СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2013. – № 1(71).** – 96–131

УДК 62-50:519.216

# РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ СОВМЕСТНОМ ИСПОЛЬЗОВАНИИ UML-ДИАГРАММ И СЕТЕЙ ПЕТРИ (ОБЗОР)*

А.В. МАРКОВ

Приводится обзор работ, посвященных созданию программного обеспечения с помощью сетей Петри и UML-диаграмм. Выделены основные аспекты, достоинства и недостатки сетей Петри. Описана взаимосвязь между поведенческими диаграммами UML и структурными. Описана методика создания программного обеспечения с использованием сетей Петри и UML-диаграмм.

**Ключевые слова:** инженерия ПО, UML-диаграммы, сети Петри, CPN Tools, параллельные процессы, автоматическая трансляция, слияние диаграмм, рекурсия, правила преобразования, методика разработки ПО.

## ВВЕДЕНИЕ

За последние годы программные продукты проникли во все сферы жизнедеятельности человека. Отдельные области требуют надежного высокопроизводительного программного обеспечения (ПО), поэтому качество разрабатываемого ПО приобретает важное значение. Существующие подходы к разработке ПО [1–7] не в силах гарантировать его качество в достаточной степени. Тестирование проектируемых систем с высокой долей вероятности может гарантировать отсутствие ошибок в программе, но некоторые из них могут проявляться в специфических условиях, которые невозможно смоделировать в тестовой среде. В связи с этим вопрос разработки качественного ПО являются достаточно актуальными.

В настоящее время разработчики используют формальные способы разработки ПО [1–24]. Особый интерес представляют такие из них, которые способны гарантировать качество и безотказность создаваемого ПО. Но сложность вышеприведенных способов разработки ограничивает их использование на практике.

Следует выделать способ формального доказательства корректности проектирования ПО, которым является применение алгоритмов на основе UML-диаграмм и сетей Петри [7–15, 17–24].

* Статья получена 16 августа 2012 г.

## 1. АНАЛИЗ ПОДХОДОВ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Правильный выбор модели, методов и инструментов для анализа, проектирования, реализации системы и организация процесса разработки требует знания всех аспектов развития современных методологий и технологий разработки. Для упрощения разработки программных продуктов необходимо придерживаться определенных парадигм программирования и способов разработки. Строгая реализация по заранее спроектированным алгоритмам позволяет создавать программный продукт поэтапно, избавляет от постоянного внесения кардинальных изменений в программный код продукта. Для более подходящего варианта разработки следует проанализировать наиболее популярные способы разработки ПО.

На сегодняшний день в программной инженерии существует достаточное количество разнообразных подходов для написания программного обеспечения со стороны управления процесса разработки. Самые распространенные: RUP, Agile, «чистая комната», RAD, итеративная разработка, MSF и другие.

– Agile (гибкая методология разработки). Данный способ представляет набор принципов и ценностей, определяющих поведение команды разработчиков. Agile в большей степени зависит от личных и профессиональных качеств всех участников команды, чем от внутренних процессов. Чаще используется небольшими группами программистов и менеджеров и больше подходит для определения ценности продукта с точки зрения ведения бизнеса и общения с заказчиком, чем для построения конечного продукта.

– Итеративная разработка – группа методологий, которые основаны на принципе разработки ПО повторяющегося цикла: планирование – реализация – проверка – оценка. Для каждой итерации существует стадия планирования и определения условий завершения итерации. Достоинством этого способа является возможность постоянно отслеживать текущий статус готовности продукта, обнаруживать проблемы на более ранних стадиях разработки.

– RUP. Является итеративным способом разработки, имеющим следующие особенности:

1) полный цикл разработки ПО разбит на фазы: начало, уточнение, построение и внедрение;

2) определены основные и дополнительные процессы при разработке программного обеспечения;

3) определен порядок работ по процессам.

Считается, что RUP более формализован, чем «простая» итеративная методология, что делает его более легким для внедрения.

– Каскадная модель[1] представляет способ разработки ПО, который основан на последовательном прохождении следующих фаз: определение требований – проектирование – реализация – интеграция – верификация – внедрение – поддержка. К недостаткам данного подхода можно отнести: сложность внесения изменений в требования, программный код продукта на поздних стадиях разработки и высокие риски получения неработоспособного продукта, так как верификация происходит только на заключительных этапах.

Таким образом, для создания качественного программного обеспечения, минимизации рисков, постоянной верификации конечного продукта, большее всего подходит итеративная модель разработки.

## 2. АНАЛИЗ ПАРАДИГМ НАПИСАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Существует несколько общепринятых парадигм программирования: структурное, императивное, декларативное, функциональное, объектно-ориентированное программирование.

*Декларативное программирование* – программа описывает не процесс создания сущности, а саму сущность. Самым ярким примером декларативного программирования может служить web-страница, написанная в html.

*Императивное программирование* описывает процесс вычисления в виде инструкций, изменяющих состояние программы.

*Структурное программирование*. Методология, в основе которой лежит представление программы в виде иерархической структуры базовых блоков (последовательное исполнение, ветвление, цикл). При составлении программы повторяющиеся участки кода оформляются как отдельные функции или процедуры.

*Функциональное программирование*. Процесс вычисления трактуется как вычисление значений функций в математическом понимании последних. Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы.

*Объектно-ориентированное программирование.* Основными концепциями которого являются понятия объектов и классов. Отношения между объектами описывается при помощи наследования (отношение частное-общее), инкапсуляция. Сами объекты вступают в такие отношения (ассоциации), как часть-целое (агрегация), использование и др.

[1] Более известная под названием «модель водопада».

Первым шаг процесса объектно-ориентированного проектирования программной системы – создание структурной диаграммы. На следующем этапе, при определенной степени проработанности структуры, разработчик отслеживает динамические аспекты архитектуры и описывает алгоритм работы системы (рис. 1).

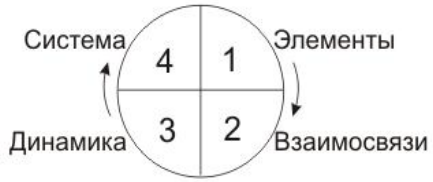



*Рис. 1.* Этапы процесса моделирования
[19, рис. 1]

Моделирование программной системы начинается с описания ее структуры, множества составляющих ее элементов. После чего устанавливаются взаимосвязи между элементами. На следующем этапе описывается процесс взаимодействия взаимосвязанных элементов, а завершающим этапом является объединение знаний о структуре, связях и динамике элементов в одно целое – т. е. систему. Структурная модель системы отражает набор ее элементов и их взаимосвязи, динамическая модель – процессы взаимодействия элементов между собой и с внешней средой. Необходимо выделить, что динамическая модель отчасти включает в себя структурную, так как любое действие системы связанно с какими-либо данными – элементами. При описании структуры какой-либо системы не требуется описание ее поведения.

Спроектированную модель необходимо проверить по ряду критериев. Правильность модели – это сложное свойство, которое включает несколько уровней, для прохождения каждого из которых следует применять специальную технику.

*Проверка корректности* (syntax checking). В последнее время задачу проверки корректности модели с точки зрения синтаксиса языка удается полностью автоматизировать. При помощи многих CASE-инструментов формирования описания моделей (программы-редакторы) вообще не позволяют создавать синтаксически некорректные конструкции.

*Верификацией* (verification) называют подтверждение смыслового соответствия модели входным требованиям с привлечением различных математических методов: синтаксический и семантический, в том числе и в автоматизированном режиме. Широко применяются также динамические методы верификации, т. е. тестирование.

*Аттестация* (validation). Предметом аттестации является прагматическое качество модели – соответствие ожиданиям потребителя.

Существуют *автоматизированные инструменты,* позволяющие *в ходе статической проверки* выявлять конструкции, которые способны вызвать нежелательные эффекты, однако недостатком является невозможность обнаружения *концептуальных и структурных расхождений между моделями*, полученных в результате выполнения различных этапов.

Наряду со статической проверкой применяется динамическая, которая заключается в пробном использовании (испытании) модели согласно ее назначению в специальном «тестовом» окружении. Завершенное программное обеспечение тестируется при заданном наборе входных параметров, для которых правильные значения результатов заранее известны. Существуют CASE-средства, позволяющие автоматически генерировать достаточно полные наборы тестовых случаев. Основной *проблемой динамической проверки моделей СИСТЕМЫ* является *невозможность* в некоторых случаях *сформировать условия реального тестового окружения*.

Из вышесказанного можно сделать вывод о том, что объектно-ориентированная парадигма написания ПО является наиболее используемой. За последние годы разработаны графические средства, ориентированные на применение ООП, например, UML.

## 3. ПРИМЕНЕНИЕ UML-ДИАГРАММ В РАЗРАБОТКЕ ПО

UML, являясь языком графического описания программных кодов, может быть транслирован в классическое представление кода для различных языков программирования. Таким образом, для создания качественного программного обеспечения целесообразно использовать языки, поддерживающие парадигму ООП. Использование нотации UML позволяет увеличить скорость разработки продукта и уменьшения количества синтаксических, семантических и др. видов ошибок [27–29].

Семантика языка UML имеет мощные средства расширения языка: стереотипы, именованные значения и ограничения. Также стоит отметить, что в состав UML входит формальный текстовый язык Object Constraint Language курс, с помощью которого есть возможность описать различные инварианты типов, пред- и постусловия.

