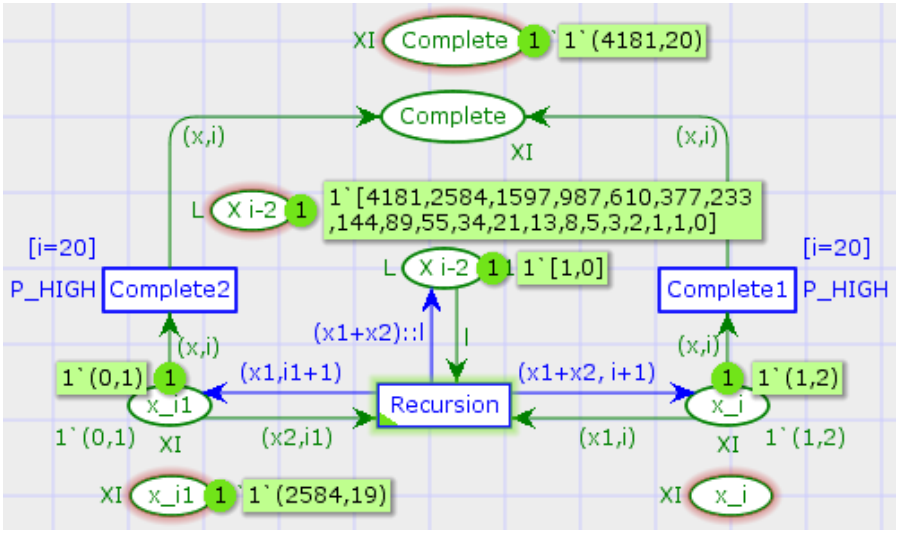


Рисунок 2.11 – Рекурсивная функция вычисления чисел Фибоначчи

На рисунке 2.12 представлена сеть Петри для рекурсивной задачи нахождения факториала числа. Место *Number* содержит метку, в которой показано искомое число факториала и произведение предыдущих двух чисел. Место *List* предназначено для хранения значений произведения, получаемых в процессе нахождения факториала числа. Места *Complete*, *Number* и *List*, отмеченные свечением, являются местами сети с найденным решением задачи. Место *List* содержит список значений произведений, получаемых при нахождении факториала двенадцати.

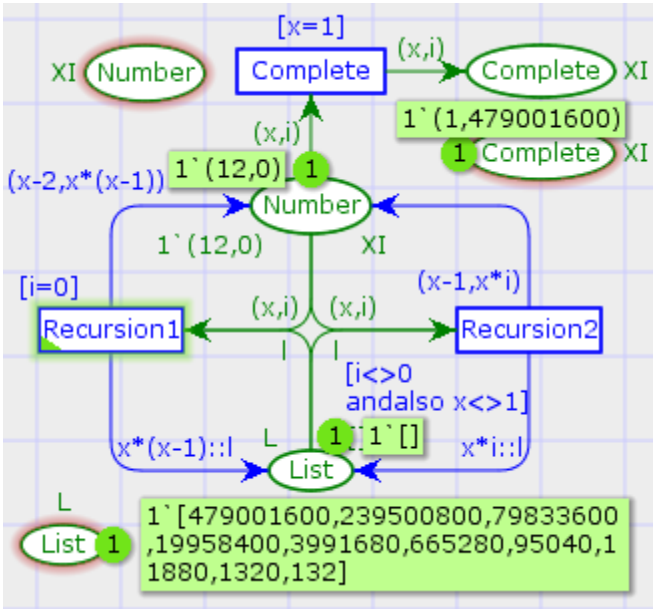


Рисунок 2.12 – Рекурсивная функция вычисления факториала

Таким образом, рекурсивные функции реализуются с использованием сетей Петри. Посредством программной среды визуального моделирования CPN Tools, показано выполнение рекурсии на задачах ханойская башня, нахождения n-го числа ряда Фибоначчи и нахождения факториала числа *X*.

2.3. Автоматическая трансляция UML диаграмм в сети Петри

В разделе предлагается ***способ автоматической трансляции UML*** диаграммы активности ***в сеть Петри***, посредством форматов подобной структуры у сохраняемых диаграмм и сетей. В работе [53] основным результатом является методика совместного использования UML диаграмм и сетей Петри в процессе разработки ПО [18, 104, 111, 117], а также правила взаимного преобразования сетей и UML диаграмм. В работе [93] предложенный ранее набор правил был расширен, а методика была модифицирована с целью возможности использования для более широкого круга задач. Но стоит отметить, что в обеих работах отсутствует формализация правил по преобразованию UML диаграмм в сети Петри и дается упоминание о возможности автоматической трансляции UML диаграмм в сети, описание которой приводится в [40]. В данном разделе формализация правил выполняется в виде алгоритма, написанном на псевдокоде [147]. Разрабатывается способ для реализации автоматической трансляции диаграммы активности в сеть Петри [124, 149, 150], который основан на формализации правил, представленных в графическом виде, и выполнении требований при проектировании диаграммы активности и позволяет сократить время на ручное преобразование диаграмм. Инструментом для автоматического преобразования диаграммы активности в сети выберем преобразование двух форматов подобной структуры *.xmi*[33] (Magic Draw) и *.cpn*[34] (CPN Tools).

Описание правил преобразования UML диаграмм в сети Петри. Рассмотрим наиболее часто встречающиеся правила преобразования UML диаграмм активности в сети Петри [53, 136, 144] и предложим алгоритмическую форму представления данных правил: состояние ожидания трансформируется в место, а состояние действия преобразуется в переход, который начинает действие (алгоритм 2.1); разделение и слияния потоков управления в UML диаграмме преобразуется в соответствующую сеть Петри (алгоритм 2.2); ветвление на диаграмме активности преобразуется

---

[33] Object Management Group был предложен формат *.xmi*, в котором хранится сжатая, но полная информация, как о создаваемых проектах, так и об отдельных диаграммах. Данный формат имеет синтаксис языка *xml*, возможность сохранить в котором присутствует не у всех пакетов визуального моделирования.

[34] Сохранённые проекты CPN Tools имеют формат *.cpn*, для записи данных используется синтаксис языка *xml.*

в эквивалент сети Петри, представленный в алгоритме 2.3; условия ограничения показываются с помощью условий ограничения переходов (алгоритм 2.3).

Алгоритм 2.1 – Преобразование последовательности состояний из UML диаграммы в сеть Петри [80]

1: $SS := SP$ // SS – начальное состояние на диаграмме активности, SP – начальное место в сети Петри,

2: $ES := EP$ // ES – конечное состояние на диаграмме активности, EP – конечное место в сети Петри,

3: $UST := PST$ // UST – начальный переход на диаграмме активности, PST – начальный переход в сети Петри,

4: $UET := PET$ // UET – конечный переход на диаграмме активности, PET – конечный переход в сети Петри,

5: $A := \{ AS_i \}$ // переменная A – множество всех состояний действия на диаграмме активности,

6: $T := \{ UT_i \}$ // переменная T – множество всех переходов системы на диаграмме активности,

7: *while* $A \neq \emptyset$ *do* // пока A не равна пустому множеству, выполняем …

8: *select an* $AS_i \in A$ // выборку всех состояний действия,

9: $AS_i := PP_i$ // трансформируем каждое состояние действия диаграммы активности в эквивалентное место сети Петри,

10: *return* PP_i // возвращаем полученное место сети Петри,

11: *end while* // конец цикла,

12: *while* $T \neq \emptyset$ *do* // пока T не равна пустому множеству, выполняем …

13: *select an* $UT_i \in T$ // выборку всех переходов диаграммы активности,

14: $UT_i := PT_i$ // и приравниваем их к соответствующим переходам сети Петри,

15: *return* PT_i // возвращаем полученный переход сети Петри,

16: *end while* // конец цикла.

Алгоритм 2.2 – Преобразование элементов разделения и слияния из UML диаграммы в сеть Петри [80]

1: $A := \{ AS_i \}$ // переменная A – множество всех состояний действия на диаграмме активности,

2: *while* $A \neq \emptyset$ *do* // пока A не равна пустому множеству, выполняем …

3: *select an* $AS_i \in A$ // выборку всех состояний действия,

4: $AS_i := PP_i$ // трансформируем каждое состояние действия диаграммы активности в эквивалентное место сети Петри,

5: *end while* // конец цикла,

6: $UF := PF$ // переход UF – разветвление на диаграмме активности, переход PF – разветвление в сети Пети,

7: $UJ := PJ$ // переход UJ – слияние на диаграмме активности, переход PJ – слияние в сети Пети.

Алгоритм 2.3 – Преобразование элементов ветвления и условий из UML диаграммы в сеть Петри [80]

1: $S := (A, T)$,

2: $A := \{ AS_i \}$ // переменная A – множество всех состояний действия на диаграмме активности,

3: $T := \{ UT_i \}$ // переменная T – множество всех переходов системы на диаграмме активности,

4: *while* $A \neq \emptyset$ *and* $T \neq \emptyset$ *do* // пока A и T не равны пустому множеству, выполняем …

5: *if* $(AS_i, UT_i, UJun) \in S$ *then* // условие: если существует множество, состоящее из последовательной цепочки: состояние действия, переход, блок условия, тогда,

6: $A := A \setminus \{ AS_i \}$ *and* $T := T \setminus \{ UT_i \}$ *and* // из множества A и T удаляем соответствующие элементы и …

7: $(AS_i, UT_i, UJun) := PJun$ // трансформируем последовательную цепочку в эквивалентное место в сети Петри,

8: *else select an* $AS_i \in A$ // иначе делаем выборку всех состояний действия,

9: $AS_i := PP_i$ // трансформируем каждое состояние действия диаграммы активности в эквивалентное место сети Петри,

10: *select an* $UT_i \in T$ // делаем выборку всех переходов диаграммы активности,

11: $(UT_i \mid condition) := \left(PT_{(i-1)} \mid condition\right)$ // добавляем условия к переходам сети Петри, которые являются выходящими для $PJun$,

12: *end while* // конец цикла.

Представление правил в алгоритмическом виде способствует их формализации. Следующим этапом развития способа совместного использования UML диаграмм и сетей Петри является разработка способа автоматического преобразования.

Описание и сравнение форматов *.xmi* и *.cpn*, требования к проектированию диаграммы активности. Автоматическое преобразование UML диаграмм и сетей Петри происходит с использованием правил [53], на основе преобразования структуры файлов *.xmi* и *.cpn* и с соблюдением корректности построения имени элементов диаграмм и сетей. Для чего предлагается логика нахождения и преобразования начального места, конечного места, обычного места, обычного перехода в структуре файла диаграммы активности и построенной на основе её сети Петри и псевдосостояний: раз-

ветвлений, слияний, условий. Даются рекомендации по именованию состояний и переходов для диаграммы активности, поскольку, в сетях Петри встречаются свойства мест: начальная маркировка, тип данных, и переходов: условие срабатывания перехода, которые в языке UML не предусмотрены, но для сетей являются обязательными.

При проектировании диаграммы активности её структура имеет три вида элементов: состояния, псевдосостояния (начало, условие, слияние и разветвление), переходы (взаимосвязи состояний), для каждого элемента определяются входные и выходные элементы. При создании сети Петри в программной среде CPN Tools используется три вида элементов: места, переходы и дуги. Каждому элементу присваивается индивидуальный *ID*, а взаимосвязь вершин происходит через описание дуг. Как таковых псевдосостояний CPN Tools не предусматривает – их заменяют различные вариации связей мест и переходов.

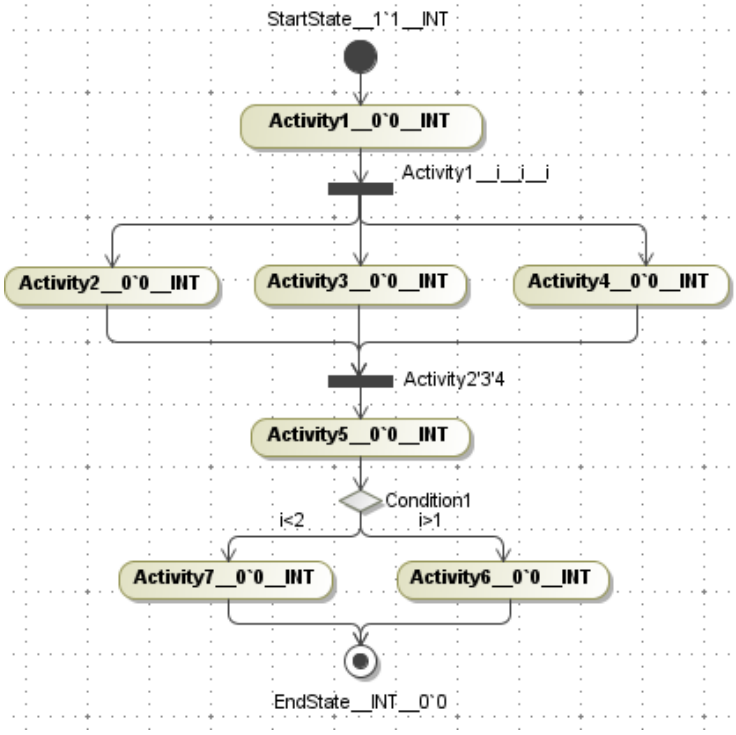


Рисунок 2.13 – Диаграмма активности, содержащая последовательные элементы, элементы разделения, слияния и условия

Построим диаграмму активности (рисунок 2.13), используя правила, приведенные выше. Данную диаграмму трансформируем в эквивалентную сеть Петри для тщательной проработки правил и рекомендаций по преобразованию UML диаграмм в сети Петри. Начальное состояние в UML у формата *.xmi* является псевдосостоянием (в контексте синтаксиса *.xmi*) с именем *StartState__1`1__INT*. В данном случае имя содержит больше информации о начальной маркировке этого состояния и о типе данных в этом состоянии. Чаще всего дуга в UML является переходом в CPN Tools. В данном примере она имеет имя *Start__i__i__i*, в котором представлена информация о названии, имени для входной и выходной дуг и условии для перехода CPN Tools, соответственно. Обычное состоя-

ние, например *Activity1*, имеет имя схожее с именем начального состояния *Activity1__0`0__INT*. В нём также присутствует название, начальная маркировка и тип данных. Дуги с именем *empty__i__i*, добавляются на диаграмму активности перед и после псевдосостояний со свойством формата *.xmi kind*: *fork*, *join*, а также только перед дугой со свойством *kind*: *junction*. Разветвление с именем *Activity1__i__i__i* на рисунке 2.13 является псевдосостоянием, а в формате *.xmi* имеет дополнительное свойство *kind* с атрибутом *fork*, и представляется в CPN Tools как переход. Первая часть имени – название, вторая – имя входящей дуги, третья – имя выходящей дуги. Псевдосостояние со свойством *kind* и атрибутом этого свойства *junction* имеет имя *Condition1*. В CPN Tools, эту роль играет место *Activity5* и переходы, образующиеся с использованием выходящих дуг из пвсевдосостояния *Condition1*. Выходящая дуга из псевдосостояния *Condition1* имеет имя *Activity5_1__i__i__i>1*, где первая часть – название перехода в CPN Tools, последующие два элемента – название входной и выходной дуги, соответственно, а четвёртый элемент отвечает за возможное условие, предъявляемое к срабатыванию перехода. Конечное состояние на диаграмме активности имеет имя *EndState__0`0__INT* и является обычным состоянием системы.

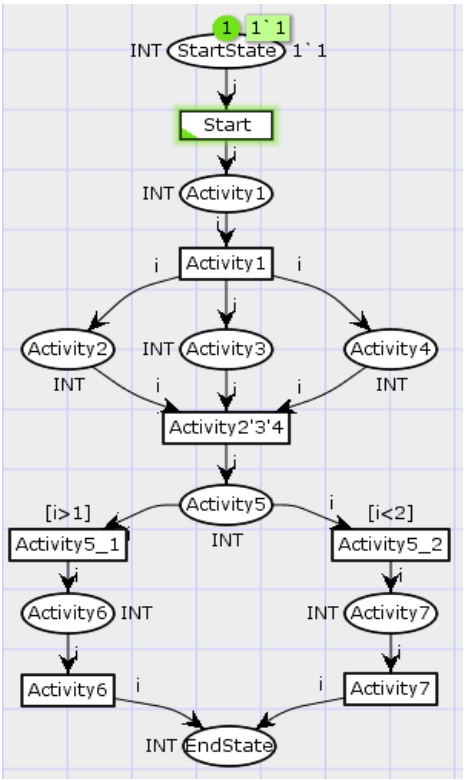


Рисунок 2.14 – Сеть Петри для диаграммы активности

Для сравнения двух форматов *.xmi* и *.cpn* преобразуем данную диаграмму в сеть Петри (рисунок 2.14). Свойства построенной сети: *colset INT = int; var i : INT.*

В CPN Tools у места присутствует имя, тип данных, который возможен в этом месте и начальная разметка места. В переходе заложена информация об имени и возможном условии, а в дуге – об ориентации, начальной и конечной вершин: либо места и перехода, либо перехода и места, в зависимости от свойств атрибута *orientation*. Так как представленная диаграмма и сеть Петри являются аналогами друг друга, необходимо сравнить структуру сохранённых файлов и выделить схожие части.

Сравнение начального места форматов

***.xmi* и *.cpn*.** Первым элементом на диаграмме активности (рисунок 2.13) и сети Петри (рисунок 2.14) является начальное место. Структуры остальных элементов: конечное место, обычное место, обычный переход, дуга сети Петри, приведены в приложении В. Структура начального места формата *.cpn*:

```
1.<placeid="ID1420720921">
2.      <posattr x="-107.000000" y="167.000000"/>
3.      <fillattrcolour="White" pattern="" filled="false"/>
4.      <lineattrcolour="Black" thick="1" type="Solid"/>
5.      <textattrcolour="Black" bold="false"/>
6.      <text>StartState</text>
7.      <ellipse w="76.000000" h="40.000000"/>
8.      <token x="-94.000000" y="0.000000"/>
9.      <marking x="-27.000000" y="0.000000" hidden="false">
10.         <snap snap_id="4" anchor.horizontal="2" anchor.vertical="3"/>
11.     </marking>
12.     <typeid="ID1420720922">
13.         <posattr x="-76.500000" y="136.000000"/>
14.         <fillattrcolour="White" pattern="Solid" filled="false"/>
15.         <lineattrcolour="Black" thick="0" type="Solid"/>
16.         <textattrcolour="Black" bold="false"/>
17.         <text tool="CPN Tools" version="3.4.0">INT</text>
18.     </type>
19.     <initmarkid="ID1420720923">
20.         <posattr x="-53.000000" y="167.000000"/>
21.         <fillattrcolour="White" pattern="Solid" filled="false"/>
22.         <lineattrcolour="Black" thick="0" type="Solid"/>
23.         <textattrcolour="Black" bold="false"/>
24.         <text tool="CPN Tools" version="3.4.0">1`1</text>
25.     </initmark>
26.</place>.
```

Структура начального места формата *.xmi*:

```
1.<UML:Pseudostate xmi.id = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000869'
2.  name = 'StartState__1`1__INT' isSpecification = 'false' kind = 'initial'>
3.<UML:StateVertex.outgoing>
4.          <UML:Transitionxmi.idref = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000874'/>
```

5.</UML:StateVertex.outgoing>
6.</UML:Pseudostate>.

У формата *.xmi* описание начального состояния более краткое по сравнению с описанием начального места формата *.cpn*, тем не менее, данный факт не сказывается на информативности.

ID места (строка 1 формата *.cpn*), *ID* типа данных места (строка 12 формата *.cpn*), *ID* маркировки места (строка 19 формата *.cpn*) являются уникальными в проекте. В строке №6 формата *.cpn* задаётся имя начального места, которое соответствует имени в названии начального состояния у формата *.xmi*. В строке 17 формата *.cpn* задаётся тип данных, соответствующий второй части в названии начального состояния у формата *.xmi*. Строка 24 формата *.cpn* содержит информацию об исходной разметке начального состояния, а у формата *.xmi* это третья часть в названии. Во второй строчке формата *.xmi* помимо названия содержится свойство *kind* с атрибутом *initial*. Это означает, что данное псевдосостояние является начальным. В строке 3 формата *.xmi* говорится о существовании выходящей дуги, направленной из этого состояния, а ниже указывается её *ID*, который уникален в проекте.

Таким образом, автоматическое преобразование диаграммы активности в сеть Петри рекомендовано выполнять через подобную структуру двух форматов *.xmi* и *.cpn*, соответственно. Сходством этих форматов является наличие мест и переходов у сетей Петри и диаграмм активности, а основным отличием – наличие у формата *.cpn* тега *<arc>*, который отвечает за связь вершин сети, а у формата *.xmi* данный тег отсутствует, но информация о взаимосвязях состояний на диаграмме активности заложена в тегах *<UML:…State>* и *<UML: Transition>*. Представлены основные признаки двух форматов: имя элемента, тип данных, присущих данному элементу и маркировка каждого элемента. Предложено формальное представление основных правил преобразования (действие, выполнение условия, разделение/слияние) в алгоритмическом виде.

2.4. Методика проектирования программного обеспечения с использованием UML диаграмм и сетей Петри

В данном разделе, опираясь на предыдущие работы (Коротиков [53], Романников [93]), предлагается методика проектирования программного обеспечения,

основанная на совместном использовании UML диаграмм и сетей Петри. Приводится её алгоритмический вид[35] и графическое представление в виде диаграммы активности.

В работах [50, 91, 94, 102, 119] используются методы совместного использования UML диаграмм и сетей Петри для создания ПО. Опираясь на анализ способов проектирования при совместном использовании UML диаграмм и сетей Петри, приведём усовершенствованную методику [34]. Напомним, что в работе [53] предлагается методика совместного использования UML диаграмм и сетей Петри, которая выполняется в два этапа: описание структуры системы и её поведение. Помимо автоматического анализа сетей Петри посредством генерации и исследования пространства состояний выполняется ручное моделирование. Также данная методика специализирована для разработки ПО центров дистанционного контроля и управления. В работе [93] методика, упомянутая выше, получила развитие: методика выполняется в один этап, диаграмма состояний используется для детального рассмотрения диаграммы классов, а детальное рассмотрение диаграммы объектов опускается. Отметим, что данный вариант методики подходит для систем локальной автоматики.

Предлагается **методика проектирования** программного обеспечения на основе анализа, представленного в разделе 1.4, которая выполняется за один этап, а для изучений состояний, в которых может оказаться система используется пространство состояний анализируемой сети Петри. Методика применима для более широкого круга задач (разработка ПО для центров дистанционного контроля и управления, для систем локальной автоматики, для задач связанных с рекурсивными функциями и т.д.) и состоит из семи шагов:

1. Составление диаграммы прецедентов с выявлением элементов системы в виде акторов и вариантов использования.
2. Составление диаграммы классов и выделение методов класса.
3. На основании диаграммы классов выделяются объекты, которые участвуют в работе системы и помогают определить начальные значения для маркировки всех вершин сети Петри.

[35] Алгоритм представлен в виде псевдокода [147].

4. Построение диаграммы активности для описания динамических свойств системы.
5. Трансляция построенной диаграммы активности в сеть Петри на основе формальных правил преобразования.
6. Анализ свойств с использованием автоматизированных программных пакетов для работы с сетями Петри, позволяющих выявить логические ошибки при моделировании диаграммы активности: нахождение неиспользуемых сценариев работы, выявление тупиковых веток алгоритмов и т.п.

6.1 Для классов или объектов, нуждающихся в детальной проработки их логики, следует составить отдельную диаграмму активности и проанализировать её при помощи сетей Петри.

6.2 Для сценариев работы, требующих проверки, моделируются диаграммы последовательности с последующим преобразованием в сеть Петри для анализа на основе исследования пространства состояний.

6.3 При необходимости отследить динамику работы по состояниям системы, следует использовать сети Петри, при анализе которых производится построение и исследование пространства состояний.

7. В случае успешного анализа и наличии полного набора диаграмм, не нуждающихся в доработке, моделируемые диаграммы готовы для автоматической генерации кода.

Представим методику проектирования ПО с использованием UML диаграмм и сетей Петри ***в виде алгоритма*** на формализованном языке:

analysis { // в функции анализа выполняются следующие процедуры,
translation diagram to Petri net; // диаграмму активности транслируется в сеть Петри,
analysis Petri net // полученная сеть Петри анализируется,
if results = ¬*satisfactory*, *then*; // если результаты неудовлетворительны, то выполняем,
correct Activity Diagram; // выполняется корректировку диаграммы активности,
else; // иначе,
end if } // условие выполнено,
1: *do Use Case Diagram* // проектируем диаграмму прецедентов,
2: *do Class Diagram* // проектируем диаграмму классов,

3: *for needed class ∈ Class Diagram do Activity Diagram* // при необходимости для некоторых классов проектируем диаграмму активности,

4: *analysis* () *of Activity Diagram Classes* // для полученной диаграммы активности выполняем функцию *analysis,*

5: *do Object Diagram* // проектируем диаграмму объектов,

6: *for needed object ∈ Object Diagram do Activity Diagram* // при необходимости для некоторых объектов проектируем диаграмму активности,

7: *analysis*() *of Activiry Diagram Objects* // для полученной диаграммы активности выполняем функцию *analysis,*

8: *do Activity Diagram for system* // проектируем диаграмму активности,

9: *for needed scenario ∈ Activity Diagram do Sequence diagram* // при необходимости для некоторых сценариев, представленных на диаграмме активности, проектируем диаграмму последовательности,

10: *analysis*() *of Sequence diagram* // для полученной диаграммы последовательности выполняем функцию *analysis,*

11: *analysis*() *of Activity Diagram for system* // для полученной диаграммы активности системы выполняем функцию *analysis,*

12: *generation code* // происходит генерация кода.

Для наглядности ***представим*** данную ***методику в виде диаграммы активности*** (рисунок 2.15).

Таким образом, предложена усовершенствованная методика проектирования программного обеспечения, полученная при анализе существующих методик и способов совместного использования UML диаграмм и сетей Петри. Данная методика состоит из семи этапов, включающих использование необходимых диаграмм: диаграмма прецедентов, диаграмма классов, диаграмма объектов и диаграмма активности, диаграмма последовательности. Предлагаемая методика применима к задачам, связанным с разработкой программного обеспечения для центров дистанционного контроля и управления, для систем локальной автоматики, для задач связанных с рекурсивными функциями и т.д.

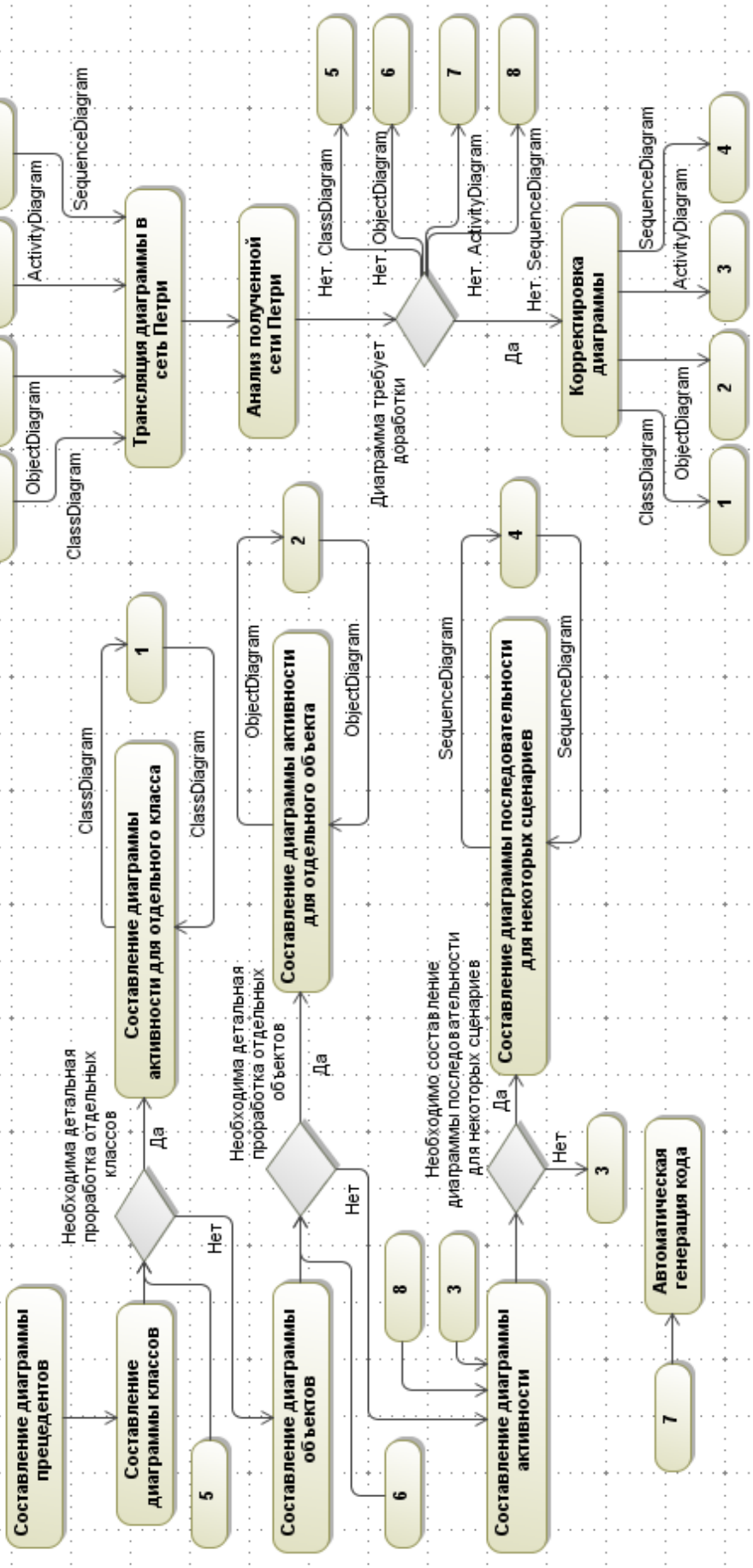


Рисунок 2.15 – Диаграмма активности предлагаемой методики проектирования ПО при совместном использовании UML диаграмм и сетей Петри

2.5. Компактное представление языков сетей Петри

На шестом этапе предлагаемой методики производится анализ сетей Петри, который возможен как стандартным способом (хранение исследованных состояний в оперативной памяти), так и альтернативными. Одним из интересных способов анализа сетей Петри видится представление и анализ *свободного языка сетей Петри* (множество всех последовательностей срабатываний переходов сети от начальной маркировки до всех достижимых разметок сети). В данном разделе предлагается ***компактное представление свободного языка*** сетей Петри [9, 14, 25] с целью анализа проектируемых сетей на примере управляемого светофора [61] и системы, моделирующей работу двухсимочного телефона [64].

Представим логику функционирования ***управляемого светофора*** с использованием свободного языка сетей Петри и его компактного представления, по которому можно отследить всевозможные цепочки срабатывания переходов, без деталей о маркировке сети. Система “Управляемый светофор” представляет собой сеть (рисунок 2.16) из двух мест P, шести переходов T и начальной маркировки m_I:

$$P = \{ State\ of\ \ traffic\ light\_1, TL\_2 \};$$
$$T = \{ State\ 1\_1, State\ 2\_2, State\ 3\_3, State\ 4\_4, State\ 5\_5, State\ 6\_6 \};$$
$$m_I = (1, 6).$$

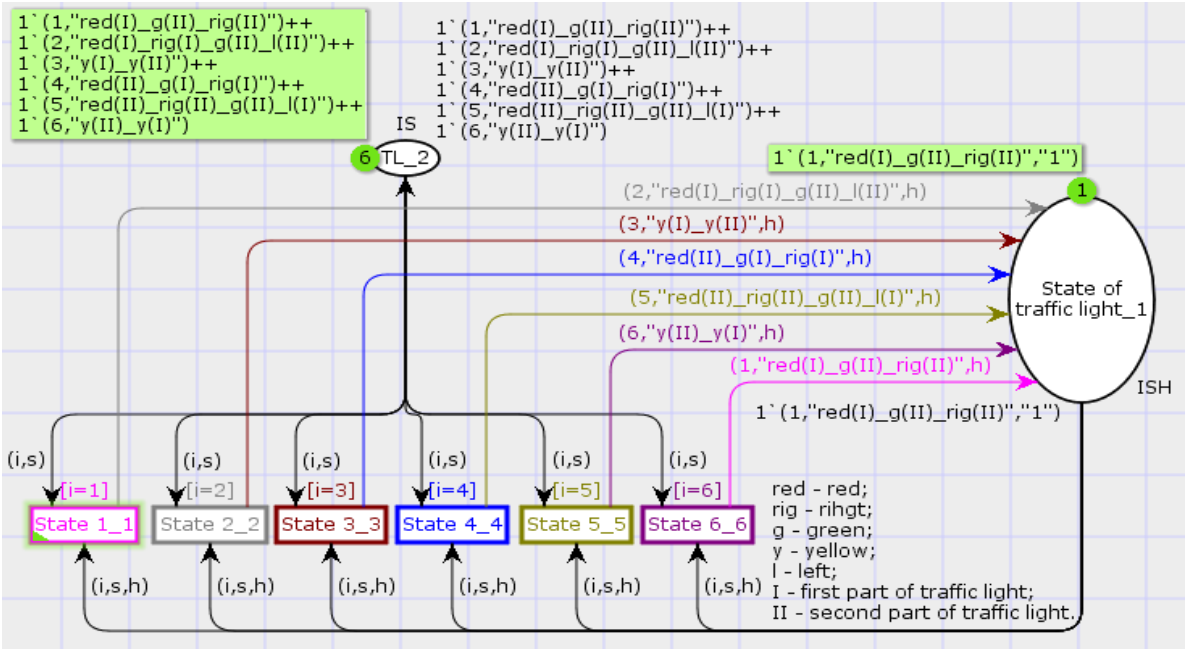


Рисунок 2.16 – Сеть Петри управляемого светофора

Срабатывание переходов происходит последовательно в зависимости от текущего состояния светофора. Место *State of traffic light_1* содержит метку, например, *1`(1,"red(I)_g(II)_rig(II)","1")*, при которой у светофора горит только красный свет *red(I)*. После каждого из срабатываний метки возвращаются в место *TL_2* и место *State of traffic light_1*, причем у последнего можно реализовать историю срабатывания переходов, используя переменную *h*.

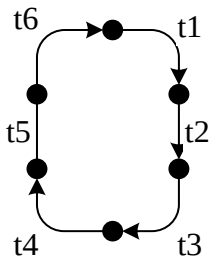


Рисунок 2.17 – Компактное представление свободного языка сети Петри, представленной на рисунке 2.16

Для компактного отображения свободного языка сетей Петри переименуем переходы сети в соответствии с их порядковым номером: $State\ n\_n = tn$. Свободный язык для данной сети выглядит как ряд, увеличивающий своё значение в арифметической прогрессии: *t1, t1t2, t1t2t3, t1t2t3t4, t1t2t3t4t5, t1t2t3t4t5t6, ...* . Представим данную структуру графически, более компактно (рисунок 2.17).

Система, моделирующая работу ***двухсимочного телефона*** (рисунок 2.18), состоит из четырёх мест, семи переходов и начальной маркировки m_I:

$$P = \{ Mobile\ phone\_1, Session\ of\ phone\_2, SMS\ archive\_3, Action\_4 \};$$
$$T = \{ Ringing\ or\ onternet\_1, Ringing\ to\ first\ SIM\_2, Ringing\ to\ second\ SIM\_3, Internet\ to\ first\ SIM\_4, Internet\ to\ second\ SIM\_5, SMS\ to\ first\ SIM\_6, SMS\ to\ second\ SIM\_7 \};$$
$$m_I = (6, 0, 0, 3).$$

В данной сети при срабатывании переходов, имитирующих работу одной из SIM карт, срабатывание переходов, имитирующих работу другой SIM карты невозможно, кроме переходов *SMS to first SIM_6* и *SMS to second SIM_7*. Используя переменную *h*, можно сохранять историю состояний телефона [64]. Для упрощения сети удалим место-счётчик *SMS archive_3*, что не влияет на выполнение основных сценариев системы. После данной операции сеть имеет разметку: $m_I = (6, 0, 3)$, а следующие взаимосвязи между вершинами будут отсутствовать: $\{ Mobile\ phone\_1 \}$ $[SMS\ to\ first\ SIM\_6 \vee \vee\ SMS\ to\ second\ SIM\_7 \rangle$ $\{Mobile\ phone\_1,\ SMS\ archive\_3\}$. Как и в предыдущем

случае, для компактного отображения свободного языка сетей Петри переименуем переходы сети в соответствии с их порядковыми номерами: … $SIM\_n = tn$.

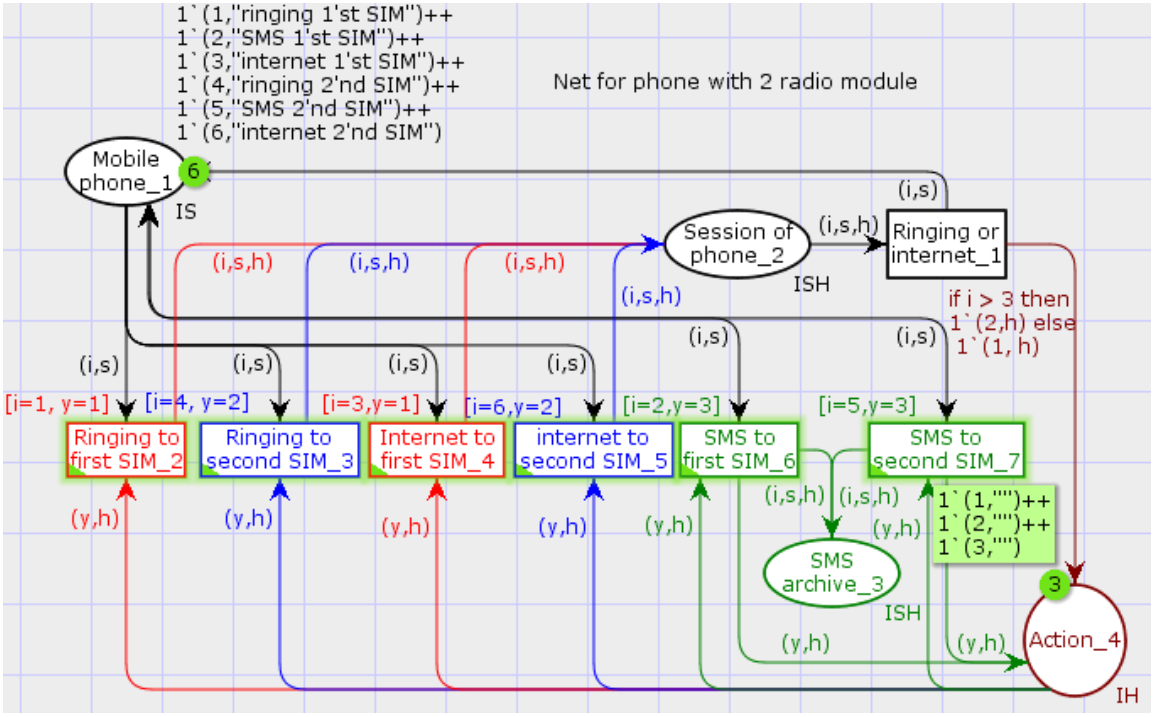


Рисунок 2.18 – Сеть Петри двухсимочного телефона

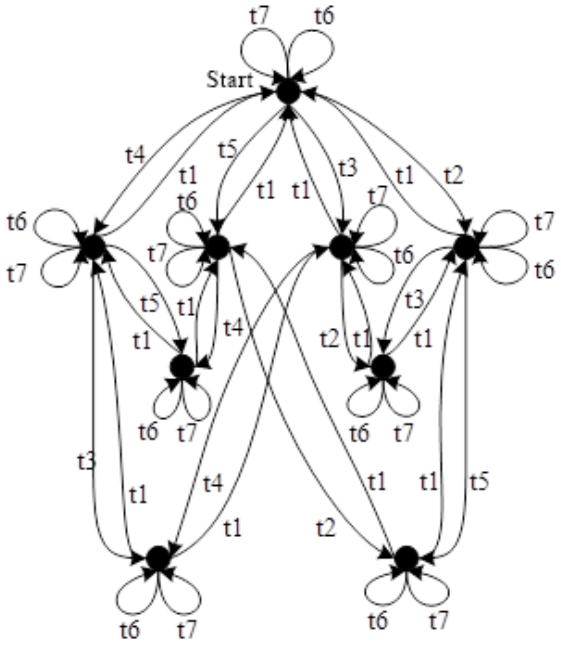


Рисунок 2.19 – Компактное представление свободного языка сети Петри, представленной на рисунке 2.18

Свободный язык для такой сети имеет вид: *t2, t3, t4, t5, t6, t7, t2t1, t2t6, t2t7, t3t1, t3t6, t3t7, t4t1, t4t6, t4t7, t5t1, t5t6, t5t7, t6t2, t6t3, t6t4, t6t5,...* . Поскольку представление языка данной сети весьма громоздко, представим свободный язык графически (рисунок 2.19). При компактном представлении не используются маркировка у вершин, а их взаимосвязи обозначены именами соответствующих переходов, при срабатывании которых происходит изменение состояний сети.