Изначально имелась возможность транслировать диаграммы классов на выбранные языки программирования, поэтому UML применялось только для начального проектирования архитектуры системы при помощи диаграмм развертывания (*deployment diagram*), компонентов (*component diagram*), классов (*class diagram*).

Сегодня появилась возможность автоматически преобразовывать UML-диаграммы в программный код, а изменения в программном коде – обратно в UML (например, Rational Rhapsody).

Приведем описание различных UML-диаграмм при разработке ПО:

– *диаграмма вариантов использования* (*use case diagram*). Данная диаграмма используется, чтобы показать отношения между актерами в системе и их последовательностью действий. Use Case-диаграмму целесообразно применять при формировании функциональных требований к системе, как показано в работах;

– *диаграмма классов* (*class diagram*) является одной из структурных диаграмм. На диаграмме классов отображается структура системы – классы, атрибуты, операторы, а также отношения между классами;

– *диаграмма деятельности* (*activity diagram*), *диаграмма состояний* (*state machine diagram*) относятся к диаграммам поведения. На данных диаграммах показано поведение объектов класса при помощи различных представлений. Например, диаграмма деятельности как блок-схема, а диаграмма состояний с точки зрения теории автоматов. Использование диаграмм деятельности и диаграммы состояний дают возможность наглядно представлять сложные алгоритмы;

– *диаграммы коммуникации (communication diagram) и последовательности (sequence diagram).* С помощью этих диаграмм можно выразить взаимодействия объектов системы.

На рис. 2 приведен пример диаграммы состояний[2] [4], которая иллюстрирует динамические свойств системы. Правильность данной диаграммы состояния можно проверить только путем тестирования конечного программного кода продукта.

В работе [9] предложен минимальный набор, соответствующий набору моделей, рекомендованных RUP:

- *диаграмма вариантов использования* является графическим описанием системы с точки зрения пользователей и полной диаграммой вариантов использования, описывающей общее функциональное назначение системы;
- критичные варианты использования, которые требуют нетривиальных подходов к проектированию и реализации, уточняются на *диаграммах последовательности* (или кооперациий);
- на *диаграмме процессов* показаны процессы, потоки системы и их взаимосвязь;

[2] На данной диаграмме изображены состояния CPU (*Inizialize*, *Wait4Run*, *BlocksExec*, *DefuneCurState*). Смена состояний достигается при вызове событий, названия которых приведены над стрелочками, например при вызове события *JumperToSTOP* происходит смена состояния с *BlocksExec* на *Wait4Run*.

- более детально операции потоков и процессов отображаются на *диаграммах деятельности*;
- объекты предметной области отображаются на *диаграмме классов*;
- отображение взаимодействия между классами можно наблюдать на *диаграмме состояний* и *диаграмме последовательности.*

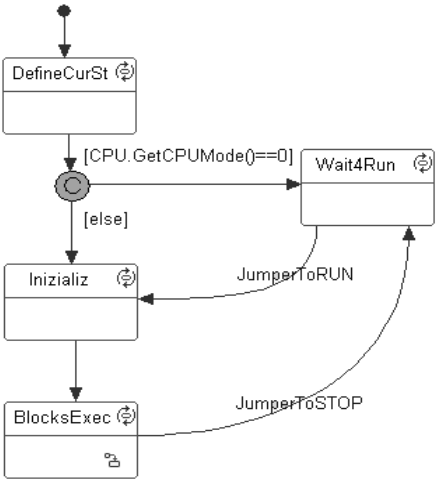



*Рис. 2.* Диаграмма состояний

Стоит подчеркнуть, что зачастую используют более узкий набор диаграмм, которые вполне удовлетворяют условиям и требованиям, предъявляемым разработчикам.

Явными достоинствами UML являются увеличение скорости разработки программы и уменьшение требований к знаниям разработчика. Среди недостатков UML выделяют отсутствие строгого формализма, перекрытие назначений различных диаграмм, отсутствие встроенных механизмов верификации и валидации диаграмм, отсутствие определенного набора UML-диаграмм, необходимого для реализации программной системы. Несмотря на все вышеперечисленные недостатки UML-диаграмм их использование в проектах с объектно-ориентированной парадигмой разработки является оправданным.

## 4. ДОСТОИНСТВА И НЕДОСТАТКИ ДИАГРАММ СОСТОЯНИЯ И ДЕЙСТВИЙ ПРИ ОПИСАНИИ ДИНАМИЧЕСКИХ СВОЙСТВ КЛАССОВ

На сегодняшний день отсутствуют четкие рекомендации применения диаграмм состояния и действий. В работе [26] описаны подходы описания динамических классов, приведены их достоинства и недостатки.

Опираясь на большинство алгоритмов разработки программного обеспечения и принципы объектно-ориентированного программирования, необходимо составить диаграмму классов для рассматриваемой задачи. Затем необходимо составить диаграмму состояния, содержащую в себе внутренние состояния объекта.

Далее рассмотрим описание той же задачи, но уже при помощи диаграмм действий. Диаграммы действий не имеют функционала для отображения состояний объекта, но их достоинством является более детальное и наглядное отображение методов классов. Поэтому для описания динамических свойств класса используется описание методов класса при помощи диаграмм действий.

Описание, которое выполнено при помощи диаграмм состояний, представляется более компактным, чем реализация при помощи диаграмм действий. Что вызвано отсутствием возможности при помощи диаграмм действий показать текущее состояние объекта, а описание, основанное на диаграммах состояния, наглядно отображает динамические свойства объекта.

Еще одним достоинством диаграмм состояния является возможность их расширения при добавлении подсостояний.

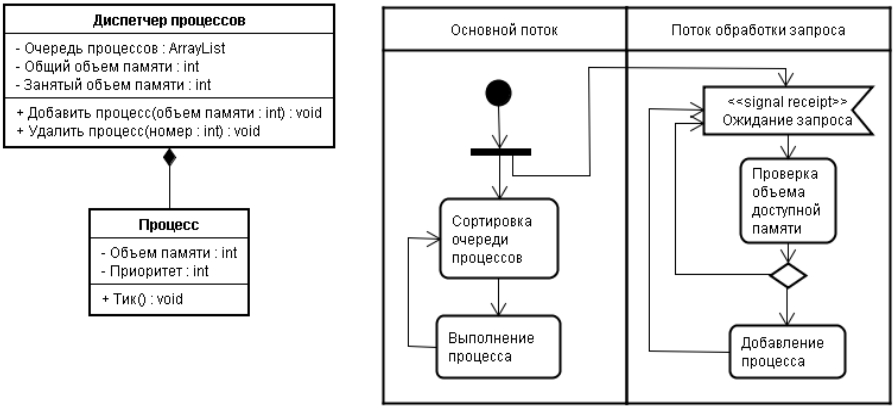
Из всего вышесказанного следует, что использование диаграмм действий обосновано при линейном исполнении программы и описании алгоритмической части ПО. Также использование диаграмм действий дает возможность более наглядно представить систему с малым количества состояния объекта, так как при этом диаграмма состояний будет представлять одно состояние с множеством переходов, замкнутых на данное состояние.

## 5. ВЗАИМОСВЯЗЬ ДИАГРАММ

В работе [30] выделяется связь между структурными и поведенческими диаграммами. Эти два вида диаграмм описывают одну и ту же систему только с разных ракурсов, и четкая связь между этими диаграммами в языке UML отсутствует. 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). Логика работы системы хорошо описывается при использовании теории машины состояний и теории графов. К примеру, диаграмма деятельности фактически является орграфом, который состоит из множества действий, т. е. вершин, и их связей (дуги). Виды связи между элементами структуры и действиями могут быть различны, например, это отношение ассоциации и отношение композиции. Связь динамики со структурой может предоставить дополнительную информацию о работе системы, отразить ее внутренние процессы, что крайне ценно для имитационного моделирования и верификации. Поэтому перспективным использованием подобного синтеза диаграмм является решение задачи автоматической трансляции UML-диаграмм в сети Петри.

Взаимосвязь диаграмм показана на примере диспетчера процессов. Основная задача диспетчера процессов – распределение процессорного времени между запущенными приложениями (согласно какому-либо алгоритму многозадачности). Соответственно процесс и диспетчер будут являться ключевыми объектами в данном проекте. Диаграмма классов изображена на рис. 3, *а*.




*а* *б*

*Рис. 3.* Диаграмма классов (*а*) и деятельности (*б*) [19, рис. 2]

В класс процесса входят такие данные, как приоритет и объем занимаемой им памяти. Класс диспетчера содержит общий и занятный объем памяти, а также список запущенных процессов. Поскольку в классе диспетчера будут создаваться и храниться объекты класса процесса, на диаграмме между этими

классами показана композитная связь. Простейшая диаграмма деятельности работы диспетчера представлена на рис. 3, *б*. У диспетчера предусмотрено два внутренних потока работы. Первый поток отвечает за цикл обработки запущенных процессов из двух стадий: сортировка списка процессов по приоритету и выполнение процесса (выделение процессорного времени). Во втором потоке осуществляется проверка возможности выделения запрошенного объема памяти и добавление процесса в общий список.

## 6. СИНТЕЗ ДИАГРАММ

На полученной схеме одновременно изображаются диаграмма классов и диаграмма деятельности. Связь определенного действия (элемента динамической диаграммы) с элементом структуры можно изобразить соединительной линией (рис. 4, *а*).

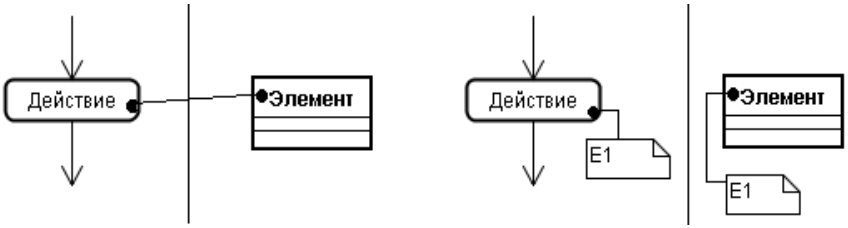



*Рис. 4.* Варианты представления связи диаграмм [19, рис. 3]

Второй способ (рис. 4, *б*) заключается в том, что ряд элементов структурной диаграммы снабжается маркерами, и эти же маркеры указываются для элементов диаграммы деятельности. Данный способ позволит сохранить читаемость диаграмм, так как связи между диаграммами в виде линий могут затруднить восприятия схемы. Разные типы связи между диаграммами также визуально отличаются на схеме. Пунктирная линия – связь ассоциации (обращение на чтение или запись) с соответствующей структурой данных (элементом), сплошная линия означает создание объекта в результате действия. Интегрированная диаграмма для рассматриваемой задачи диспетчера процессов принимает вид, изображенный на рис. 5. Во время трансляции полученного синтеза диаграмм в сеть Петри основной источник информации – это диаграмма деятельности. Для отображения связи со структурой сеть Петри дополняется элементами, которые моделируют процессы работы с данными.

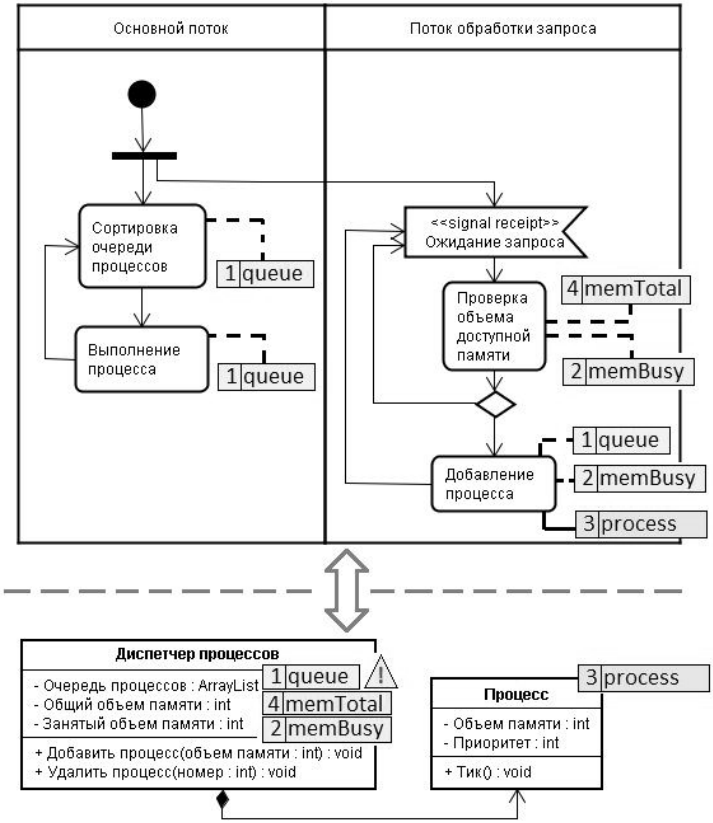



*Рис. 5.* Синтез структурной и динамической диаграмм [19, рис. 4]

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 сет Петри.

## 7. ИСПОЛЬЗОВАНИЕ СЕТЕЙ ПЕТРИ В РАЗРАБОТКЕ ПО

Весьма интересным и полезным средством разработки программного обеспечения являются сети Петри, предназначенные для анализа не только правильности каждой диаграммы, на основе которой будет автоматически сгенерирован или реализован вручную код, но и согласованности UML-диа-

грамм. В работах представлены комплексные подходы к преобразованию в раскрашенную иерархическую сеть Петри совокупности диаграмм: классов, состояний класса на его жизненном цикле и взаимодействия классов и анализу. Чаще всего используют раскрашенную иерархическую сеть Петри для моделирования UML-диаграммами объектно-ориентированного ПО. Данные сети позволяют с помощью цветов задавать типы данных, определять переменные и использовать их значения для управления изменением состояний и передачи потока управления.

Сети Петри получили широкое распространение в разработке многих сфер деятельности: от проектирования сетевых протоколов до проверки многопоточного программного обеспечения [30–54].

Сети Петри – математический аппарат для моделирования динамических дискретных систем. Представляют собой двудольный ориентированный граф, соединяющий вершины двух типов – позиции (места) и переходы, связанные между собой дугами, вершины одного типа не могут быть соединены непосредственно. В позициях могут размещаться метки (маркеры), способные перемещаться по сети.

Графически места изображаются как овалы, переходы как прямоугольники. Например, P1, P2, P3 – места, T1,T2 – переходы сети Петри, изображенной на рис. 6.

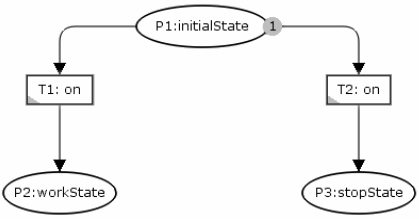



*Рис. 6.* Пример сети Петри

Каждое место и переход имеет надпись, которая поясняет назначение элемента сети (P1 – *initialState*, T1 – *on* и д.р.).

Наличие состояния или ресурса показано с помощью метки. При начальной маркировке место P1 содержит одну метку. При срабатывании перехода T1 метка из P1 перейдет в место P2, при T2 в P3 соответсвенно.

Для более точного представления работы сетей Петри приведем правила переходов:

1. Переход может сработать при наличии метки в месте, входящем в этот переход. Количество меток должно быть не меньше, чем количество входящих дуг в переход. Например, для срабатывания перехода T1 необходимо наличия в месте P1 не менее 2 фишек (см рис. 7, *а*).

2. Число фишек, которые появляются в месте после срабатывания перехода, равно количеству дуг, входящих в данное место. Если в месте P3 была одна фишка и сработал переход T2, то в месте P4 будет 2 фишке, а в P3 – ни одной (см. рис. 7, *б*).

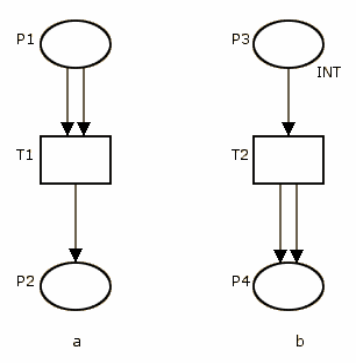



*Рис. 7.* Сети Петри для примеров кратности переходов

За свое существование к сетям Петри применялись всевозможные модификации и разновидности, среди которых основными являются:

– *цветные сети Петри*. Цвет метки соответствует определенному типу метки. Каждый тип метки может быть использован как аргумент в функциональных сетях. Цветные сети Петри широко используются для моделирования асинхронных процессов, в том числе и в разработке ПО;

– *временные сети Петри.* В данных сетях переходы обладают весом, определяющим продолжительность срабатывания (задержку). Примеры данных сетей даны в работах, многие из которых посвящены разработке программного обеспечения реального времени или встраиваемых систем.

Существуют и другие виды[3] сетей Петри, но они являются менее распространенными.

Применение сетей Петри дает возможность достаточно простого анализа, позволяющего выявить тупики, зацикливания, мертвые участки сети и т. д., которые указывают на возможные логические неточности системы. Кроме того, современные инструменты разработки позволяют моделировать поведение сложных систем, что упрощает работу с разрабатываемой системой.

Классические сети Петри редко используются, это связано с необходимостью дополнительных затрат на составление сети из исходных условий. Также существует проблема «взрыва» пространства состояний. Но несмотря на все возможные недостатки, сети Петри широко применяются совместно с другими инструментами моделирования, например с UML-диаграммами.

Одним из представителей визуального моделирования сетей Петри является программный пакет CPN Tools.

[3] *Функциональные сети Петри*: задержки определяются как функции некоторых аргументов, например, количества меток в каких-либо позициях, состояния некоторых переходов. *Ингибиторные сети Петри*: возможны ингибиторные дуги, запрещающие срабатывания перехода, если во входной позиции, связанной с переходом ингибиторной дугой, находится метка. Ингибиторные дуги также могут быть частью цветных сетей Петри, что способно расширить возможности описания сети Петри. *Стохастическая сеть Петри*: задержки являются случайными величинами.

## 8. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО РАБОТЕ С ПРОГРАММНЫМ ПАКЕТОМ

Программный пакет CPN Tools представляет собой экран, разделенный на две части: слева – меню, позволяющее выполнять работу с сетями, справа – рабочее поле, где располагается сеть и/или панель инструментов.