Таким образом, предлагается компактное представление свободного языка сетей Петри. Из приведенных при-

меров видно, что компактное представление свободного языка является сокращенным вариантом графа состояний сети, а именно в компактном представлении скрыта маркировка сети, что делает ее более удобной для восприятия свободного языка. Предложенное компактное представление языков может быть применено для анализа и использовано для синтеза сети Петри по заданному языку.

2.6. Выводы

В данной главе представлена методика совместного использования UML диаграмм и сетей Петри для анализа поведенческих UML диаграмм. В данной методике используется диаграмма прецедентов, диаграмма классов и диаграмма объектов, а также диаграмма активности и диаграмма последовательности, которые в последующем транслируются в сеть Петри для анализа. Отличием от более ранних работ [53, 93] является отказ от использования детализированной диаграммы прецедентов и диаграммы состояний, которую заменяет построение и исследование пространства состояний сети Петри.

Предложен способ автоматического преобразования UML диаграмм в сети Петри, реализуемый посредством двух форматов подобной структуры *.xmi* (UML) и *.cpn* (сети Петри). Для реализации данного способа были формализованы правила преобразования UML диаграмм в сети Петри и предложены рекомендации по построению UML диаграмм. Опираясь на анализ отдельных фрагментов файлов форматов, разработан способ преобразования элементов UML диаграмм и сетей Петри.

Предложен способ проектирование систем со сложной структурой на примере задачи перемещения манипулятора до определённого поля в ограниченном пространстве с препятствиями на основе сетей Петри с нагруженными метками. Нагруженные метки необходимы для хранения информации о текущих координатах манипулятора, а также истории об его движении по замкнутому пространству. Посредством программной среды визуального моделирования CPN Tools продемонстрировано выполнение рекурсии при переносе упорядоченного массива с ограничением выборки по одному элементу, нахождении n-го числа ряда Фибоначчи и нахождении факториала.

Одним из способов анализа является представление и анализ свободного языка сетей Петри. В общем виде представлен способ компактного представления свободного языка сетей Петри, по которому продолжаются исследования.

3. АНАЛИЗ СЕТЕЙ ПЕТРИ: ПРОСТРАНСТВО СОСТОЯНИЙ, ИНВЕРСИЯ, МАТРИЧНОЕ ПРЕДСТАВЛЕНИЕ

В предложенной ранее методике проектирования ПО выполнение первых пяти шагов при использовании наработок, представленных в диссертации, затруднений не вызывает. На следующем могут возникнуть сложности, связанные с анализом больших систем, у которых использование стандартного способа малоэффективно. Для решения данной проблемы предлагаются альтернативные способы анализа сетей Петри.

Рассматривается способ, основанный на построении сценариев работы системы критичных к ошибкам, которые преобразуются в сети Петри. Полученные сети анализируются при исследовании пространства состояний. В *п. 3.2* приводится способ, который заключается в представлении полученной на пятом этапе сети Петри в иерархическом виде. При разбиении на отдельные подсети они становятся отдельными сетями и анализируются независимо от основной.

В следующем разделе предлагается проверка полученной простой сети Петри, заключающийся в анализе отдельных фрагментов пространства состояний системы. Полученные результаты показали, что при анализе отдельных фрагментов пространства состояний присущие всей сети свойства сохраняются. Обязательным условием для данного способа является проверка достижимости выбранной маркировки, с которой начинается анализ. В *п. 3.4* демонстрируется способ проверки достижимости выбранной маркировки посредством инвертирования сети Петри и нахождении начальной маркировки сети на примере системы “Протокол передачи данных”. Инвертирование простых и ординарных сетей происходит с использованием предложенных правил. Также для проверки достижимости предлагается реализация инверсии графа состояний, преимуществом которой является однозначность преобразования.

Приводится анализ сетей Петри с использованием матриц на примере интернет магазина, и программное приложение, преобразующее сети Петри, проектируемые в пакете CPN Tools (version 3.4.0), в матрицы с их последующим анализом.

3.1. Анализ отдельных сценариев системы с использованием сетей Петри и пространств состояний

Предлагается способ, основанный на ***построении отдельных сценариев*** (один из вариантов выполнения программы) работы системы [74, 123]. В методике, предложенной в диссертационной работе [53], диаграммы последовательности используются для отображения взаимодействия классов, что демонстрируется на примере разработки ПО системы управления и контроля таксофонами [47, 51, 52]. В методике, предложенной в разделе 2.4, сценарии работы, требующие проверки, моделируют с использованием диаграмм последовательности с последующим преобразованием в сеть Петри для анализа на основе исследования пространства состояний. Покажем, как данный способ реализуется на примере сценариев с положительным исходом **взаимодействия пользователя с банкоматом** [45].

Опишем основные действия пользователя при работе с банкоматом. Первое, что требуется от пользователя – вставить карту (*Insert card*), далее ввести PIN-код (*Inject PIN*). При корректном вводе PIN-кода пользователю предлагается выполнить одну из следующих операций: просмотреть баланс счета (*Checking balance*), снятие наличных (*Withdrawal*), пополнение баланса на мобильном номере (*Replenishment mobile balance*). Представим логику работы в виде диаграммы активности (рисунок 3.1). Для проверки корректности построения и выявления логических ошибок полученную диаграмму активности транслируем в советующую сеть Петри (рисунок 3.2) по правилам, предложенных в п. 1.4. Для

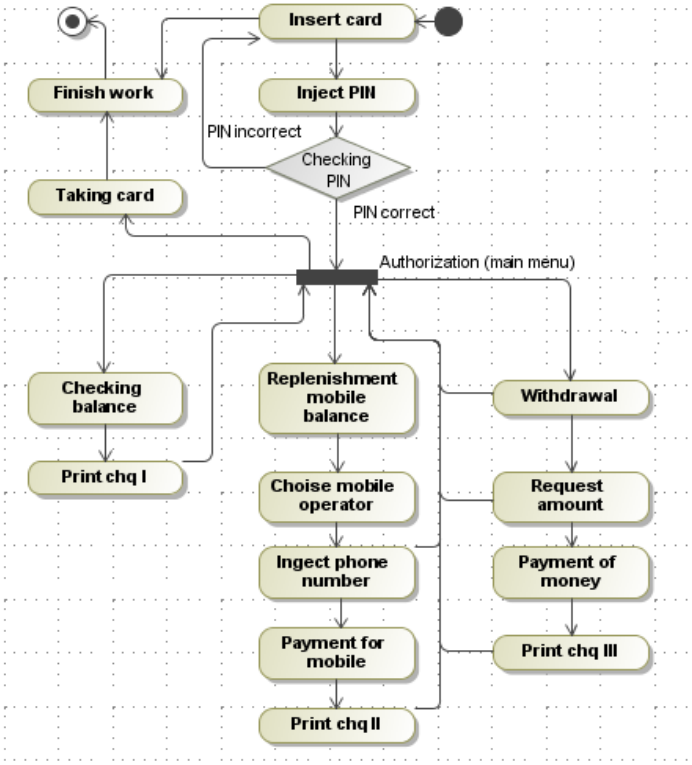


Рисунок 3.1 – Диаграмма активности взаимодействия пользователя с банкоматом

построенной сети Петри переходы соответствуют действиям на диаграмме активности. Для возврата в главное меню данная сеть дополнена местами *From balance to main menu*, *From menu withdrawal to main menu*, *From menu ingect sum for mobile to main menu*.

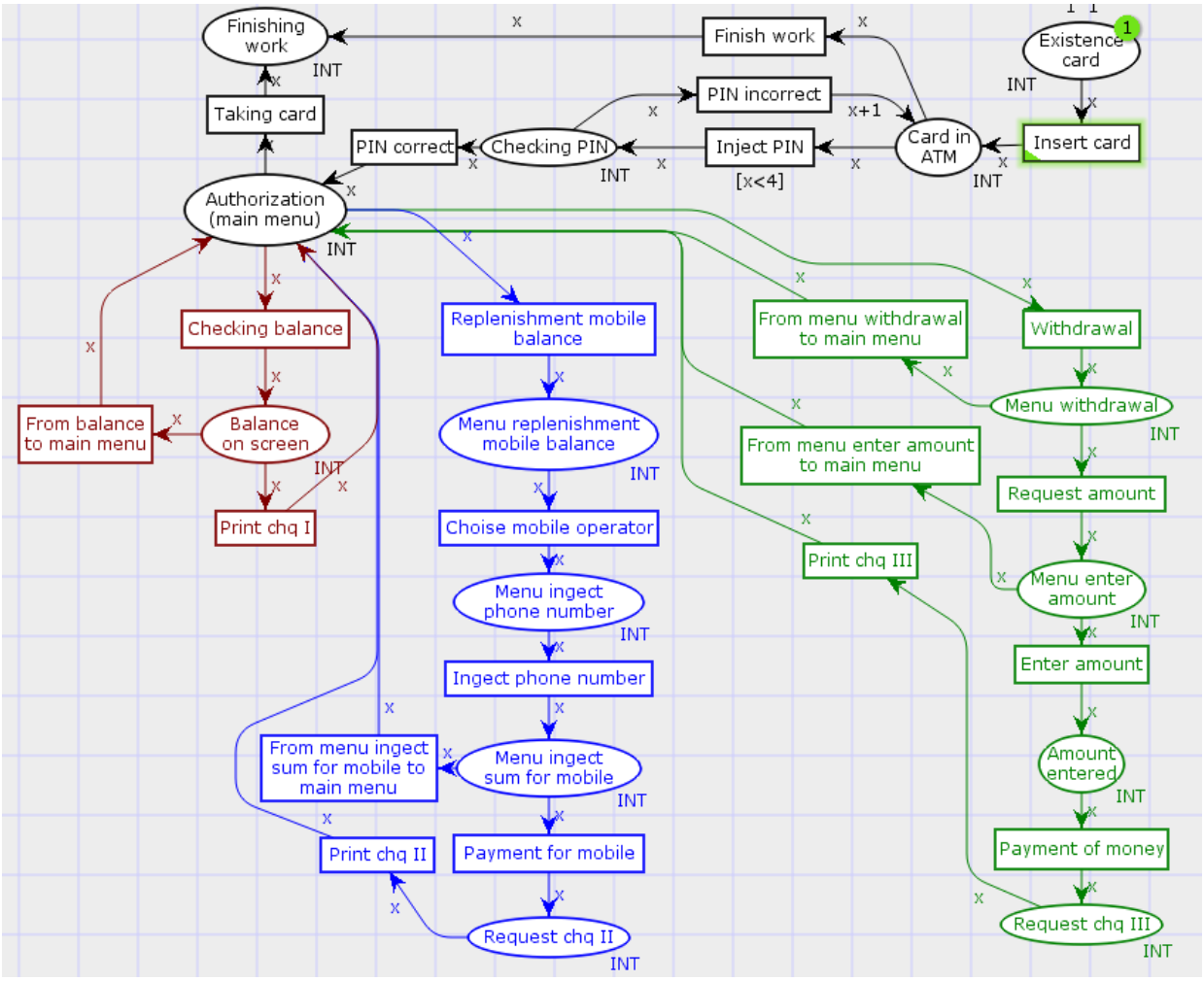


Рисунок 3.2 – Сеть Петри взаимодействия пользователя с банкоматом

Проанализируем полученную сеть посредством исследования пространства состояний, часть из которого проанализировано методом плавающей линии (раздел 1.6), которое содержит 42 узла и 65 дуг. Маркировки 4, 8, 17, 29 (рисунок 1.10) являются конечными для данного графа состояний, который разделяется на три схожие ветки[36], сформировавшиеся из-за возможности некорректного ввода PIN-кода (пользователю предоставляется три попытки). Черным цветом отмечены состояния окончания работы алгоритма – попадание фишки в место *Finishing work*.

Поскольку предлагается анализировать пространство состояний отдельными частями и делать вывод о корректности проектирования сети при помощи их

[36] *Ветка* – состояния, имеющие одного родителя в графе состояний.

анализа, выделим из диаграммы активности несколько возможных сценариев и оценим каждый в отдельности.

Выберем один из сценариев работы системы – проверка баланса счета (рисунок 3.3). В данном сценарии пользователь безошибочно с первого раза вводит PIN-код (*Inject PIN*), проверяет баланс (*Checking balance*), печатает чек (*Print chq I*) и забирает пластиковую карту (*Taking card*).

Построенная диаграмма последовательности транслируется в соответствующую сеть Петри (рисунок 3.4), а пространство состояний данного сценария выглядит следующим образом:

$$V = \{ m_1(1, 0, 0, 0, 0, 0),\ m_2(0, 1, 0, 0, 0, 0),\ m_3(0, 0, 1, 0, 0, 0),$$
$$m_4(0, 0, 0, 1, 0, 0),\ m_5(0, 0, 0, 0, 0, 1),\ m_6(0, 0, 0, 0, 1, 0)\};$$
$$E = \{(m_1, \mathit{Insert\ card}, m_2), (m_2, \mathit{Inject\ PIN}, m_3), (m_3, \mathit{PINcorrect}, m_4),$$
$$(m_4, \mathit{Taking\ card}, m_5), (m_4, \mathit{Checking\ balance}, m_6), (m_6, \mathit{Print\ chq\ I}, m_4)\}.$$

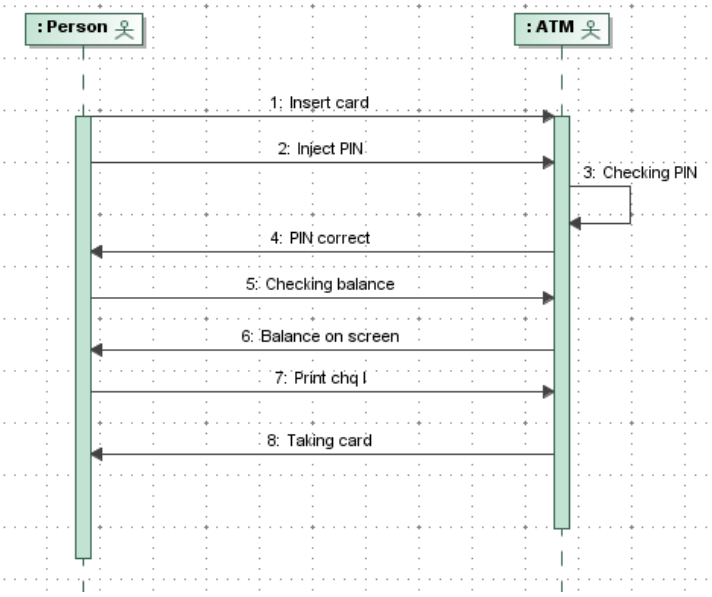


Рисунок 3.3 – Диаграмма последовательности сценария проверки баланса

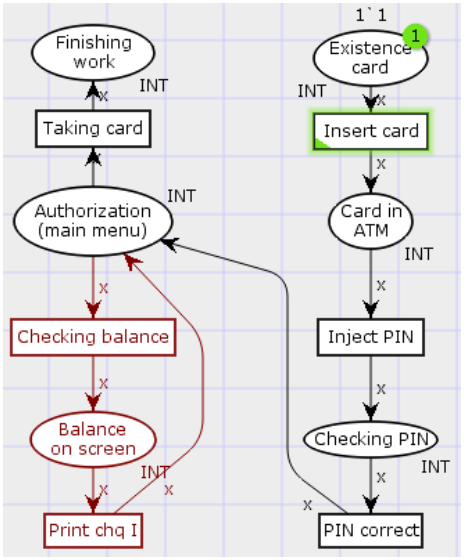


Рисунок 3.4 – Сеть Петри сценария проверки баланса

Пространство состояний данного сценария содержит шесть узлов и шесть дуг. Сеть является безопасной, так как количество фишек в местах не превышает одну.

Выберем другой сценарий работы системы, который заключается в снятии наличных из банкомата (рисунок 3.5). В данном сценарии пользователь также безошибочно с первого раза вводит PIN-код (*Inject PIN*), выбирает пункт меню “Снятие наличных” (*Withdrawal*), вводит необходимую сумму (*Enter amount*), получает платеж (*Payment of money*), печатает чек (*Print chq III*) и забирает карту (*Taking card*).

Результат трансляции диаграмм последовательности (рисунок 3.5) в сеть Петри представлен на рисунке 3.6, а пространство состояний для данной сети имеет девять состояний, девять взаимосвязей и выглядит следующим образом:

$$V = \{ m_1(1,0,0,0,0,0,0,0,0),\ m_2(0,1,0,0,0,0,0,0,0),\ m_3(0,0,1,0,0,0,0,0,0),$$
$$m_4(0,0,0,1,0,0,0,0,0),\ m_5(0,0,0,0,0,1,0,0,0),\ m_6(0,0,0,0,1,0,0,0,0),$$
$$m_7(0,0,0,0,0,0,1,0,0),\ m_8(0,0,0,0,0,0,0,1,0),\ m_9(0,0,0,0,0,0,0,0,1)\};$$
$$E = \{(m_1, Insert\ card, m_2), (m_2, Inject\ PIN, m_3), (m_3, PINcorrect, m_4),$$
$$(m_4, Taking\ card, m_5), (m_4, Withdrawal, m_6), (m_6, Reqest\ amount, m_7),$$
$$(m_7, Enter\ amount, m_8), (m_8, Payment\ of\ money, m_9), (m_9, Print\ chq\ III, m_4).$$

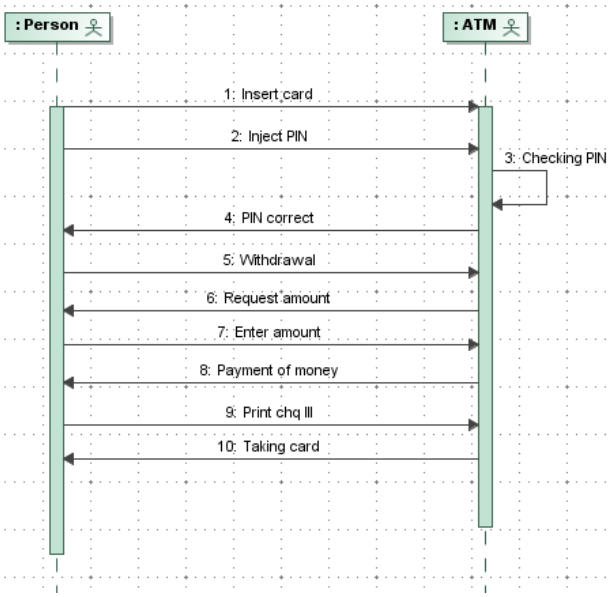


Рисунок 3.5 – Диаграмма последовательности сценария снятия наличных со счета

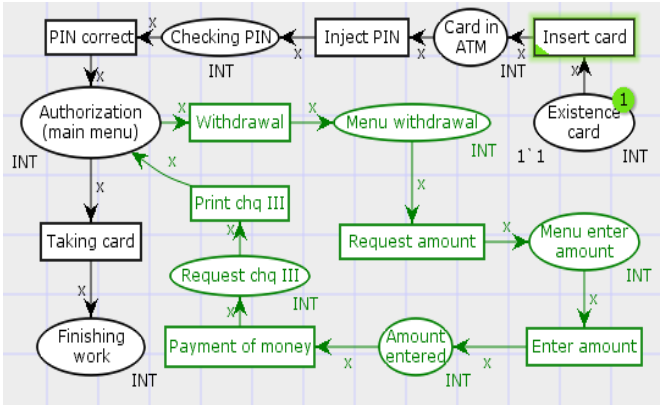


Рисунок 3.6 – Сеть Петри сценария снятия наличных со счета

Покажем на примере, что графы состояний для первого и второго сценария могут быть совмещены с последующим восстановлением целого графа состояний. Для построения общего графа состояний смоделируем общий алгоритм работы двух сценариев, который представлен на частичной диаграмме активности (рисунок 3.7). Действия, используемые в двух сценариях, отображаются на диаграмме активности. Построенная диаграмма была транслирована в сеть Петри (рисунок 3.8). Выполнение двух сценариев можно отследить и на основной сети Петри (рисунок 3.2). Представленный на рисунке 3.9 граф состояний получен на основе анализа общего алгоритма для двух сценариев. Одной из идей данного пункта является *“склеивание” (совмещение)* отдельных сценариев алгоритма воедино, т.е. построение графа состояний выбранных сценариев с их последующим соединением.

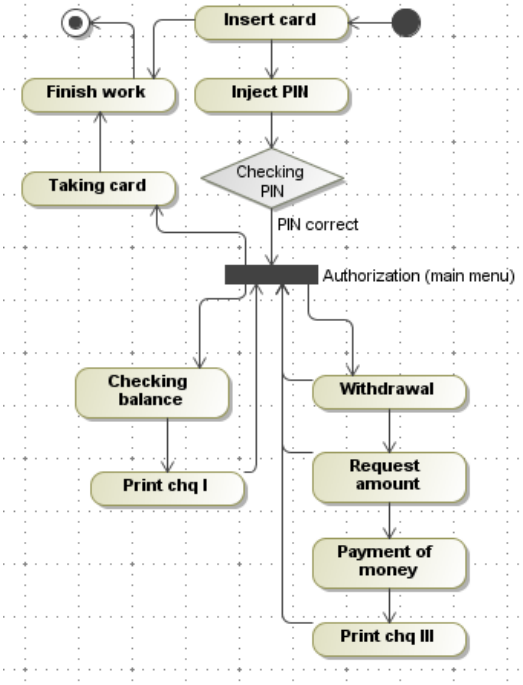


Рисунок 3.7 – Диаграмма активности выделенных сценариев работы

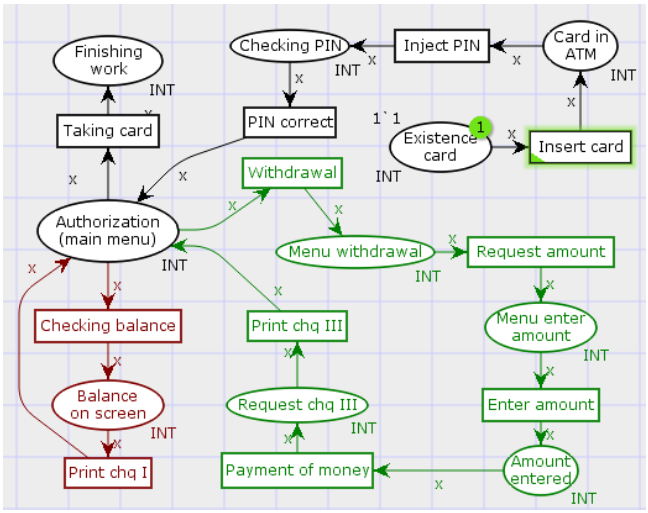


Рисунок 3.8 – Сеть Петри выделенных сценариев работы

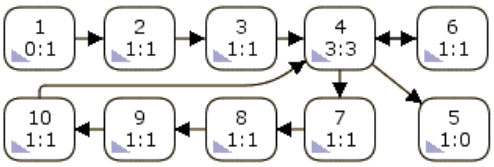


Рисунок 3.9 – Граф состояний выделенных сценариев работы

“Склеивание” нескольких графов состояний происходит при совпадении разрядов у состояний. Совпадение маркировки у нескольких состояний означает, что это одно и то же состояние, следовательно, необходимо оставить одно из них с учетом существующих взаимосвязей. Если количество разрядов в маркировке отличается необходимо добиться их совпадения через добавления к нужным маркировкам нулевых разрядов.

Представим **маркированный граф состояний** для сценария снятия наличных со счёта и сценария проверки баланса. Отметим, что при сравнении построенных графов состояний, количество элементов у маркировки состояний одного графа отличается от другого, что приводит к невозможности соединить их непосредственно. Чтобы избежать данных затруднений, предлагается использовать одинаковое количество разрядов в маркировке, т.е. необходимо добиться одинакового количества мест для сети каждого из сценариев, которое будет равно количеству мест на их общей сети Петри (рисунок 3.8). При отсутствии конкретного места в сети Петри для данного сценария используется нулевая маркировка.

Таким образом, можно избежать трудностей, связанных с различным количеством разрядов. В таблице 3.1 приводится нумерация мест, что необходимо для отображения маркировки графов сети (рисунок 3.8) для двух данных сценариев.

Таблица 3.1 – Нумерация мест у сетей Петри

Existence card	01	Cardin ATM	02	Checking PIN	03
Authorization	04	Menu withdrawal	05	Balance on screen	06
Finishing work	07	Menu enter amount	08	Amount entered	09
Request chq III	10				

Придерживаясь таблицы 3.1 составим маркированные графы состояний для сценария проверки баланаса и сценария снятия наличных со счёта (рисунок 3.10). В квадратных скобках указывается номер состояния, которое приводится в виде элементов в двочной системе исчисления (элемент в круглых скобках) и в системе исчислений с основанием четыре (элементы без скобок)[37]. В маркированном графе (рисунок 3.10.а) не используются маркировки пять, восемь – десять, в маркированном графе (рисунок 3.10.б) не используются маркировка шесть. Отметим, что существуют одинаковые состояния у графов состояний (рисунок 3.10.а, б), которые являются общими для обоих графов состояний и будут совмещены при слиянии возможных разметок сценариев. Совмещенный маркированный граф представлен на рисунке 3.10.в.

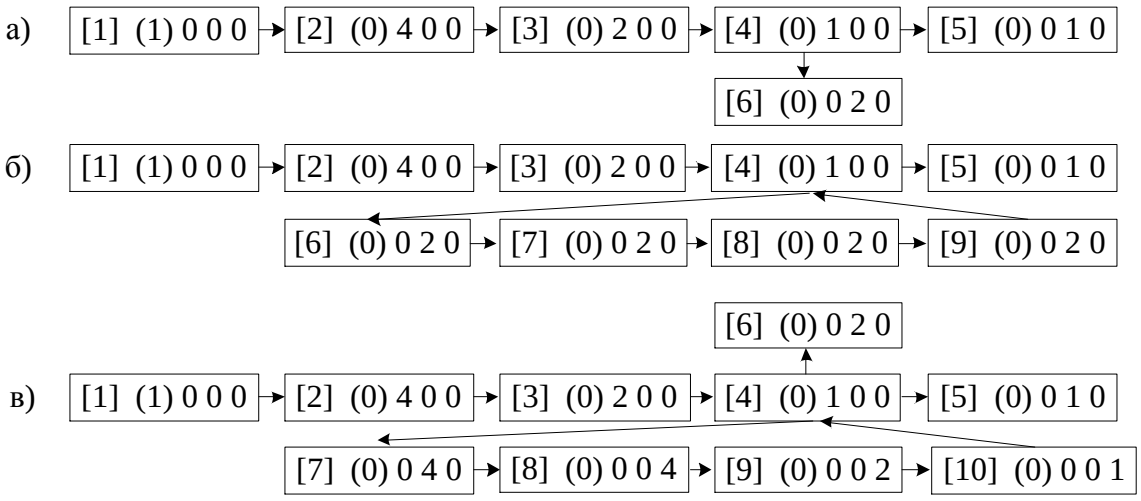


Рисунок 3.10 – Маркированный граф состояний: а) сценарий проверки баланса; б) сценарий снятия наличных со счета; в) сценарий проверки баланса и снятия наличных со счета

[37] Например, на рисунке 3.10.а первое состояние имеет значение (1) 0 0 0. Расшифруем элемент в скобках и следующий за ним (1) 0: $\left(1 \cdot 2^0\right)\ 0 \cdot 4^2\ \ 0 \cdot 4^1\ \ 0 \cdot 4^0$.

Сравнивая детализированный граф (рисунок 3.10.в), и граф, сгенерированный с использованием программной среды CPN Tools (рисунок 3.9), можно сделать вывод об их сходстве и возможности “склеивания” графов (отдельных сценариев одного алгоритма). Таким образом, существует возможность анализа графа состояний посредством исследования отдельных сценариев, полученных из общего алгоритма (диаграмма активности) с последующим их совмещением при необходимости.

Таким образом, продемонстрирован способ построения сценариев работы системы, полученных из диаграммы активности, на примере взаимодействия пользователя с банкоматом. В данном примере были выделены два сценария работы, смоделированные на основе UML и проанализированные в сетях Петри посредством построения графов состояний. В связи с различным количеством вершин (состояний) у графов состояний различных сценариев был предложен способ “склеивания” отдельных веток алгоритма в основной граф состояний, учитывая общие состояния для каждого сценария. При отсутствии одного из элементов сети у выбранного сценария считать маркировку данного элемента нулевой. Сравнение результатов, полученных вручную и с использованием программной среды CPN Tools дали положительные результаты, что может свидетельствовать о работоспособности предложенного способа.

3.2. Иерархическая сеть Петри. Анализ свойств отдельных подсетей для оценки всей системы

В предыдущем разделе продемонстрирована возможность анализа пространства состояний посредством анализа отдельных сценариев на примере взаимодействия банкомата и пользователя. Альтернативным способом анализа пространства состояний является ***представление*** всей ***структуры*** системы, а именно сети Петри ***в иерархическом виде,*** с последующим разбиением некоторых частей на отдельные подсети. Если результаты, полученные при анализе свойств подсетей и основной сети, удовлетворяет желаемым качествам, то можно утверждать, что при анализе свойств иерархической сети получают подобные результаты [70]. Стоит отметить, что анализ подсети нужно начинать со всех возможных входных состояний в подсеть. Продемонстрируем данный способ на примере интернет-магазина, который представим алгоритмически. Анализ моделируе-

мой системы выполним посредством генерации и исследования пространства состояний.

Интернет-магазин – сайт, который организует торговлю товарами через интернет, позволяя пользователю найти нужный товар, осуществить покупку, выбрать способ оплаты. Также пользователю доступны функции регистрации, авторизации в системе и посещение личного кабинета. Все эти возможности, как видно на рисунке 3.11, отделены друг от друга для более легкого восприятия.

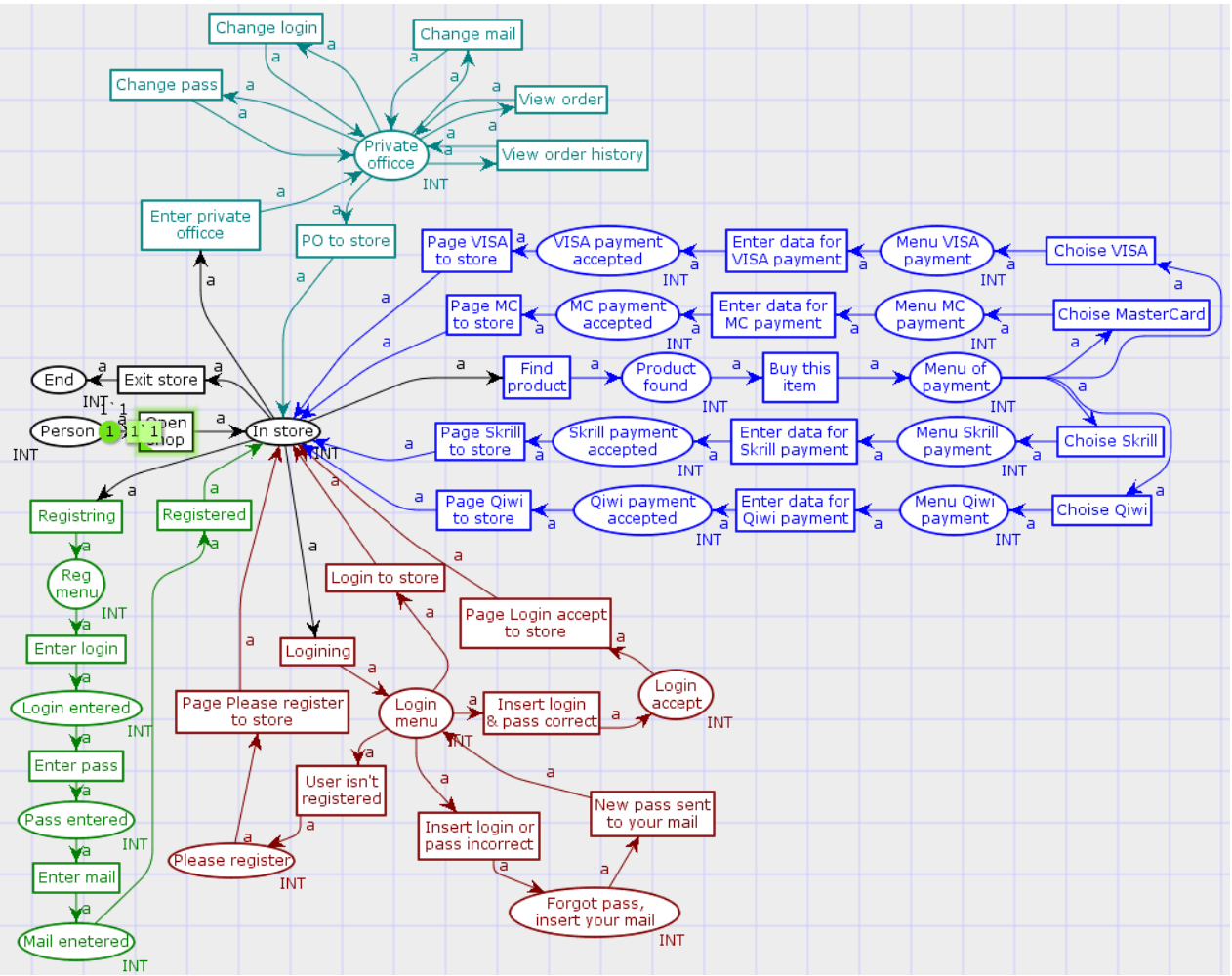


Рисунок 3.11 – Сеть Петри интернет-магазина

Представим **данную сеть Петри в иерархическом виде.** На рисунке 3.11 представлена сеть Петри, которая отображает основные процессы, задействованные при функционировании интернет-магазина. Система состоит из *пяти* основных сценариев: нахождение в интернет-магазине, регистрация, авторизация, поиск и оплата товара, а также “Личный кабинет”. Каждый из пяти сценариев может функционировать отдельно, тем самым напрашивается разделение основных сценариев на подсети (рисунок 3.12). При анализе пространства состояний

иерархической сети получен граф состояний, который имеет *42* вершины и *70* дуг. В системе нет зацикливаний, *4ая* и *25ая* маркировки являются тупиковыми. Данные маркировки – это состояния выхода из интернет-магазина, следовательно, их стоит рассматривать как состояния завершения работы системы.

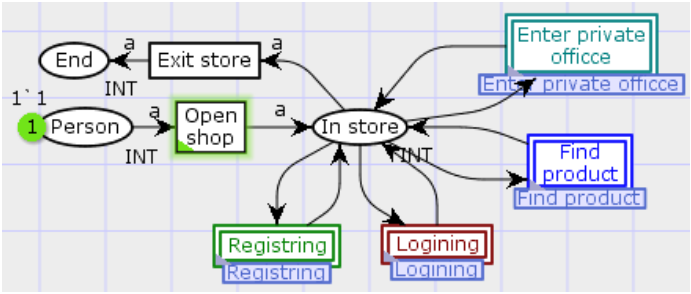


Рисунок 3.12 – Главная страница иерархической сети Петри интернет-магазина

Анализ отдельных частей системы. Поскольку подсети (*Registring*, *Loginig*, *Find product*, *Enter private office*) представляют собой подсистемы, то их можно анализировать отдельно от основной сети. Проведем анализ подсети “Личный кабинет”. На рисунке 3.13 представлена сеть для выполнения данного сценария. Если сеть для проверки системы была поделена на иерархии, то анализ подсети нужно начинать со всех возможных входных состояний в эту подсеть – для данного примера существует только одно входное состояние.

Пространство состояний для данной подсети выглядит следующим образом:

$$V = \{ m_1(1,0),\ m_2(0,1)\};$$
$$E = \{(m_1, Enter\ \ private\ office, m_2), (m_2, Change\ \ pass, m_2),$$
$$(m_2, Change\ login, m_2), (m_2, Change\ mail, m_2), (m_2, View\ order, m_2),$$
$$(m_2, View\ order\ history, m_2), (m_2, PO\ to\ store, m_1).$$

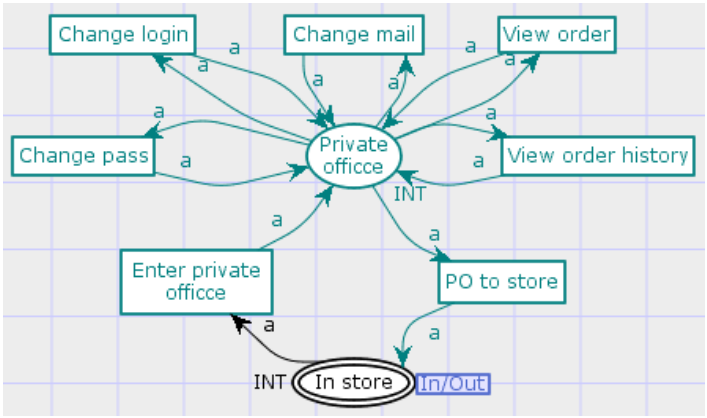


Рисунок 3.13 – Подсеть “Личный кабинет”

Анализ пространства состояний показал наличие двух вершин и семи дуг с полностью живой маркировкой сети, отсутствием зацикливаний, тупиковых разметок и возможностью срабатывания всех переходов сети.

Аналогичные результаты были получены при анализе трёх оставшихся подсетей, что дает возможность упростить сеть Петри, представленную на рисунке 3.11, до сети – на рисунке 3.14. Места каждой подсети заменяются

единственным аналогичным местом (местом входа в подсеть) в основной сети. Граф состояний для данной сети представлен на рисунке 3.15: *13* вершин, *21* дуга, в сети две мертвые маркировки – *третья* и *девятая*, которые связаны с местом выход из системы и тем самым эти ошибки игнорируются, в сети нет зацикливаний, все переходы имеют возможность сработать.

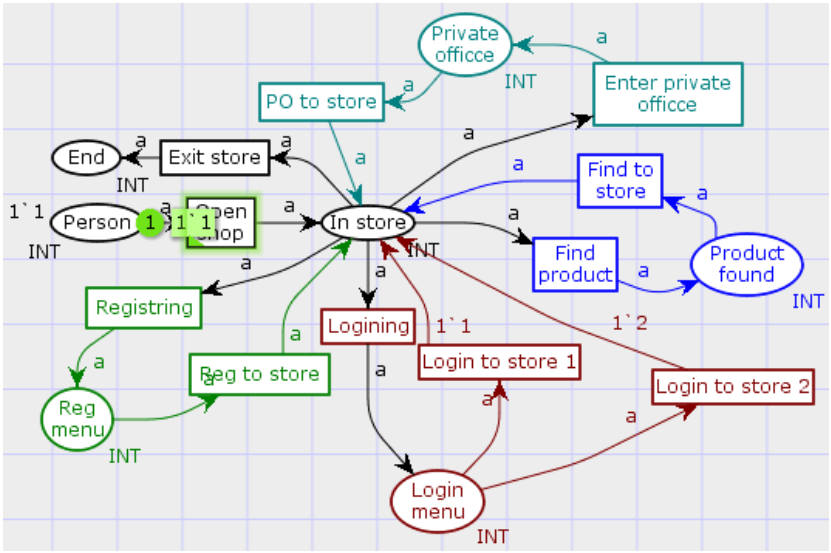


Рисунок 3.14 – Упрощенная сеть системы “Интернет-магазин”

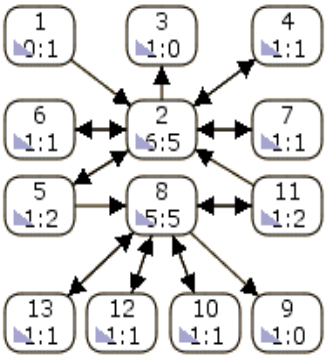


Рисунок 3.15 – Граф состояний для упрощенной сети интернет-магазина

Результаты, полученные при анализе иерархической сети целиком, и результаты, полученные при анализе подсетей и главной сети по отдельности, совпадают: в системе имеется две мертвые маркировки (выход из системы), отсутствуют зацикливания, все переходы могут сработать.