Работа с сетью начинается с выбора необходимых инструментов. Меню Tool box следующие наборы инструментов (см. рис. 8):

– Create. Набор инструментов, позволяющий создавать места, переходы и дуги;

– Hierarchy. Набор инструментов, позволяющий создавать иерархические сети;

– Net. Набор инструментов, позволяющий выполнять операции создания, сохранения, загрузки с сетью;

– Simulation. Набор инструментов, позволяющий выполнять моделирование сети вручную;

– State space. Набор инструментов, позволяющий выполнять анализ сети Петри.

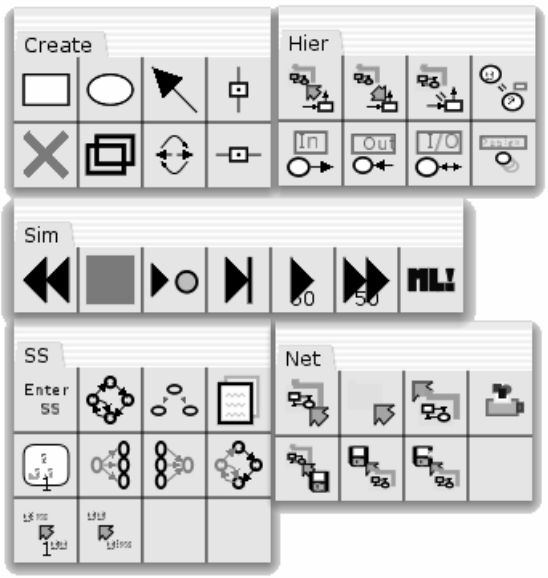



*Рис. 8.* Набор инструментов CPN Tools

Для создания сети необходимо в панели инструментов Net выбрать инструмент Create a new net (первый в первом ряду) и кликнуть на рабочем поле. Создание мест и переходов осуществляется с помощью Create a place (или Create a transition). Каждому месту сети необходимо задать ему тип и начальные метки, если они требуются.

После создания мест и переходов их необходимо соединить с помощью дуг Create an arc. После соединения необходимо назначить переменные для дуг, при помощи которых будут «переноситься» метки.
Пример построенной сети приведен на рис. 9.

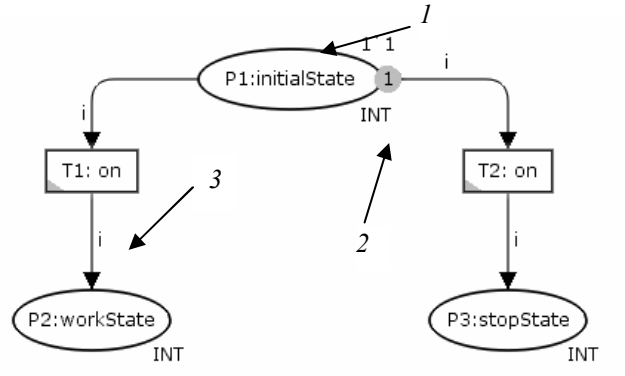



*Рис. 9.* Сеть Петри, в программном пакете CPN Tools

На рис. 9 цифрами обозначено следующее:

1) описание метки в месте (1`1), первая цифра показывает количество меток, вторая – значение меток. Например, запись 20`10 показывала бы 20 меток со значением 10. Также в месте могут находиться несколько меток – 1`1++1`2 – означает, что в месте находится 2 метки со значениями 1 и 2;

2) тип места;

3) переменная, которая «переносит» метку из одного места в другое. В данном примере из «initalState» в «stopState».

Анализ спроектированных сетей Петри осуществляется на основании их свойств. Основными, позволяющими судить о системе, являются:

– *безопасность*. Сеть безопасна, когда во всех достижимых маркированиях ее позиции не могут иметь больше одной метки;

– *ограниченность*. Сеть называется ограниченной, когда все позиции ограничены. Место является ограниченным, когда количество меток в нем не может превышать целое число. Соблюдение свойства ограниченности очень важно, так как в реальных условиях число ресурсов ограничено, а неограни-

ченность места приводит к бесконечному накапливанию меток, что является неприемлемым для модели реальной системы;

– *живость.* Возможность срабатывания любого перехода при функционировании моделируемого объекта.

Определение свойств сети Петри можно реализовать при помощи программного пакета CPN Tools. Для анализа свойств сети выбирается панель инструментов «SS», которая изображена на рис. 10.

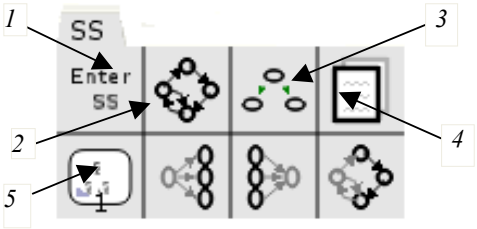



*Рис. 10.* Панель инструментов «SS»
из программного пакета CPN Tools

Цифрами на рис. 10 обозначены:

1 – инструмент, предназначенный для генерации пространства состояний для выбранной сети.

2 – инструмент, позволяющий вычислить пространство состояний для выбранной сети.

3 – инструмент, предназначенный для вычисления графа состояний выбранной сети.

4 – инструмент, позволяющий получить отчет на основе вычисленных пространства состояний и графа состояний.

5 – инструмент, позволяющий строить дерево достижимости моделируемой сети Петри.

Опираясь на перечисленные выше свойства можно определить некоторые виды программных ошибок, которые нельзя определить на этапе компилирования программы и которые не всегда можно проявляют себя при тестировании.

## 9. ИСПОЛЬЗОВАНИЕ ИНТЕГРАЦИИ UML-ДИАГРАММ И СЕТЕЙ ПЕТРИ В РАЗРАБОТКЕ ПО

Создание программных продуктов на основе совместно использования UML-диаграмм и сетей Петри можно разделить на несколько этапов: разработка набора UML-диаграмм, трансляция полученных диаграмм в сети Петри, анализ сетей Петри и внесение необходимых изменений в UML-диаграммы в соответствии с результатом анализа.

Ранее полученный набор диаграмм состояний преобразовывается в ординарную сеть Петри, после чего производится анализ сетей Петри с целью выявления некоторых логических ошибок: зацикливание, мертвые переходы, накопление неограниченного числа меток. Недостаток данной методики – ограничение в использовании диаграмм UML-диаграммами состояния и классов, что не позволяет использовать программисту дополнительных возможностей UML-диаграмм.

В ряде работ рассматривается применение UML-диаграмм с различными вариациями сетей Петри. Например, с классическими, цветными, временными. Применение различных типов сетей Петри позволяет анализировать всевозможные аспекты системы. В работах приводятся примеры, демонстрирующие анализ временной сети Петри в программных пакетах TINA и INTP-PerfBound получить временные оценки переходов меток от произвольного места к заданному.

Также необходимо упомянуть о работах, посвященных оценке быстродействия разрабатываемой системы. В этих работах сеть Петри используются для получения количественных оценок быстродействия системы или ее узлов.

## 10. ТРАНСЛЯЦИЯ ДИАГРАММ В СЕТЬ ПЕТРИ

При помощи математического аппарата сетей Петри возможна верификация архитектуры программной модели. Для этого исходная поведенческая диаграмма (как правило, это диаграмма состояний или диаграмма деятельности) согласно набору правил преобразуется в сеть Петри, которая позволяет осуществить имитационное моделирование. Если при работе сети возникли ошибки (например, тупиковые ветви) – значит при проектировании архитектуры тоже допущены ошибки, и результаты моделирования сети Петри позволяют их локализовать. В условиях все возрастающей сложности разрабатываемого программного обеспечения подобный подход призван значительно повысить его качество. Однако на сегодняшний день задача автоматической трансляции UML-диаграмм в сети Петри полностью не решена. Обеспечить трассовую эквивалентность исходный диаграммы и сети Петри на уровне автоматизации возможно, однако немалая часть информации о поведенческой модели системы не представима структурой графа деятельности, и при трансляции она теряется. Если между динамической и структурной диаграммами определены взаимосвязи, то при автоматической трансляции первой в сеть Петри эти взаимосвязи способны передать больший объем информации о поведении системы.

## 11. ПРАВИЛА ПРЕОБРАЗОВАНИЯ UML-ДИАГРАММ В СЕТЬ ПЕТРИ И ОБРАТНО

Трансляция UML-диаграмм в раскрашенную иерархическую сеть Петри и обратно должно осуществляться по формальным правилам, которые были предложены в [19]. Перечислим их.

**R1:** состояние ожидания трансформируется в место, а состояние действия преобразуется: в переход, который начинает действие; место, отражающее состояние выполнения действия; переход, завершающий выполнение действия. Возможно, что два состояния действия расположены последовательно, то окончание первого и начало второго действия совмещают и показывают одним переходом для упрощения диаграммы, как показано на рис. 11.

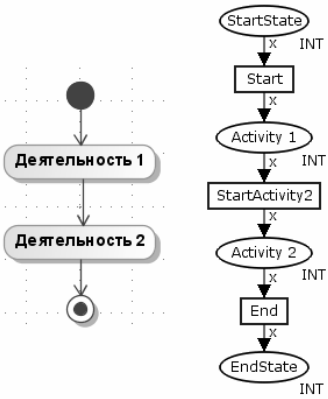



*Рис. 11.* Пример преобразования последовательности состояний

**R2:** разделение и слияния параллельных потоков управления UML-диаграмм преобразуется в соответствующий эквивалент сети Петри, представленный на рис. 12.

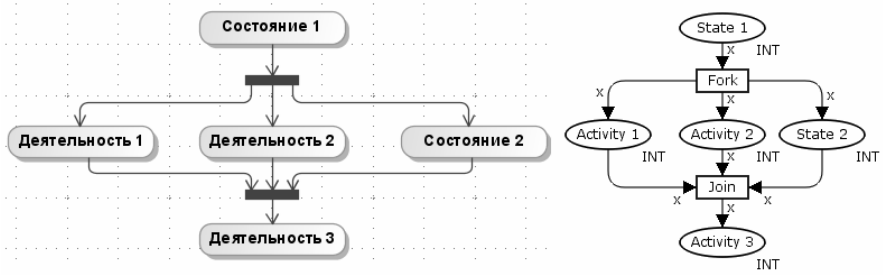



*Рис. 12.* Пример преобразования элементов разделения и слияния

**R3:** ветвление на диаграмме деятельности, обозначаемое символом решения, преобразуется в соответствующий эквивалент сети Петри представленный на рис. 13.

**R4:** условия ограничения показывают с помощью условий ограничения переходов, как показано на рис.13: «Условие 1» транслируется в «защитное выражение» (guard) «Condition1», а «Условие 2» в «Condition2».

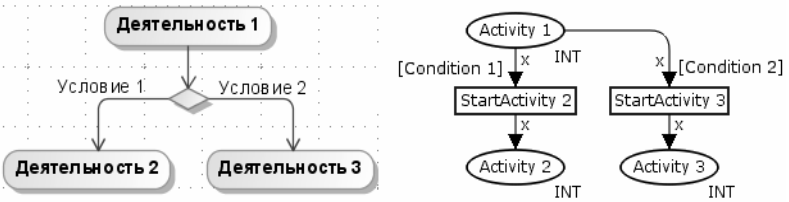



*Рис. 13.* Пример преобразования ветвления и условий переходов

**R5:** критическая секция (доступ к критическому ресурсу, реализуемый с помощью критической секции) преобразуется в место с одной меткой в начальной маркировке. Пример преобразования приведен на рис. 14. Критическая секция «Соединение с сервером приложений» трансформируется в место «AppServer» типа BOOL (булевы значения true и false) с начальной маркировкой 1`true. При выполнении критической секции метка изымается из места и возвращается по завершению критической секции.

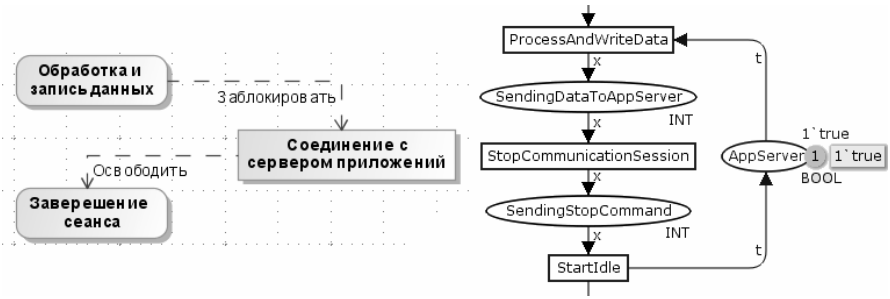



*Рис. 14.* Пример преобразования критической секции

**R6:** 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. Пример преобразования приведен на рис.15. Семафор «Канал связи» преобразуется в место Channels типа CHANNELS (colset CHANNELS = index c with 1..8;) – набора значений от 1 до 8 с начальной маркировкой CHANNELS.all() – все значения цветового множества. Метка с произвольным индексом должна изыматься из места и возвращаться при завершении ее использования. В простом семафоре должны быть использованы метки одного типа, и их количество в начальной маркировке равно числу ресурсов.

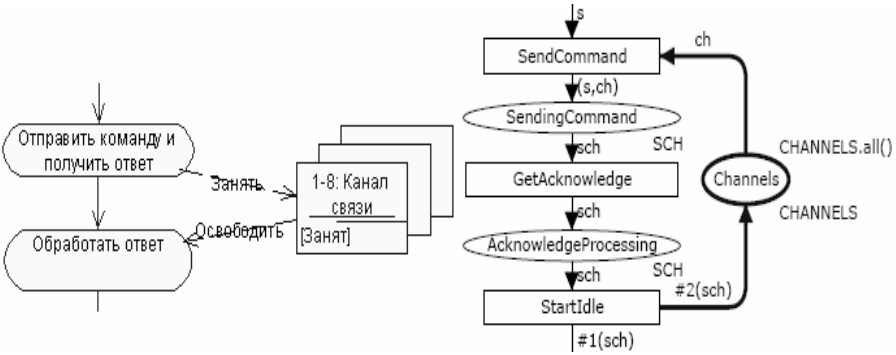



*Рис. 15.* Пример преобразования семафора [19, рис. 3.10]

**R7:** дорожка (swimlane) – область на диаграмме деятельности, содержащая элементы модели, которые выполняет отдельная подсистема, отражающаяся в виде отдельной подсети (страницы). Пример преобразования приведен на рис. 16. «Главный процесс», «Поток передачи» и «Поток слежения» выполнены отдельными страницами. «Поток передачи» заменяется страницей составного перехода DoSending, а «Поток слежения» заменяется страницей составного перехода DoMonitoring. Составные переходы, расположенные на главной страницы, упрощают восприятие модели и отображают функциональность.

Предложенные выше правила преобразования UML-диаграмм в сеть Петри были дополнены правилами для сетей критичных ко времени исполнения и иерархический сетей Петри.

**R8:** к критическим по времени исполнения секциям необходимо добавить место-счетчик. Для проверки максимального времени исполнения переходов критической секции нужно добавить защитные условия. Место-счетчик надо обеспечить переходом с противоположным защитным условием (рис. 17).

**R9:** при взаимном переходе из/в иерархическую сеть Петри необходимо добавить пустые места и соответствующие им переходы, которые соединены последовательно [19] (рис. 18).

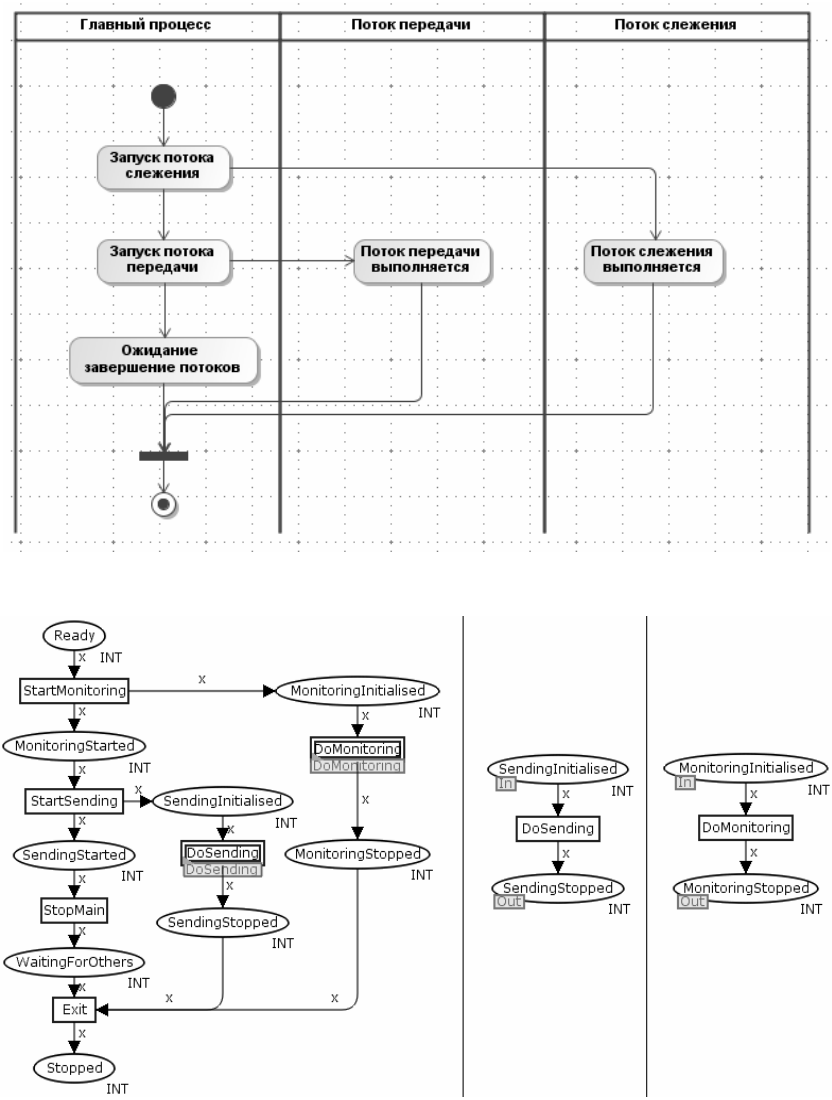



*Рис. 16.* Пример преобразования «дорожек»