Таким образом, на примере интернет-магазина продемонстрирована возможность представления сетей Петри в иерархическом виде с целью анализа отдельных подсетей. Это позволяет анализировать по отдельности все участки сети и, опираясь на полученные результаты, делать вывод о корректности построения всей сети. При наличии связей между подсетями следует внести дополнительные критерии анализа сетей, заключающиеся в добавлении в основную сеть мест и переходов между подсетями, количество которых будет зависеть от количества возможных состояний взаимодействия между подсетями.

3.3. Анализ отдельных частей графа состояний сетей Петри

В данном разделе предлагается вариант анализа сетей Петри, который основан на предложенных выше способах [70, 74], и заключается в ***анализе некоторых частей графа состояний*** [76]. При чем данные части, содержащие состояния могут быть выбраны произвольно. В сравнении с хешированием состояний и с методом “плавающей” линии (раздел 1.6) данный позволяет анализировать граф состояний без потери информации об изученных ранее состояниях. Продемонстрируем данный способ на примере цветной сети Петри (рисунок 3.16).

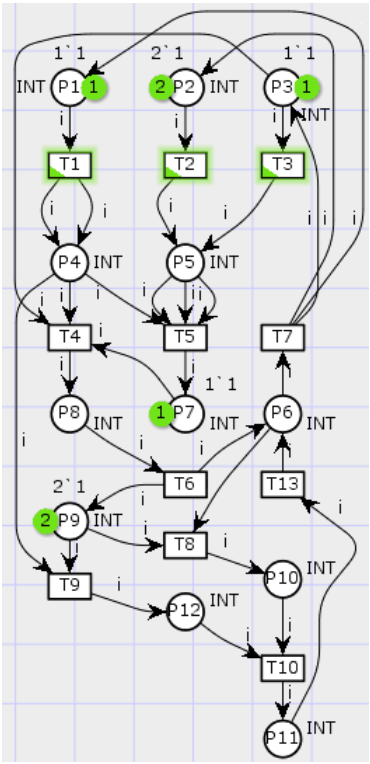


Рисунок 3.16 – Пример цветной сети Петри [76]

Данную сеть анализируем с использованием генерации пространства состояний [76], которое имеет *193* вершины и *401* дугу, зацикливаний в сети нет, существуют тупиковые маркировки – состояния, при которых прекращается работа сети, на графе состояний они изображены черным цветом. Представим структуру графа по частям и проанализируем данные части. После этого сравним свойства полученных графов состояний со свойствами общего графа.

Граф состояний делится на три части (ветки). Эти три ветки анализируются и при необходимости разбиваются еще на несколько частей. *Анализ любого участка графа состояний* используют при соблюдении некоторых условий: проверена достижимость состояния, с которого начинается анализ, нет возвратных взаимосвязей с более ранними состояниями. При наличии данных связей их необходимо перенести в анализируемую часть. Данный способ позволяет анализировать наиболее критичные к ошибкам состояния и участки сети, а также избегать анализа всего пространства состояний. Из вышесказанного сделаем вывод, что возможен анализ, заключающийся, в построении графа состояний тех участков, которые необходимо проанализировать.

Представление и описание фрагментов графа состояний. Фрагмент графа состояний представлен на рисунке 3.17. Первое разделение осуществлялось

по второму состоянию. В данном случае были достигнуты *44ая* и *45ая* маркировки сети, которые являются состояниями, завершающими работу системы.

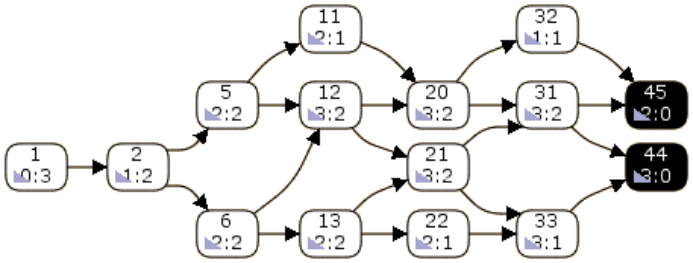


Рисунок 3.17 – Первый фрагмент графа состояний сети Петри

На рисунке 3.18 демонстрируется второй фрагмент графа состояний, начинающийся с s_3, и его последующие состояния с выходными состояниями *44, 45, 130, 131*. Остальная часть графа состояний разбивается на 12 фрагментов [76], после чего отдельные фрагменты становятся отдельными графами. Анализ свойств полученных фрагментов можно сравнить с анализом на примерах разбиения сети на иерархии и оценки получившихся сетей по отдельности.

Для **нахождения предела количества исследуемых состояний** в пакете **CPN Tools (version 3.4.0)** сеть, представленная на рисунке 3.16, расширяется местами, переходами и метками (рисунок 3.19). После чего анализируется динамика роста пространства состояний (количественное увеличение состояний с течением времени при исследовании пространства состояний) при увеличении времени на анализ. Динамика роста пространства состояний представлена в табличном (таблица 3.2) и графическом виде (рисунок 3.20).

Таблица 3.2 – Динамика роста пространства состояний

Seconds (se)	Nodes (n)	Arcs (a)	Seconds (se)	Nodes (n)	Arcs (a)	Seconds (se)	Nodes (n)	Arcs (a)
5	6320	19770	15	12659	41867	25	17169	58037
50	26033	91179	100	39908	144222	150	51509	189664
200	61509	229050	300	78898	298551	400	93917	358730
500	107561	413878	600	119299	461739	800	140386	547563
1000	159718	626898	1200	173431	682913	1600	200017	792542
2000	223053	887683	2400	244008	974603	2800	264372	1059026
3200	282966	1136296	3600	295487	1188236	4000	307087	1236799
4100	320948	1294341						

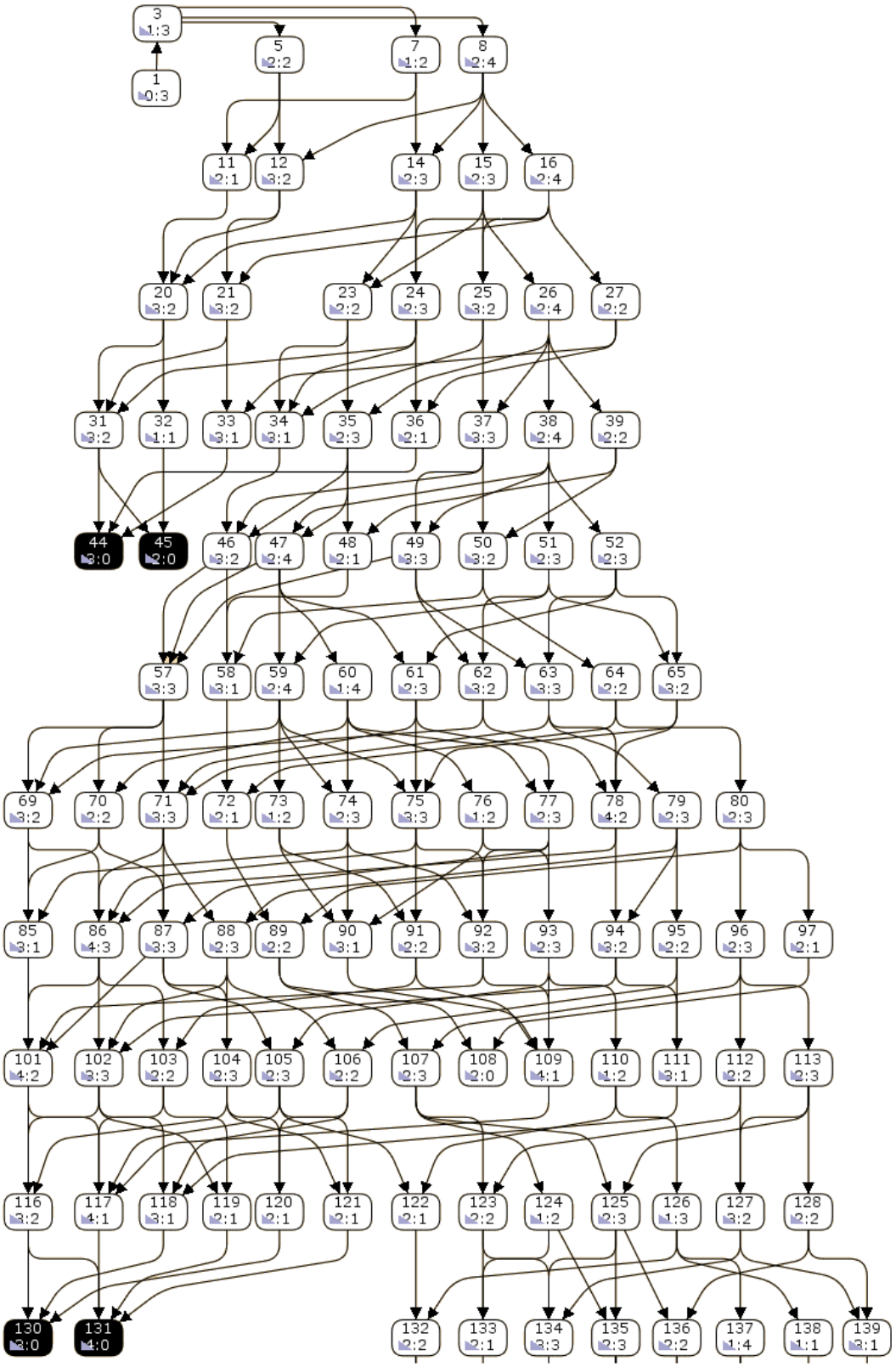


Рисунок 3.18 – Второй фрагмент графа состояний сети Петри

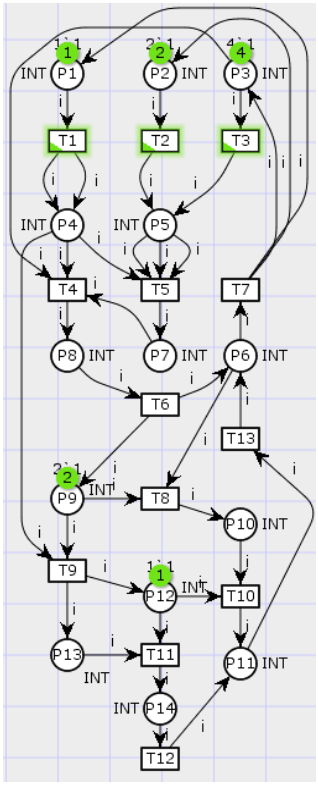


Рисунок 3.19 – Усложненная сеть Петри

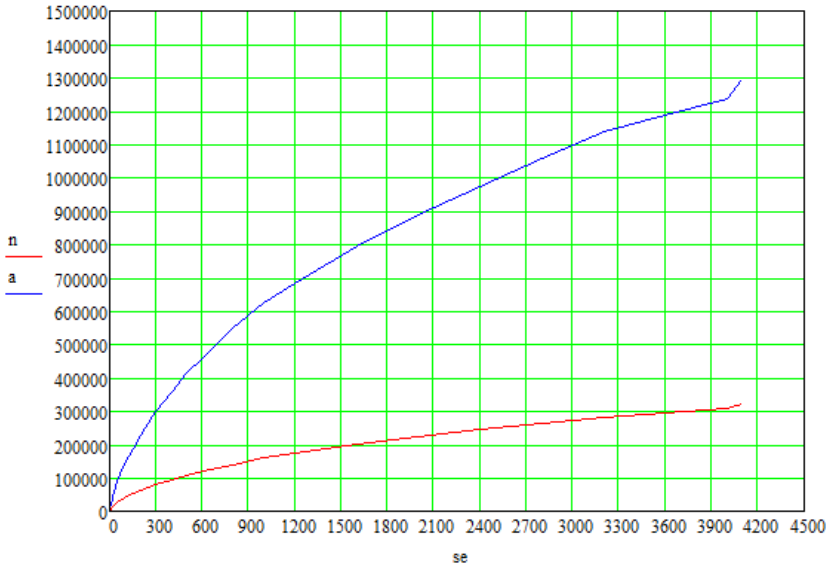


Рисунок 3.20 – График динамики роста пространства состояний

И так предложен способ, в котором предполагается выделение критичных участков пространства состояний с последующим их анализом, с условием проверки достижимости начального состояния. При подтверждении достижимости данного состояния анализируется граф, получаемый при исследовании с данного состояния без потери свойств, присущих выбранному фрагменту пространства состояний: число меток в любом месте не превышает четыре – сеть ограничена; система является несохраняющей – во время работы, появляются новые и исчезают старые метки.

3.4. Инверсия сетей Петри для проверки достижимости выбранных состояний

В данном разделе предлагается ***инверсия***[38] ***сетей Петри*** для проверки достижимости выбранного состояния, при которой происходит смена ориентаций взаимосвязей между элементами сети, способствуя движению меток в обратном направлении, с целью нахождения начального состояния от заранее выбранного.

[38] *Инверсия* – изменение расположения, либо ориентации, элементов сети Петри в особом порядке, нарушающем обычный (прямой) порядок, с целью изменить движение меток на обратное.

Приводится алгоритм для реализации инверсии и анализа пространства состояний (алгоритм 3.1), а также девять правил, по которым происходит преобразование из ординарной [77] или простой [78] сети Петри в инвертированную. Данный способ анализа может потребоваться, когда перед проектировщиками стоит задача проверки определенной части пространства состояний системы (п. 3.3). При изучении отдельных частей пространства состояний системы [76] необходима уверенность в достижимости выбранного для анализа состояния, что не всегда доступно стандартным способом. Предлагаемый способ проверки достижимости выбранной маркировки применим к случаям, если заранее неизвестно находится ли состояние во всём множестве состояний системы, а заранее известна только начальная маркировка.

Покажем **инверсию простой ординарной сети Петри** на системе, состоящей из двух мест $P = \{ A, B \}$ и одного перехода $T = \{ a \}$ (рисунок 3.21.а). Инвертируем представленную модель, посредством смены ориентации взаимосвязей между вершинами (рисунок 3.21.б), а начальным состоянием выберем состояние №2 не преобразованной сети. Рассмотрим возможность инвертирования на примере простой ординарной сети Петри $\mathcal{N} = \left(P, T, F, m_I \right)$ (рисунок 3.6) с начальной маркировкой: $m_I = (1,0,0,0,0,0,0,0,0)$, $p \in p : m(p) = 1$.

а) б)

Рисунок 3.21 – Система, состоящая из двух мест: а) сеть Петри и её граф состояний, б) инвертированная сеть Петри и её граф состояний

Представленная сеть (рисунок 3.6) является одним из сценариев работы [74] взаимодействия пользователя и банкомата. Данный сценарий проанализирован с последующим построением графа состояний, достижимость всех маркировок проверена. Выберем маркировку, при которой происходит инвертирование сети: $m_i = (0,0,0,0,0,0,1,0,0)$. При таком состоянии в системе место *Amount entered* имеет одну фишку. Чтобы проверить находится ли выбранное состояние в про-

странстве состояний всей системы, нужно проверить достижимость этой маркировки из начального состояния. Для чего инвертируем сеть (рисунок 3.6), посредством простой смены ориентации всех дуг $\mathcal{N}_i = \left(P, T, F_i, m_i \right)$, где $\mathcal{N}_i$ – инвертированная простая сеть Петри, F_i – инвертированные направленные взаимосвязи, m_i – маркировка, при которой произошла инверсия.

Алгоритм 3.1 – Инверсия простой ординарной сети Петри и вычисление пространства состояний

1: *while* $F \subseteq P \times T \cup T \times P$ // пока существует цепочка данного вида,

2: *select* $p \xrightarrow{t} p'$ // выбираем последовательно цепочку данного вида,

3: *for all* $\left(t, p' \right)$ *such that* $p \xrightarrow{t} p'$ *do invert* $p' \xrightarrow{t} p$ // реализуем инверсию,

4: *return* $\left(P, T \right)$ // возвращаем преобразованные данные,

5: *end while* // конец цикла,

6: $V := \left\{ m_i \right\}$ // переменной V (множество посещённых состояний) присваивается множество с единственной переменной m_i,

7: $W := \left\{ m_i \right\}$ // переменной W (множество не посещённых состояний) присваивается множество с единственной переменной m_i,

8: $E := 0$ // переменной E (множество рёбер) присваивается значение $\emptyset$,

9: *while* $W \neq \left\{ m_I \right\}$ *do* // пока множество W не станет пустым, выполняем следующие действия,

10: *select an* $s \in W$ // выбираем состояния m из множества W,

10: $W := W \setminus \left\{ m \right\}$ // убираем из множества W состояние m,

11: *for all* t, m' *such that* $m \xrightarrow{t} m'$ *do* // для всех t, m', удовлетворяющих $m \xrightarrow{t} m'$, выполняем следующие действия,

12: $E := E \cup \left\{ \left(m, t, m' \right) \right\}$ // добавляем новое ребро во множество,

13: *if* $m' \notin V$ *then* // если найдено новое состояние, не входящее в множество *V*, тогда,

14: $V := V \cup \left\{ m' \right\}$ // добавляем его к этому множеству,

15: $W := W \cup \left\{ m' \right\}$ // и к множеству W,

16: *return* $\left(V, E_i, W \right)$ // возвращаем обновленные значения переменных V и E_i,

17: *end while* // конец цикла.

После выполнения алгоритма, представленного выше, получаем сеть (рисунок 3.22) и следующее пространство состояний:

$V = \{ m_1(0,0,0,0,0,0,0,0,1),\ m_2(0,0,0,0,0,0,1,0,0),\ m_3(0,0,0,0,0,1,0,0,0),$
$m_4(0,0,0,0,1,0,0,0,0),\ m_5(0,0,0,0,0,0,0,1,0),\ m_6(0,0,0,1,0,0,0,0,0),$
$m_7(0,1,0,0,0,0,0,0,0),\ m_8(1,0,0,0,0,0,0,0,0)\};$
$E_i = \{(m_1, Enter\ amount, m_2),\ (m_2, Request\ amount, m_3),\ (m_3, Withdrawal, m_4),$
$(m_4, Print\ chq\ III, m_5),\ (m_4, PIN\ correct, m_6),\ (m_6, Inject\ PIN, m_7),$
$(m_7, Insert\ card, m_8),\ (m_5, Payment\ of\ money, m_1).$

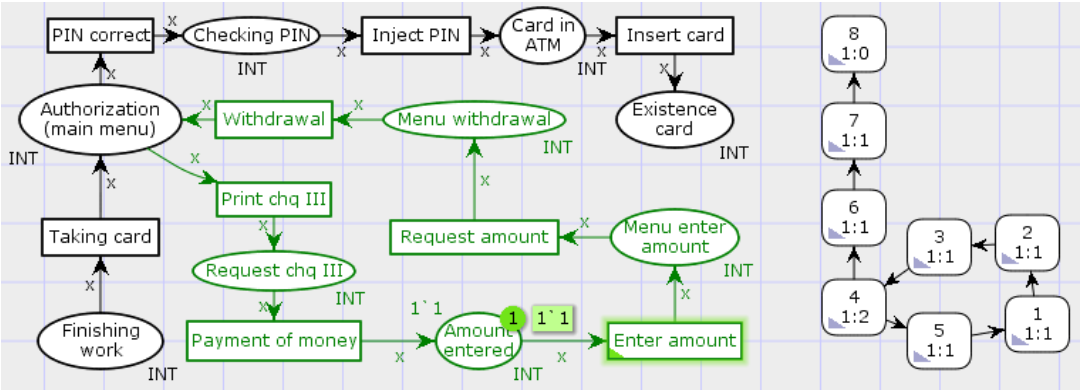


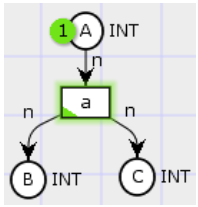
Рисунок 3.22 – Инвертированная сеть Петри (слева)
и граф состояний инвертированной сети Петри (справа)

Граф состояний для инвертированной сети Петри (рисунок 3.22) (V, E_i, src, trg), где E_i – инвертированное множество рёбер, показывает, что начальная маркировка сценария достижима из места *Amount entered*. Если сравнить графы для сети, представленной на рисунке 3.6 и рисунке 3.22, видно, что состояние 3 для сети (рисунок 3.6) отсутствует в графе инвертированной сети (рисунок 3.22). Это говорит о возможности потери состояний во время инверсии сетей. А именно, часть сети, состоящая из места *Finishing work* и перехода *Taking card*, не присутствует в данном пространстве состояний, так как нет возможности достигнуть этих вершин из выбранной разметки, тем самым они выпадают из пространства состояний. Поскольку, задачей ставилась проверка достижимости выбранной маркировки, то данной потерей можно пренебречь.

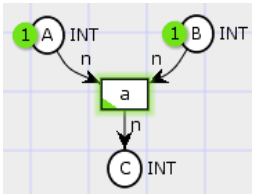
Стоит отметить, что прямая инверсия[39] возможно только для простых ординарных сетей. При наличии набора условий у переходов и дуг, стоит ввести определенный набор правил, при выполнении которых разработчик получит начальную маркировку, используя данный способ.

[39] *Прямая инверсия* – смена ориентации дуг между вершинами сети.

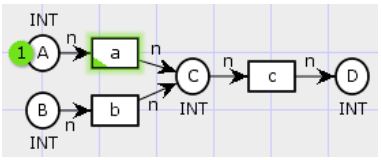
Ниже на примерах рассматриваются **правила инверсии простой сети Петри**, у которой, в отличие от простой ординарной, используется любое количество взаимосвязей, направленных от мест к переходам и от переходов к местам. Правила получены, опираясь на анализ небольших сетей Петри (рисунок 3.23), которые могут являться фрагментами более крупной сети Петри. Приведённые примеры излагаются более подробно с отображением графов состояний и представлением сетей Петри инвертированного вида в приложении Г.



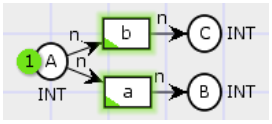
а) пример №1



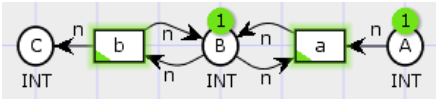
б) пример №2



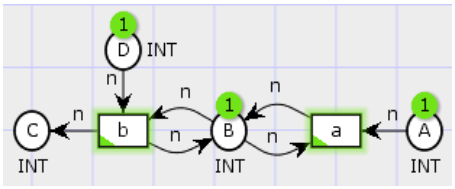
в) пример №3



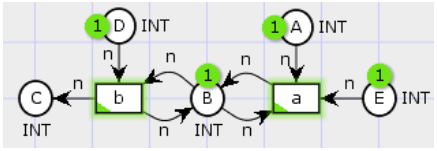
г) пример №4



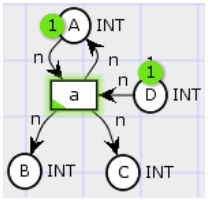
д) пример №5



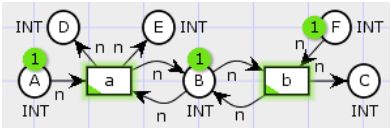
е) пример №6



ж) пример №7



з) пример №8



и) пример №9

Рисунок 3.23 – Примеры фрагментов сетей Петри

Таким образом, для проверки достижимости состояний предложено использовать инверсию сети Петри. Данный способ показан на примере простой ординарной сети Петри и представлен алгоритм для его реализации. Стоит уточнить, что для простой ординарной сети подходит прямая инверсия, а для простых сетей Петри, в свою очередь, стоит добавить правила, так как в системе возможно неограниченное количество входных и выходных дуг у вершин.

3.5. Инверсия графа состояний сети Петри для проверки достижимости выбранных состояний на примере протокола передачи данных

В данном разделе демонстрируется инверсия на примере протокола передачи данных. Также предлагается ***инверсия графа состояний***, как одного из способов проверки достижимости выбранного состояния [81]. Преимуществом данного способа является однозначность (изменение ориентации взаимосвязей вершин сети без преобразования её структуры) преобразования. После на основании инвертированного графа состояний восстанавливается сеть.

Система “Протокол передачи данных” [7, 8] предназначена для передачи информации между отправителем и приёмником, т.е. отправитель может передавать необходимое количество сообщений приёмнику (рисунок 3.24). Предложенная структура системы имеет три основных потока: отправление сообщения от источника к приёмнику, подтверждение получения сообщения от приёмника к источнику и запрос на получение данных от приёмника к источнику. Таким образом, отправитель передаёт пакеты данных получателю и принимает подтверждение об их доставке. В приёмник поступают отправленные сообщения, после чего он отсылает информацию о получении. Данная сеть (рисунок 3.24) состоит из 14 мест P, восьми переходов T и начальной маркировкой m_I:

$$P = \{F, E, Received, B, C, S3, D, A, nextsend, send, S1, S2, Count, Count1\},$$
$$T = \{transmit\ packet2, send\ packet2, Receive, receiveAck, send\ Packet, Receive\ Packet2, transmit2, transmitAck\};$$
$$m_I = (0,0,0,0,0,1,0,0,1,1,1,1).$$

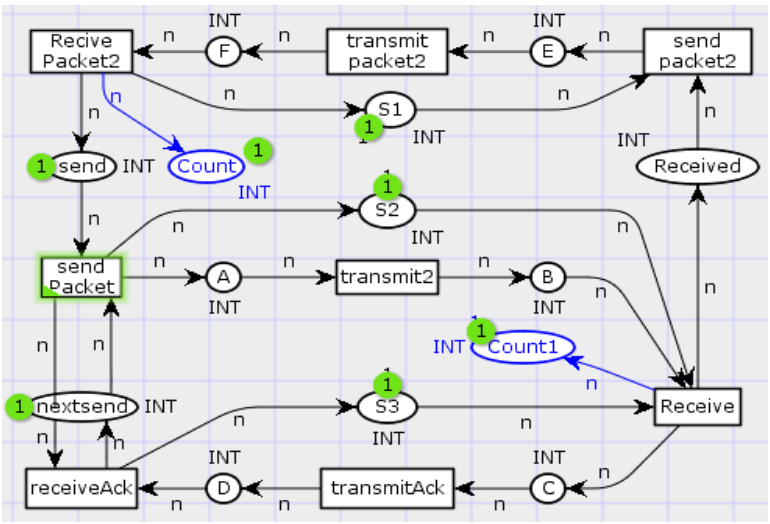


Рисунок 3.24 – Сеть Петри протокола передачи данных

Полученную сеть Петри анализируют на основе пространства состояний, полученного автоматически в программном пакете CPN Tools. Поскольку анализ проводился в системе с местами-счётчиками[40] *Count* и *Count1*, способствующими циклическому анализу, пространство состояний вычислено и показано частично. А проверка частичного, хоть и повторяющегося пространства состояний, не может полностью подтвердить корректность построения системы. Поэтому принято решение об удалении этих мест, с целью провести анализ данной сети и реализовать для неё инверсию. Полученная сеть (рисунок 3.24, без мест *Count* и *Count1*) имеет туже структуру, а отсутствие мест *Count* и *Count1* на логику всей системы никак не влияет. Граф состояний (рисунок 3.26) для системы (рисунок 3.25, без мест *Count* и *Count1*) имеет 18 состояний: $V = \{m_1, \ldots, m_{18}\}$.

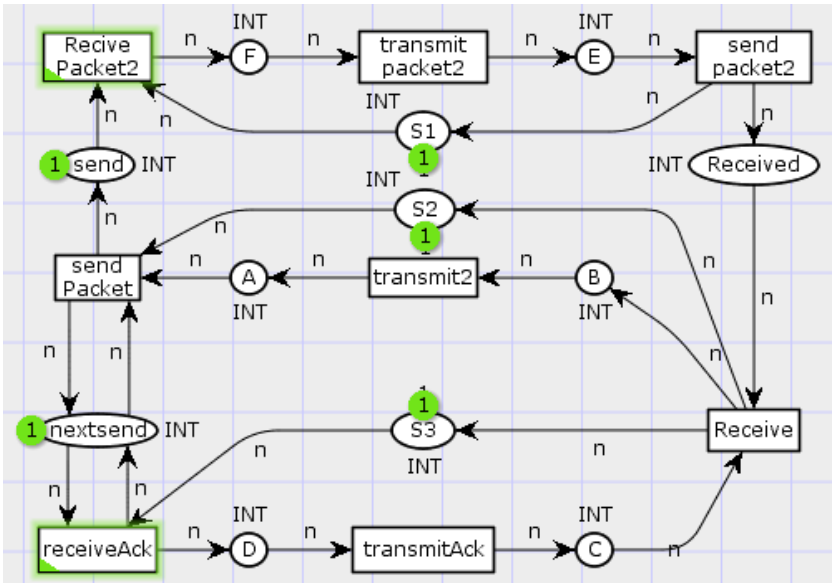


Рисунок 3.25 – Инвертированная сеть Петри протокола передачи данных

Для инверсии сети (рисунок 3.24, без мест *Count* и *Count1*) воспользуемся правилами, предложенными в предыдущем разделе, а сама инверсия будет заключаться в простой смене ориентации дуг (алгоритм 3.1 строка 1-5).

Визуально, полученная инвертированная сеть (рисунок 3.25) не отличается от исходной сети Петри (рисунок 3.24, без мест *Count* и *Count1*). Но чтобы убедиться в этом, представим графы состояний исходной и инвертированной сети (рисунок 3.26, 3.27), а разряды каждого из состояний сгруппируем по три элемента и отобразим в четверичной системе исчисления.

[40] Реализация пространства состояний, включающего счётчик, представлена в [78].

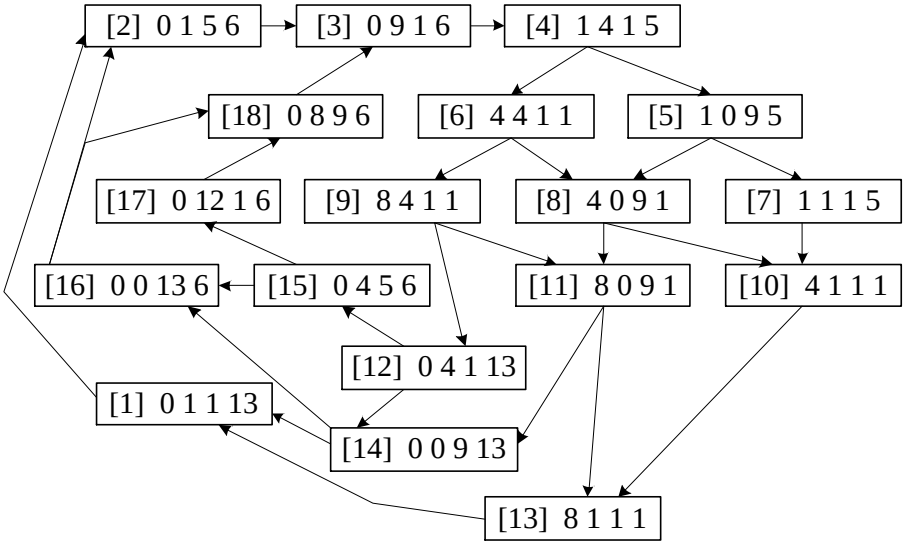


Рисунок 3.26 – Разметка графа состояний исходной сети Петри

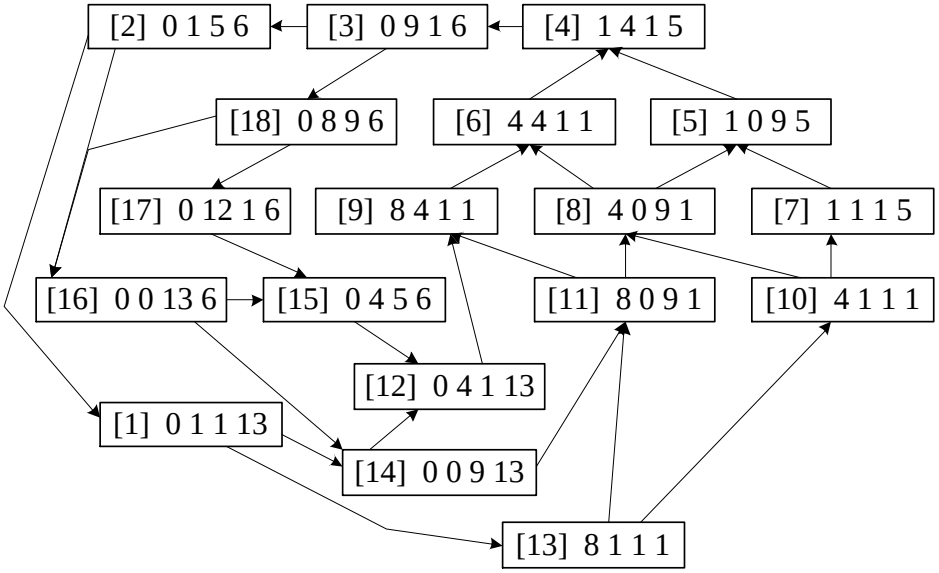


Рисунок 3.27 – Разметка графа состояний инвертированной сети Петри

Инверсия графа состояний. При инвертировании сети Петри существует возможность столкнуться с различными проблемами, связанными со сложностью структуры системы. Например, будет недостаточно изменить направление дуг между вершинами, а потребуется преобразование структуры сети, что у больших систем весьма затруднительно. В связи с этим, более простым видится инверсия графа состояний. Преимущество данного способа заключается в однозначности преобразования, которая заключается в смене ориентаций связей между состояниями. Недостатком является то, что для безошибочного инвертирования графа состояний необходимо знать его целиком. Данное преобразование демонстрируется на примере протокола передачи данных (рисунок 3.24, без мест *Count* и *Count1*).

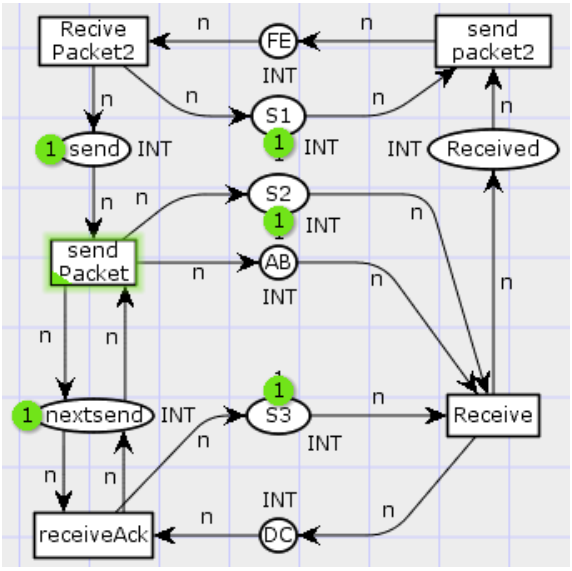


Рисунок 3.28 – Преобразованная сеть Петри протокола передачи данных

Для упрощения исходного пространства состояний в предложенной сети Петри объединим места F и E, A и B, D и C и удалим переходы между ними (рисунок 3.28), что не изменит логику работы сети. После преобразования система содержит девять мест P, пять переходов T с начальной маркировкой m_I:

$P = \{FE,\ Received,\ S3,\ DC,\ AB,\ nextsend,\ send,\ S1,\ S2\}$;

$T = \{send\ \ packet2,\ Receive,\ receiveAck,\ send\ Packet,\ Receive\ Packet2\}$;

$m_I = (0,0,1,0,0,1,1,1,1)$.

Полученная сеть Петри имеет следующие взаимосвязями между вершинами:

$$\{S1,\ Received\}\left[\ send\ \ packet2\right\rangle\{FE\},$$
$$\{S2,\ AB,\ S3\}\left[\ Receive\right\rangle\{Received,\ DC\},$$
$$\{DC,\ nextsend\}\left[\ receiveAck\right\rangle\{S3,\ nextsend\},$$
$$\{send,\ nextsend\}\left[\ send\ \ Packet\right\rangle\{S2,\ AB,\ nextsend\},$$
$$\{FE\}\left[\ Receive\ Packet2\right\rangle\{S1,\ send\}.$$

Количество мест в сети уменьшилось с четырнадцати до девяти, что привело к уменьшению количества узлов в графе состояний (рисунок 3.29).

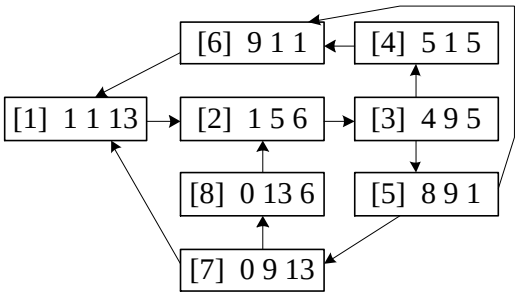


Рисунок 3.29 – Граф состояний преобразованной сети Петри

Реализуем инверсию графа состояний, которое после преобразования (рисунок 3.30): (V_i, E_i, src, trg) повторяет структуру исходного графа, но взаимосвязи между вершинами поменяли своё направление.

На следующем этапе происходит восстановление сети Петри из конечного графа состояний (приложение Г). После восстановления сети из инвертированного графа состояний получаем сеть (рисунок 3.31).

При сравнении сетей, представленных на рисунке 3.28 и 3.31, отметим, что для инверсии исходной сети достаточно изменить ориентацию дуг, что много проще при неизвестном пространстве состояний. Но применив инверсию к графу состояний, мы использовали значительное преимущество – однозначность трансформации графа состояний, что может быть весьма полезно для систем со сложной структурой сети.

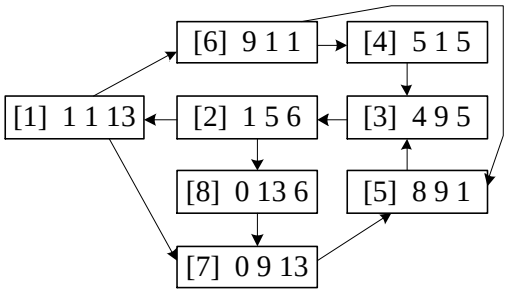


Рисунок 3.30 – Инвертированный граф состояний преобразованной сети Петри

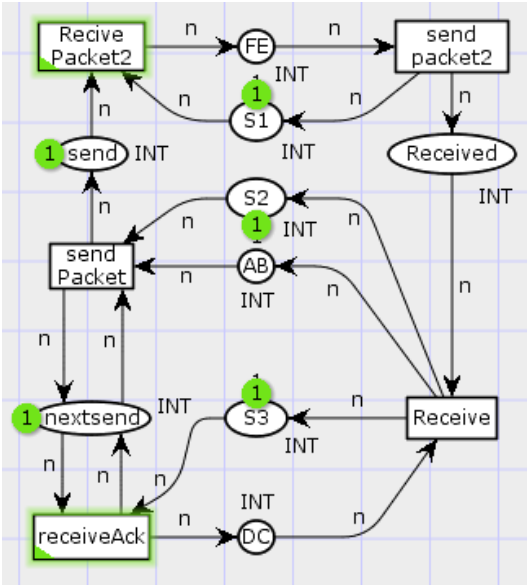


Рисунок 3.31 – Инверсия сети Петри, представленной на рисунке 3.27

Итак, на примере протокола передачи данных была реализована инверсия, заключающаяся в изменении ориентации дуг между вершинами сети. Проверка корректности преобразования реализовывалась через построение разметки графа состояния исходной сети и инвертированной. Сравнение полученных результатов показало, что сеть инвертирована корректно. Помимо инверсии самой сети предлагается реализовать инверсию графа состояний. Стоит отметить, что значительным преимуществом является однозначность преобразования графа состояний, это может быть полезно для систем со сложной структурой сети. Но восстановление сети из графа состояний занимает некоторое время, но трудности не вызывает. Таким образом, приходится делать выбор между сложностью преобразования сети и временными ресурсами на восстановление системы из инвертированного графа состояний.