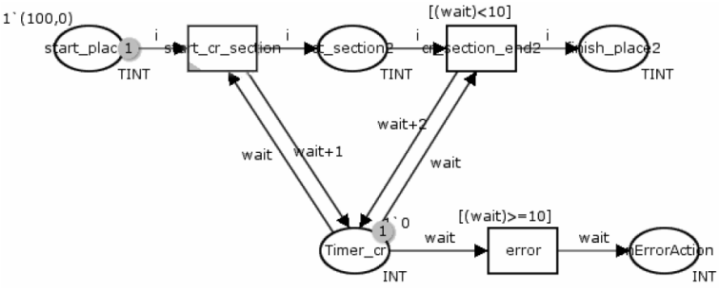



*Рис. 17.* Реализация защиты в критической секции в сетях Петри [19]

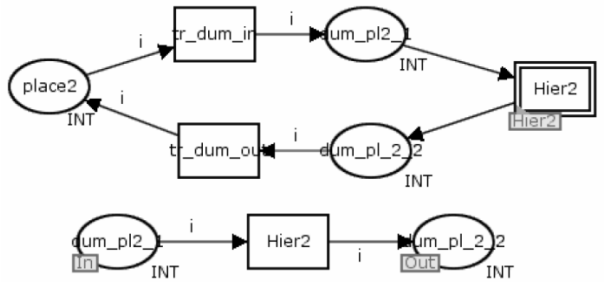



*Рис. 18.* Взаимный переход в иерархической сети Пери [19]

Правила преобразования, описываемые в данном разделе были опробованы в ряде работ [9, 26] и, несмотря на свою простоту, справляются с задачей трансляции UML-диаграмм в сеть Петри.

## 12. РЕКОМЕНДАЦИИ ПРЕОБРАЗОВАНИЯ UML-ДИАГРАММ В СЕТИ ПЕТРИ

Одним из недостатков сетей Петри, полученной при формальном преобразовании UML-диаграммы состояния, описывающей поведение классов, является отсутствие возможности моделирования поведения нескольких объектов одновременно [26], что приводит к неполноценному моделированию системы, а также к вероятности потенциального пропуска ошибок моделирования.

Для решения этой проблемы были предложены ***изменения*** в правилах формирования сети Петри на основе UML-диаграмм:

– в переходы сети нужно добавить защитные условия, определяющие какая метка (метка представляет собой объект) может совершить переход (т. е. для какого объекта будет вызван метод). Защитное условие работает на осно-

ве уникального идентификатора объекта, к которому может относиться как имя объекта, так и обычный целочисленный номер. Идентификатор хранится в метке. Для реализации этого необходимо расширить тип метки и ввести дополнительное значение для хранения идентификатора;

– для вызова методов необходимо использовать дополнительные места, соединенные с переходом, обозначающим срабатывание метода; возврат программы на место вызова метода и возвращение значения выполнения метода осуществляется при помощи дополнительного места, соединенного с переходом, в котором заканчивается выполнение метода;

– для моделирования статических методов нужно использовать схему, как и для моделирования обычных методов, исключением является то, что переходы и места такого метода не соединены с местами и объектами самого метода.

## 13. ПРАВИЛА ВЫЗОВА МЕТОДОВ И КОНСТРУКТОРОВ КЛАССА, ПРАВИЛА ОБРАЩЕНИЯ К ОБЪЕКТАМ

В работе [26] отдельно рассматривается структура сети Петри при описании вызова методов класса. На рис. 19 приведен пример вызова метода *On* класса *Valve*. Здесь вызов метода *On* происходит при помощи мест *On1* и *On1_Ret*. При моделировании вызова метода необходимо срабатывание перехода *On1*, при этом одна метка с идентификатором и параметрами переместится в место *On1*, вторая метка, сохраняющая значения до вызова метода, переходит в место *On1Buff*.

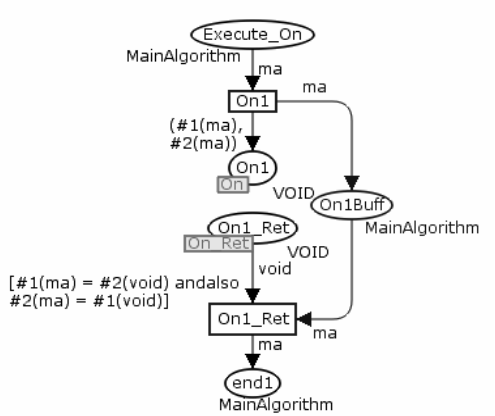



*Рис. 19.* Вызов метода *On* класса *Valve*

Выход из метода происходит во время срабатывания перехода *On1_Ret* и при помощи мест *On1_Ret* и *On1Buff*. При вызове метода метка попадает в место *On1Buff*. После завершения выполнения метода метка перемещается в место *On1_Ret*. Таким образом, переход *On1_Ret* становится доступен при выполнении защитного условия *[#1(ma) = #2(void) andalso #2(ma) = #1(void)]*, определяющее участок сети должна возвращения метки.

Моделирование ***вызова конструкторов*** осуществляется аналогичным образом. Однако существует отличие. Поскольку объект еще не существует, то отпадает необходимость передачи и проверки идентификатора объекта. При переходе метки во входное место конструктора объекту присваивается уникальный идентификатор.

## 14. ОПРЕДЕЛЕНИЕ ОШИБКИ СЕГМЕНТАЦИИ[4] ПРИ ОБРАЩЕНИИ К НЕСУЩЕСТВУЮЩЕМУ ЭЛЕМЕНТУ МАССИВА

В работе [26] предложен пример ошибки сегментации при обращении к несуществующему элементу массива. Формально запуск предложенной программы (пример 1) прошел успешно, однако программа содержит ошибку. Данную ошибку удалось определить при помощи построения и анализа аналогичной сети Петри (рис. 20).

Пример 1

```
#include "stdio.h"
int main()
{
int const nmax=3;
int a[3];
a[5] = 1;
printf("a(5)=%i\r\n", a[5]);
}
```

[4] Ошибка сегментации (*Segmentation fault, segfault*) – ошибка программного обеспечения, возникающая при попытке обращения к недоступным для записи участкам памяти либо при попытке изменения памяти запрещённым способом.

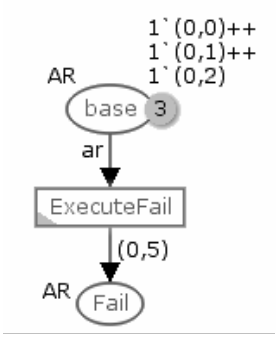



*Рис. 20.* Сеть Петри, соответствующая программе из примера 3.2

Стоит отметить, что данный способ не является приоритетным, так как ручное моделирование, по сути, является аналогом тестирования, а в последнее время упор делается на автоматизацию тестирования.

Также в работе [26] предлагается способ определения утечек памяти для программ, которые реализованы на языках программирования, не поддерживающих автоматическое освобождение памяти или его механизмы несовершенны. К таким языкам можно отнести C/C++[5], PHP[6].

## 15. МЕТОДИКА ПОСТРОЕНИЯ И ПРОВЕРКИ МОДЕЛИ ПРОЕКТА

Предлагается [9] методика построения и проверки модели проекта центра дистанционного управления и контроля технологии преобразования UML-диаграмм в сеть Петри, включающая следующие шаги:

**Шаг 1.** Создается *диаграмма классов* с учетом следующего: процесс и потоки, отображенные на *диаграмме процессов*, реализуются как активные классы; задача запуска потока работы с управляемыми объектами при создании экземпляра класса и его останова при завершении работы программного модуля (удалении класса) возлагается на активный класс;

**Шаг 2.** Для каждого класса системы создается *диаграмма состояний*, отражающая последовательности состояний, в которые попадает класс при его создании, удалении и вызове его методов. Вызов каждого метода отображается в виде сложного состояния со своей внутренней последовательностью состояний. Дополнительно указывается использование элементов синхронизации потоков и доступа к ресурсам, а также флагов, влияющих на ход выполнения методов класса;

[5] В C/C++ при динамическом выделении памяти при помощи функций C *malloc/alloc* и им подобных и конструкций C++ *new* необходимо ручное ее освобождение при помощи функций *free* в C и конструкции *delete* в C++.

[6] Несмотря на то, что в официальной документации присутствует информация о том, что PHP поддерживает автоматическое освобождение памяти, на практике такое освобождение не всегда работает. Например, если в коде присутствуют циклические ссылки, то память не всегда будет освобождаться корректно, и для ее освобождения требуется вызов конструкции *unset*.

**Шаг 3.** Взаимодействие классов отображается на *диаграмме последовательности* с указанием вызовов конструкторов и деструкторов (создание и удаление классов), вызов методов классов;

**Шаг 4.** Каждая диаграмма состояний с помощью набора правил преобразуются в страницу раскрашенной иерархической сети Петри;

**Шаг 5.** Все полученные страницы раскрашенной иерархической сети Петри связываются в единую сеть с помощью страницы связки реализуемой на основании диаграммы последовательности;

**Шаг 6.** Выполняется прогон (simulation) модели по различным последовательностям выполнения. На основании проведенного исследования делается вывод о правильности функционирования модели, а в случае обнаружения ошибок и неточностей производится корректировка сети Петри и UML-диаграмм;

**Шаг 7.** Выполняется генерация пространства состояний модели и отчета с различными исходными данными. На основании анализа полученных отчетов делается вывод о правильности модели, а в случае обнаружения ошибок и неточностей производится корректировка сети Петри UML-диаграмм.