3.6. Матричное представление сетей Петри. Разработка приложения, преобразующего комбинацию мест и переходов маркированных сетей Петри в матричную форму

В данном разделе приводится программное приложение [32, 71], преобразующее сети Петри, проектируемые в программной среде CPN Tools (version 3.4.0*)*, из графической формы в матричную. Матричная форма представления сетей Петри $\left(P, T, D^{-}, D^{+} \right)$, предложенная в (Дж. Питерсон [87]), эквивалентна стандартной форме $N = \left(P, T, F, m_I \right)$ и позволяет дать определения в терминах векторов и матриц. Отличие заключается лишь в появлении двух матриц D^{-} и D^{+}, представляющих входную и выходную функции [69, 75]. Полученные матрицы анализируются через задаваемые векторы запусков.

Продемонстрируем **матричное представление и анализ сетей Петри** на примере **интернет-магазина.** Приведем входную функцию и выходную функцию (приложение Д) интернет-магазина (рисунок 3.11). Для построения матриц и векторов в данной работе использовалась система компьютерной алгебры *MathCAD* (*version 14.0.0.163*). Входная функция сети Петри интернет-магазина выглядит следующим образом:

$$D^{-} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ \vdots \\ 36 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & & 22 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & & 0 \\ & & & & \ddots & \\ 0 & 0 & 0 & 0 & & 1 \end{pmatrix} \end{array}. \qquad (3.1)$$

Составная матрица изменений $D = D^{-} - D^{+}$ имеет вид (приложение Д). Важно заметить, что составная матрица изменений полностью не отображает структуру проектируемой сети, но в совокупности с входной и выходной функциями способна отразить спроектированную ранее сеть. Кроме того, составная матрица изменений необходима для последующего анализа сетей, а точнее для анализа возможных состояний сети. Зададим вектор начальной маркировки *M*:

$$M = (1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0), \quad (3.2)$$

и несколько возможных вариантов срабатывания переходов, а после оценим полученные результаты. Для первого варианта вектор запусков F_1 выглядит следующим образом:

$$F_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \dots & 22 & 23 & 24 & \dots & 29 & 30 & 31 & 32 & 33 & 34 & 35 & 36 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \quad (3.3)$$

Для того чтобы рассчитать получившееся состояние необходимо к начальной маркировке прибавить произведение вектора запусков и составной матрицы изменений $M_1 = M + (F_1 \cdot D)$:

$$M_1 = (0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0). \quad (3.4)$$

Вычисление маркировки через использование вектора запуска, еще для трёх случаев показано в приложении Д. Стоит отметить, что при некоторых вариантах векторов запуска система попадает в состояния (M_2, M_4), которые невозможно достичь, так как отсутствует информация о последовательности срабатываний переходов в векторе запуска.

Разработаем приложение, которое преобразует моделируемые в CPN Tools сети Петри из графической формы в матричную. Файл с расширением *.cpn*, полученный в программной среде CPN Tools, имеет тегообразную структуру, т.е. разбит на блоки, каждый из которых отвечает за определенную часть сети Петри, построенной пользователем с использованием графического интерфейса [71].

Приведем пример содержания сохраняемого файла CPN Tools:

<cpnet> <globbox> <block id=”ID1412310166”> <place id=”ID1420720921”> <posattr x=”-107.000000” y=”167.000000”/> <fillattrcolour=”White” pattern=”” filled=”false”/> <lineattrcolour=”Black” thick=”1” type=”Solid”/>.

С целью получения информации по конфигурации конкретной сети, отбираются следующие параметры: идентификационный номер элемента сети, порядковый номер каждого элемента. Во избежание дополнительных вычислительных затрат в представленной программе считывание файла происходит дважды. Во время первого прохода определяется количество элементов и наличие связей между ними, что необходимо для правильной работы динамического выделения памяти. При втором чтении файла происходит считывание конкретной информации о

конфигурации сети. На основе информации, полученной из файла формата *.cpn*, производится формирование матриц для возможности дальнейшей обработки сети Петри другими программами, так как представление сети в виде матриц легче интерпретируется цифровой техникой.

Рассмотрим функционирование разработанного приложения на примере простой сети Петри (рисунок 3.32). Стоить отметить, что все названия мест и переходов заканчиваются соответствующим порядковым номером, который был присвоен разработчиком.

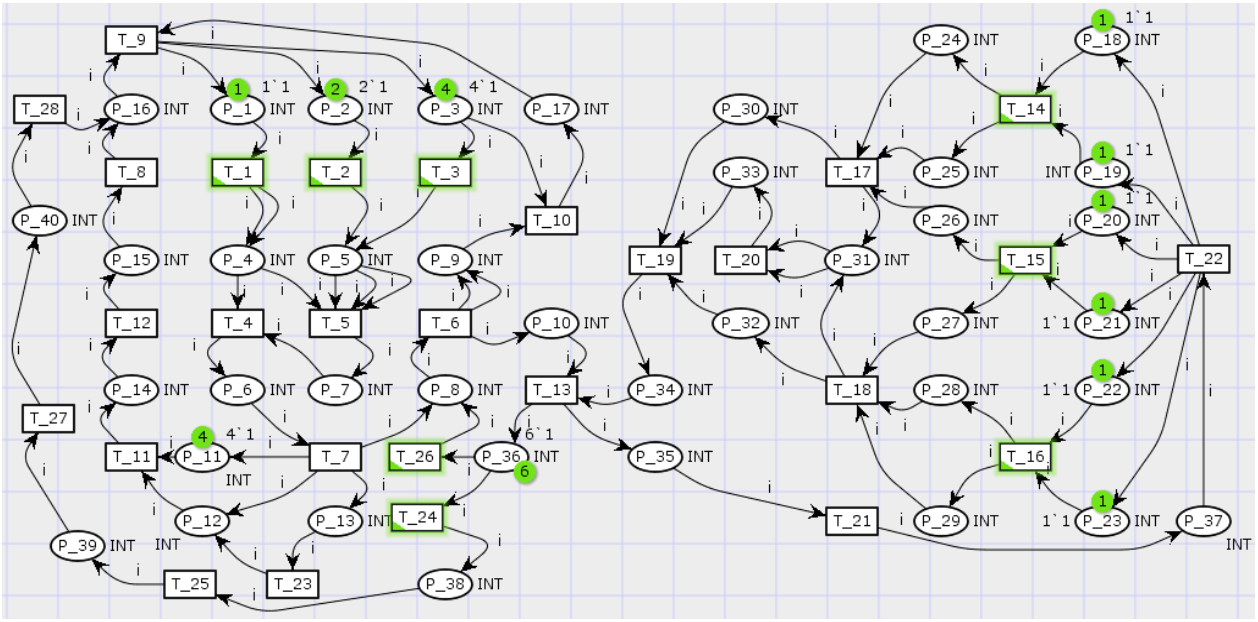


Рисунок 3.32 – Простая сеть Петри [71]

Приложение выполняет алгоритм представления и моделирования на основе матриц – вычисляются матрицы D^-, D^+, D и векторы начального состояния и достигнутой маркировки (приложение Д, рисунки Д.9-Д.11)

Таким образом, разработано программное приложение, преобразующее графическую форму сетей Петри, спроектированных в программной среде CPN Tools (version 3.4.0), к матричному виду с последующим анализом, предложенным Дж. Питерсоном [87]. Матричный подход к анализу сетей Петри весьма перспективен, но, в свою очередь, имеет и некоторые недостатки. Так, например, матрица D не полностью отражает структуру сети Петри, отсутствуют указания порядка срабатывания элементов в векторе запуска. Тем не менее, представление сети в виде матриц легче интерпретируется вычислительной техникой, чем представление с использованием комбинации мест и переходов, и требует более глубокого изучения.

3.7. Выводы

Предложены способы исследования сетей Петри и пространств состояний, при которых происходит анализ отдельных сценариев системы и различных частей пространства состояний. Разработано приложение, анализирующее сеть Петри на основе матриц входной и выходной функций, а также результирующих матриц и векторов начальной и полученной маркировки.

Один из способов заключается в построении UML диаграмм последовательности отдельных сценариев работы с последующим их анализом на основе сетей Петри. Пространство состояний данных сетей при необходимости объединяются в одно. На данном этапе появляется проблема, связанная с размерностью состояний для разных сценариев, которая решена заданием общего количества элементов маркировки для всей системы и каждого отдельного сценария. Неиспользуемые разряды получают нулевое значение. Предложен и протестирован способ, который заключается в разделении всей системы на иерархические уровни с последующим анализом полученных подсетей. Если в подсетях не найдены ошибки, то они и отсутствуют во всей сети. При наличии связей между подсетями следует внести дополнительные критерии анализа, заключающиеся в исследовании подсетей с несколькими входными (для данных подсетей начальных) состояниями. Предложен способ анализа отдельных фрагментов пространств состояний с проверкой следующих свойств: ограниченность, безопасность, но необходима проверка, выбранной в качестве начальной, маркировки.

Проверка достижимости маркировки, с которой начинается анализ выбранной части, реализована с использованием инверсии сети Петри. Действия при данной процедуре заключаются в смене направления взаимосвязей между вершинами ординарной сети, для простых сетей предлагается воспользоваться предложенными правилами инверсии. Также описана возможность инверсии графа состояний как однозначной процедуры, заключающейся в изменении направлений взаимосвязей между вершинами. К недостатком данного преобразования относится необходимость полного восстановления сети из инвертированного графа с использованием всех маркировок.

4. ПРИМЕНЕНИЕ МЕТОДИКИ СОВМЕСТНОГО ИСПОЛЬЗОВАНИЯ UML ДИАГРАММ И СЕТЕЙ ПЕТРИ ПРИ ПРОЕКТИРОВАНИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

На основе модифицированной методики совместного использования UML диаграмм и сетей Петри проектируется программное обеспечение для автоматизированной системы регулирования температуры обжига железорудных окатышей и автоматической системы управления технологическим процессом (ТП) водоснабжения. Проектирование ТП с использованием упомянутых инструментов рассматривается в работах[41] [56, 94].

При проектировании ПО для автоматизированной системы обжига окатышей учитываются управляющие, управляемые и возмущающие величины, рассчитываются коэффициенты, необходимые для регулирования температуры в зоне обжига печи. На основе описания технологического процесса строится диаграмма прецедентов, диаграмма классов и диаграмма объектов. В *п. 4.2* проектируется динамическая часть разрабатываемого ПО на основе UML диаграммы активности и сети Петри. Используя дискретизированные аналоговые величины, в сетях Петри моделируется регулятор поддержания температуры, который отслеживает её отклонения от номинальных значений и регулирует в зависимости от величины отклонения.

В следующем разделе проектируется ПО для АСУ ТП водоснабжения на основе методики совместного использования UML диаграмм и сетей Петри. Диаграмма активности преобразуется в эквивалентную сеть Петри, в которой поддержание регулируемых величин системы водоснабжения происходит с использованием дискретных величин. Приводится зависимость времени достижения выбранного количества состояний для локальных водонапорных станций: количество состояний всей системы превышает значение порядка 10^6.

В *п 4.4* получена матричная форма сетей Петри с использованием разработанного приложения на примерах управляемого светофора, двухсимочного телефона и взаимодействия пользователя с банкоматом.

---

[41] Бандман, М.К. Территориально-производственные комплексы [Текст] / М.К. Бандман, О.Л. Бандман, Т.Н. Есикова. – Новосибирск: Наука. Сиб. отд-ние. – 1990. – 303 с.

Дорофеев, Р.С. Модели структурного описания объектов для оценки их качества [Текст]: дис. … канд. техн. наук: 05.13.01 / Р.С. Дорофеев. – Иркутск, 2014. – 139 с.

4.1. Проектирование программного обеспечения для системы автоматизации обжига окатышей: разработка модели

Предлагается ***применение методики*** проектирования ПО для системы автоматизации обжига железорудных окатышей, а именно поддержания температуры в зоне обжига печи [84, 86]. Приводится описание ТП обжига окатышей с детализаций по технологическим зонам, что отображается на диаграмме прецедентов, диаграмме классов и диаграмме объектов. При создании автоматизированной системы для данной зоны учитываются управляющие, управляемые и возмущающие величины, рассчитываются коэффициенты поддержания температуры и подачи газа для реализации функции регулирования температуры в зоне обжига на основе статических характеристик по управлению и возмущению.

4.1.1. Описание технологического процесса подготовки железорудных окатышей. Горно-обогатительное предприятие АО «ССГПО» состоит из нескольких карьеров по добыче железной руды, участка дробления и сухой магнитной сепарации, участка мокрой магнитной сепарации, участка по производству железорудных окатышей. На рисунке 4.1.а представлена схема цепи аппаратов производства концентрата и окатышей, начиная с добычи руды и заканчивая отгрузкой готовой продукции. Рассмотрим более детально процесс обжига железорудных окатышей, который является одной из самых важных и сложных стадий по производству окатышей. Основная цель обжига – получение окатышей с оптимальными металлургическими свойствами: максимальное понижение уровня серы и придание им прочности за счет поддержания определенного температурного режима.

В процессе обжига окатыши проходят пять технологических зон: сушка, подогрев, обжиг, рекуперация, охлаждение (рисунок 4.1.б). В каждой зоне поддерживают определенный температурный (*temperatureZone*) и газовый режим, а скорость движения конвейера (*Conveyor*) изменяется от *1* до *3 м/мин* (*speed*), толщина слоя окатышей (*layerHeight*) на тележках конвейера (паллетах) составляет примерно *300 мм*. Первой технологической зоной в обжиговой машине (*Calcar*) является зона сушки, температура в которой составляет *350-400 °C*. Второй технологической зоной в обжиговой машине является зона подогрева с температурным режимом *700-1100 °C*. Из зоны подогрева окатыши поступают в зону обжига. Тем-

пературный режим в зоне устанавливают в соответствии с технологической картой для обжиговых машин (*1250-1270 °C*). Окатыши спекаются между собой при температуре *1300 °C*. Следовательно, незначительное превышение подачи газа над слоем в этой зоне заметно влияет на среднюю температуру всего слоя и как следствие на их качество, поэтому особенно важно выдерживать установленный температурный режим в зоне обжига. Следующей является зона рекуперации, температурный режим которой – *900-1000 °C*. В зоне охлаждения окатыши должны охлаждаться до температуры ниже *400 °C*.

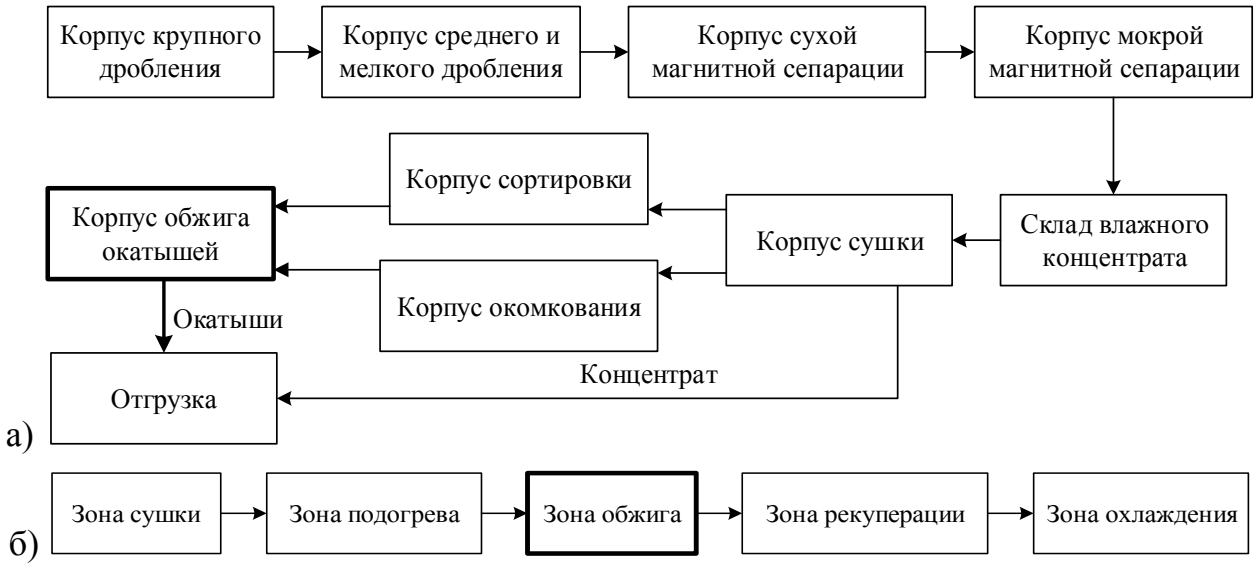


Рисунок 4.1 – Схемы производства окатышей: а) схема цепи аппаратов производства концентрата и окатышей; б) стадии прохождения окатышей в обжиговой машине

Обжиговая печь представляет собой сложный объект управления с множеством входных, выходных и режимных параметров, а типовые воздействия технологического объекта представлены на рисунке 4.2. Основными сигналами являются температура (*temperature*) в зонах обжига печи, удельная подача газа (*feedingGas*), расход воздуха (*feedingAir*) и подача окатышей (*feedPellets*).

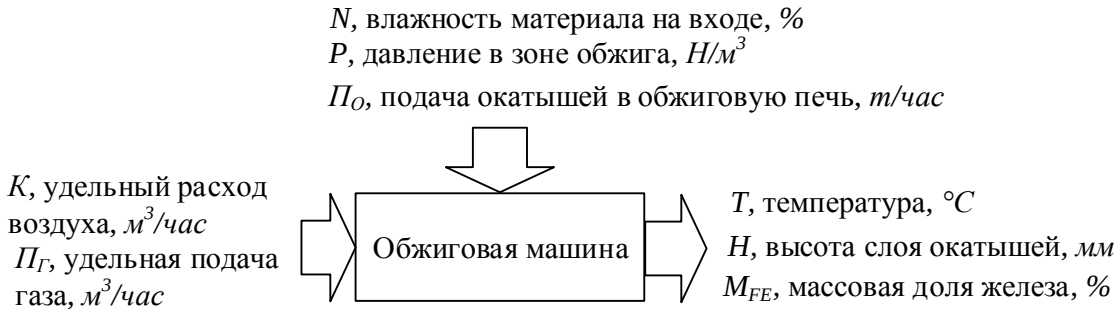


Рисунок 4.2 – Типовые сигналы обжиговой машины

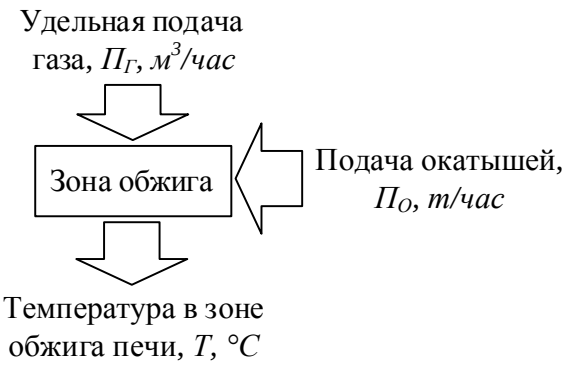


Рисунок 4.3 – Зависимость температуры в зоне обжига печи

Представим зависимость температуры в зоне обжига печи (рисунок 4.3), которая выглядит следующим образом:

$$T = K \cdot П_Г \,/\, П_О, \qquad (4.1)$$

где T – температура окатышей в зоне обжига печи, K – коэффициент, характеризующий объект управления, $П_Г$ – подача газа, $П_О$ – подача окатышей. Из (4.1) коэффициент, характеризующий объект управления, найдём следующим образом: $K = T \cdot П_О \,/\, П_Г$; $K = 245\ °C \cdot т \,/\, м^3$. Опираясь на данное значение, построим статические характеристики температуры в зоне обжига печи по управлению и по возмущению для определения соответствующих коэффициентов. На основе семейства статических характеристик по управлению, получим зависимость изменения температуры от изменения подачи газа, которая выглядит следующим образом:

$$\Delta T = K_У \cdot \Delta П_Г, \qquad (4.2)$$

где ΔT - изменение температуры в зоне обжига печи, *21.5 °C*, $\Delta П_Г$ - изменение подачи газа, *9 $м^3$/час*. Из (4.2) вычислим: $K_У = \Delta T \,/\, \Delta Г_{ПГ} = 2.38\ °C \cdot час \,/\, м^3$.

Опираясь на статические характеристики по возмущению, получим зависимость изменения температуры от изменения подачи окатышей, которая выглядит следующим образом:

$$\Delta T = K_В \cdot \Delta П_О, \qquad (4.3)$$

где ΔT - изменение температуры в зоне обжига печи, *26 °C*; $\Delta П_О$ - изменение подачи окатышей, *2 т/час*. Из (4.3) находим: $K_В = \Delta T \,/\, \Delta П_О = 13\ °C \cdot т \,/\, м^3$.

4.1.2. Разработка диаграмм. *Проектирование диаграммы прецедентов* осуществляется на основе технологического процесса, который описан выше. На диаграмме прецедентов (рисунок 4.4), описывающей функционирование ТП обжига окатышей, представлено три актора: администратор (*Administrator*), сервис-инженер (*Service*) и оператор системы (*Operator*). Администратор добавляет (*AddUser*), изменяет (*ChangeUserProfile*) и удаляет (*RemoveUser*) пользователей, просматривает их полный список (*ViewUserList*), а также профиль каждого пользова-

теля (*ViewUserProfile*). Сервис-инженер может просматривать свой профиль, а также устраняет аварийные режимы (*EliminationAlarm*), которые фиксирует оператор. Оператор следит за поддержанием температуры в зонах обжиговой машины (*ViewTemperature*), контролирует подачу газа (*Gas*) и подачу воздуха (*Air*). При необходимости оператор изменяет состояние задвижек (открытие/закрытие) (*TurnOn/OffValve…*), отвечающих за подачу воздуха или газа. При возникновении аварийной ситуации (*SeeAlarm*) и отсутствии автоматического закрытия задвижек оператор в аварийном режиме изменяет их состояние (*AlarmOffValveAir/Gas*), после чего сообщает об аварии (*AckAlarm*), для её устранения сервис-инженером.

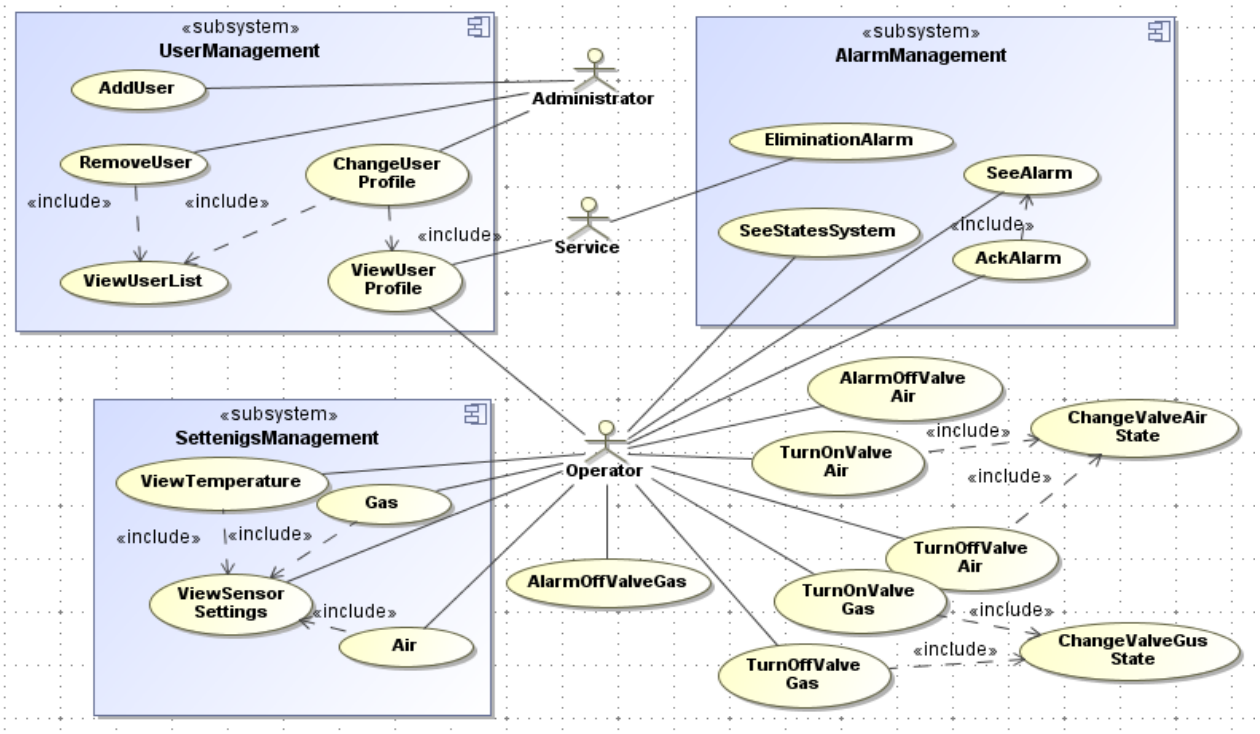


Рисунок 4.4 – Диаграмма прецедентов ТП обжига окатышей

Диаграмма классов – это следующий этап проектирования статических свойств системы (рисунок 4.5), на которой представлено семь классов. *Periphery* отвечает за сбор значений различных датчиков (*GetPinValue*). *SystemInputOutput* – система ввода/вывода предназначена для установления значений устройствам и вывода оперативной информации. *Calcar* – в обжиговой печи необходимо стабильно поддерживать нужную температуру (*temperature*). В *Conveyor* поддерживаются следующие величины: скорость (*speed*), высота слоя окатышей (*layerHeight*), подача окатышей (*feedPellets*). При помощи заслонок *(Valve)* регулируется подача газа и воздуха (*feeding*) для поддержания оптимальной температуры.

Sensor – устройство необходимое для установки значений (*min/max*). Класс *System* выполняет запуск (run) и останов (*stop*) всего процесса.

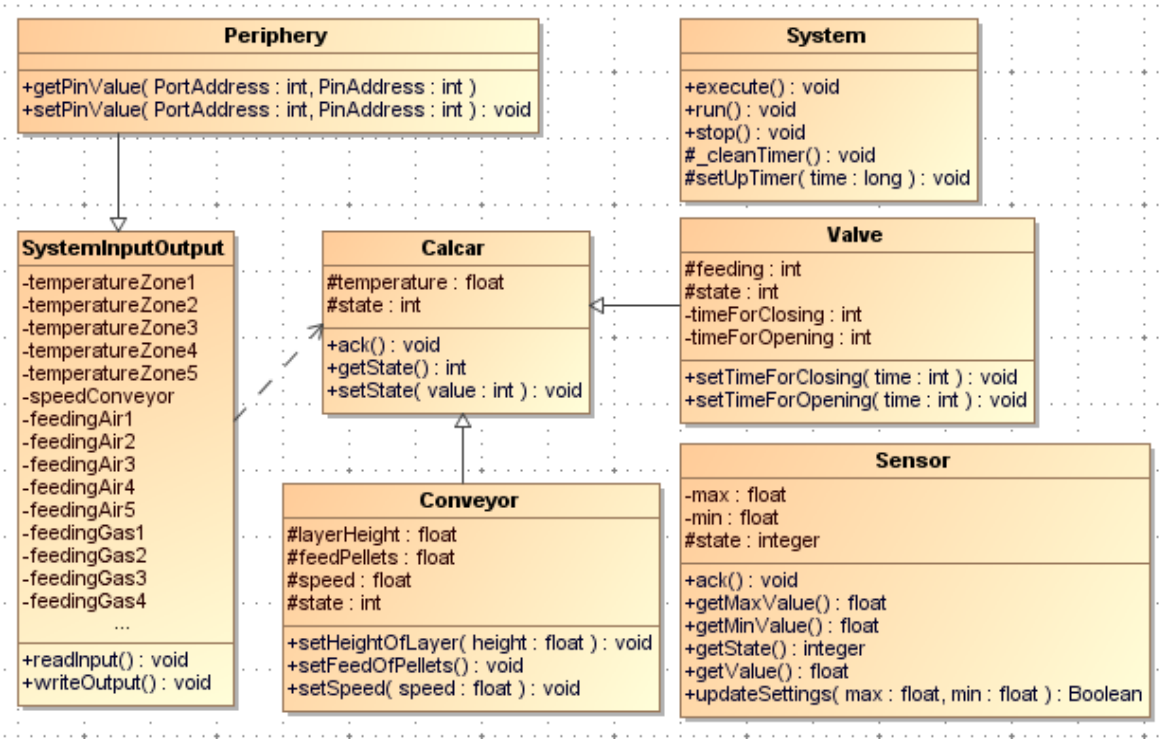


Рисунок 4.5 – Диаграмма классов ТП обжига окатышей

Класс *SystemInputOutput* устанавливает значения температуры (*temperatureZonen*), открытие воздушных (*feedingAirn*) и газовых (*feedingGasn*) заслонок для каждой зоны обжиговой машины. А также считывает (*readInput*) и выводит (*writeOutput*) оперативную информацию.

К разработке диаграммы объектов приступают после создания диаграммы классов. Полученная диаграмма (рисунок 4.6) состоит из 15 элементов, описывающих каждую из зон обжиговой печи, конвейер, заслонки для подачи воздуха и газа. Объекты, соответствующие каждой зоне обжига, имеют одинаковые параметры: состояние (*state*), максимальную (*temperatureMax*) и минимальную температуру (*temperatureMin*). Объекты класса *Valve*, газовые (*Gas*) и воздушные (*Air*) задвижки, имеют параметры состояния (*state*), времени на открытие (*timeForClosing*) и закрытие (*timeForOpening*) задвижек. Параметры объекта *Conveyor* следующие: подача окатышей (*feedPellets*), высота слоя окатышей (*layerHeight*), скорость (*speed*) и его текущее состояние (*state*). Уделим большее внимание поддержанию температуры в зоне обжига *1250-1270°C* (*temperatureMin/Max*) и рассмотрим её динамические свойства.

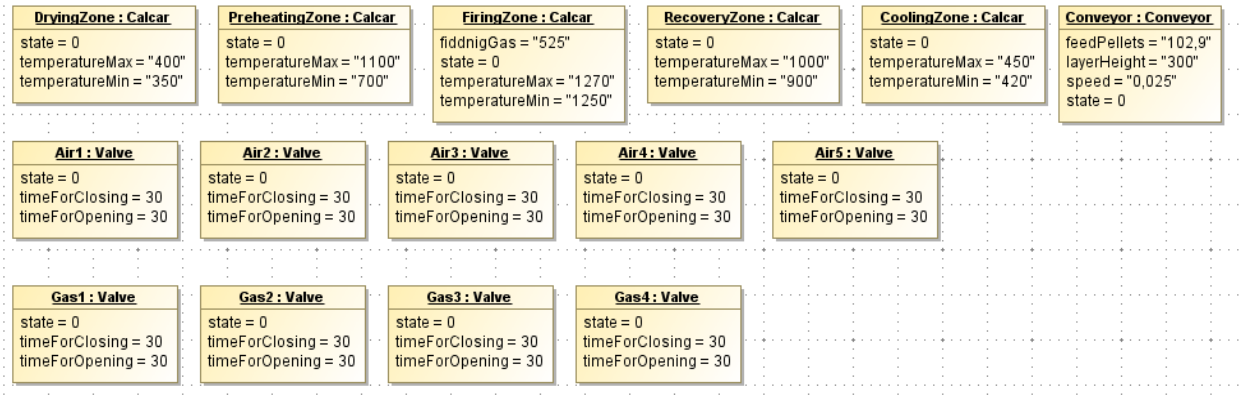


Рисунок 4.6 – Диаграмма объектов технологического процесса обжига окатышей

Таким образом, на основе управляющих воздействий, управляемых и возмущающих величин, полученных статических характеристик и рассчитанных коэффициентов поддержания температуры и подачи газа спроектированы диаграмма прецедентов (акторы: *Administrator, Service, Operator*), диаграмма классов (*Periphery, SystemInputOutput, Calcar, Conveyor, Valve, Sensor, System*) и диаграмма объектов, состоящей из 15 элементов.

4.2. Проектирование программного обеспечения для системы автоматизации обжига окатышей: моделирование поддержания температуры в зоне обжига печи

Динамические свойства проектируемой автоматизированной системы обжига окатышей отражаются на диаграмме активности, которая транслируется в соответствующую сеть Петри. Для данной системы описание регуляторов происходит с использованием переходов сети Петри, нагруженных имитационным временем с целью учесть реальные условия в модели [19, 21, 48].

Спроектировав структурные диаграммы (диаграмма прецедентов, диаграмма классов, диаграмма объектов), приступим к созданию ***диаграммы активности*** (рисунок 4.7). На данной диаграмме проверяют температуру в зоне обжига печи (*Checking temperature*), которая должна соответствовать диапазону *1250-1270 °C*. Процесс проверки температуры повторяют при оптимальном режиме работы, а при отклонении параметров температуры происходит проверка значений подачи газа (*Checking Feeding Gas*) и подачи окатышей (*Checking Feeding pellets*). Если данные значения отклоняются от нормы, их регулируют (*Change Feeding Gas/pellets*), после чего измеряют температуру и цикл поддержания за-

данных значений повторяется. Полученную диаграмму активности транслируем в соответствующую ей сеть Петри.

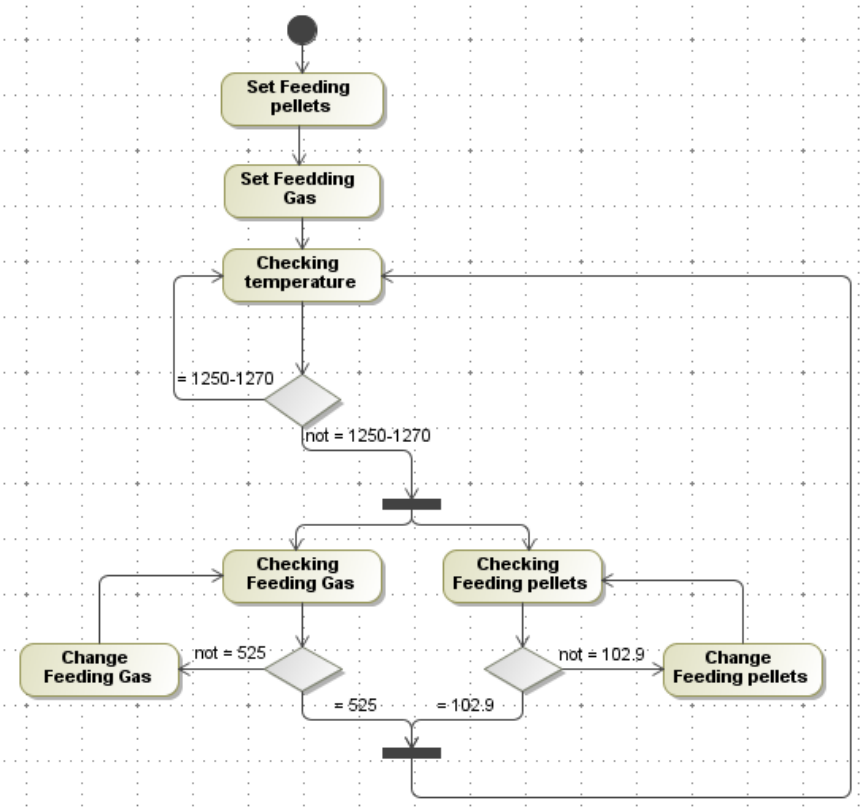


Рисунок 4.7 – Диаграмма активности регулирования температуры в зоне обжига печи (итерация 1)

Важно заметить, что после преобразования сеть Петри не отражает многих особенностей данного технологического процесса – по ней можно лишь отследить последовательность действий и проверку значений с последующим при необходимости их исправлением. Поэтому скорректируем сеть Петри и отразим необходимые технологические процессы (рисунок 4.8).

К переходу *Change corner* добавим полученную из фактических номинальных данных функцию: $t1 / 2.38$, которая задает подачу газу относительно температуры в зоне обжига печи (4.2). К переходу *Interaction* также добавляется функция: $245.0 \cdot g / p$ согласно (4.1). На второй итерации сеть состоит из восьми мест *P*, пяти переходов *T* и начальной маркировки m_I:

$$P = \{ Temperature\ of\ \ Zone3,\ Gas,\ Feeding\ \ pellets,\ temperatureZone3,$$
$$Regulator,\ Change\ Fidding\ \ pellets,\ Checking1,\ Checking2 \}$$
$$T = \{ Interaction,\ Regulation1,\ Regulation2,\ Change\ corner,$$
$$Change\ Fidding\ \ pellets \}$$
$$m_I = (1, 1, 1, 1, 0, 1, 0, 1).$$

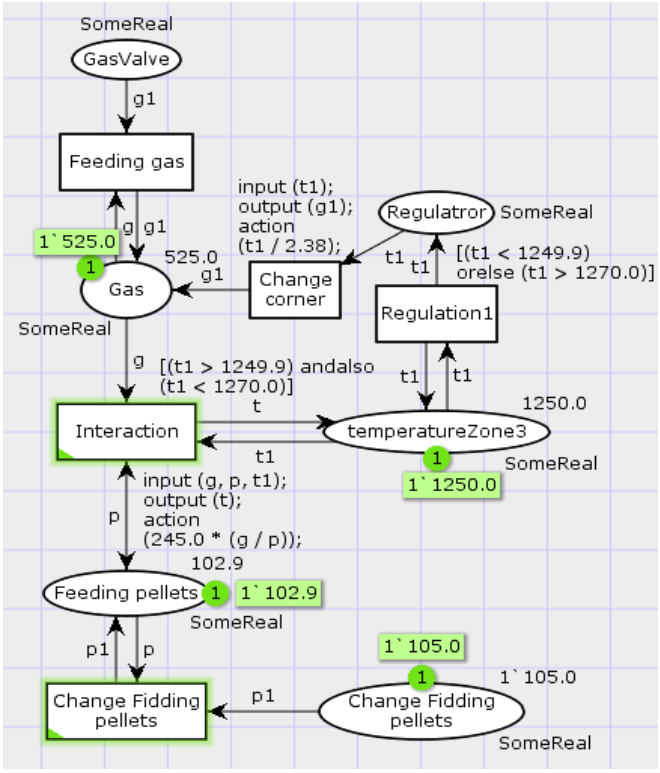


Рисунок 4.8 – Сеть Петри, соответствующая диаграмме активности регулирования температуры в зоне обжига печи (итерация 1)

На второй итерации проектирования к переходам сети Петри добавим временную составляющую[42] (рисунок 4.9) [17, 22, 27, 28] и приоритет на срабатывание перехода *Change corner*, что позволит данному переходу срабатывать в первую очередь относительно всех переходов сети.

У перехода *Change corner* изменим функцию: $g+(t_2/2.38)$, с целью добавить приращение подачи газа на необходимую величину при поддержании оптимального температурного режима.

[42] Для представления временных сетей Петри используют *имитационное время*, а не время реального мира. При использовании имитационного времени к переходам добавляется время в виде целочисленного значения, которое является глобальным для всей сети и символизирует выбранную единицу времени (сутки, часы, минуты, секунды и т.д.). При срабатывании перехода, нагруженного временем, глобальное значение времени увеличивается на указанную величину. Формально, в какой момент времени сработает переход, например, переход *Interaction* на рисунке 4.9 увеличивает общее имитационной время сети на 47 секунд: @+47.