**Шаг 8.** Набор UML-диаграмм проекта готов к автоматической генерации проекта и реализации кода.

Последовательность разработки СИСТЕМЫ при использовании предлагаемой методики на этапе проектирования представлена на рис. 21.

Этапы процесса разработки с применением предлагаемой методики соответствуют этапу проектирования технологического цикла разработки, принятого в качестве стандарта в инженерии программного обеспечения, и усовершенствуются применением сетей Петри для проверки правильности и согласованности UML-диаграмм на этапах анализа и проектирования. Процесс проверки и корректировки UML-диаграмм на обоих этапах является итерационным и продолжается до получения свойств сети Петри, обеспечивающих корректность и безотказность работы СИСТЕМЫ и соответствующих требованиям к проектируемой СИСТЕМЕ.

Описанная в данном разделе методика позволяет разработчику системы получить набор UML-диаграмм проекта, правильность и согласованность которых доказана с помощью формального метода. Представленная методика является одной из первых, которая сочетает в себе использование UML-диаграмм и их анализ с помощью сетей Петри. В данной работе были успешно использованы правила преобразования, описанные выше, UML-диаграмм в сеть Петри.


*Рис. 21.* Последовательность разработки СИСТЕМЫ (проверка проекта) [9, рис. 4.1]

В работе [26] предложен еще один *алгоритм разработки ПО*, позволяющий выявить их общие достоинства и недостатки[7].

1. Опираясь на диаграммы, полученные на этапе анализа, разрабатывают диаграммы классов с учетом следующего: *процесс и потоки, отображенные на диаграмме процессов, реализуются как активные классы*; задача запуска потока работы с управляемыми объектами при создании экземпляра класса и его останова при завершении работы программного модуля (удалении класса) возлагается на активный класс.

2. Данный шаг следует рассматривать по-разному в разных способах разработки:

а) для всех классов системы разрабатывается диаграмма состояний, которая отражает последовательности состояний. В эти состояния класс попадает класс при его создании, удалении и вызове его методов. *Вызов каждого метода отображается в виде сложного состояния со своей внутренней последовательностью состояний*. Следует дополнительно указать использование

[7] В вышеизложенном алгоритме разработки ПО курсивом отмечены его недостатки и неточности.

элементов синхронизации потоков и доступа к ресурсам и флагов, влияющих на ход выполнения методов класса;

b) *поведение каждого класса описано при помощи диаграммы состояний.* Взаимодействие диаграмм состояний друг с другом происходит при помощи обмена сообщениями. Состояния состоят из идентификатора и действий, которые совершаются в данный момент. Переходы включают в себя названия, которое обозначает событие срабатывания перехода, и совершаемое действие при срабатывании перехода.

3. В некоторых работах данный шаг трактуется по-разному:

a) на диаграмме последовательности отображается взаимодействие между классами с указанием вызовов конструкторов и деструкторов (создание и удаление классов), методов классов;

b) диаграммы взаимодействия определяют соединения среди подсетей в соответствии с различными классами.

4. Диаграммы состояний при помощи набора правил преобразуется в страницы раскрашенной иерархической сети Петри.

5. Все полученные страницы раскрашенной иерархической сети Петри связываются в единую сеть с помощью страницы связки, реализуемой на основании диаграммы последовательности.

6. *Выполняется моделирование (simulation) по различным последовательностям выполнения*. После анализа смоделированной системы делается вывод о корректности функционирования модели, при обнаружении ошибок производится корректировка сети Петри и UML-диаграмм.

7. Выполняется генерация пространства состояний модели и отчета с различными исходными данными. *На основании анализа полученных отчетов делается вывод о правильности модели*, а в случае обнаружения ошибок производится корректировка сети Петри и UML-диаграмм.

8. Набор UML-диаграмм проекта готов к автоматической генерации проекта и реализации кода.

Предложенный выше алгоритм [26] также имеет недостатки. Ниже приведены некоторые из них:

– использование диаграмм состояний не является обязательным – они могут быть заменены на диаграммы, описывающие поведение системы, например, диаграммы действий;

– использование только диаграммы последовательности не способно отразить взаимодействие между различными классами в достаточной мере;

– отсутствие конкретных шаблонов или примеров выявления ошибок UML-диаграммах на основании результатов моделирования системы в сетях Петри.

Предложенный выше алгоритм разработки ПО был использован для создания методики проектирования моделей, графическое представление которой дано на рис. 22. Моделирование системы разбивается на несколько этапов.

1. На первом этапе происходит сбор требований к разрабатываемой системе – построение диаграмм вариантов использования. Выделяются используемые в системе актеры, прецеденты и варианты использования системы.

2. Далее составляется диаграмма классов, выделяются сущности системы, для которых составляются классы или иерархия классов в системе. Выделяются методы классов.

3. Следующим шагом является проектирование диаграмм, описывающих динамические свойств системы. Для выделенных классов составляются диаграммы состояний или действий. Диаграммы действий необходимо составить для каждого метода из диаграммы классов.

4. На четвертом этапе необходимо составить диаграммы объектов. На основании диаграммы классов выделяется объекты. С помощью диаграммы объектов определяют начальные значения объектов системы, следовательно, значение меток для сети Петри.

5. После чего проектируются диаграммы основного алгоритма системы, отображающие динамические свойства системы.

6. На шестом шаге UML-диаграммы преобразуются в сеть Петри с помощью правил формального преобразования.

7. Завершающий этап – анализ полученной сети Петри. Выполняется при помощи автоматизированного программного пакета для работы с сетями Петри, например, CPN Tools. В случае отсутствия ошибок составленный набор UML-диаграмм можно считать верным и приступать к генерации исполняемого кода.

Таким образом, можно сделать вывод, что предложенная имеет возможность выявлять неточности и логические ошибки в составленных UML-диаграммах на этапе разработки системы.

Для создания представленной методики [26] были изучены и проанализированы более ранние работы, связанные с совместным использованием UML-диаграмм и сетей Петри. Предлагаемый способ включает в себя использование основных поведенческих (диаграмма деятельности) и структурных диаграмм (диаграмма классов, объектов, состояний), что позволяет использовать его в различных областях разработки программного обеспечения.

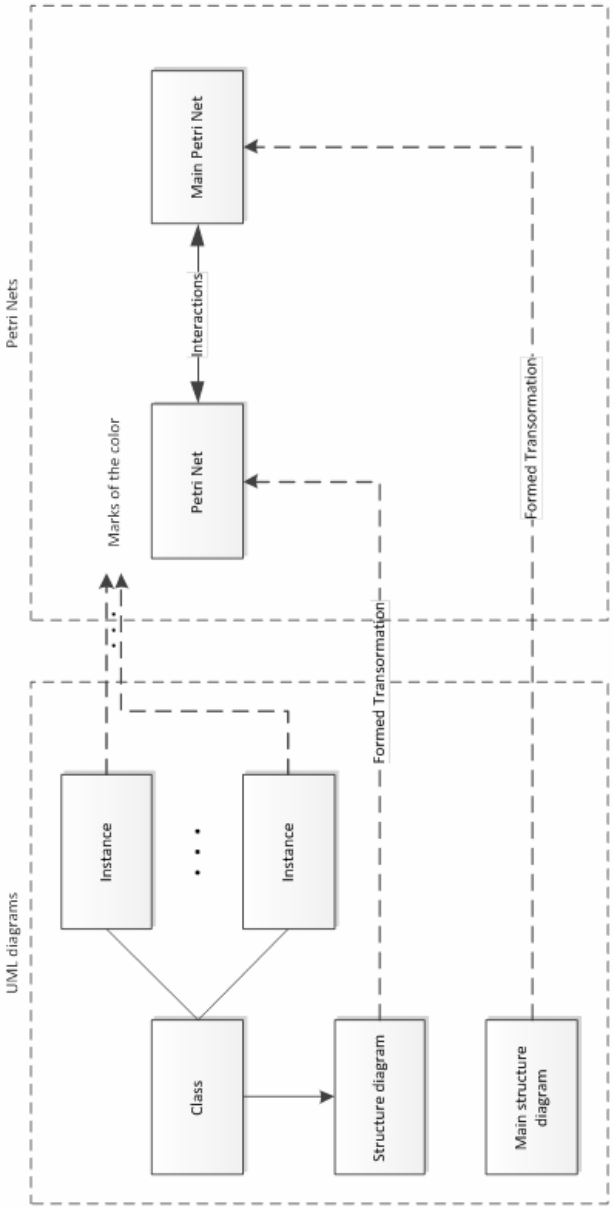



*Рис. 22.* Графическое представление предлагаемой методики разработки ПО

## ЗАКЛЮЧЕНИЕ

В данной статье подробно описаны подходы разработки программного обеспечения такие как: agile (гибкая методология разработки), итеративная разработка, RUP, каскадная модель. Произведен анализ декларативного программирования, императивного программирования, структурного программирования, функционального программирования, объектно-ориентированного программирования, а также подходов проектирования программного обеспечения с использованием UML и аппарата сетей Петри.

Непосредственно описан язык UML с присущими ему положительными моментами и недостатками, описана взаимосвязь между поведенческими и структурными диаграммами UML, рассказаны преимущества сетей Петри – свойство параллелизма сетей.

Представлено описание пакета CPN Tools и даны указания по его использованию для моделирования сетей Петри. Детально описаны инструменты необходимые для проектирования сетей, приведены основные свойства, по которым происходит анализ сетей.

Параллелизм сетей наглядно продемонстрирован на примере системы нахождения выхода из лабиринта меткой и манипулятором. В статье приводится подробное описание этапа разработки программного обеспечения, который включает в себя: описание структуры системы, т. е. множества составляющих ее элементов, установление взаимосвязей между элементами, описание процесса взаимодействия взаимосвязанных элементов и объединение знаний о структуре, связях и динамике элементов в одно целое – систему.

Описана методология проектирования при совместном использовании UML и сетей Петри, состоящая из восьми этапов:

1) создание диаграммы классов;

2) создание диаграммы состояний для каждого класса;

3) построение диаграммы последовательности, отображающей взаимодействие классов;

4) каждая диаграмма состояний транслируется в сеть Петри;

5) все полученные страницы связываются в одну сеть;

6) тестирование сети;

7) генерация пространства состояний модели и отчета с различными исходными данными;

8) набор UML-диаграмм проекта готов к автоматической генерации проекта и реализации кода.

А также методология более общего применения, состоящая из 7 этапов:

1) проектирования диаграммы вариантов использования;

2) разработка диаграммы классов;

3) составления диаграммы состояний и диаграммы действий;

4) создания диаграммы объектов;

5) проектирования диаграммы основного алгоритма системы;

6) преобразование UML-диаграмм в сеть Петри;

7) анализ полученной сети Петри. При отсутствии ошибок можно считать составленный набор UML-диаграмм верным и приступить к генерации исполняемого кода.

Приведены правила для трансляции и преобразования UML в сеть Петри.

Также авторами описываемых работ замечена необходимость, а самое главное – возможность автоматической трансляции UML-диаграмм в сеть Петри

[1] *Бек К.* Экстремальное программирование / К. Бек. – СПб.: Питер, 2002. – 224 с.

[2] *Брауде Э.* Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004. – 655 с.

[3] *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++: пер. с англ. / Г. Буч. – 2-е изд. – М.: Бином; СПб.: Невский диалект, 1998 г. – 560 с.

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

[5] *Дал У.* Структурное программирование / У. Дал, Э. Дейкстра, К. Хоор. – М.: Мир, 1975. – 247 с.

[6] *Ларман К.* Применение UML 2.0 и шаблонов проектирования. – 3-е изд.: пер. с англ. / К. Ларман; – М.: Вильямс, 2007. – 736 с.

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

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

[9] *Коротиков С.В.* Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления: дис. … канд. техн. наук / С.В. Коротиков. – Новосибирск: Изд-во НГТУ, 2007.

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

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

[12] *Воевода А.А.* О применении сетей Петри в процессе объектно-ориентированного анализа и проектирования / А.А. Воевода, Д.В. Прытков // Сб. науч. тр. НГТУ. – 2009. – № 4(58). – С. 131–138.

[13] *Воевода А.А.* Применение UML-диаграмм и сетей Петри при разработке встраиваемого программного обеспечения / А.А. Воевода, Д.О. Романников, И.В. Зимаев // Научный вестник НГТУ. – 2009. – № 4(37). – С. 169–175.

[14] *Романников Д.О.* Применение UML-диаграмм и сетей Петри в разработке программного обеспечения для систем критичных ко времени исполнения: магис. Дис / Д.О. Романников. – Новосибирск: НГТУ, 2009.

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

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

[17] *Воевода А.А.* Тестировании UML-диаграмм с помщью аппарата сетей Петри на примере разработки ПО для игры «Змейка» / А.А. Воевода, А.В. Марков // Сб. науч. тр. НГТУ. – 2010. – № 3(61). – С. 51–61.

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

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

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

[21] *Марков А.В.* Обзор работ посвященным разработке ПО с использованием UML и сетей Петри / А.В. Марков, Д.О. Романников // Сб. науч. тр. НГТУ. – 2011. – № 1(63). – С. 85–95.

[22] *Романников Д.О.* Применение методики разработки ПО на примере разработки игры «Змейка» / Д.О. Романников / Сб. науч. тр. НГТУ. – 2011. – № 3(65).

[23] *Романников Д.О.* Пример применения методики разработки ПО с использованием UML-диаграмм и сетей Петри / Д.О. Романников, А.В. Марков // Научный вестник НГТУ. – 2012. – № 1(67). – С. 175–181.

[24] *Романников Д.О.* Нахождение ошибок обращения к несуществующим элементам массива на основании результатов анализа сети Петри / Д.О. Романников // Сб. науч. тр. НГТУ. – 2012. – № 1(67). – С. 115–120.

[25] *Шоба Е.В.* Методология проектирования современного программного обеспечения приминительно к станции управления лифтомахождение ошибок обращения к несуществующим элементам массива на основании результатов анализа сети Петри / Е.В. Шоба, А.В. Марков // Сб. науч. тр. НГТУ. – 2012. – № 1(67). – С. 121–132.

[26] *Романников Д.О.* Разработка программного обеспечения с применением UML-диаграмм и сетей Петри для систем управления локальным оборудованием: дис. … канд. техн. наук / Д.О. Романников. – Новосибирск: Изд-во НГТУ, 2012.

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

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

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

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

[31] *Питерсон Дж.* Теория сетей Петри и моделирование: пер. с англ / Дж. Питерсон. – М.: Мир, 1984.

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

[33] *Зайцев Д.А.* Моделирование телекоммуникационных систем в CPN Tools / Д.А. Зайцев, Т.Р. Шмелева. – Одесса: Онат, 2006. – 60 с.

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

[35] *Воевода А.А.* Применение сетей Петри при разработке протоколов / А.А. Воевода, С.В. Коротиков, Д.О. Саркенов // Научный вестник НГТУ. – 2004. – № 2(17). – С. 183–189.

[36] *Воевода А.А.* О моделировании системы дистанционного контроля и управления информацией цветной временной сетью Петри в пакете CPN Tools: инициализация модели / А.А. Воевода, С.В. Коротиков, В. Хассоунех // Научный вестник НГТУ. – 2004. – № 3(18). – С. 184–187.

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

[38] *Саркенов Д.О.* Сети Петри: графы состояний / Д.О. Саркенов // Сб. науч. тр. НГТУ. – 2005. – № 1(39). – С. 159–160.

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

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

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

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

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

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

[45] *Воевода А.А.* О компактном представлении языков сетей Петри: сети с условиями и временные сети / А.А. Воевода, А.В. Марков // Сб. науч. тр. НГТУ. – 2010. – № 2(60). – С. 77–83.

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

[47] *Прытков Д.В.* О применении сетей Петри для использования алгоритмов на примере решения задачи о кратчайших путях с единственным источником / Д.В. Прытков // Сб. науч. тр. НГТУ. – 2010. – № 3(61). – С. 91–98.

[48] *Марков А.В.* Моделирование процесса поиска пути в лабиринте при помощи сетей Петри / А.В. Марков // Сб. науч. тр. НГТУ. – 2010. – № 4(62). – С. 133–141.

[49] *Воевода А.А.* WorkFlow-моделей с применением сетей Петри / А.А. Воевода, И.В. Зимаев // Научный вестник НГТУ. – 2010. – № 4(41). – С. 151–155.

[50] *Марков А.В.* Моделирование процесса поиска пути в лабиринте при помощи сетей петри для системы из двух связных звеньев / А.В. Марков, А.А. Воевода // Сб. науч. тр. НГТУ. – 2011. – № 3(65). – С. 75–85.

[51] *Марков А.В.* Описание разрабатываемой системы «поиск манипулятором кратчайшего пути в лабиринте» / А.В. Марков, А.А. Воевода // Сб. науч. тр. НГТУ. – 2011. – № 3(65).

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

[53] *Марков А.В.* Поиск манипулятором кратчайшего пути в лабиринте / А.В. Марков // Сб. науч. тр. НГТУ. – 2011. – № 3(65).

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

***Марков Александр Владимирович*** – аспирант кафедры автоматики Новосибирского государственного технического университета. Основное направление – исследование свойств сетей Петри и UML-диаграмм. Имеет более 10 публикаций.

E-mail: muviton3@mail.ru.

**A.V. Markov**

**Studies devoted to the development of software when used UML diagrams and Petri nets**

This article analyzed the approach of software development, analysis paradigms of writing software. Detailed design phase. The review of papers on software development with Petri nets and UML diagrams. The description of the package and the CPN Tools provides guidance on its use for simulation of Petri nets. This article highlights the main aspects, strengths and weaknesses of Petri nets. Such as concurrency networks, to the movement of the manipulator and the mark on the labyrinth, and UML diagrams - visual simulation system. Described the relationship between the behavioral and structural UML diagrams. Identified strengths and weaknesses of diagrams UML. The description of sharings mathematical apparatus of Petri nets and the UML, UML transformation rules to the network and vice versa. Described previously proposed technique for creating software using Petri nets and UML diagrams. Told about the possibility and necessity of the automatic tranformation UML diagrams to Petri nets.

**Key words:** software engineering, UML-diagrams, Petri nets, CPN Tools, automatic translation, merge of diagrams, recursion, transformation rules, methods of software development.