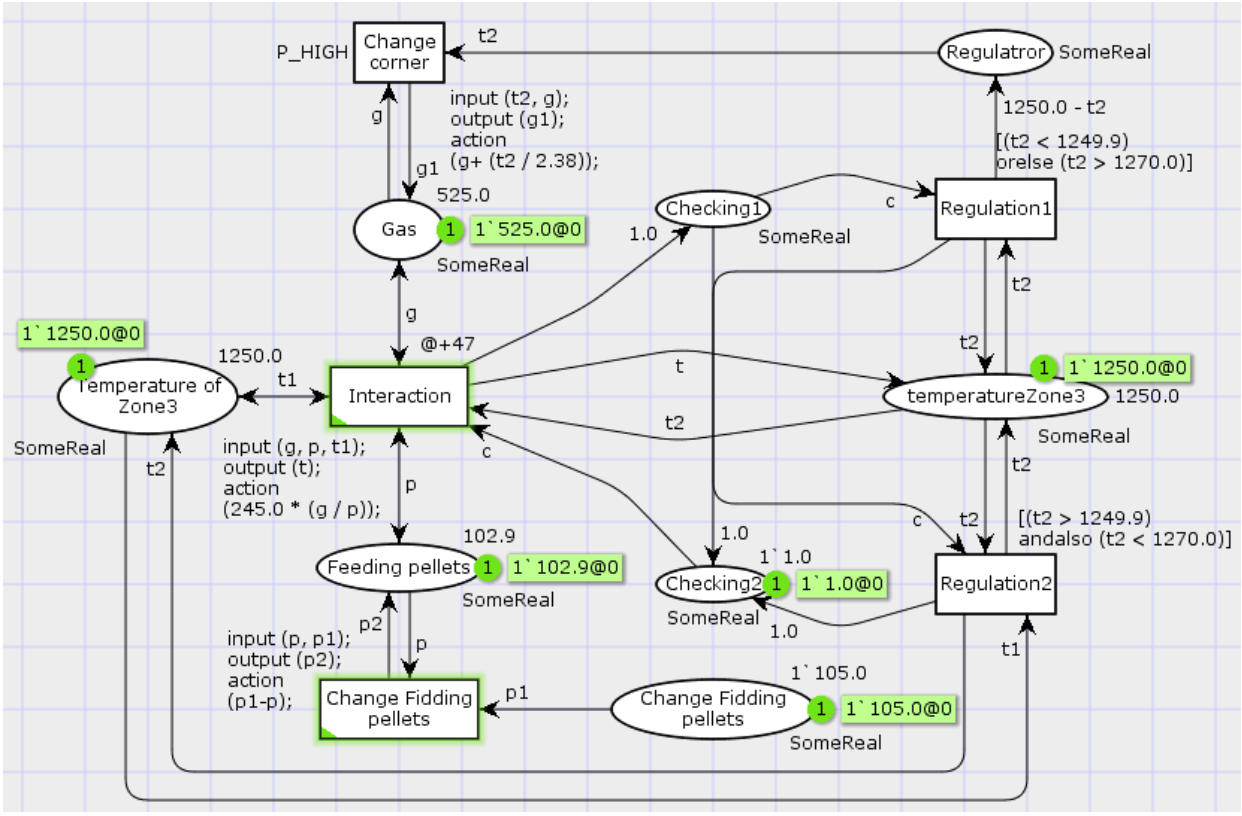


Рисунок 4.9 – Сеть Петри, соответствующая диаграмме активности регулирования температуры в зоне обжига печи (итерация 2)

На третьей итерации получаем сеть Петри (рисунок 4.10), в которой моделируется поддержание необходимого уровня температуры в зоне обжига печи. Добавим к сети шесть переходов *Change corner_n*, для каждого из которых установим условие срабатывания. Поскольку в программной среде CPN Tools нет возможности добавить условие на время, разграничим температурное отклонение на несколько уровней: $[0;\pm5);[\pm5;\pm10);[\pm10;\pm15);[\pm15;\pm30);[\pm30;\pm60);[\pm60;\pm100);[\pm100;\pm150)$.

К переходу *Change Fidding pellets* добавим условие

$$(p1 > 94.9) \wedge (p1 < 115.1),$$

которое позволяет изменить возмущающее воздействие на систему в пределах подачи окатышей от *94.4 м*^{3} до *115.1 м*^{3}. При большей подаче окатышей может произойти завал конвейера, а при меньшей – выпекание железорудных окатышей даже при номинальной подачи газа. При изменении подачи окатышей вне установленных пределов, следует вывод о некорректности работы других систем обжиговой машины. После завершения проектирования сети Петри приступают к проверке её построения посредством программной генерации и анализа пространства состояний.

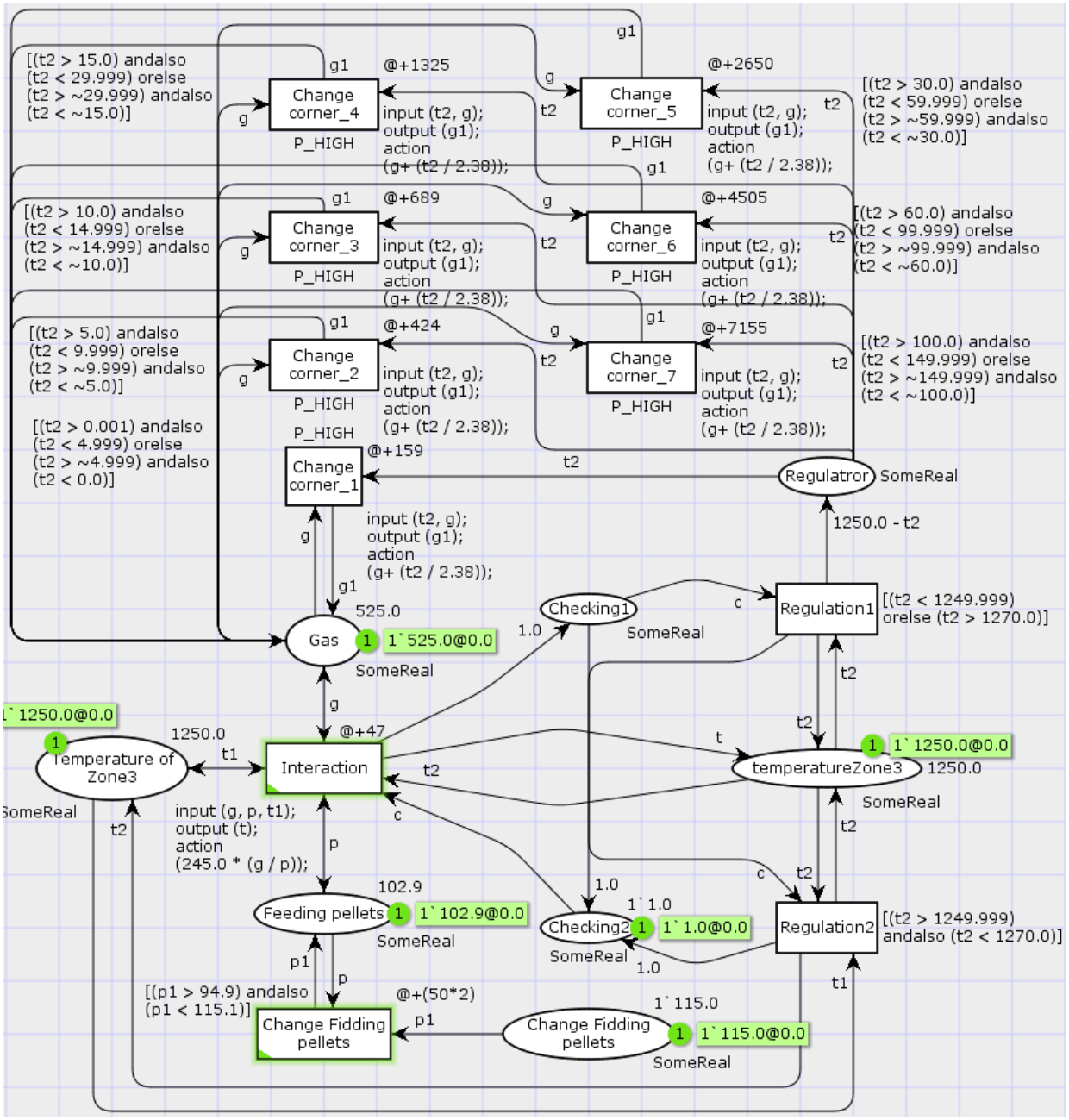


Рисунок 4.10 – Сеть Петри, соответствующая диаграмме активности регулирования температуры в зоне обжига печи (итерация 3)

Граф состояний данной сети Петри поддержания температуры в зоне обжига печи при одном возмущающем воздействии содержит 32 вершины и 36 взаимосвязей, в сети нет мёртвых переходов и зацикливаний (рисунок 4.11). На рисунке 4.11 приведены *32* состояния, каждое из которых имеет три значения: верхнее означает порядковый номер состояния, левое нижние – количество входных взаимосвязей, правой нижнее – количество выходящих взаимосвязей. Чёрным цветом отмечено конечное состояние в пространстве состояний.

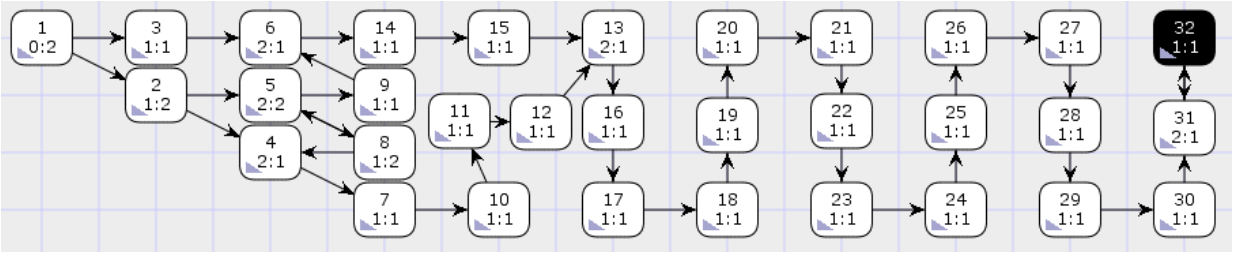


Рисунок 4.11 – Граф состояний сети Петри регулирования температуры в зоне обжига печи

Разберем несколько состояний более детально. Поскольку в сети присутствует восемь мест, то для каждого состояния разметка содержит восемь разрядов. В названии маркировки соответствующего места первым указывается название страницы, на которой представлена сеть Петри, в данном случае *"New_Page"*. Следующим является название места, далее разметка, соответствующая данному месту. Представим полную разметку первого и конечного состояния пространства состояний сети Петри регулирования температуры в зоне обжига печи:

1@0.0:
*New_Page'temperatureZone*3 1:1`1250.0@0.0
New_Page'Feeding_pellets 1:1`102.9@0.0
New_Page'Regulator 1: *empty*
New_Page'Gas 1:1`525.0@0.0
New_Page'Change_Fidding_pellets 1:1`115.0@0.0
*New_Page'Temperature_of_Zone*3 1:1`1250.0@0.0
*New_Page'Checking*1 1: *empty*
*New_Page'Checking*2 1:1`1.0@0.0

32@7202.0:
*New_Page'temperatureZone*3 1:1`1249.9982516@7202.0
New_Page'Feeding_pellets 1:1`115.0@7202.0
New_Page'Regulator 1: *empty*
New_Page'Gas 1:1`586.73461181@7202.0
New_Page'Change_Fidding_pellets 1: *empty*
*New_Page'Temperature_of_Zone*3 1:1`1249.99982516@7202.0
*New_Page'Checking*1 1:1`1.0@7202.0
*New_Page'Checking*2 1: *empty* .

После изменений функции у перехода *Change corner*, а также добавлении еще пяти переходов *Change corner_n* с соответствующим для каждого температурным режимом и других незначительных изменений, сеть Петри моделирует поддержание температуры в зоне обжига печи при возможном изменении вели-

чины подачи окатышей. Измененную сеть Петри преобразуем в UML диаграмму активности (рисунок 4.12).

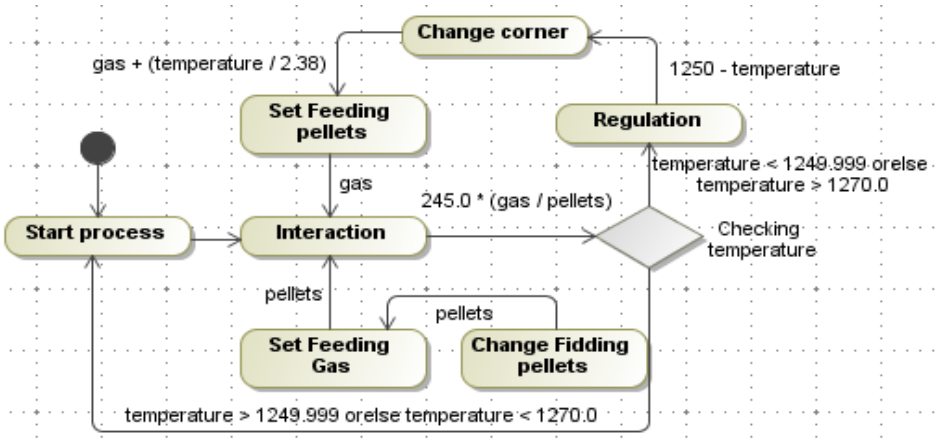


Рисунок 4.12 – Диаграмма активности технологического процесса обжига окатышей (итерация 2)

После корректировки диаграммы активности разработчик переходит к следующему этапу разработки программного обеспечения, а именно к написанию или генерации программного кода.

Таким образом, проектирование динамических свойств разрабатываемого ПО осуществлялось на основе диаграммы активности, которая транслировалась в сеть Петри для проверки корректности построения. Анализ полученной сети показал, что необходимо добавить ряд условий и функций к переходам, что позволило регулировать температуру в зоне обжига при изменении подачи окатышей. Переходы сети были нагружены имитационным временем с целью учесть реальные условия в модели. Используя семь переходов в сети Петри, спроектирован регулятор, поддерживающий номинальный температурный режим. Диаграмма активности была изменена согласно построенной в несколько итераций сети Петри. Полученный набор UML диаграмм готов к генерации программного кода.

4.3. Проектирование программного обеспечения для АСУ ТП водоснабжения: поддержание регулируемых величин

Используем ***методику*** проектирования ПО ***для АСУ ТП водоснабжения***, которая представляет собой комплекс, состоящий из определенного набора насосов, задвижек и резервуаров для очистки и хранения воды [33, 83]. Параметром регулирования выберем поддержание номинального значения давления в трубопроводе. Регу-

лирование давления в сетях Петри реализуем через систему переходов, изменяющих мощность двигателя насоса и имеющих несколько уровней дискретности. По сравнению с работой (Д.О. Романников [93]) предлагается моделирование статических свойств всей АСУ ТП водоснабжения. При моделировании динамических свойств локальных насосов в диаграмме деятельности и сети Петри используем дискретные значения в отличие от простой последовательности действий.

На первых этапах происходит перекачка воды *n* насосами (*WPumpn*) из скважин при открытых задвижках (*WValven*) до резервуаров (*Reservoirn*) (рисунок 4.13).

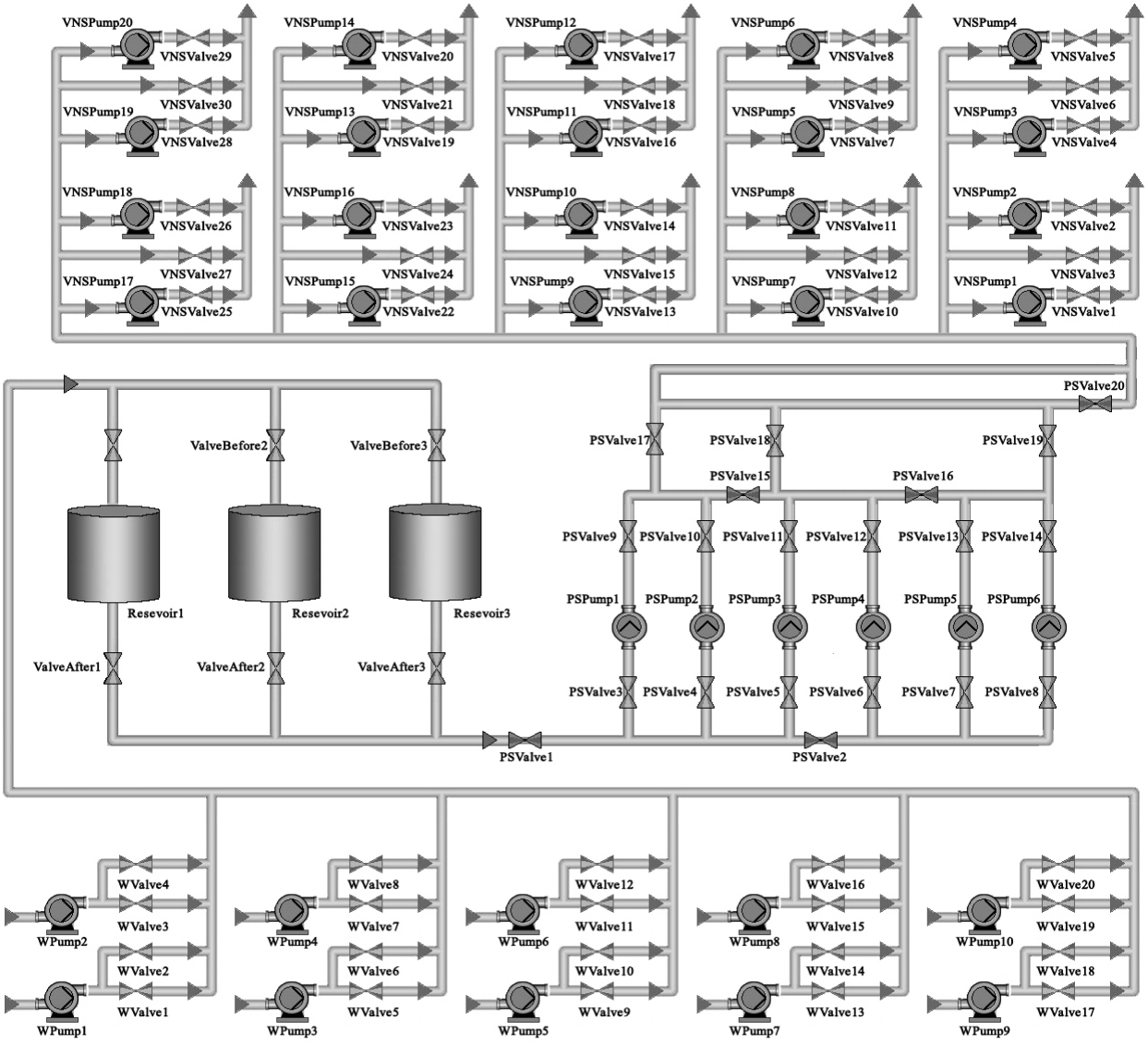


Рисунок 4.13 – Функциональная схема АСУ ТП водоснабжения

В резервуарах происходит очистка воды и дальнейший её перегон до насосной станции второго подъема, которая при помощи шести насосов (*PSPumpn*) перекачи-

вает воду до локальных насосных станций. Данной системой управляет оператор, который открывает/закрывает задвижки, включает/выключает насосы и следит за состоянием всех элементов (скважинные насосы, локальные насосы, задвижки на всех уровнях системы, резервуары с чистой водой и т.д.). В проектируемой системе необходимо поддерживать давление, равное пяти атмосферам. В трубопроводе локальной водонапорной станции при колебании давления необходима его корректировка увеличением или уменьшением мощности работы двигателей насосных станций. Функционирование технологического процесса происходит при поддержании и регулировании аналоговых величин, но в данном случае для создания ПО на этапе проектирование выберем несколько уровней дискретности для описания и моделирования величин, нуждающихся в регулировании: давление в трубопроводе, мощность работы двигателя насоса, а также состояние насосов, задвижек и резервуаров.

Во время проектирования **диаграммы прецедентов** (рисунок 4.14), способствующей лучшему пониманию всей системы, определены три актора: администратор, сервис-инженер, оператор.

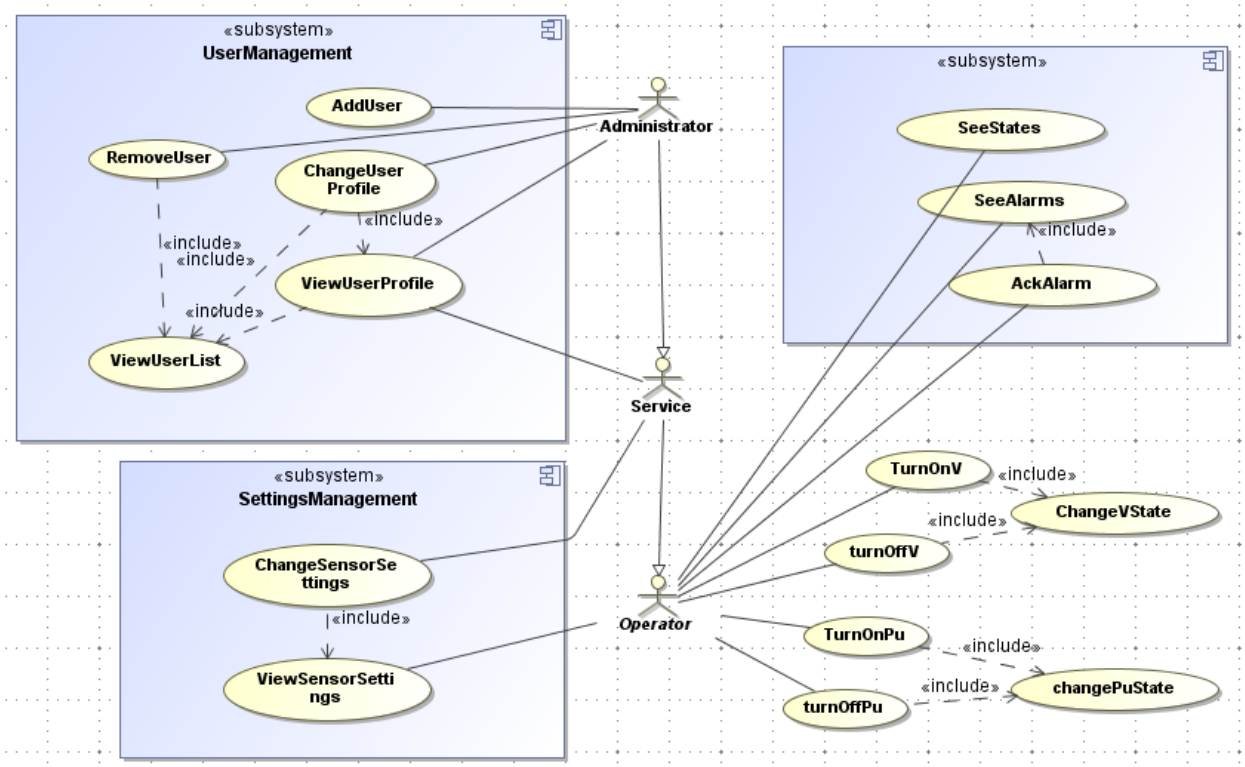


Рисунок 4.14 – Диаграмма прецедентов АСУ ТП насосной станции

Администратор добавляет, редактирует и удаляет профили пользователей, а также просматривает их полный список. Сервис-инженер имеет возможность просматривать свой профиль, а также задавать параметры для регулирования величин

объекта управления. Оператор контролирует параметры, которые поддерживает система, включает/выключает насосы, открывает/закрывает задвижки, а также контролирует состояние каждого элемента системы и при возникновении аварийной ситуации, обязан зафиксировать тревогу и вызвать специалистов для её устранения.

Диаграмма классов проектируется на следующем шаге. Данная диаграмма состоит из 12 классов, основными являются *PuVNS*, *ValveVNS*, родители которых – *Pu* и *Valve* со свойствами: значение давления и состояние (*pressure, state*), для *Pu* (насос), характерен параметр частоты (*freq*) (рисунок 4.15).

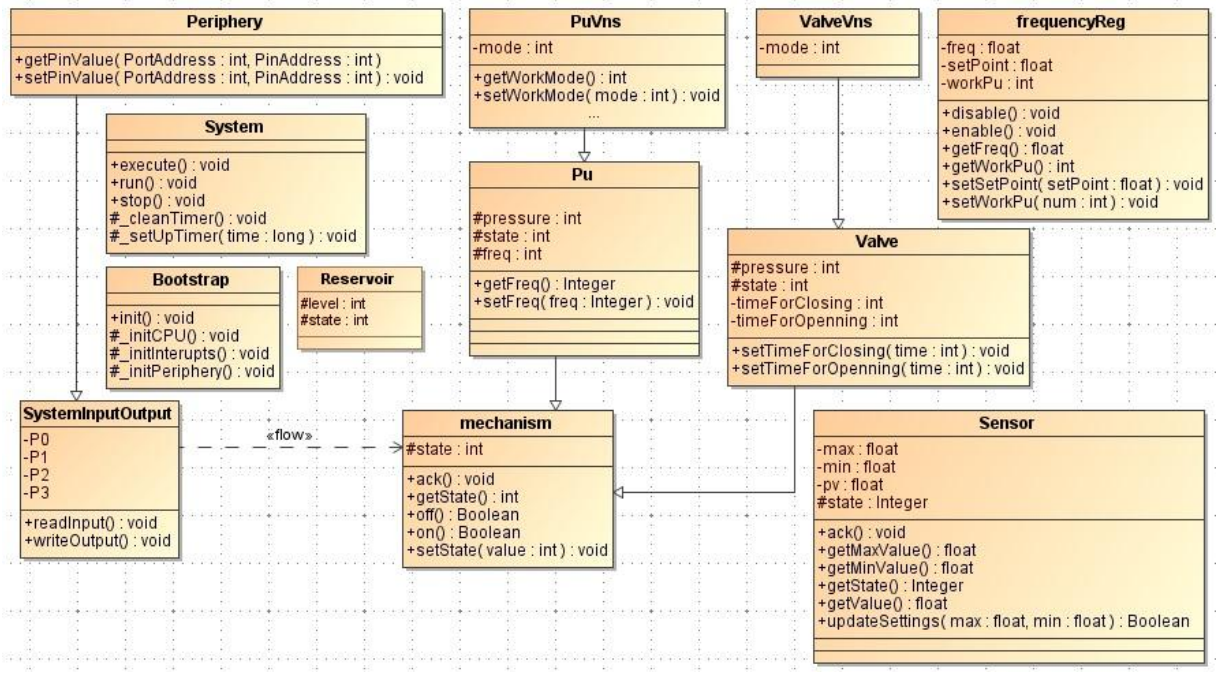


Рисунок 4.15 – Основная часть диаграммы классов АСУ ТП насосной станции

Методы данных классов нацелены на открытие и закрытие задвижек, а также на поддержание частоты работы двигателя насоса. Остальные классы служат для снятия и/или поддержания параметров периферийных устройств (поддерживание частоты работы двигателя – *frequencyReg*, снятие значений уровня наполнения резервуара – *Resrvoir*, взаимосвязи элементов при помощи класса Periphery и т.д.). Необходимость в детальной проработке отдельных классов отсутствует, поэтому после составления диаграммы классов следует этап построения диаграммы объектов.

Так как вся система содержит большое количество элементов, следовательно, и **объектов**, приведём по одному экземпляру каждой группы схожих объектов (рисунок 4.16): *WPump1* – насос №1, находящийся на скважинах с показателями давления (*pressure*), частоты работы (*freq*) и состояний (*state*) двигателя; *PSPump1* – насос

№1, находящийся на водонапорной станции; *VNSPump1* – насос №1, находящийся на локальной водонапорной станции; *WValve1* – задвижка №1, находящаяся на скважинах с установленным временем на открытие (*timeForClosing*) и закрытие (*timeForOpening*) задвижек и т.д. Свойства одного объекта в каждой группе объектов идентичны свойствам других объектов в группе.

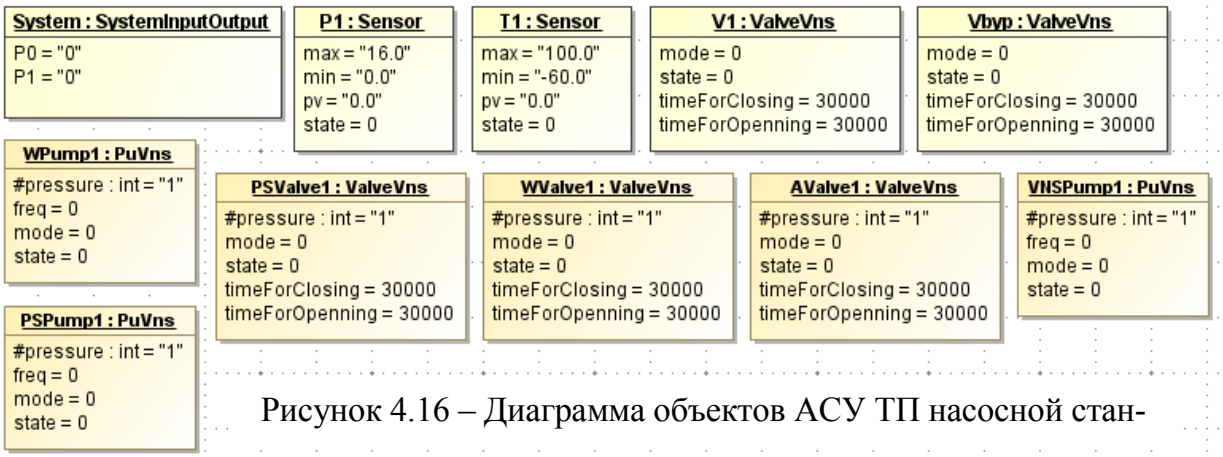


Рисунок 4.16 – Диаграмма объектов АСУ ТП насосной стан-

На следующем шаге проектируем **диаграмму активности** локальной водонапорной станции [32] (рисунок 4.17).

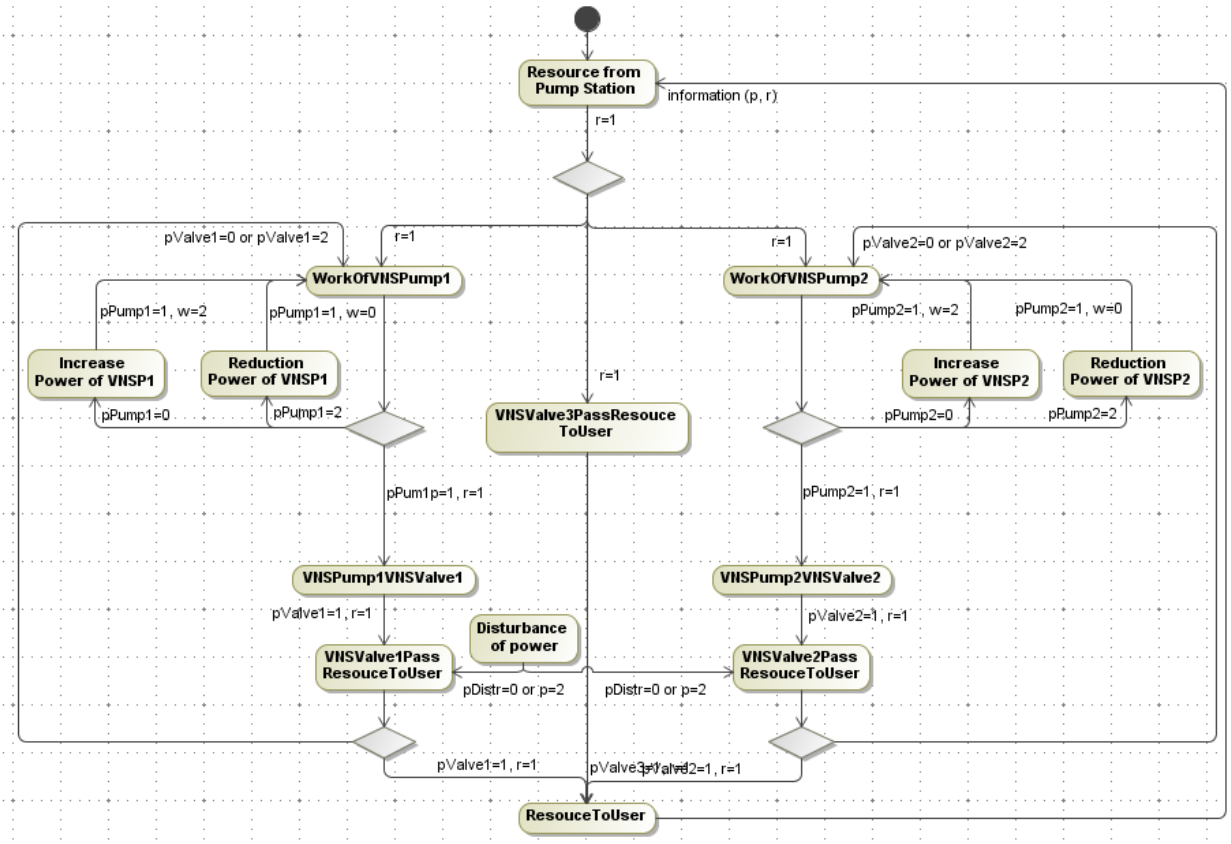


Рисунок 4.17 – Диаграмма активности АСУ ТП локальной насосной станции

Локальная водонапорная станция состоит из двух насосов (*VNSPumpn*) и трех задвижек (*VNSValven*). Насосы поочередно включаются (*WorkOfVNSPumpn*) – при поломке одного из них запускается другой. Если произошло отключение обоих насосов, то происходит открытие третьей задвижки и вода самотёком поступает до части потребителей. При влиянии возмущающего фактора (*Disturbance of power*), происходит корректировка (*Increase/Reduction of VNSPn*) частоты работы двигателя насоса для нормализации давления в трубопроводе.

Сеть Петри (рисунок 4.18, 4.19), которая состоит из девяти основных мест *P*, 16-ти переходов *T* с начальной маркировкой m_I, получаем из соответствующей диаграммы активности с целью проверки корректности её построения:

$P = \{ FromPS, VNSPump1, VNSPump2, VNSValve1, VNSValve2, VNSValve3, RtoUser1, R1toUser1, R2toUser1, User1, Distrubance \}$;
$T = \{ PStoVNSP1, PStoVNSP2, PStoVNSV3, VNSP1toVNSV1, VNSP2\ to, VNS2, VNS1toU, VNS2toU, VNS3toU, Controller\ P1, Controller\ P2, P1RtoUser1, RtoUser1, P2RtoUser1, Power\ below of\ VNSP1, Power\ above of\ VNSP1, Power\ below of\ VNSP2, Power\ above of\ VNSP2 \}$;
$m_I = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$.

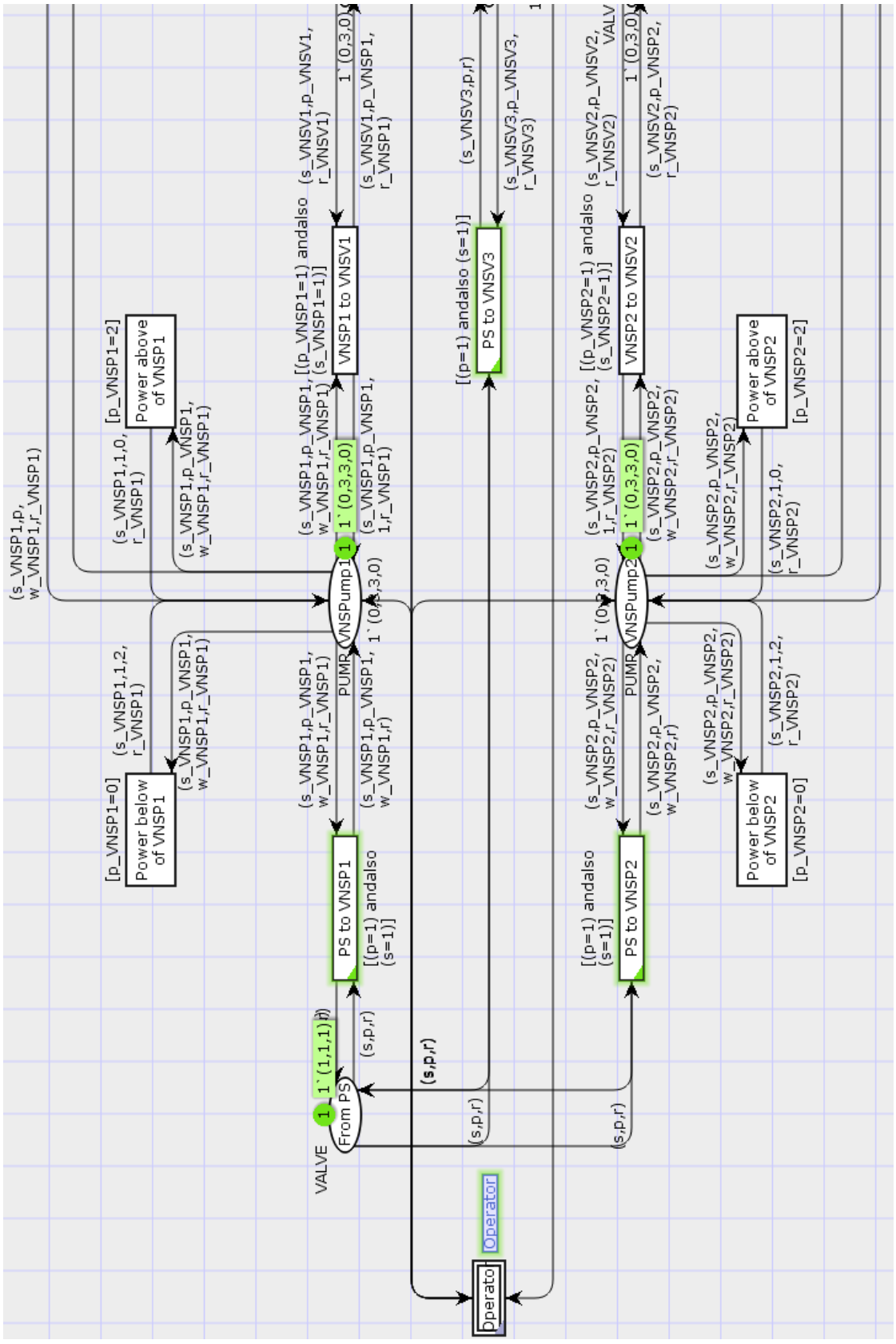


Рисунок 4.18 – Сеть Петри АСУ ТП локальной насосной станции (часть 1)

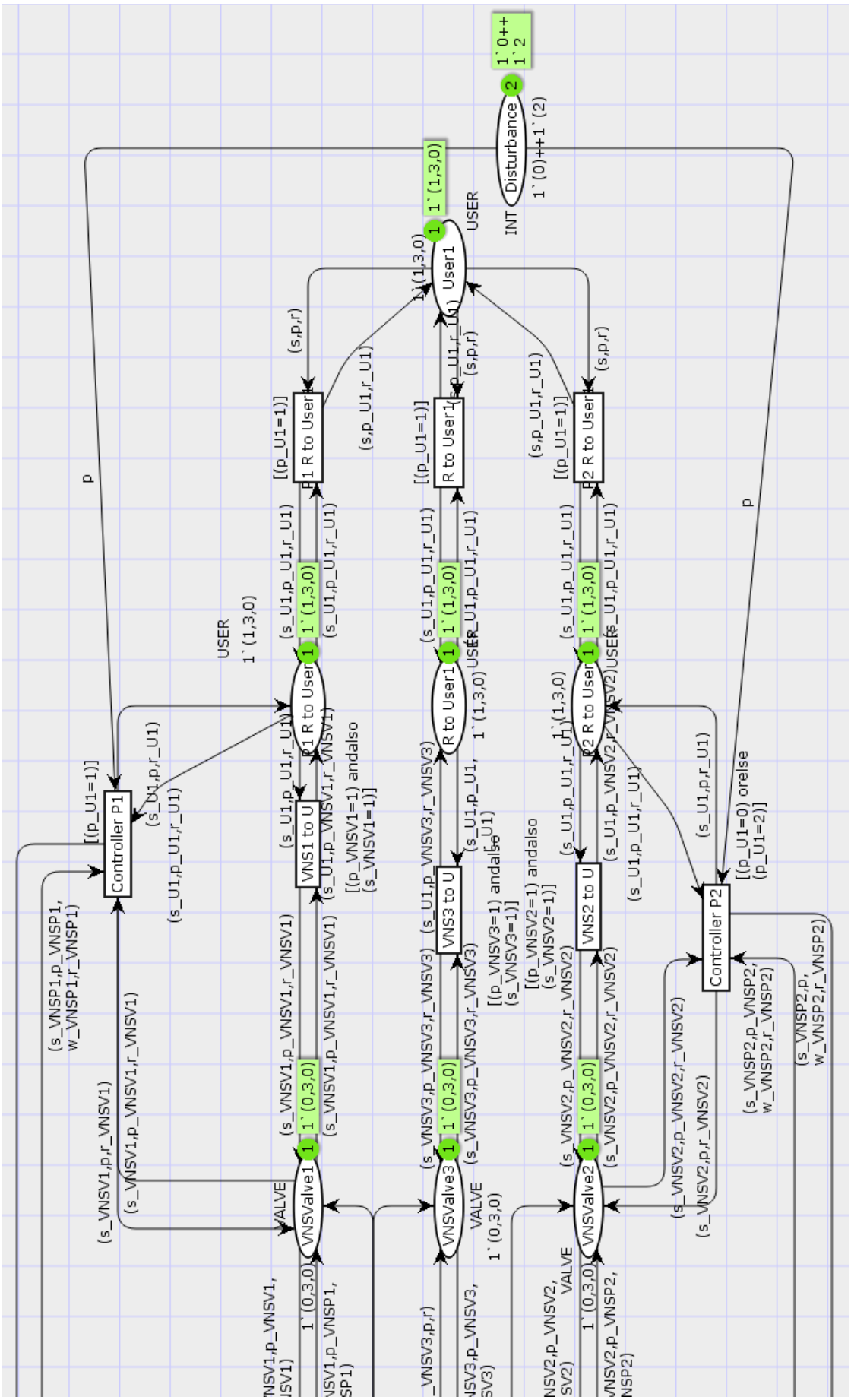


Рисунок 4.19 – Сеть Петри АСУ ТП локальной насосной станции (часть 2)

После проведения исследования данной СП получены результаты, которые помогли скорректировать сеть и тем самым избежать ложных состояний, которые могли навредить работе системы. Например, при включении и выключении задвижек оператором, данные элементы получали текущие состояния до выбранного действия. Анализ показал, что состояния задвижек должны меняться. После корректировки ошибок работа системы начала выполняться корректно. Для понимания более общей картины пространства состояний отдельных элементов, так и всей системы, проведём анализ для различного числа локальных водонапорных станций. Моделирование выполнено на вычислительной машине Intel Corei5-3330 CPU 3.00GHz, ОЗУ – DDR3 8Gb с операционной системой Windows 8.1 x64 и показало, что для исследования 10 000 состояний одной локальной водонапорной станции потребовалось 22 с, двух – 18 с, трёх – 9с; для исследования 25 000 состояний одной локальной водонапорной станции потребовалось 178 с, двух – 193 с, трёх – 79 с; для исследования 100 000 состояний одной локальной водонапорной станции потребовалось 6 310 с, двух – 6 234с, трёх – 28 965 с. Поскольку количество состояний для одной водонапорной станции превышает значение 10^5, можно утверждать, что для десяти водонапорных станций и всей системы водоснабжение количество состояний будет превышать порядок 10^6.

Итак, показано применение модифицированной методики проектирования ПО на основе UML диаграмм [20] и сетей Петри на примере АСУ ТП водоснабжения. Используя диаграммы прецедентов, классов и объектов проектируем свойства системы : давление в трубопроводе, частота работы двигателя, уровень в накопительных контейнерах и т.д. С использованием диаграммы деятельности и сети Петри выполнено моделирование динамических свойств системы. Посредством двух переходов, изменения значения задвижек на открытие/закрытие и увеличения/уменьшения частоты работы двигателя получен регулятор поддержания давления для каждого из насосов локальной водонапорной станции.

4.4. Использование матричного представления сетей Петри

В данном разделе ***используется матричное представление***, которое описано в п. 3.6. Данное представление на основе разработанного приложения [71] де-

монстрируется на примерах управляемого светофора [61], двухсимочного телефона [64, 68] и классической задачи взаимодействия пользователя с банкоматом [74].

4.4.1. Матричное представление автоматического режима работа управляемого светофора (рисунок 2.16) – это часть рассматриваемого примера использования методики разработки программного обеспечения с применением диаграмм UML и аппарата сетей Петри. Управляемый светофор работает в ручном и автоматическом режимах. В автоматическом режиме состояния светофора (изменение цветов) изменяются по ранее заданному алгоритму. При ручном управлении оператор следит за движением на перекрёстке и переключает цвета светофора в зависимости от загруженности дорог. Проектирование работы светофора с использованием сетей Петри, позволяет проектировать программное обеспечение, при котором одновременное включение одинаковых цветов светофора для разных направлений невозможно. Преобразование сети осуществляется способом, предложенным в [62], а именно создание массива меток с историей. Полученная сеть представлена в матричной форме на рисунке 4.20.

```
Матрица D-  Матрица D+  Матрица D
1|1|        1|1|       0|0|
1|1|        1|1|       0|0|
1|1|        1|1|       0|0|
1|1|        1|1|       0|0|
1|1|        1|1|       0|0|
1|1|        1|1|       0|0|

Вектор начального состояния
6|1|
```

Рисунок 4.20 –Матричное представление сети Петри управляемого светофора

4.4.2. Матричное представление логики работы двухсимочного телефона. В настоящее время на рынке мобильной технике представлены двухсимочные телефоны, укомплектованные одним или двумя радиомодулями (антенной для приёма и передачи сигнала). При работе с одним радиомодулем в режиме ожидания активны обе SIM-карты. Во время звонка на одну или с одной из SIM, вторая остаётся зарегистрированной в сети, но для звонящих переводится в состояние “занято”. Идентичная реакция возникает, когда с одной из SIM-карт подключаются к интернету. Исключением являются SMS, которые доставляются как при разговоре, так и при активной интернет сессии.

На рисунке 2.18 показана сеть Петри, демонстрирующая логику работы двухсимочного телефона с одним радиомодулем. Скорректированная сеть из графического вида была преобразована в матричную форму (рисунок 4.21).

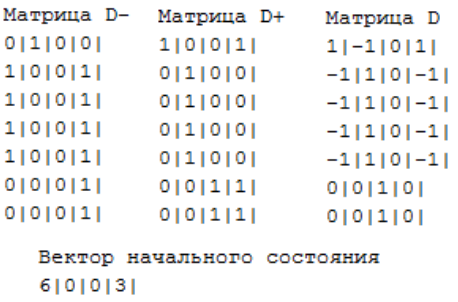


Рисунок 4.21 – Матричное представление сети Петри двухсимочного телефона

4.4.3. Матричное представление основных взаимодействий пользователя с банкоматом. Представим в матричной форме (рисунок 4.22) сеть Петри (рисунок 3.8), в которой смоделирована работа двух сценариев системы взаимодействия пользователя с банкоматом: просмотр баланс счета и снятие наличных, предварительно от пользователя требуется вставить карту, после чего корректно ввести личный PIN-код.

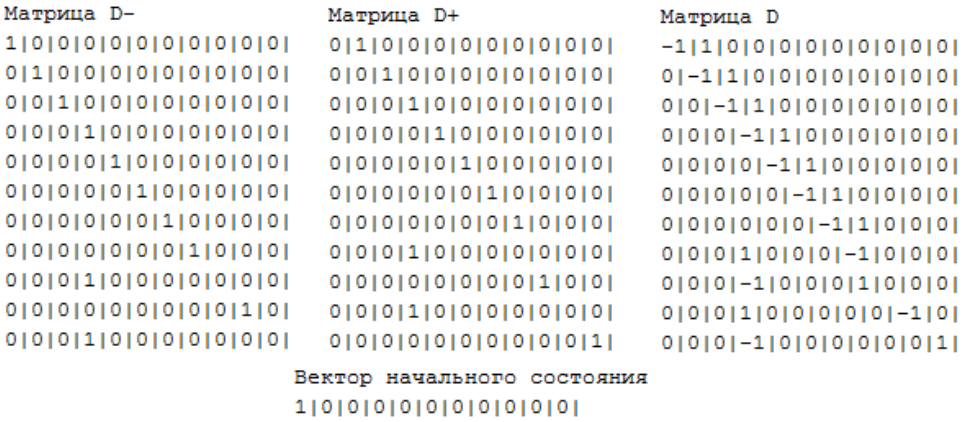


Рисунок 4.22 – Матричное представление сети Петри взаимодействия пользователя с банкоматом

Таким образом, предлагается матричное представления сетей Петри на примере управляемого светофора, двухсимочного телефона, взаимодействие пользователем с банкоматом. Стоит отметить, что алгоритм анализа с использованием матриц не является идеальным и требует доработок, которые связанны с возможностью задания последовательности срабатывания переходов в векторе запуска, а также отсутствует запрет на срабатывание переходов, которые при определенных сценариях не задействованы в системе.

4.5. Выводы

На основе модифицированной методики совместного использования UML диаграмм и сетей Петри, применимой для более широкого круга задач за счет строго выбранного набора диаграмм, нацеленных на проверку в сетях Петри, спроектировано программное обеспечения для автоматизированной системы обжига железорудных окатышей и АСУ ТП водоснабжения. Для каждой из систем

построены диаграмма прецедентов, диаграмма классов, диаграмма объектов и диаграмма активности, последняя транслируется по предложенным алгоритмам в сеть Петри, которая анализируется через автоматическую генерацию и исследования пространства состояний.

На этапе проектирования ПО для АСУ зоны обжига разработаны необходимые UML диаграммы и сеть Петри, переходы которой нагружены имитационным временем, с целью учесть реальные условия в модели. При проектировании регулятора поддержания номинального температурного режима использованы переходы сети Петри, срабатывание которых происходит в зависимости от изменения значений температуры. Для АСУ ТП водоснабжения построены диаграмма прецедентов, диаграмма классов и диаграмма объектов, которые отображают статические свойства системы. Проектирование диаграммы активности и сети Петри происходило с использованием дискретных значений. При создании регулятора поддержания давления для каждого из насосов локальной водонапорной станции, использованы два перехода, которые изменяют состояния положения задвижек и частоты работы двигателя.

При проектировании ПО для управляемого светофора, двухсимочного телефона и системы взаимодействия пользователем с банкоматом используется матричное представления сетей Петри, которое получено из графического посредством разработанного приложения по преобразованию сетей Петри, проектируемых в программной среде CPN Tools.

ЗАКЛЮЧЕНИЕ

Разработана методика проектирования программного обеспечения (ПО) при совместном использовании UML диаграмм и сетей Петри, основанная на анализе подходов к разработке ПО, парадигм написания программных продуктов, а также методов их анализа, как с точки зрения написания кода, так и управления процессом. Предлагается способ для реализации автоматической трансляции UML диаграмм в сети Петри. Разработаны способы анализа сетей Петри и их пространств состояний, при которых исследуются отдельные сценарии системы и различные части пространства состояний. На основе модифицированной методики совместного использования UML диаграмм и сетей Петри спроектировано программное обеспечение для автоматизированной системы регулирования температуры обжига железорудных окатышей и автоматической системы управления технологическим процессом на водонапорной станции.

Сформулируем основные положения и результаты диссертации.

1. Разработана методика проектирования ПО (архитектуры и исполняемого поведения), состоящая из семи этапов и включающая разработку UML диаграмм (прецедентов, классов, объектов, последовательности и активности) и сетей Петри для анализа диаграммы активности и последовательности. Данная методика применима к задачам, связанным с разработкой программного обеспечения для центров дистанционного контроля и управления, для систем локальной автоматики, для задач связанных с рекурсивными функциями и т.д.

2. Разработан алгоритм и правила реализации инверсии в сетях Петри для проверки достижимости выбранного состояния сети. Для простой ординарной сети подходит прямая инверсия (смена ориентации дуг между вершинами сети), а для простых сетей Петри, в свою очередь, предложены правила трансформации, поскольку в системе возможно неограниченное количество входных и выходных дуг у вершин.

3. Предложены алгоритмы для преобразования UML диаграмм в сети Петри (действие, выполнение условия, разделение/слияние), а также способ выполнения автоматической трансляции UML диаграммы активности в сети Петри посредством

подобной структуры форматов, основные признаки которых: имя элемента, тип данных, присущих данному элементу и маркировка каждого элемента.

4. Разработано программное обеспечение для преобразования комбинации мест и переходов маркированных сетей Петри к матричной форме, предложенного Дж. Питерсоном, преобразующее графическую форму сетей Петри, проектируемых в программной среде CPN Tools (version 3.4.0).

5. Разработан способ проектирования сетей Петри, при котором задание исходных данных о структуре проектируемой системы производится без использования мест и переходов, а при помощи информации, хранящейся в метках, что демонстрируется на примере перемещения манипулятора в ограниченном пространстве. Представлена реализация рекурсивных функций в сетях Петри.

6. Предложен способ анализа пространства состояний посредством моделирования сценариев работы проектируемой системы, требующих проверки, с использованием диаграмм последовательности и преобразованием их в сеть Петри для анализа на основе исследования пространства состояний. Предложено использовать представление сетей Петри в иерархическом виде с целью анализа отдельных подсетей сети с учетом начальных маркировок. Разработан способ анализа отдельно взятых частей пространства состояний при условии выполнения достижимости их начальной маркировки и отсутствия возвратных рёбер.

7. Предложено моделирование работы регуляторов с использованием сетей Петри, которое показано на примерах поддержания давления в трубопроводе при использовании дискретных величин и температуры в зоне обжига печи на основе дискретизированных аналоговых величин.

Предлагаемая методика реализуема при выполнении последовательных пошаговых процедур, основана на использовании UML диаграмм и сетей Петри и применима к широкому кругу задач. Результаты диссертационной работы активно внедряются на горно-обогатительном комбинате Акционерного Общества Соколовско-Сарбайского Горно-обогатительного Объединения (Республика Казахстан, г. Рудный), в проектировании ПО локальных подсистем АСУТП водоснабжения (г. Тюмень), в разработке интернет сайтов в ООО «Дабаз» (г. Новосибирск). Кроме того, полученные результаты также используются в учебном процессе в курсе лекций и лабораторных работ.

СПИСОК ЛИТЕРАТУРЫ

1. Ачасова, С.М. Корректность параллельных вычислительных процессов [Текст] / С.М. Ачасова, О.Л. Бандман. – Новосибирск: Наука. Сиб. отд-ние. – 1990. – 253 с.

2. Бек, К. Экстремальное программирование [Текст] / К. Бек. – Питер. – 2002. – 224 с.

3. Брауде, Э. Технология разработки программного обеспечения[Текст] / Э. Брауде. – Питер. – 2004. –655 с.

4. Буч, Г. UML. Руководство пользователя [Текст] / Г. Буч, Д. Рамбо. – ДМК. – 2001. – 432 с.

5. Буч, Г.Объектно-ориентированный анализ и проектирование с примерами приложений на С++[Текст] / Г. Буч. – СПб. : Бином. – 2006. – 560 с.

6. Вирт, Н. Алгоритмы и структуры данных [Текст] / Н. Вирт. – Москва: Мир. – 1989. – 264 с.

7. Воевода, А.А. Моделирование протокола связи таксафона и центра дистанционного управления таксафонами при помощи сетей Петри[Текст] / А.А. Воевода, Д.О. Саркенов. – Сб. науч. тр. НГТУ. – 2004. – №2(36). – С. 3–8.

8. Воевода, А.А. Моделирование протоколов с учётом времени на цветных сетях Петри [Текст] / А.А. Воевода, Д.О. Саркенов, В. Хассоунех. – Сб. науч. тр. НГТУ. – 2004. – №3(37). – С. 133–136.

9. Воевода, А.А. О компактном представлении языков сетей Петри [Текст] / А.А. Воевода, С.В. Коротиков. – Сб. науч. тр. НГТУ. – 2005. – №1(39). – С. 1–4.

10. Воевода, А.А. О модификации полного покрывающего дерева и графа разметок сети Петри [Текст] / А.А. Воевода, С.В. Коротиков. – Науч. вестн. НГТУ. – 2005. – №1(19). – С. 171–172.

11. Воевода, А.А. Сети Петри: семмитричные графы состояний [Текст] / А.А. Воевода, Д.О. Саркенов. – Сб. науч. тр. НГТУ. – 2005. – №3(41). – С. 1–6.

12. Воевода, А.А. Моделирование системы многоканальной визуализации с использованием аппарата сетей Петри [Текст] / А.А. Воевода, И.В. Зимаев. – Сб. науч. тр. НГТУ. – 2008. – №3(53). – С. 43–48.

13. Воевода, А.А. Моделирование сетей Петри в CPN TOOLS [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2008. – №3(53). – С. 49–54.

14. Воевода, А.А. О компактном представлении языков раскрашенных сетей Петри [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2008. – №3(53). – С. 105–108.

15. Воевода, А.А. Соотнесение структурных и временных масштабов UML-диаграмм [Текст] / А.А. Воевода, И.В. Зимаев. – Сб. науч. тр. НГТУ. – 2008. – №4(54). – С. 59–62.

16. Воевода, А.А. Особенности проектирования систем реального времени при по-мощи UML и сетей Петри [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2009. – №1(55). – С. 57–62.

17. Воевода, А.А. О моделировании систем реального времени с использованием UML и сетей Петри [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2009. – №1(55). – С. 63–66.

18. Воевода, А.А. Об особенностях преобразования UML диграмм деятельности в сети Петри [Текст] / А.А. Воевода, И.В. Зимаев. – Сб. науч. тр. НГТУ. – 2009. – №2(56). С. 77–86.

19. Воевода, А.А. Применение аппарата сетей Петри для моделирования системы организации и контроля доступа к услугам IP-телефонии [Текст] / А.А. Воевода, Д.В. Прытков, О.В. Прыткова. – Сб. науч. тр. НГТУ. – 2009. – №3(57). – С. 83–88.

20. Воевода, А.А. О проектировании программного обеспечения для микроконтроллера с использованием UML [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2009. – №4(58). – С. 35–40.

21. Воевода, А.А. Применение UML диаграмм и сетей Петри при разработке встраиваемого программного обеспечения [Текст] / А.А. Воевода, Д.О. Романников. – Науч. вестн. НГТУ. – 2009. - №4(37). –С. 169–174.

22. Воевода, А.А. Временные сети Петри и диаграммы UML [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2010. – №1(59).– С. 79–84.

23. Воевода,А.А. О возможностях некоторых популярных CASE-средств [Текст] / А.А. Воевода, Д.В. Прытков, О.В. Прыткова. – Сб. науч. тр. НГТУ. – 2010. – №1(59). – С. 143–148.

24. Воевода, А.А. Применение сетей Петри на этапе объектно-ориентированного проектирования [Текст] / А.А. Воевода, Д.В. Прытков. – Сб. науч. тр. НГТУ. – 2010. – №2 (60). – с. 65–76.

25. Воевода, А.А. О компактном представлении языков сетей Петри: сети с условиями и временные сети [Текст] / А.А. Воевода, **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2010. – №2(60). – С.77–82.

26. Воевода, А.А. Тестирование UML-диаграмм с помощью аппарата сетей Петри на примере разработки ПО для игры "Змейка" [Текст] / А.А. Воевода, **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2010. – №3(61). –С. 51–60.

27. Воевода, А.А. Использование UML и временных сетей петри при разработке программного обеспечения [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2010. – №3(61). – С. 61–70.

28. Воевода, А.А. Использование UML-диаграмм и временных сетей петри в методе разработки ПО ч.2 [Текст] / А.А. Воевода, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2010. – №4(62). – С. 117–126.

29. Воевода, А.А. Редуцирование пространства состояний сети Петри для объектов из одного класса [Текст] / А.А. Воевода, Д.О. Романников. – Науч. вестн. НГТУ. – 2011. – №4(45). – С. 146–150.

30. Воевода А.А. Совместное использование UML-диаграмм и сетей Петри на этапе проектирования программного обеспечения: обзор. [Текст] /А.А. Воевода, С.В. Коротиков, **А.В. Марков. –** Сб. науч. тр. НГТУ. – 2012. – № 2(66) С. 75–98.

31. Воевода, А.А. Рекурсия в сетях Петри [Текст] / А.А. Воевода, **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2012. – №3(69). – С. 115–122.

32. Воевода, А.А. Описание работы приложения для преобразования графического представления сетей Петри в матричную форму [Текст] / А.А. Воевода, **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2013. – № 3(73). – С. 49–57.

33. Воевода, А.А. Разработка программного обеспечения: проектирование с использованием UML диаграмм и сетей Петри на примере АСУ ТП водонапорной станции [Текст] / А.А. Воевода, **А.В. Марков**, Д.О. Романников. – Труды СПИИРАН. – 2014. – №3(34). – С. 218–231.

34. Воевода, А.А. Методика автоматизированного проектирования программного обеспечения функционирования сложных систем на основе совместного использования UML диаграмм и сетей Петри [Текст] / А.А. Воевода, **А.В. Марков**.

– Современные технологии. Системный анализ. Моделирование. – 2014. – №2(42). – С. 110–115.

35. Гуров, В. Исполняемый UML из России [Текст] / В. Гуров, А. Нарвский, А. Шалыто. – PCWeek. – 2005. – №26. – С. 18–19.

36. Дал, У. Структурное программирование [Текст] / У. Дал, Э. Дейкстра, К. Хоор. – М.: Мир. – 1975. – С. 247.

37. Доля, А.В. Алгоритмы безопасного перехода в сетях петри для лицензионной защиты программных систем [Текст]: дис. … канд. техн. наук: 05.13.11 / А.В. Доля. – Ростов-на-Дону, 2007. – 156 с.

38. Зайцев, Д.А. Моделирование телекоммуникационных систем в CPN Tools [Текст] / Д.А. Зайцев, Т.Р. Шмелёва. – Одесса: Онат. – 2006. – 60 с.

39. Зимаев, И.В. Моделирование асинхронной сети автоматов обработки данных [Текст] // И.В. Зимаев, А.А. Воевода. – Сб. науч. тр. НГТУ. – 2009. – №4 (58). – С. 147–154.

40. Зимаев, И.В. О возможности автоматической трансляции UML диаграмм деятельности в сети Петри [Текст] / И.В. Зиамев. – Сб. науч. тр. НГТУ. – 2010. – №1(59). – С. 149–156.

41. Зимаев, И.В. Интеграция структурных и динамических UML-моделей [Текст] / И.В. Зиамев. – Сб. науч. тр. НГТУ. – 2010. – №3(61). С. 77–84.

42. Зимаев, И.В.Блоки анализирующей сети Петри [Текст] / И.В. Зиамев. – Сб. науч. тр. НГТУ. – 2010. – №3(61). – С. 169–172.

43. Зимаев, И.В. Верификация WorkFlow-моделей с применением сетей Петри [Текст] / И.В. Зиамев, А.А. Воевода. – Науч. вестн. НГТУ. – 2010. – №4(41). – С. 151–154.

44. Ищенко, М.А. Разработка моделей и методов синтеза модульной структуры автоматизированных информационных систем с использованием сетей Петри [Текст]: дис. … канд. техн. наук: 05.25.05 / М.А. Ищенко. – Москва, 2009. – 146 с.

45. Козлов, В.А. Моделирование работы банкомата [Текст] / В.А. Козлов, О.А. Комалёв. – проект. – СПБГУ ИТМО, 2006. – 56 с.

46. Колесников, Д.А. Разработка математического и алгоритмического обеспечения управления режимами работы ситуационного центра регионального

уровня на базе сетей Петри [Текст]: дис. … канд. техн. наук: 05.13.01 / Д.А. Колесников. – Краснодар, 2011. – 167 с.

47. Коротиков, С.В. Представление логики взаимодействия таксафона и СКУТ в виде цветной иерархический сети Петри [Текст] / С.В. Коротиков, А.А. Воевода. – Сб. науч. тр. НГТУ. – 2004. – №2(36). – С. 147–148.

48. Коротиков, С.В. Применение цветных иерархических сетей Петри для верификации UML-диаграмм на этапе анализа требований к системе дистанционного контроля и управления [Текст] / С.В. Коротиков. – Сб. науч. тр. НГТУ. – 2007. – №1(47). – С. 81–92.

49. Коротиков, С.В. Проверка согласованности UML-диаграмм проекта службы контроля и управления ДЦ БРЗ с помощью сетей Петри [Текст] / С.В. Коротиков. – Сб. науч. тр. НГТУ. – 2007. – №2(48). – С. 51–62.

50. Коротиков, С.В. Применение шаблонов UML и сетей Петри при разработке системной службы центра дистанционного управления и контроля [Текст] // С.В. Коротиков. – Сб. науч. тр. НГТУ. – 2007. – №2 (48). – С. 135–140.

51. Коротиков, С.В. Применение спецификации эквивалентности в моделировании сеанса связи таксофона и центра дистанционного контроля и управления таксофонами раскрашенной сетью Петри [Текст] / С.В. Коротиков, Д.О. Саркенов. – Сб. науч. тр. НГТУ. – 2007. – №3(49). – С. 87–94.

52. Коротиков, С.В. Применение сетей Петри в разработке программного обеспечения центров дистанционного управления и контроля [Текст] // С.В. Коротиков, А.А. Воевода. – Науч. вестн. НГТУ. – 2007. – №4(29). – С. 16–30.

53. Коротиков, С.В. Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления [Текст]: дис. … канд. техн. наук.: 05.13.11 / С.В. Коротиков. – Новосибирск, 2007. – 216 с.

54. Котов, В.Е. Сети Петри [Текст] / В.Е. Котов. – М.: Наука. – 1984. – 160 с.

55. Крэг, Л. Применение UML 2.0 и шаблонов проектирования. – 3-е издание [Текст] / Л. Крэг. – М.: ООО «И.Д. Вильямс». – 2007. – 736 с.

56. Лескин, А.А. Сети Петри в моделировании и управлении [Текст] / А.А. Лескин, П.А. Мальцев, А.М. Спиридонов. – Ленинград: Наука. – 1989. – 133 с.

57. Ломазова, И.В. Вложенные сети Петри: моделирование и анализ распределенных систем с объектной структурой [Текст] / И.В. Ломазова. – Москва: Научный мир. – 2004. – 208 с.

58. Малышкин, В.Э. Параллельное программирование мультикомпьютеров: учебник [Текст] / В.Э. Малышкин, В.Д. Корнеев. – Новосибирск: Изд-во НГТУ, 2011. – 296 с.

59. Мамиконов, А.Г. Использование сетей Петри при проектировании систем обработки данных [Текст] / А.Г. Мамиконов. – Наука, 1988. – 103 с.

60. **Марков, А.В.** Моделирование процесса поиска пути в лабиринте при помощи сетей Петри [Текст] / **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2010. – №4(62). – С. 133–140.

61. **Марков, А.В.** Совокупное использование сетей Петри и UML-диаграмм при разработке программного обеспечения [Текст] / **А.В. Марков**, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2011. – №2(64). – С. 85–94.

62. **Марков, А.В.** Моделирование процесса поиска пути в лабиринте при помощи сетей Петри для системы из двух связных звеньев [Текст] / **А.В. Марков**, А.А. Воевода – Сб. науч. тр. НГТУ. – 2011. – №3(65). – С. 95–104.

63. **Марков, А.В.** Описание разрабатываемой системы “Поиск манипулятором кратчайшего пути в лабиринте” [Текст] / **А.В. Марков**, А.А. Воевода. – Сб. науч. тр. НГТУ. – 2011. – № 3(65). – С. 105–112.

64. **Марков, А.В.** Описание работы двухсимочных мобильных телефонов с помощью сетей Петри. Камбиев метод [Текст] / **А.В. Марков**, Д.В. Прытков. – Сб. науч. тр. НГТУ. – 2011. – №3(65). – С. 113–118.

65. **Марков, А.В.** Поиск манипулятором кратчайшего пути в лабиринте [Текст]/ **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2011. – №4(66). – С. 75–90.

66. **Марков, А.В.** Моделирование процесса поиска пути в лабиринте при помощи UML диаграмм и сетей Петри [Текст] / **А.В. Марков**, Д.О. Романников, А.А. Воевода. – Международная заочная научно-практическая конференция «Наука и техника XXI века», 14 ноября 2011 г. – Новосибирск: Изд-во «Априори», 2011. – С. 84–89.

67. **Марков, А.В.** Использование UML диаграмм и сетей Петри для моделирования процесса поиска пути в лабиринте [Текст] / **А.В. Марков**, Д.О. Романников. – XIII Международная научно-практическая конференция Наука и современность, 15 ноября 2011 г. – Новосибирск: Изд-во НГТУ, 2011. – С. 220–225.

68. **Марков, А.В.** Описание работы двухсимочных мобильных телефонов при помощи сетей Петри [Текст] / **А.В. Марков**. – XIII Международная конференция

"Информатика: проблемы, методология, технологии", 7-8 февраля 2013 г. – Воронеж: Изд-во ВГУ, 2013.

69. **Марков, А.В.** Применение матричного представление сетей Петри к различным системам [Текст] / **А.В. Марков**. – Международная научная конференция «Математическое и компьютерное моделирование», 18-19 октября 2013 г. – Омск: Изд-во ОГУ, 2013. – С. 29–34.

70. **Марков, А.В.** Анализ сетей Петри при помощи деревьев достижимости [Текст] / **А.В. Марков**. – 61-я международная молодёжная научно-техническая конференция «Молодёжь. Наука. Инновации», 20-21 ноября 2013 г. – Владивосток: Мор. гос. ун-т, 2013. – С. 93–96.

71. **Марков, А.В.** Описание приложения, преобразующего графическое представление сетей Петри к матричной форме [Электронный ресурс] / **А.В. Марков**, А.А Воевода. – VIII Международная конференция «Современные информационные технологии и ИТ-образование», 8-10 ноября 2013 г. – Москва: МГУ, 2013. – С. 264–269. – Режим доступа: http://conf.it-edu.ru/sites/default/files/elektronnyy_sbornik_tom_1.pdf, свободный.

72. **Марков, А.В.** Разработка программного обеспечения при совместном использовании UML-диаграмм и сетей Петри (обзор) [Текст] / **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2013. – №1(71). – С. 96–131.

73. **Марков, А.В.** Понятие рекурсии в сетях Петри: факториал числа, числа Фибоначчи [Текст] / **А.В. Марков**, А.А. Воевода. – Сб. науч. тр. НГТУ. – 2013. – №1(71). – С. 72–77.

74. **Марков, А.В.** Анализ сетей Петри при помощи деревьев достижимости [Текст] / **А.В. Марков**, А.А. Воевода. – Сб. науч. тр. НГТУ. – 2013. – №1(71). – С. 78–95.

75. **Марков А.В.** Матричное представление сетей Петри [Текст] / **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2013. – № 2(72). – С. 99–108.

76. **Марков, А.В.** Анализ отдельных частей дерева достижимости сетей Петри [Текст] / **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2013. – № 3(73). – С. 58–74.

77. **Марков, А.В.** Инверсия простой ординарной сети Петри [Текст] / **А.В. Марков**, А.А. Воевода. – Науч. вест. НГТУ. – 2013. – №4(53). – С. 215–218.

78. **Марков, А.В.** Инверсия сетей Петри [Текст] / **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2013. – № 4(74). – С. 97–121.

79. **Марков, А.В.** Развитие системы “Перемещение манипулятора в пространстве с препятствиями” при помощи рекурсивных функций [Текст] / **А.В. Марков**, А.А. Воевода. – Автоматика и программная инженерия. – 2013. – №2(4). – С. 35–41.

80. **Марков, А.В.** Алгоритм автоматической трансляции диаграммы активности в сеть Петри [Текст] / **А.В. Марков**, Д.О. Романников. – Доклады АН ВШ РФ. – 2014. – №1(22). – С. 104–112.

81. **Марков, А.В.** Проверка достижимости маркировки сетей Петри при помощи инвертирования деревьев состояний для протокола передачи данных [Текст] / **А.В. Марков**, А.А. Воевода. – Доклады ТУСУР. – 2014. – №1(31). – С. 143–148.

82. **Марков, А.В.** Анализ методик исследования пространства состояний сетей Петри [Текст] / **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2014. – №1(75). – С. 105–123.

83. **Марков, А.В.** Описание структуры АСУ ТП водонапорной станции при помощи UML диаграмм [Текст] / **А.В. Марков**, А.А. Воевода, Д.О. Романников. – Двенадцатая международная научно-техническая конференция “Актуальные проблемы электронного приборостроения”, 2-4 октября 2014 г. – Новосибирск: Изд-во НГТУ, 2014. – С. 65–67.

84. **Марков, А.В.** Применение UML-диаграмм и сетей Петри для проектирования ПО технологического процесса обжига окатышей [Текст] / **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2014. – №3 (77). – С. 99–118.

85. **Марков, А.В.** Обзор работ журнала Journal of Systems and Software за 2012-2014 годы, посвященных анализу программного обеспечения [Текст] / **А.В. Марков**, Д.О. Романников. – Сб. науч. тр. НГТУ. – 2014. – №3 (77). – С. 125 – 136.

86. **Марков, А.В.** Проектирование программного обеспечения для АСУ ТП обжига окатышей [Текст] / А.В. Марков. – Международная научно-практическая конференция «Тенденции формирования науки нового времени», 18 октября 2014 г. – Уфа: РИО МЦИИ ОМЕГА САЙНС, 2014. – С. 39–46.

87. Питерсон, Дж. Теория сетей Петри и моделирование [Текст] / Дж. Питерсон. – М: Мир. – 1984. – 264 с.

88. Прытков, Д.В. О применении сетей Петри для исполнения алгоритмов на примере решения задачи о кратчайших путях с единственным источником [Текст] / Д.В. Прытков. – Сб. науч. тр. НГТУ. – 2010. – №3(61). – С. 77–82.

89. Романников, Д.О. Перспективы развития методики разработки программного обеспечения с использованием UML-диаграмм и сетей Петри [Текст] / Д.О. Романников. – Сб. науч. тр. НГТУ. – 2010. – №2(60). – С. 181–184.

90. Романников, Д.О. Обзор работ, посвященных разработке по с использованием UML и сетей Петри [Текст] / Д.О. Романников, А.В. Марков, И.В. Зимаев. – Сб. науч. тр. НГТУ. – 2011. – №1(63). – С. 91–104.

91. Романников, Д.О. Пример применения методика разработки ПО с использованием UML-диаграмм и сетей Петри[Текст] / Д.О. Романников, **А.В. Марков**. – Науч. вест. НГТУ. – 2012. – №1(67). – С. 175–181.

92. Романников, Д.О. Об использовании программного пакета CPN Tools для анализа сетей Петри [Текст] / Д.О. Романников, **А.В. Марков**. – Сб. науч. тр. НГТУ. – 2012. – №2(68). – С. 105–116.

93. Романников, Д.О. Разработка программного обеспечения с применением UML диаграмм и сетей Петри для систем управления локальным оборудованием [Текст]: дис. … канд. техн. наук: 05.13.11 / Д.О. Романников. – Новосибирск, 2012. – 195 с.

94. Шоба, Е.В. Методология проектирования современного программного обеспечения применительно к станции управления лифтом [Текст] / Е.В. Шоба, **А.В. Марков** // Сб. науч. тр. НГТУ. – 2012. – №1(46). – С. 121–132.

95. Якобсон, А. Унифицированный процесс разработки программного обеспечения [Текст] / А. Якобсон, Г. Буч, Дж. Рамбо. – СПб.: Питер. – 2002. – 496 с.

96. Applications and Theory of Petri Nets // 30th International Conference, PETRI NETS 2009. 22–26 June 2009, Paris, France – 363 p.

97. Applications and Theory of Petri Nets // 32th International Conference, PETRI NETS 2011. 20–24 June 2011, Newcastle, UK – 358 p.

98. Applications and Theory of Petri Nets // 33th International Conference, PETRI NETS 2012. 25–29 June 2012, Hamburg, Germany – 427 p.

99. Application and Theory of Petri Nets and Concurrency. – 34th International Conference, PETRINETS 2013. June 2013, Milan, Italy. – 420 p.

100. Baresi, L. Improving UML with Petri Nets [Text] / L. Baresi, M. Pezze. – Electronic notes in theoretical computer science. – 2001. – Vol. 44. – P. 107–119.

101. Basile, F. A two-stage modelling architecture for distributed control of real-time industrial systems: Application of UML and Petri Net [Text] / F. Basile, P. Chiacchio, D.D. Grosso. – Computer Standards & Interfaces. – 2009. – Vol. 39. – P. 529–538.

102. Bouabana-Tebibel, T. An interleaving semantics for UML 2 interactions using Petri nets [Text] / T. Bouabana-Tebibel, S.H. Rubin. – Information Sciences. – 2013. – Vol. 232. – P. 276–293.

103. Burch, J.R. Symbolic Model Checking: 1020 States and Beyond [Text] / J.R. Burch, E.M. Clarke, K.L. McMillan. – Logic in Computer Science, 1990. LICS '90.4–7 June 1990. Philadelphia, USA. – P. 428–439.

104. Campos, J. On the Integration of UML and Petri Nets in Software Development [Text] / J. Campos, J. Merseguer. – 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency. 26–30 June 2006, Turku, Finland. – P. 19–36.

105. Christensen, S.A Sweep-Line Method for State Space Exploration / S. Christensen, L.M. Kristensen, T. Mailund. – TACAS 2001, Berlin, Germany. – P. 450–464.

106. Delatour, J. ArgoPN: a CASE Tool Merging UML and Petri Nets [Text] / J. Delatour, F.D. Lamotte. – 1st International Workshop on Validation and Verification of software for Enterprise Information Systems. – 2003. – P. 94–102.

107. Dillinger, P.C. Fast and Accurate Bitstate Verification for SPIN / P.C. Dillinger, P. Manolios. – 11th International SPIN Workshop. 1–3 April 2004, Barcelona, Spain. – P. 57–75.

108. Eichner, C. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets [Text] / C. Eichner, H. Fleischback, R. Meyer, U. Schrimpf, C. Stehno. – 12th International SDL Forum. 20–23 June 2005,Grimstad, Norway. – P. 133–148.

109. Emadi, S. From UML component diagram to an Executable model based on Petri nets [Text] / S. Emadi, F. Shams. – Information Technology. 2008, Kuala Lumpur, Malaysia. – P. 1–8.

110. Emadi, S. Transformation of Use case and Sequence Diagrams to Petri Nets [Text] / S. Emadi, F. Shams. – Computing, Communication, Control, and Management. – 2009. – Vol. 4. – P. 399–403.

111. Garrido, J.L. A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling [Text] / J.L. Garrido, M. Gea. – 9th International Workshop, DSV-IS 2002. 12–14 June 2002, Rostock, Germany. – P. 16–28.

112. Hack, M.H.T. Analysis of production schemata by Petri nets [Text] / M.H.T. Hack. – Massachusetts. Cambridge. – 1972. – 119 p.

113. Haddad, S. Theory and application to discrete event systems [Text] / S. Haddad, D. Poitrenaud. – Acta Informatica. – 2007. – 44 (7-8). P. 463–508.

114. Holzmann, G.J. An Analysis of Bitstate Hashing [Text] / G.J. Holzmann. – An early version of this paper peared in the Proceedings of the 15th Symposium on Protocol Specification, Testing and Verification. 1995, London, Great Britan. – P. 301 – 314.

115. Holzmann, G.J. An Improved Protocol Reachability Analysis Technique [Text] / G.J. Holzmann. – Software: Practice and Experience. – 1998. – Vol. 18. – P. 137–161.

116. Holzmann, G.J. On Limits and Possibilities of Automated Protocol Analysis [Text] / G.J. Holzmann. – Testing and Verification VII. – 1987. – P. 339–344.

117. Jensen, K. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems [Text] / K. Jensen, L.M. Kristensen, L. Wells. – International Journal on Software Tools for Technology Transfer. – 2007. – Vol. 9, issue 3–4. – P. 213–254.

118. Jensen, K. Coloured Petri Nets [Text] / K. Jensen, L.M. Kristensen. – Modelling and Validation of Concurrent Systems. – 2009. – 381 p.

119. Jørgensen, J.B. Coloured Petri Nets in UML-Based Software Development – Designing Middleware for Pervasive Healthcare [Text] / J.B. Jørgensen. – 20 p.

120. Kessentini, M. Example-Based Sequence Diagrams to Colored Petri Nets Transformation Using Heuristic Search [Text] / M. Kessentini, A. Bouchoucha, H. Sahraoui, M. Boukadoum. – 6th European Conference, ECMFA 2010. 15–18 June 2010, Paris, France. – P. 156–172.

121. Khan, R.H. Translation from UML to SPN Model: A Performance Modeling Framework [Text] / R.H. Khan, P.E. Heegaard. – Computer Design and Applications (ICCDA). – 2010. – volume 5. – P. 72–80.

122. Kristensen, L.M.A Generalised Sweep-Line Method for Safety Properties / Kristensen L.M., Mailund T. – FME. 2002, Berlin, Germany. – P. 549–567.

123. Lee, J. Verifying scenarios with time Petri-nets [Text] / J. Lee, J. Pan, J. Kuo. – Information and Software Technology. – 2001. – Vol. 43. – P. 769–781.

124. Lian-Zhang, Z. Automatic conversion from UML to CPN for software performance evaluation [Text] / Z. Lian-Zhang, K. Fan-Sheng. – Procedia Engineering. – 2012. – Vol. 29. – P. 2682–2686.

125. McMillan, K.L. Symbollic Model Checking [Text] / K.L. McMillan. – PhD thesis. – 1992. – 214 p.

126. Merseguer, J.J.H. Software Performance Modeling based on UML and Petri nets [Text] / J.J.H. Merseguer. – PhD thesis. – 2003. – 239 p.

127. Miyamoto, T. Synthesis of state machine diagrams from communication diagrams using Petri nets [Text] / T. Miyamoto, H. Kurahata, T. Fujii, R. Hosokawa. – Innovations in Systems and Software Engineering. – 2010. – Vol. 6. – P. 39–46.

128. Nogueraa, M. Ontology-driven analysis of UML-based collaborative processes using OWL-DL and CPN [Text] / M. Noguerra, M.V. Hurtado, M.L. Rodriguez, L. Chungb, J.L. Garrido. – Science of Computer Programming. – 2010. – Vol. 75. – P. 726–760.

129. Oliveira, S.A.R.D. Colored Petri Nets in the Animation of UML Models for Requirements Validation [Text] / S.A.R.D. Oliveira. – 2006. – Guimaraes, Julho. – 100 p.

130. Petri Nets and Other Models of Concurrency – ICAPTN 2006. – 2006. – Berlin, Germany. – 441 p.

131. Petri Nets and Other Models of Concurrency – ICAPTN 2007. – 2007. – Berlin, Germany. – 515 p.

132. Rabova, I. Using UML and Petri nets for visualization of business document flow [Text] / I. Rabova. – Acta universitatis agriculturae et silviculturae mendelianae brunensis. – 2012. – Vol. LX. – P. 299–306.

133. Reisig, W. Petri nets. An introduction [Text] / W. Reisig. – Springer-Verlag. – 1982. – 161 p.

134. Stern, U. Improved Probabilistic Verification by Hash Compaction [Text] / U. Stern, D.L. Dill. – IFIP WG 10.5 Advanced Research Working Conference, CHARME '95. 2–4 October 1995, Frankfurt/Main, Germany. – P. 206–224.

135. Szell, M. Understanding mobility in a social petri dish [Text] / M. Szell, R. Sinatra, G. Petri, S. Thurner, V. Latora. – Scientific Reports. – 2012. – Vol. 457. – P. 1–8.

136. Thierry-Mieg, Y. UML behavioral consistency checking using instantiable Petri nets [Text] / Y. Thierry-Mieg, L. Hillah. – Innovations in Systems and Software Engineering. – 2008. – Vol. 4. – P. 293–300.

137. Transactions on Petri Nets and Other Models of Concurrency I. – 2008. – Berlin, Germany. – 251 p.

138. Transactions on Petri Nets and Other Models of Concurrency II. – 2009. – Berlin, Germany. – 297 p.

139. Transactions on Petri Nets and Other Models of Concurrency III. – 2009. – Berlin, Germany. – 275 p.

140. Transactions on Petri Nets and Other Models of Concurrency IV. – 2010. – Berlin, Germany. – 225 p.

141. Transactions on Petri Nets and Other Models of Concurrency V. – 2012. – Berlin, Germany. – 293 p.

142. Transactions on Petri Nets and Other Models of Concurrency VI. – 2012. – Berlin, Germany. – 365 p.

143. Transactions on Petri Nets and Other Models of Concurrency VII. – 2013. – Berlin, Germany. – 435 p.

144. Trickovic, I. Formalizing activity diagram of UML by Petri nets [Text] / I. Trickovic. – Novi Sad J. Math. – 2000. – Vol. 30. – P. 161–175.

145. Voevoda, A.A. Petri Nets Modeling with supporting of system history storing [Text] / A.A. Voevoda, **A.V. Markov**, D.O. Romannikov. – Proceedings on RFBR and DST sponsored. The second Russian-Indian Joint Workshop «Computational intelligence and modern heuristics in automation and robotics», September 10–13, 2010. – Novosibirsk, Russia, 2010. – P. 47–49. – [Моделирования сетей Петри с поддержкой сохранения системной истории].

146. Voevoda, A.A. Using timed Petri nets in approach of software design with UML diagrams [Text] / A.A. Voevoda, D.O. Romannikov. – Proceedings on DST-RFBR Sponsored Indo-Russian Joint Workshop on «Computational intelligence and modern heuristics in automation and robotics». September 20–22, 2010. Surat, India. – P. 95–97 [Использование временных сетей Петри в методе разработке программного обеспечения с использованием UML диаграмм].

147. Westergaard, M. Behavioural Verification and Visualisation of Formal Models of Concurrent Systems [Text] / M. Westergaard. – PhD thesis. – 2007. – 183 p.

148. Wolper, P. Reliable Hashing without Collision Detection [Text] / P. Wolper, D. Leroy. – 5th International Conference, CAV '93 Elounda. 28 June – 1 July 1993, Elounda, Greece. – P. 59–70.

149. Yang, N. Modeling UML sequence diagrams using extended Petri nets [Text] / N. Yang, H. Yu, H. Sun, Z. Qian. – Information Science and Applications (ICISA). 21–23 April 2010, Seoul, South Korea. – P. 1–8.

150. Zhu, L.Z. Research of Automatic Conversion from UML Sequence Diagram to CPN Based on Modular Conversion [Text] / L.Z. Zhu, F.S. Kong. – International Conference, ICCIP 2012. 7–11 March 2012, Aveiro, Portugal. – P. 95–102.

ПРИЛОЖЕНИЕ А

АКТЫ ВНЕДРЕНИЯ

«Соколов-Сарыбай
кен-байыту өндірістік
бірлестігі» АҚ
«КЕНАВТОМАТИКА»
ӨНДІРІСТІК-ТЕХНОЛОГИЯЛЫҚ
БАСҚАРМАСЫ

SSGPO

АО «Соколовско-Сарбайское
горно-обогатительное
производственное объединение»
ПРОИЗВОДСТВЕННО-
ТЕХНОЛОГИЧЕСКОЕ УПРАВЛЕНИЕ
«РУДОАВТОМАТИКА»

Ленин даңғылы, 26, Рудный қаласы,
Қостанай облысы,
Қазақстан Республикасы, 111500
T 8 (71431) 22887
F 8 (71431) 28916, 28917
E main@ssgpo.enrc.com
www.enrc.com

26,Lenin Avenue, Rudny town,
Kostanai region,
Republic of Kazakhstan, 111500
T 8 (71431) 22887
F 8 (71431) 28916, 28917
E main@ssgpo.enrc.com
www.enrc.com

пр. Ленина, 26, г. Рудный,
Костанайская область,
Республика Казахстан, 111500
T 8 (71431) 22887
F 8 (71431) 28916, 28917
E main@ssgpo.enrc.com
www.enrc.com

Исх. № 1510/965
«12» 09 2014 г

УТВЕРЖДАЮ:
И.о.директора ПТУ «Рудоавтоматика»
АО «ССГПО»
_____А.К.Иевлев

АКТ
о внедрении результатов исследования

Выдан аспиранту кафедры «Автоматики» Новосибирского Государственного Технического Университета Маркову Александру Владимировичу для предоставления в диссертационный Совет.

Настоящий акт, подтверждает, что результаты исследования методики проектирования программного обеспечения с использованием UML диаграмм и сетей Петри были переданы для использования в разработке автоматизированной системы участка обжига окатышей АО ССГПО, а именно, для регулирования температуры в зоне обжига печи, равной 1250 ^{o}C.

Предложена методика проектирования ПО с использованием UML диаграмм и сетей Петри, а также рекомендации по проектированию регулирующих элементов в сетях Петри. Данная методика, а так же рекомендации по выбору и составлению набора UML диаграмм позволили успешно спроектировать автоматизированную систему. Применение данной методики позволило сократить время (34 рабочих часа), затрачиваемое на тестирование ПО для автоматизированной системы. Предложенные исследования будут использоваться в дальнейшем для разработки автоматизированных систем.

Главный инженер Центральной лаборатории
автоматизации и метрологии
ПТУ «Рудоавтоматика»_____И.В.Фаткуллин
М.П. Ф.И.О.

ЗАО «СИНЕТИК», адрес центрального офиса:
630009, г.Новосибирск, ул.3-го Интернационала, 127
тел.: (383) 266-51-40, 266-47-28, факс (383) 266-07-51
e-mail: root@sinetic.ru, http://www.sinetic.ru

УТВЕРЖДАЮ
Генеральный директор
_____А.В.Салдаев

19 сентября 2014 г.

АКТ № 12397/14

о практическом применении результатов исследований Маркова А. В., полученных в ходе выполнение диссертационной работы на соискание ученой степени кандидата технических наук

Выдан аспиранту кафедры «Автоматика» ФГБОУ ВПО «Новосибирского государственного технического университета» Маркову Александру Владимировичу для предоставления в диссертационный Совет.

Настоящий акт подтверждает, что результаты исследований, заключающиеся в совместном использовании UML диаграмм и сетей Петри при проектировании программного обеспечения, применялись в разработке программного комплекса локальных подсистем автоматизированной системы управления технологическим процессом водоснабжения г. Тюмень.

В проводимых исследованиях предлагается сократить число диаграмм и применить анализ пространства состояний, как одного из методов исследования спроектированной системы. Предложенная методика, а так же рекомендации по выбору и составлению набора UML диаграмм позволили успешно разработать и отладить программный код системы, при этом время этапа проектирования уменьшилось на 20% за счет применения современных технологий визуального программирования, а именно UML. Анализ полученного кода по результатам моделирования системы в сетях Петри позволил выявить неточности логического характера, заключающие в нахождении неиспользуемых сценариев работы системы.

к.т.н., Главный инженер Г.П. Голодных

Юридический адрес: 630075, г.Новосибирск, ул.Народная, 7/1, ИНН 5410119182, КПП 546050001,
р/с 40702810544070111112 в Сибирском банке Сбербанка РФ г.Новосибирск, БИК 045004641, к/с 30101810500000000641,
коды ОГРН 1025403908100, ОКПО 23584736, ОКОНХ 80400, 95400, 72200.

УТВЕРЖДАЮ
Директор ООО «Дабаз»
Рапопорт М.А.
«16» апреля 2014г.

АКТ №15
о внедрении результатов исследования

Выдан аспиранту кафедры «Автоматики» Новосибирского Государственного Технического Университета Маркову Александру Владимировичу для предоставления в диссертационный Совет.

Настоящий акт, подтверждает, что результаты исследования различных способов анализа спроектированных сетей Петри были переданы для использования в разработке интернет сайтов.

В частности использовался способ разделения сети Петри, соответствующей разрабатываемому сайту, на иерархии с последующим анализом необходимых участков сети. Предоставленные исследования способствовали ускорению создания программной части интернет сайтов, сократить появление ситуаций с появлением ошибок на сервере, связанных с отсутствием запрашиваемых страниц, до 80%.

Директор ООО «Дабаз» М.П. Рапопорт М.А.

«Утверждаю»
Ректор НГТУ, д.т.н., проф.
_____ Н.В. Пустовой
«___» _____ 2014 г.

СПРАВКА

о внедрении результатов научных исследований в учебный процесс
Факультета автоматики и вычислительной техники НГТУ

Настоящим подтверждается, что результаты научных исследований Маркова Александра Владимировича, аспиранта кафедры «Автоматика» Новосибирского Государственного Технического Университета, были использованы в учебном процессе НГТУ в курсах лекций «Теория вычислительных процессов», «Теоретическая информатика» и «Технология программирования», читаемых для студентов специальностей 230105 – «Программное обеспечение вычислительной техники и автоматизированных систем» и 230100 – «Информатика и вычислительная техника».

Декан АВТФ, к.т.н. Рева И.Л.

Зав. кафедрой Автоматики, д.т.н., профессор Жмудь В.А.

Ученый секретарь каф. Шахтшнейдер В.Г.

ПРИЛОЖЕНИЕ Б

СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ

Программа “Алгоритм управления насосным агрегатом водонапорной станции”

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2012618139

Алгоритм управления насосным агрегатом водонапорной станции

Правообладатель(ли): ***Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Новосибирский государственный технический университет» (НГТУ) (RU)***

Автор(ы): ***Романников Дмитрий Олегович,***
Воевода Александр Александрович,
Марков Александр Владимирович (RU)

Заявка № **2012615873**
Дата поступления **13 июля 2012 г.**
Зарегистрировано в Реестре программ для ЭВМ
7 сентября 2012 г.

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

Программа “Приложение для преобразования графического представления сетей Петри в матричную форму”

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2014611333

Приложение для преобразования графического представления сетей Петри в матричную форму

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Новосибирский государственный технический университет» (НГТУ) (RU)***

Авторы: ***Марков Александр Владимирович (RU),***
Воевода Александр Александрович (RU)

Заявка № **2013661729**
Дата поступления **17 декабря 2013 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***30 января 2014 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Б.П. Симонов

Программа “Расширения для пакета MATHCAD при решении задачи полиноминального синтеза”

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2013614151

Расширение для пакета MATHCAD
при решении задачи полиноминального синтеза

Правообладатель(ли): ***Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Новосибирский государственный технический университет» (НГТУ) (RU)***

Автор(ы): ***Шоба Евгений Владимирович (RU), Воевода Александр Александрович (RU), Марков Александр Владимирович (RU), Вороной Вадим Владимирович (RU)***

Заявка № **2012661318**
Дата поступления **19 декабря 2012 г.**
Зарегистрировано в Реестре программ для ЭВМ
24 апреля 2013 г.

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

ПРИЛОЖЕНИЕ В

АНАЛИЗ И СРАВНЕНИЕ ФОРМАТОВ *.XMI* И *.CPN*

Для формализации правил преобразования UML диаграмм в сети Петри предлагается анализ и сравнение форматов *.xmi* и *.cpn* (п. 2.3) начального места, конечного места, обычного места, перехода, описание дуги формата *.cpn* и определение связей между элементами формата *.xmi* [18, 80]. По данному описанию и сравнению разрабатывается способ для автоматической трансляции из диаграммы активности в соответствующую сеть Петри, что позволяет сократить время на проектирование и анализ программного обеспечения.

В.1. Описание и сравнение конечного места форматов *.xmi* и *.cpn*. В листинге представлено конечное состояния формата *.xmi*:

```
1.<place id="ID1420722112">
2.      <posattr x="-104.000000" y="-420.000000"/>
3.      <fillattrcolour="White" pattern="" filled="false"/>
4.      <lineattrcolour="Black" thick="1" type="Solid"/>
5.      <textattrcolour="Black" bold="false"/>
6.      <text>EndState</text>
7.      <ellipse w="60.000000" h="40.000000"/>
8.      <token x="-10.000000" y="0.000000"/>
9.      <marking x="0.000000" y="0.000000" hidden="false">
10.         <snap snap_id="0" anchor.horizontal="0" anchor.vertical="0"/>
11.     </marking>
12.     <type id="ID1420722113">
13.         <posattr x="-104.000000" y="-389.000000"/>
14.         <fillattrcolour="White" pattern="Solid" filled="false"/>
15.         <lineattrcolour="Black" thick="0" type="Solid"/>
16.         <textattrcolour="Black" bold="false"/>
17.         <text tool="CPN Tools" version="3.4.0">INT</text>
18.     </type>
19.     <initmark id="ID1420722114">
20.         <posattr x="-47.000000" y="-396.000000"/>
21.         <fillattrcolour="White" pattern="Solid" filled="false"/>
22.         <lineattrcolour="Black" thick="0" type="Solid"/>
23.         <textattrcolour="Black" bold="false"/>
24.         <text tool="CPN Tools" version="3.4.0"/>
25.     </initmark>
26.</place>.
```

Конечное состояния диаграммы активности имеет собственное название *FinalState*, но его атрибуты совпадают с атрибутами начального места. В листинге представлено конечное состояния формата формата *.cpn*:

1.<UML:FinalState xmi.id = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000898'
2. name = 'EndState__INT__0`0' isSpecification = 'false'>
3.<UML:StateVertex.incoming>
4.<UML:Transitionxmi.idref = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000899'/>
5.<UML:Transitionxmi.idref = '-64--88-0-100--4260c10:1410c69956e:-8000:000000000000089A'/>
6.</UML:StateVertex.incoming>
7.</UML:FinalState>.

Структура конечного места в формате *.cpn* аналогична структуре начального состояния, отличие заключается лишь в том, что у конечного состояния отсутствует разметка. Формат *.xmi*, в свою очередь, содержит в последней части названия (строка №2) разметку вида: *0`0*. Также, отметим, что в данном случае у формата *.xmi* присутствуют две входящие дуги – строчки №4 и №5 со своим уникальным идентификатором в проекте. Представленные данные могут быть использованы для реализации автоматической трансформации данных элементов из диаграммы активности в сеть Петри.

B.2. Описание и сравнение обычного места форматов *.xmi* и *.cpn*. Наиболее часто встречающимися элементами, как в диаграмме активности, так и в сети Петри, являются обычные места. Обычное место формата *.cpn*:

1.<placeid="ID1420721136">
2.<posattr x="-107.000000" y="42.000000"/>
3.<fillattrcolour="White" pattern="" filled="false"/>
4.<lineattrcolour="Black" thick="1" type="Solid"/>
5.<textattrcolour="Black" bold="false"/>
6.<text>Activity1</text>
7.<ellipse w="60.000000" h="40.000000"/>
8.<token x="-10.000000" y="0.000000"/>
9.<marking x="0.000000" y="0.000000" hidden="false">
10.<snap snap_id="0" anchor.horizontal="0" anchor.vertical="0"/>
11.</marking>
12.<typeid="ID1420721137">

13.<posattr x="-151.500000" y="42.000000"/>
14.<fillattrcolour="White" pattern="Solid" filled="false"/>
15.<lineattrcolour="Black" thick="0" type="Solid"/>
16.<textattrcolour="Black" bold="false"/>
17.<text tool="CPN Tools" version="3.4.0">INT</text>
18.</type>
19.<initmarkid="ID1420721138">
20.<posattr x="-50.000000" y="65.000000"/>
21.<fillattrcolour="White" pattern="Solid" filled="false"/>
22.<lineattrcolour="Black" thick="0" type="Solid"/>
23.<textattrcolour="Black" bold="false"/>
24.<text tool="CPN Tools" version="3.4.0"/>
25.</initmark>
26.</place>.

Обычное место формата *.xmi*:

1.<UML:ActionState xmi.id = '-64--88-0-100--4260c10:1410c69956e:-8000:000000000000086A'
2. name = 'Activity1__0`0__INT' isSpecification = 'false' isDynamic = 'false'>
3.<UML:StateVertex.outgoing>
4.<UML:Transitionxmi.idref = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000879'/>
5.</UML:StateVertex.outgoing>
6.<UML:StateVertex.incoming>
7.<UML:Transitionxmi.idref = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000874'/>
8.</UML:StateVertex.incoming>
9.<UML:State.entry>
10.<UML:UninterpretedAction xmi.id = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000875'
11.isSpecification = 'false' isAsynchronous = 'false'>
12.<UML:Action.script>
13.<UML:ActionExpression xmi.id = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000877'
14.language = '' body = 'Activity1'/>
15.</UML:Action.script>
16.</UML:UninterpretedAction>
17.</UML:State.entry>
18.</UML:ActionState>.

В обычном месте структура данных у формата *.cpn* и *.xmi* идентична структуре начального и конечного состояния. В данном состоянии у формата *.xmi* существует одна выходящая (строка 3) и одна входящая (строка 6) дуги. Также добавляется свойство *body* (строка 14) с атрибутом *Activity1*, который соответствует первой части в названии (строка 2) самого состояния и необходим для отображения имени на диаграмме активности.

В.3. Описание и сравнение обычного перехода форматов *.xmi* и *.cpn*. Приведем сравнение перехода, представленного в формате *.xmi*:

```
1.<trans id="ID1420721026" explicit="false">
2.      <posattr x="-107.000000" y="102.000000"/>
3.      <fillattrcolour="White" pattern="" filled="false"/>
4.      <lineattrcolour="Black" thick="1" type="solid"/>
5.      <textattrcolour="Black" bold="false"/>
6.      <text>Start</text>
7.      <box w="60.000000" h="40.000000"/>
8.      <binding x="7.200000" y="-3.000000"/>
9.      <condid="ID1420721027">
10.             <posattr x="-146.000000" y="133.000000"/>
11.             <fillattrcolour="White" pattern="Solid" filled="false"/>
12.             <lineattrcolour="Black" thick="0" type="Solid"/>
13.             <textattrcolour="Black" bold="false"/>
14.             <text tool="CPN Tools" version="3.4.0"/>
15.     </cond>
16.     <timeid="ID1420721028">
17.             <posattr x="-62.500000" y="133.000000"/>
18.             <fillattrcolour="White" pattern="Solid" filled="false"/>
19.             <lineattrcolour="Black" thick="0" type="Solid"/>
20.             <textattrcolour="Black" bold="false"/>
21.             <text tool="CPN Tools" version="3.4.0"/>
22.     </time>
23.     <codeid="ID1420721029">
24.             <posattr x="-42.500000" y="50.000000"/>
25.             <fillattrcolour="White" pattern="Solid" filled="false"/>
26.             <lineattrcolour="Black" thick="0" type="Solid"/>
27.             <textattrcolour="Black" bold="false"/>
28.             <text tool="CPN Tools" version="3.4.0"/>
29.     </code>
30.     <channelid="ID1420745696">
```

```
31.             <posattr x="-43.500000" y="102.000000"/>
32.             <fillattrcolour="White" pattern="Solid" filled="false"/>
33.             <lineattrcolour="Black" thick="0" type="Solid"/>
34.             <textattrcolour="Black" bold="false"/>
35.             <text tool="CPN Tools" version="3.4.0"/>
36.         </channel>
37.         <priorityid="ID1420721031">
38.             <posattr x="-175.000000" y="71.000000"/>
39.             <fillattrcolour="White" pattern="Solid" filled="false"/>
40.             <lineattrcolour="Black" thick="0" type="Solid"/>
41.             <textattrcolour="Black" bold="false"/>
42.             <text tool="CPN Tools" version="3.4.0"/>
43.         </priority>
44.</trans>.
```

Обычный переход формата *.xmi*:

```
1.<UML:Transition xmi.id = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000874'
2.  name = 'Start__i__i__i' isSpecification = 'false'>
3.<UML:Transition.source>
4.<UML:Pseudostatexmi.idref = '-64--88-0-100--4260c10:1410c69956e:-8000:0000000000000869'/>
5.</UML:Transition.source>
6.<UML:Transition.target>
7.<UML:ActionStatexmi.idref = '-64--88-0-100--4260c10:1410c69956e:-8000:000000000000086A'/>
8.</UML:Transition.target>
9.</UML:Transition>.
```

У формата *.cpn* в строчке №6 указывается имя перехода. В строке 14 показано условие, которое может быть задано для перехода, у формата *.xmi* – это последняя часть в названии (строка 2). Если в формате *.cpn* ничего не указано, то в *.xmi* название также состоит из четырёх частей, а в последней части указывается имя входящей дуги (вторая часть в названии у формата *.xmi*). У формата *.xmi* во второй строке указано имя перехода, которое состоит из четырёх частей: первая – имя, вторая – имя входящей дуги, третья – имя выходящей дуги, четвёртая – условие для срабатывания перехода. Также у формата *.xmi* можно увидеть, какое состояние предшествует переходу (строка 3), а какое следует после него (строка 6).

В.4. Описание дуги формата *.cpn* и определение связей между элементами формата *.xmi.* У формата *.cpn* присутствует ещё один тег *<arc>*, который отвечает за связь у вершин сети. Одно из основных свойств этого тега – *orientation* с атрибутами двух видов: *PtoT, TtoP*, что означает направление дуги от места к переходу или от перехода к месту. Строка 7 и строка 8 указывает на идентификаторы перехода и места, к которому и от которого направлена дуга, соответственно. Строка 14 содержит информацию об имени дуги. У формата *.xmi* данный тег отсутствует, но информация о взаимосвязях между состояниями на диаграмме активности заложена в тегах *<UML:…State>* и *<UML:Transition>*. Дуга в *CPN Tools*:

```
1.<arc id="ID1420721566" orientation="PtoT" order="1">
2.      <posattr x="0.000000" y="0.000000"/>
3.      <fillattrcolour="White" pattern="" filled="false"/>
4.      <lineattrcolour="Black" thick="1" type="Solid"/>
5.      <textattrcolour="Black" bold="false"/>
6.      <arrowattrheadsize="1.200000" currentcyckle="2"/>
7.      <transendidref="ID1420721026"/>
8.      <placeendidref="ID1420720921"/>
9.      <annotid="ID1420721567">
10.         <posattr x="-101.500000" y="135.000000"/>
11.         <fillattrcolour="White" pattern="Solid" filled="false"/>
12.         <lineattrcolour="Black" thick="0" type="Solid"/>
13.         <textattrcolour="Black" bold="false"/>
14.         <text tool="CPN Tools" version="3.4.0">i</text>
15.     </annot>
16.</arc>.
```

Таким образом, взаимосвязь между форматами *.xmi* и*.cpn*, а именно: начального места, конечного места, обычного места, перехода, дуги формата *.cpn* и определение связей между элементами формата *.xmi*, является основой для реализации автоматической трансляции диаграмм активности в соответствующие сети Петри.

ПРИЛОЖЕНИЕ Г

ПРАВИЛА РЕАЛИЗАЦИИ ИНВЕРСИИ СЕТЕЙ ПЕТРИ

Поясняется предложенные правила (п. 3.4) для реализации инверсии простых сетей Петри: разделение и слияние потоков передачи информации через переход, разделение и слияние через место с различным количеством ресурсов у потоков, а также сетей с наличием возвратных рёбер. Инверсия сети Петри выполняется с целью проверки достижимости выбранной ранее маркировки, что необходимо при анализе любого участка пространства состояний сети. Описывается восстановление сети Петри из инвертированного графа состояний [77, 78, 81].

Г.1. Примеры инверсии. Рассмотрим ***примеры №1 и №2***, для которых сеть Петри имеет три места: $P = \{A, B, C\}$ и один переход $T = \{a\}$, а граф состояний имеет две возможные маркировки $m_1 [a\rangle m_2 = (1,0,0)[a\rangle(0,1,1)$, $m_I = m_1$ (рисунок Г.1). Как и в случае простой ординарной сети, данная система будет преобразовываться путём прямой инверсии. Трансформированная сеть с графом состояний представлена на рисунке Г.2. Стоит отметить, что примеры сети Петри №1 и №2 являются инвертированными сетями по отношению друг к другу. То есть при *выполненной инверсии*[43] для первого случая можно с уверенностью сказать, что будет получена сеть, представленная в примере №2.

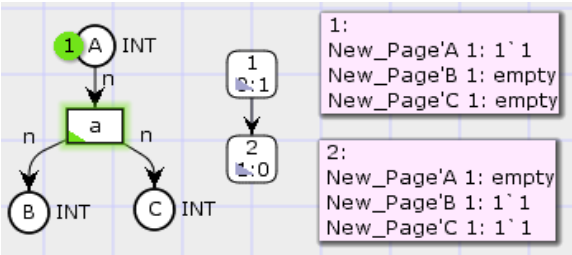


Рисунок Г.1 – Сеть Петри и граф состояний, пример №1

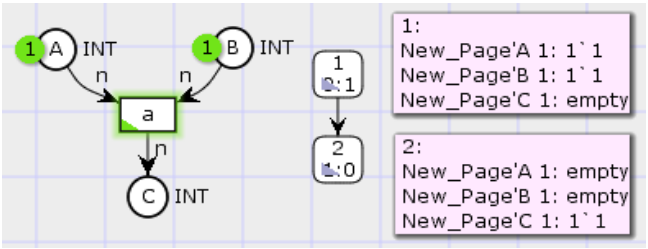


Рисунок Г.2 – Инвертированная сеть с графом состояний, пример №1

В *примере №3* сеть имеет четыре места и три перехода с начальной маркировкой $m_I = (1,1,0,0)$ (рисунок Г.3).

Граф состояний (рисунок Г.3) имеет восемь состояний $V = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8\}$ и десять рёбер между ними $E = \{m_1[b\rangle m_2,$ $m_1[a\rangle m_3,\ m_2[a\rangle m_5,\ m_3[b\rangle m_5,\ m_2[c\rangle m_4,\ m_3[c\rangle m_6, m_5[c\rangle m_7,\ m_4[a\rangle m_7,\ m_6[b\rangle m_7,\ m_7[b\rangle m_8\}$.

[43] *Выполненная инверсия* – инверсия, после которой получена полностью трансформированная сеть с возможностью восстановления графа состояний.

К данному примеру также применим способ прямой инверсии, который при смене направления взаимосвязей между вершинами приводит к начальной маркировке из любого состояния сети.

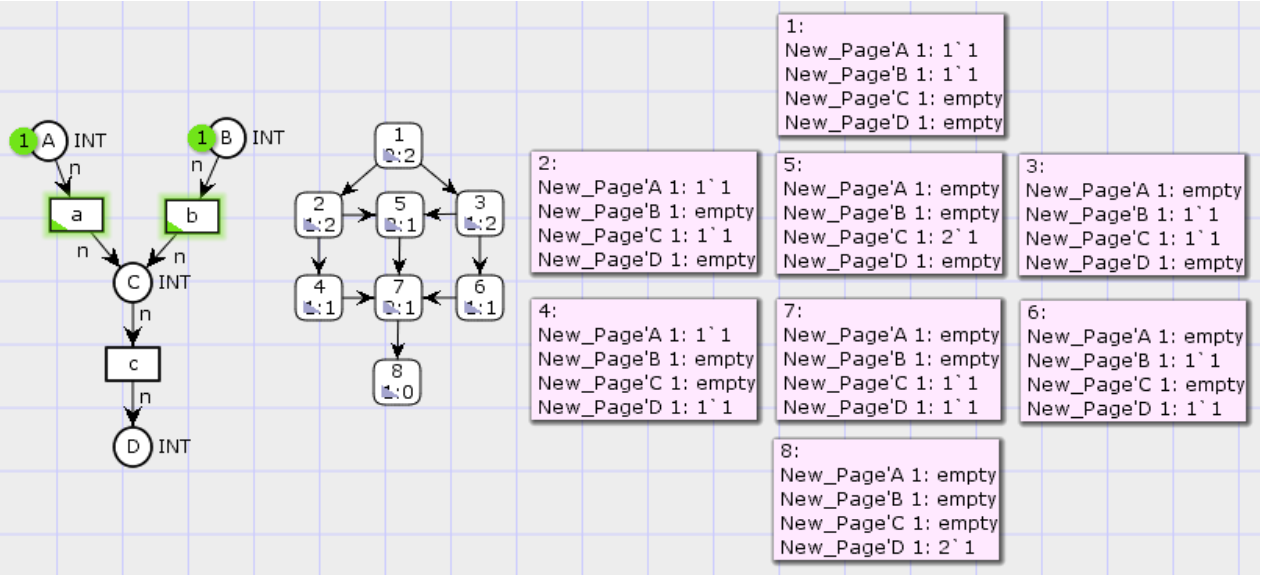


Рисунок Г.3 – Сеть Петри и граф состояний, пример №3

После инвертирования сеть принимает вид (рисунок Г.4) схожий с начальной сетью, а отличие заметно лишь в направлении дуг. Стоит отметить, что в графе состояний, представленном на рисунке Г.4, после инвертирования появились дополнительные маркировки m_8, m_{10}, наличие которых в примере №3 не оговаривается. Следовательно, данные состояния нужно считать неучтенными. Тем не менее, выполняется поставленная цель инвертирования – достижения начальной маркировки из выбранного состояния.

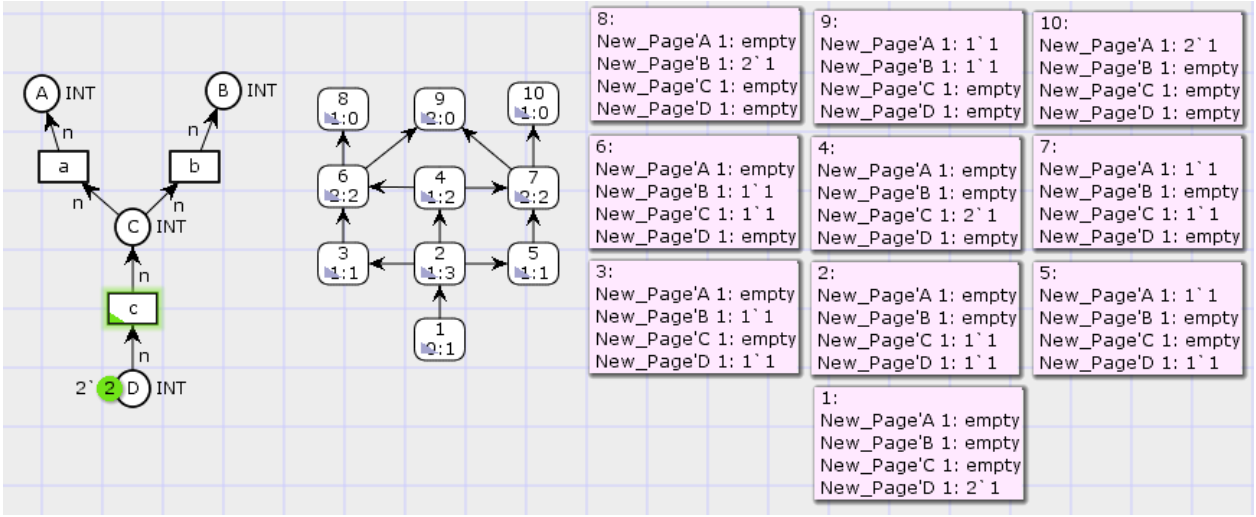


Рисунок Г.4 – Инвертированная сеть с графом состояний, пример №3

Зададим для сети ***примера №3*** дополнительно две маркировки, которые исключают фишку из одного места: $m_I = (1,0,0) \neg m_I = (0,1,0)$. В первом случае (рисунок Г.5) сеть выглядит также как и на рисунке Г.3, только в месте *B* отсутствует фишка. При инвертировании сети (рисунок Г.6) у пространства состояний появляется лишняя маркировка m_4, которая невозможна у исходной сети Петри.

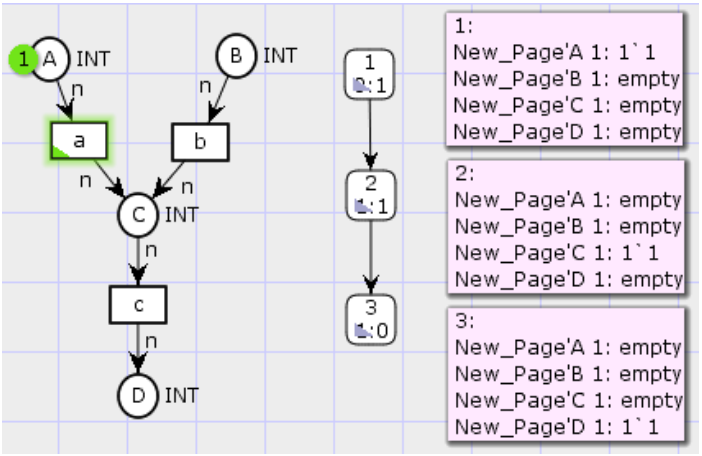


Рисунок Г.5 – Сеть Петри и граф состояний с начальной маркировкой $m_I = (1,0,0)$, пример №3

Данная маркировка m_4 является ошибочной, но и при её наличии можно достигнуть начального состояния. Аналогичные результаты были получены при начальной маркировке $m_I = (0,1,0)$.

В примере №4 (рисунок Г.7) может сработать любой из переходов $(A[a\rangle B \neg A[b\rangle C)$. После чего место *B*, либо место *C* содержит фишку.

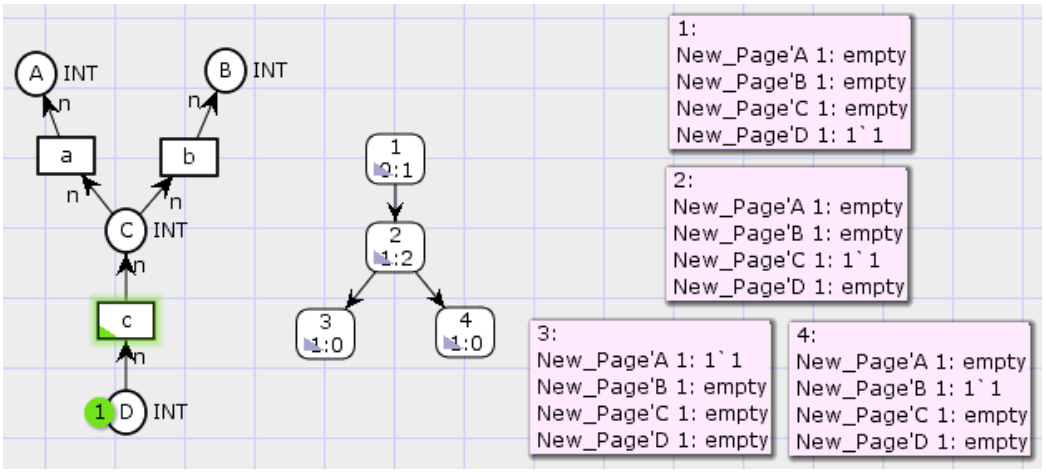


Рисунок Г.6 – Инвертированная сеть Петри и граф состояний с начальной маркировкой $m_I = (1,0,0)$, пример №3

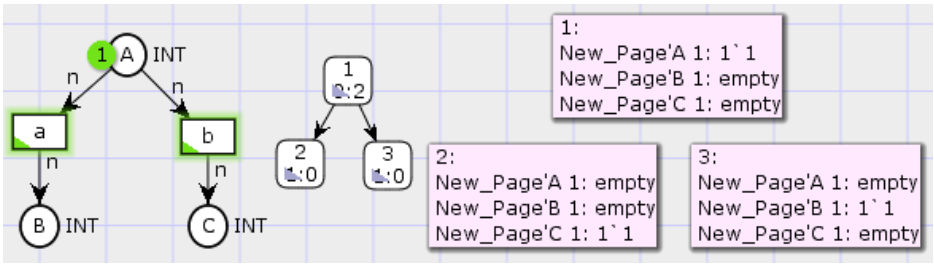


Рисунок Г.7 – Сеть Петри и граф состояний, пример №4

Так как в данном случае можно инвертировать по двум состояниям $m_2 = (0,0,1)$ и $m_3 = (0,1,0)$, представим инвертированные сети и их графы состояний для обоих случаев (рисунок Г.8).

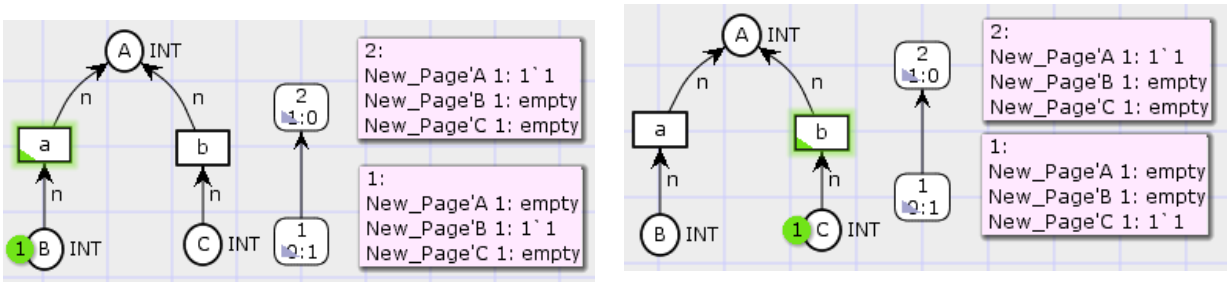


Рисунок Г.8 – Инвертированные сети с графом состояний, пример №4

Во время инверсии в пространстве состояний каждого из случаев (рисунок Г.8) происходит потеря одного из мест, тем не менее, начальная маркировка достигается с сохранением структуры исходного графа, что свидетельствует о корректно проведенной инверсии.

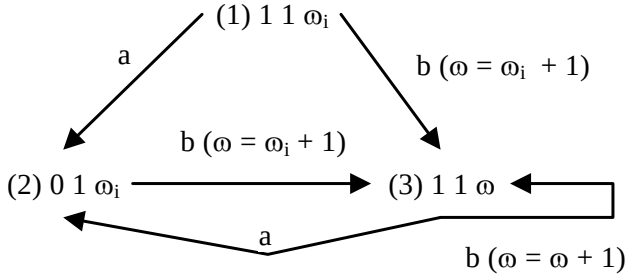


Рисунок Г.9 – граф состояний сети Петри, пример №5

В примере №5 (рисунок 3.22.д) сеть усложняется добавлением возвратных дуг и добавлением места-счётчика[44]. В связи с тем, что в моделируемой среде CPN Tools (version 3.4.0) нет возможности сгенерировать пространство состояний системы, содержащей счётчик, представим граф состояний, построенный вручную (рисунок Г.9). Здесь, ω_i – начальное значение счётчика, а ω – текущее значение.

[44] *Место-счётчик* – счётчиком можно считать место, в котором количество фишек увеличивается на постоянную величину.

К некоторым дугам пространства состояний добавлено условие, которое инкрементирует ω, тем самым увеличивая его значение в самом пространстве состояний. Данный способ может компактно и наглядно отобразить пространства состояний систем, в которых присутствуют счётчики. Нужно отметить, что затруднительно инвертировать место-счётчик, т.к. заранее неизвестно, сколько фишек он накопит и как неопределенное количество фишек будет преобразовываться в определенное.

Пример №6 (рисунок Г.10) отличается от примера №5 тем, что место-счётчик *C* может накапливать только одну фишку. Пространство состояний для данной сети имеет четыре состояния.

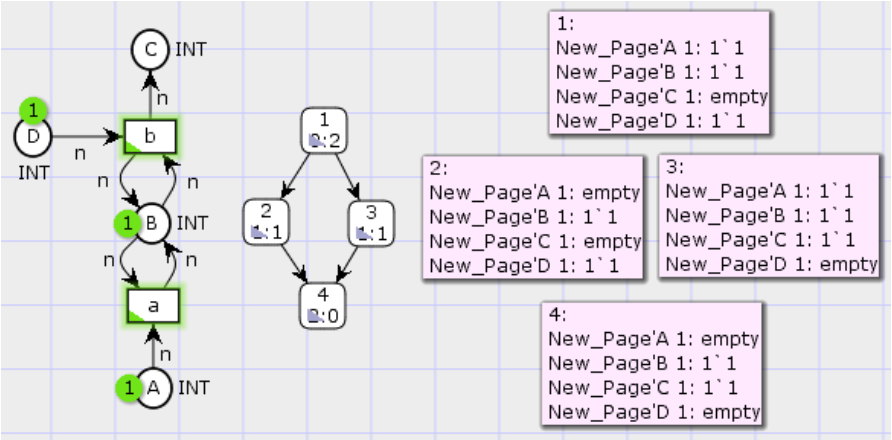


Рисунок Г.10 – Сеть Петри и граф состояний, пример №6

При прямом инвертировании данной структуры имеем сеть (рисунок Г.11), которая получена при изменении ориентации взаимосвязей между вершинами.

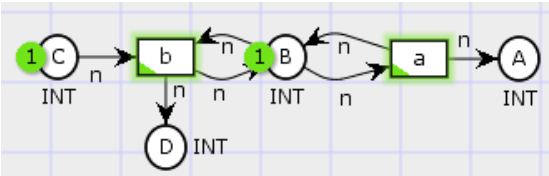


Рисунок Г.11 – Прямая инверсия сети Петри, пример №6

Стоит обратить внимание на то, что для данного случая прямая инверсия не подходит, так как ребро (*B*, *a*, *A*) образует счётчик. Место-счётчик A значительно повлияет на структуру графа состояний, а его сходство с графом состояний (рисунок Г.10) будет незначительным.

Для сети (рисунок Г.10) стоит использовать иной метод преобразования, который сможет сохранить структуру графа состояний (рисунок Г.12).

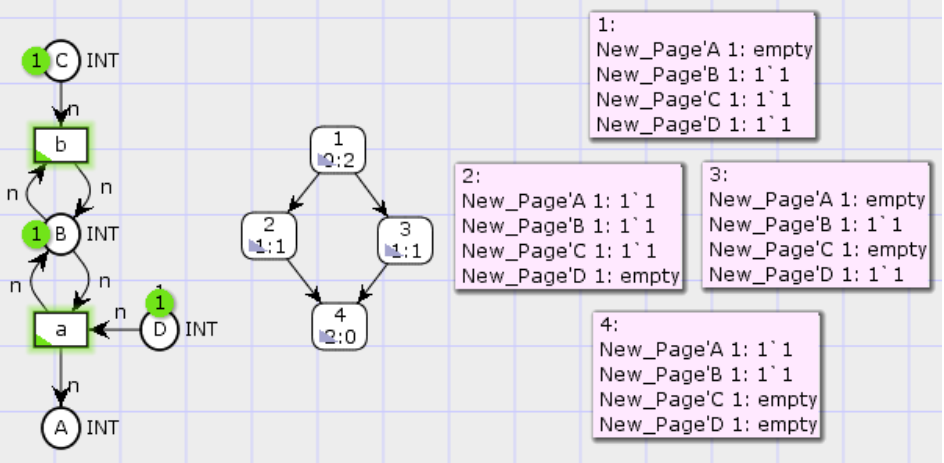


Рисунок Г.12 – Инвертированная сеть с графом состояний, пример №6

Из данной сети видно, что в каждой разметке пространства состояний $V = \{(0,1,1,1),(1,1,1,0),(0,1,0,1),(1,1,0,0)\}$ четвёртая цифра соответствует *месту-условию*[45] и не влияет на граф состояний.

Сеть Петри ***примера №7*** (рисунок Г.13) является расширением примера №6 – добавлено дополнительное место–условие *E*.

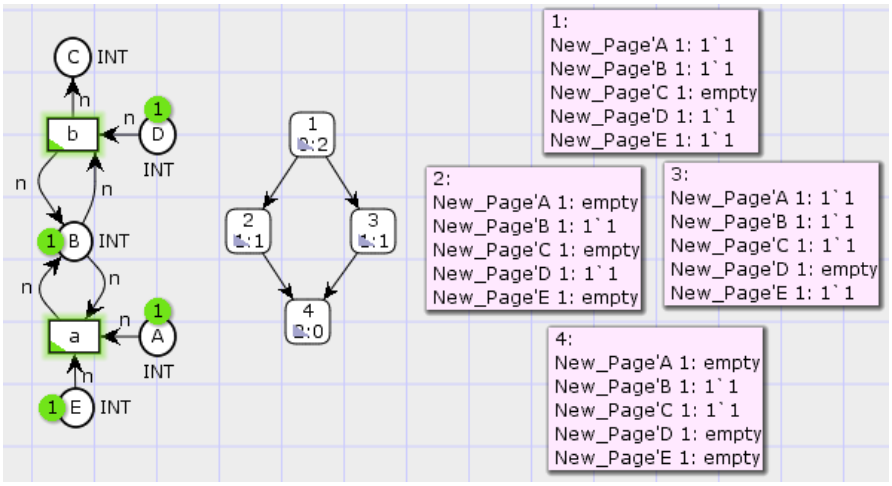


Рисунок Г.13 – Сеть Петри и граф состояний, пример №7

При инвертировании сети посредством смены ориентации дуг у рёбер $F_{DbC} \subseteq D \times b \cup b \times C$ и $F_{EaA} \subseteq E \times a \cup a \times A$, получим сеть (рисунок Г.14), в которой появляется сразу два места-счетчика *A* и *E*.

[45] *Место-условие* – место, при наличии фишки в котором разрешено срабатывания связанного с ним перехода.

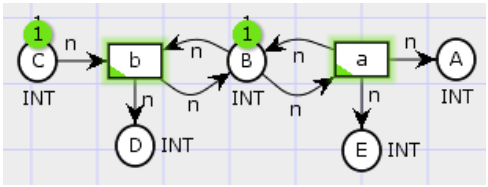


Рисунок Г.14 – Прямая инверсия сети Петри, пример №7

Так как в системе присутствует два места-счётчика, обратный граф состояний отличается от исходного, следовательно, преобразование сети (рисунок Г.13) выполнено неверно.

В данном случае прямая инверсия не удовлетворяет условию одинаковой структуры графов состояний исходной и инвертированных сетях. Для корректного преобразования следует изменить направление дуг только у рёбер $F_{BaA} \subseteq B \times a \cup a \times A$ и $F_{CbB} \subseteq C \times b \cup b \times B$ (рисунок Г.15).

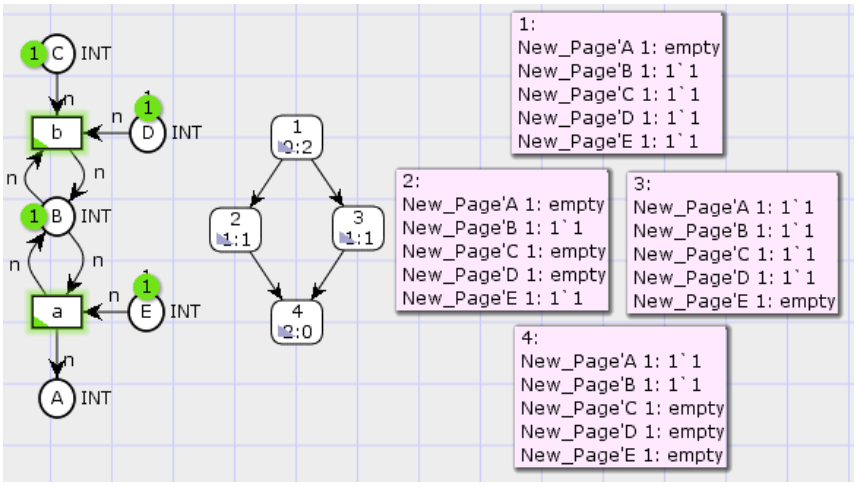
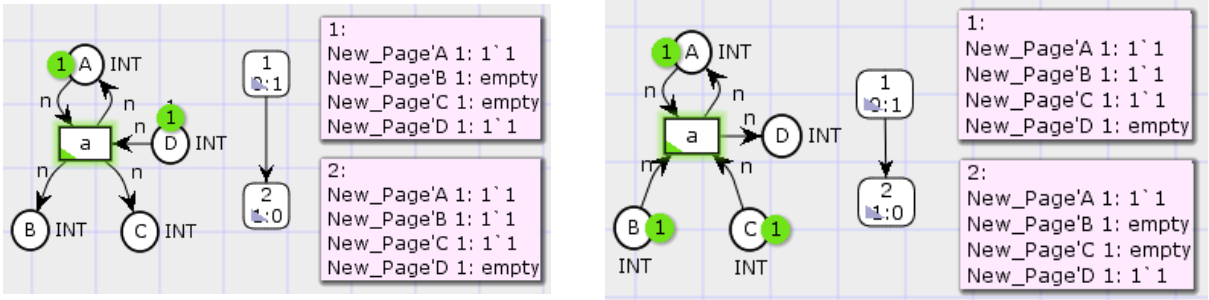


Рисунок Г.15 – Инвертированная сеть с графом состояний, пример №7

После предложенной инверсии необходимость в смене расположения мест *D* и *E* отпадает.

Сеть Петри ***примера №8*** (рисунок Г.16.а) имеет два места-счётчика *B* и *C* и одно место-условие *D*. При прямой инверсии сети (рисунок Г.16.б) получаем обратную сеть с графом состояний, структура которой подобна графу состояний исходной сети.



а) б)

Рисунок Г.16 – Сеть Петри и граф состояний, пример №8: а) исходная сеть, б) инвертированная сеть

В данном случае видно, что граф состояний сети (рисунок Г.16.б) является зеркальным отражением графа состояний сети (рисунок Г.16.а), что означает корректное преобразовании. Стоит отметить, что для структуры системы данного вида (рисунок Г.17) можно использовать прямую инверсию и быть уверенным, что не будут утеряны существующие или получены новые состояния.

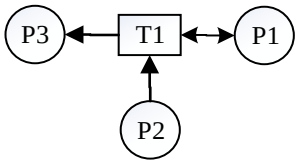


Рисунок Г.17 – Пример структуры сети Петри, при которой допустимо использование прямой инверсии

Опираясь на правила, предложенные в данном приложении, можно утверждать, что структура системы ***примера №9***: $P=\{A,B,C,D,E,F\}$, $T=\{a,b\}$, $m_I=(1,1,1,0,0,0)$; $\{A,B\}[a\rangle\{B,D,E\}$, $\{B,F\}[b\rangle\{B,C\}$ (рисунок Г.18), позволяет использовать прямое инвертирование сети без потери информации о графе состояний для нахождения начального состояния системы.

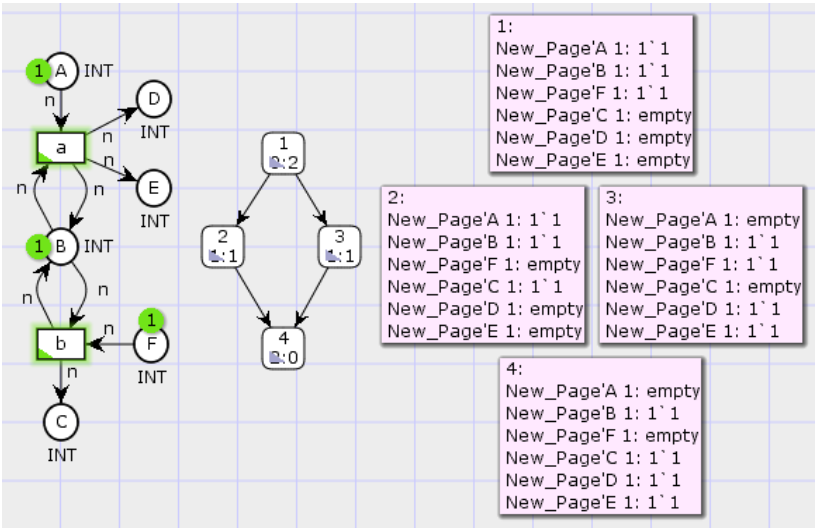


Рисунок Г.18 – Сети Петри и граф состояний, пример №9

Обратная сеть выглядит следующим образом: $P=\{A,B,C,D,E,F\}$, $T=\{a,b\}$, $m_I=(0,1,0,1,1,1)$; $\{B,D,E\}[a\rangle\{A,B\}$, $\{C,B\}[b\rangle\{B,F\}$ (рисунок Г.19).

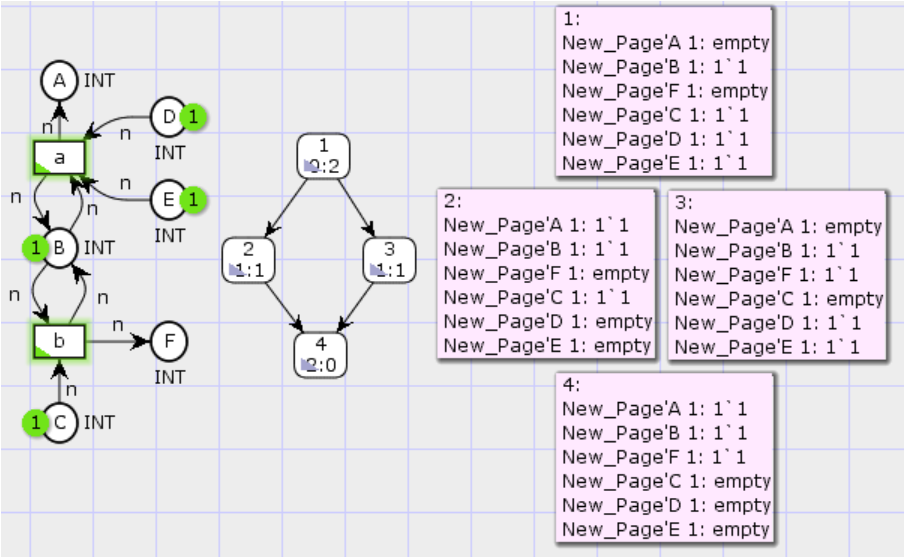


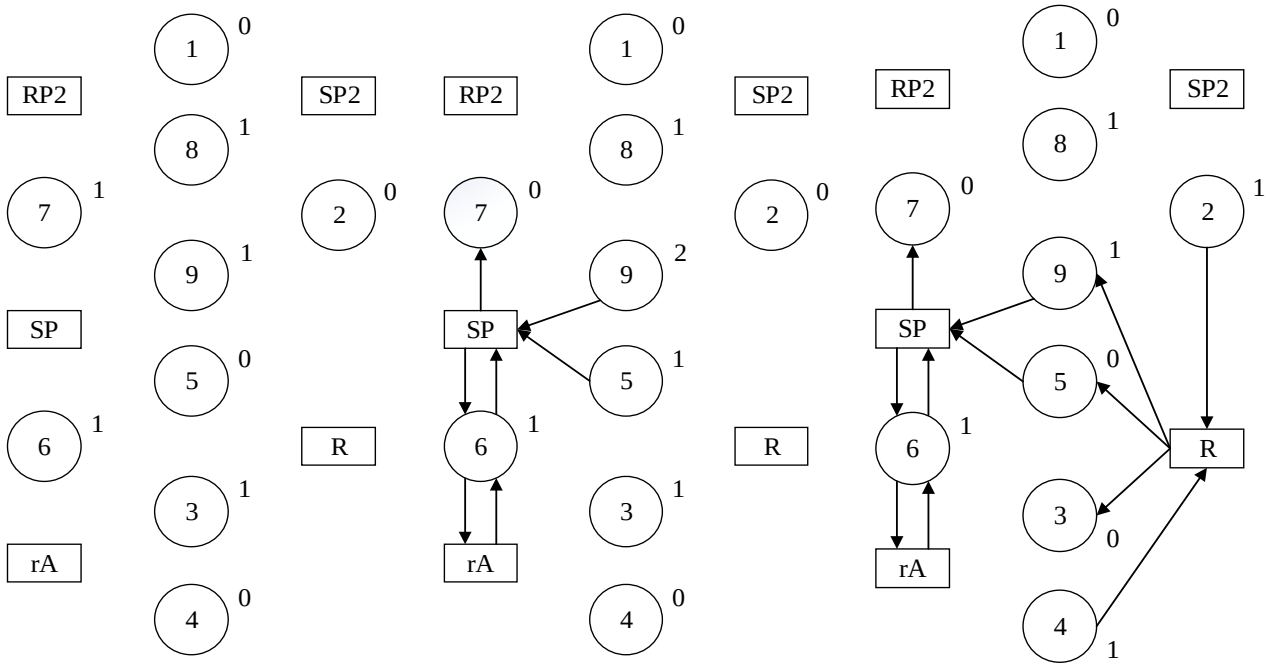
Рисунок Г.19 – Инвертированная сеть с графом состояний, пример №9

А граф состояний имеет схожую структуру: $V = \{ m_1, m_2, m_3, m_4 \}$, $E = \{ (m_1, a, m_2), (m_1, b, m_3), (m_2, b, m_4), (m_3, a, m_4) \}$.

Таким образом, представлены правила инверсии для простых сетей Петри: с разделением и слиянием потоков передачи информации и с возвратными рёбрами.

Г.2. Восстановление сети Петри из конечного графа состояний. Приводится восстановление сети Петри из *конечного* (исследованного) инвертированного графа состояний на примере протокола передачи данных. Инвертирование графа состояний имеет значительное преимущество перед инвертированием сети Петри, а именно, однозначность трансформации, которая заключается в смене ориентации взаимосвязей у состояний.

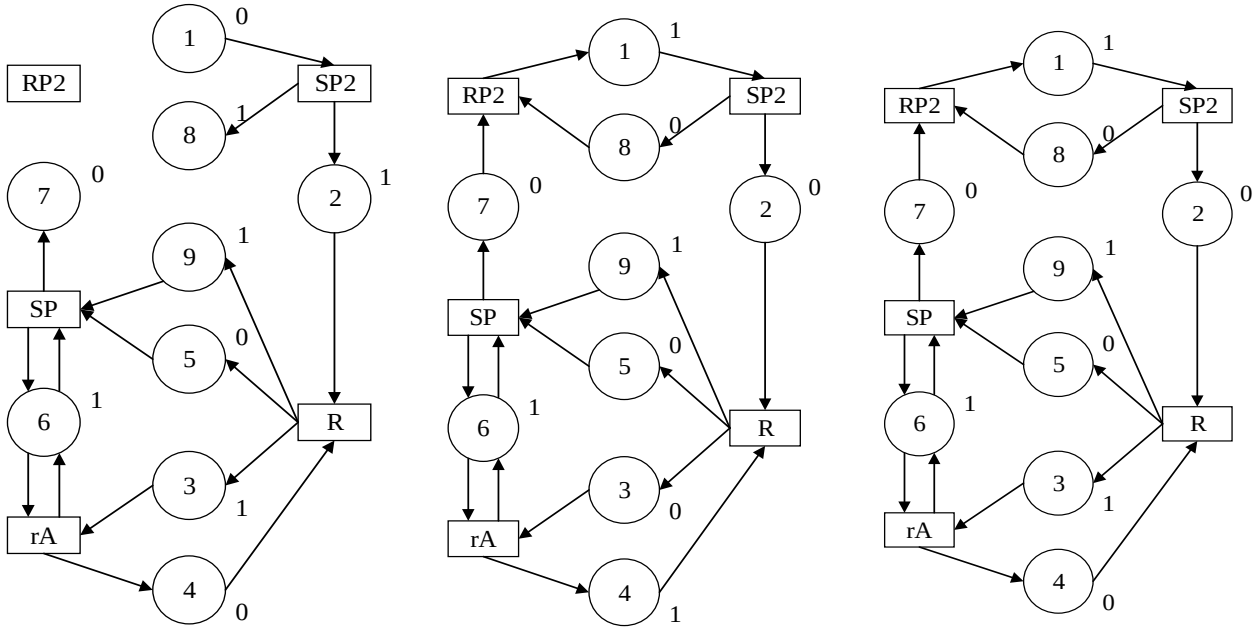
Восстановим структуру сети (рисунок Г.20-Г.27). При изучении первого состояния видно, что в системе присутствует девять мест (рисунок Г.20.а). Взаимосвязи (переходы и дуги) между этим типом вершин (местами) нам неизвестны, следовательно, на данном этапе мы получаем сеть, состоящую только из мест. При анализе первого и второго состояния с соответствующей разметкой для каждого восстановление сети продвинется ещё на один шаг: появится один переход (рисунок Г.20.б), связывающий места *5*, *6*, *7*, *9*.



а) состояние №1 б) состояние №2 в) состояние №3

Рисунок Г.20 – Восстановление сети Петри: состояние №№1–3

Как видно из рисунка 3.29, второе состояние было получено из третьего состояния. Это означает, что для преобразования графа состояний в сеть Петри необходимо сравнить эти два состояния. После сравнения разметок был добавлен переход *R* и взаимосвязи между ним и пятью местами *2*, *3*, *4*, *5*, *9* (рисунок Г.20.в).



а) состояние №4 б) состояние №5 в) состояние №6

Рисунок Г.21 – Восстановление сети Петри: состояние №№4–6

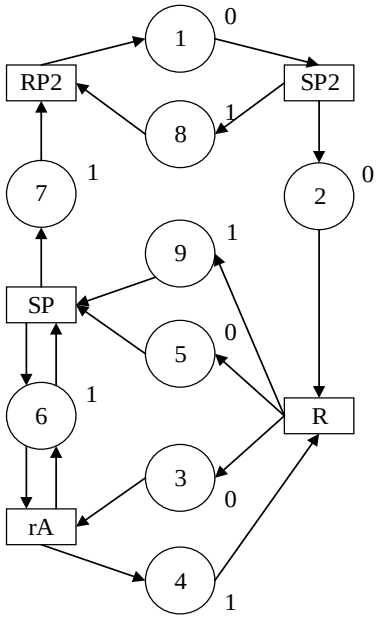


Рисунок Г.22 – Восстановление сети Петри, состояние №7

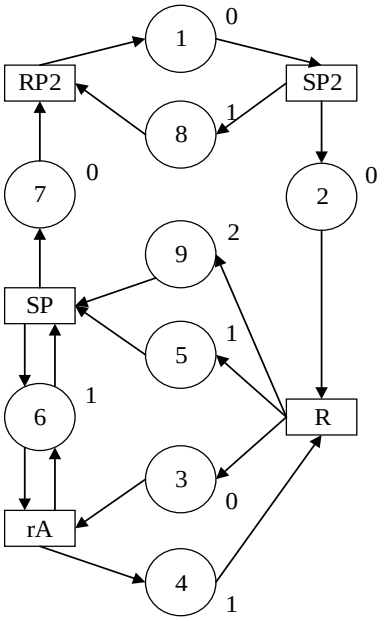


Рисунок Г.23 – Восстановление сети Петри, состояние №8

В третье состояние система попадает из четвёртого и пятого, рассмотрим последовательно эти два случая. После сравнения третьего и четвертого состояний (рисунок Г.21.а), добавим переход *rA*, связывающий места *3, 4, 6*. А при сравнении третьего и пятого состояний добавим в сеть переход *SP2* с дугами между *1, 2, 8* местами (рисунок Г.21.б). Как видно из рисунка 3.29, состояние шесть получено из первого, а его потомками являются четвёртое и пятое состояние, следовательно, при анализе взаимосвязей между данным состоянием с родителем и потомками в сети не появится новых переходов и дуг (рисунок Г.21.в). При сравнении седьмого состояния и пятого добавляем в сеть переход *RP2*, связывающий *1, 7, 8* места (рисунок Г.22). После добавления перехода *RP2*, его входных и выходных дуг сеть полностью восстановлена, а анализ состояния восемь (рисунок Г.23) служит для проверки корректности построенной сети из графа состояний. После восстановления сети из инвертированного графа состояний получаем инвертированную сеть.

Таким образом, описаны правила, при помощи которых проводится инверсия у простых сетей Петри. Инверсия сети Петри выполняется с целью проверки достижимости выбранной ранее маркировки, что необходимо при анализе любого участка графа состояний сети. Описывается восстановление сети Петри из инвертированного графа состояний.

ПРИЛОЖЕНИЕ Д

ВЫЧИСЛЕНИЕ МАРКИРОВОК СЕТИ ПЕТРИ ПРИ ИСПОЛЬЗОВАНИИ МАТРИЦ И ВЕКТОРОВ ЗАПУСКОВ

Приводится способ преобразования и анализа сетей Петри с использованием матриц, предложенных Дж. Питерсоном [87], который заключается в формировании входной и выходной функций с последующим представлением их в составной матрице изменений, а также составлении вектора начального состояния системы и задания вектора запусков (срабатывание выбранных переходов), на примере интернет-магазина [70]. Демонстрируется работа приложения по преобразованию сетей Петри, проектируемых в программной среде CPN Tools (version 3.4.0), из графического вида в матричную [75].

Для построения матриц и векторов использовалась система компьютерной алгебры MathCAD (version 14.0.0.163). Представим выходную функцию D^+:

$$D^+ = \begin{array}{c} \\ 1-3 \\ 4-6 \\ 7-9 \\ 10-12 \\ 13-15 \\ 16-18 \\ 19-21 \\ 22-24 \\ 25-27 \\ 28-30 \\ 31-33 \\ 34-36 \end{array} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 \\ 16 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 21 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 21 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 16 & 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 16 & 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 & 1 \end{pmatrix}, \quad (Д.1)$$

и входную функцию D^- рисунка 3.15:

$$D^- = \begin{array}{c} \\ 1-3 \\ 4-6 \\ 7-9 \\ 10-12 \\ 13-15 \\ 16-18 \\ 19-21 \\ 22-24 \\ 25-27 \\ 28-30 \\ 31-33 \\ 34-36 \end{array} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 \\ 0 & 0 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 21 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 16 & 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 16 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 \end{pmatrix}, \quad (Д.2)$$

элементы которых объединим по вертикали – по одному из каждой строки, и представим их в четверичной системе исчисления[46].

[46] Например, в (Д.1) первый элемент имеет значение 16 в четверичной системе исчисления, а в десятичной это значение равно 100: $1 \cdot 4^2 + 0 \cdot 4^1 + 0 \cdot 4^0 = 16$. Поскольку до преобразования элементы были сгруппированы по три, то $D_{11}^+ = 1,\ D_{12}^+ = 0,\ D_{13}^+ = 0$.

Составная матрица $D = D^{-} - D^{+}$ изменений представлена ниже:

D =

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0
16	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0
19	0	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0
22	0	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0
23	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-1	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0
30	0	1	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0
31	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1
35	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1

(Д.3)

Как видно составная матрица изменений (Д.3) не может полностью показать структуру проектируемой сети, но в совокупности с входной и выходной функциями отражает построенную ранее сеть. Кроме того, составная матрица изменений необходима для последующего анализа сетей, а точнее для анализа возможных состояний сети. Далее представим вектор начальной маркировки сети (Д.4), по которому рассчитываются возможные состояния:

$$M = (1 \quad 0). \qquad (Д.4)$$

Достижимость сети Петри интернет-магазина. Зададим несколько возможных вариантов срабатывания переходов и оценим полученные результаты. Для первого варианта вектор запусков, элементы которого сгруппированы по три и представлены в четверичной системе исчисления, выглядит следующим образом:

$$F_1 = (21 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 16 \quad 0). \qquad (Д.5)$$

Для вычисления получаемого состояния прибавим к начальной маркировки произведение вектора запусков и составной матрицы изменений $M_1 = M + (F_1 \cdot D)$:

$$M_1 = (0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0). \quad (Д.6)$$

Для второго варианта вектор запусков, элементы которого сгруппированы по три и представлены в четверичной системе исчисления, представлен ниже:

$$F_2 = (21 \quad 0 \quad 0 \quad 17 \quad 0 \quad 0 \quad 1 \quad 17 \quad 0 \quad 0 \quad 5 \quad 21). \qquad (Д.8)$$

После вычислений $M_2 = M + (F_2 \cdot D)$, получаем состояние *M*2:

$$M_2 = (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0). \quad (Д.9)$$

Для третьего варианта вектор запусков (Д.10) объединим каждые три элемента вектора и представим их в четверичной системе исчисления:

$$F_3 = (20 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 5 \quad 21). \qquad (Д.10)$$

После вычислений: $M_3 = M + (F_3 \cdot D)$, получаем состояние *M*3:

$$M_3 = (0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0). \quad (Д.11)$$

Для четвертого варианта вектор запусков (Д.12), объединим каждые три элемента вектора и представим их в четверичной системе исчисления:

$$F_3 = (4 \quad 0 \quad 0 \quad 21 \quad 21 \quad 21 \quad 21 \quad 20 \quad 0 \quad 0 \quad 0 \quad 0). \qquad (Д.12)$$

После вычислений: $M_4 = M + (F_4 \cdot D)$, получаем состояние *M*4:

$$M_3 = (0 \quad 3 \quad 2 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0). \quad (Д.13)$$

Представление сетей Петри в матричной форме при использовании разработанного приложения. Запускаем приложение и вводим название файла, содержащего анализируемую сеть Петри. Получаем матрицы D^-, D^+ объединим три элемента по горизонтали – по одному из каждого столбца, и представим их в четверичной системе исчисления:

$$D^{+}=\begin{pmatrix}
0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 2 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 4 & 5 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
21 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 16 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 16 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 21 & 20 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}. \qquad \text{(Д.14)}$$

После чего получим D:

$$
\begin{array}{l}
-1|0|0|2|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|-1|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|-1|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|-1|0|1|-1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|-1|-3|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|-1|2|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|-1|0|1|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|-1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|-1|0|0|0|0|0|-1|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|-1|-1|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|-1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|-1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|1|1|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|-1|0|0|0|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|-1|0|0|0|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|-1|0|0|0|1|1|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|-1|-1|0|0|1|1|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|0|-1|-1|1|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-2|0|1|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|0|1|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|1|1|1|1|1|0|0|0|0|0|0|0|0|0|0|0|-1|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|1|-1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|0|1|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|1|0| \\
0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|0|0|0|0| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|1| \\
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|-1|
\end{array}
$$

Также определим вектор начального состояния:

$$
1|2|4|0|0|0|0|0|0|0|4|0|0|0|0|0|0|1|1|1|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|6|0|0|0|0|. \qquad \text{(Д.16)}
$$

Все полученные данные были сохранены в текстовом файле, под названием *“result.txt”*, находящийся в директории приложения. При очередном моделировании данные в файле заменяются.

В данном приложении используется способ преобразования комбинации мест и переходов маркированной сети Петри в матричную форму с последующим получением достижимых маркировок при использовании векторов запусков на примере интернет-магазина. Показана работа приложения по данному преобразованию сетей Петри, построенных в программной среде CPN Tools (version 3.4.0), на примере простой сети Петри.

ПРИЛОЖЕНИЕ Е

АЛГОРИТМЫ ИССЛЕДОВАНИЯ ПРОСТРАНСТВА СОСТОЯНИЙ

В данном приложении приводятся алгоритмы исследования графа состояний методом “плавающей” линии (алгоритм Е.1, алгоритм Е.2), а также методом возвратных рёбер (алгоритм Е.3), которые базируются на известных алгоритмах (S. Christensen [105], L.M. Kristensen [122], M. Westergaard [147]), *дополненные поясняющими комментариями* [77], позволяющими при необходимости модифицировать данные алгоритмы.

Алгоритм Е.1 – Алгоритм “плавающей” линии

1: $Unprocessed \leftarrow \{ m_I \}$ // выбираем все не посещённые состояния,

2: $Nodes.Add(m_I)$ // добавляем выбранные состояния к вершинам,

3: $while \neg Unprocessed.Empty() \; do$ // пока существуют не посещённые состояния, выполняем …

4: $s \leftarrow Unprocessed.GetMinElement()$ // выборку состояний с минимальным индексом,

5: $for\ all(t, m')\ such\ that\ m \xrightarrow{t} m'\ do$ // для всех корней, полученных из рёбер, выполняем …

6: $if\ \varphi(m) \not\sqsubseteq \varphi(m')\ then$ // Если прогрессивная оценка m не входит в прогрессивную оценку m', тогда,

7: $Stop(\text{"}Progress\ measure\ rejected\text{:"}, (m, t, m'))$ // останавливаем выполнение и не сохраняем данное ребро,

8: $end\ if$ // конец если,

9: $if\ \neg(Nodes.Contains(m'))\ then$ // если вершины не содержат m', тогда,

10: $Nodes.Add(m')$ // добавляем к вершинам состояние m',

11: $Unprocessed.Add(m')$ // добавляем к не посещённым состояниям состояние m',

12: $end\ if$ // конец если,

13: $end\ for$ // конец цикла *for*,

14: $Nodes.GarbageCollect(\min\{\varphi(m) \mid m \in Unprocessed\})$ // Собираем состояния с наименьшей прогрессивной оценкой,

15: $end\ while$ // конец цикла *while*.

Алгоритм Е.2 – Алгоритм обобщённого sweep-line метода

1: $Roots \leftarrow \{ m_I \}$ // Из корней получаем множество состояний,

2: $Nodes.Add(m_I)$ // Добавляем состояния к узлам,

3: $while \neg(Roots.Empty())\ do$ // пока корни непустые выполняем …

4: $Unprocessed \leftarrow Roots$ // выборку не посещённых состояний из корней,

5: $Roots \leftarrow \emptyset$,

6: $while \neg \left(Unprocessed.Empty() \right) do$ // пока список не посещённых состояний не станет пустым,

7: $m \leftarrow Unprocessed.GetMinElement()$ // Выбираем из не посещённых состояний состояния с минимальным индексом,

8: $for\ all\ \left(t, m' \right)\ such\ that\ m \xrightarrow{t} m'\ do$ // для всех корней, входящих в существующие рёбра, выполняем условие,

9: $if\ \neg \left(Nodes.Contains\left(m' \right) \right) then$ // если узлы содержат состояние m', тогда,

10: $Nodes.Add\left(m' \right)$ // добавляем состояние m' в узлы,

11: $if\ \varphi\left(m \right) \supset \varphi\left(m' \right) then$ // если прогрессивная оценка m входит в прогрессивную оценку m', тогда,

12: $Nodes.MarkPersistent\left(m' \right)$ // добавляем состояние в список постоянных состояний,

13: $Roots.Add\left(m' \right)$ // добавляем состояние m' в корни,

14: $else$ // иначе,

15: $Unprocessed.Add\left(m' \right)$ // добавляем состояние m' в не посещённые состояния,

16: $end\ if$ // конец если,

17: $end\ if$ // конец если,

18: $end\ for$ // конец цикла *for*,

19: $Nodes.GarbageCollect\left(\min\left\{ \varphi\left(m \right) \mid m \in Unprocessed \right\} \right)$ // Собираем состояния с наименьшей прогрессивной оценкой,

20: $end\ while$ // конец цикла *while*,

21: $end\ while$ // конец цикла *while*.

Алгоритма Е.3 – Алгоритм ComBack метода

1: $n \leftarrow 1$ // присваиваем номеру состояния 1,

2: $StateTable.Init(); StateTable.Insert\left(H\left(m_I \right), 1 \right)$ // инициализируем глобальную структуру данных: таблица состояний (хеш-сумма для состояния и порядковый номер состояния),

3: $WaitingSet.Init(); WaitingSet.Insert\left(m_I, 1 \right)$ // инициализируем глобальную структуру данных: список ожидающих состояний (состояние и номер состояния),

4: $BackEdgeTable.Init(); BackEdgeTable.Insert\left(1, \perp \right)$ // инициализируем глобальную структуру данных: таблица возвратных рёбер (с номер состояния и возможных корней),

5: $while \neg WaitingSet.Empty()\ do$ // пока список ожидающих состояний не пуст,

6: $\left(m, n' \right) \leftarrow WaitingSet.Select()$ // из данного списка выбираем пару: состояние и соответствующий ей номер,

7: *for all* t, m' *such that* $(m, t, m') \in \Delta$ *do* // для всех корней, входящих во множество рёбер, выполняем …

8: *if* $\neg$ *Contains*(m') *then* // если состояние m' не было исследовано, то,

9: $n \leftarrow n + 1$ // инкрементируем номер состояния,

10: *StateTable.Insert*$(H(m'), n)$ // добавляем в таблицу состояний хеш-значение исследуемого состояния и его номер,

11: *WaitingSet.Insert*(m', n) // в список ожидания добавляем состояние и его номер,

12: *BackEdgeTable.Insert*$(n, (m', t))$ // в таблицу возвратных рёбер добавляем номер состояния и возможные корни,

13: *proc Contains*(m') *is* // процедура *Contains* выполняет цикл, при котором,

14: *for all* $n \in$ *StateTable.Lookup*$(H(m'))$ *do* // для всех номеров состояний из таблицы состояний выполняем выборку,

15: *if Matches*(n, m') *then* // если данный номер и состояние существует, тогда,

16: *return tt* // возвращаем *true,*

17: *return ff* // иначе возвращаем *false,*

18: *proc Matches*(n, m') *is* // процедура *Matches* реализует,

19: *return* $m' =$ *Reconstruct*(n) // возвращение состояния m' равное номеру *Reconstruct* ,

20: *proc Reconstruct*(n) *is* // процедуре *Reconstruct* соответствует условие,

21: *if* $n = 1$ *then* // если номер состояния равен 1, тогда,

22: *return* m_I // возвращаем состояние m_I ,

23: *else* // иначе,

24: $(n', t) \leftarrow$ *BackEdgeTable.Lookup*(n) // выполняем поиск нужного номера состояния,

25: $m \leftarrow$ *Reconstruct*(n') // и присваиваем соответствующему состоянию,

26: *return Execute*(m, t) // возвращаем вычисленный дескриптор состояния.

Таким образом, приведены алгоритмы, дополненные поясняющими комментариями, для исследования пространства состояний системы: метод “плавающей” линии, метод возвратных рёбер. Из приведённых способов метод возвратных рёбер является лучшей альтернативой между ограниченными ресурсами, выделяемыми на анализ, и полнотой исследования пространства состояний.