СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2012. – № 2(68) –**75–98

УДК 62-50:519.216

СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ UML-ДИАГРАММ И СЕТЕЙ ПЕТРИ НА ЭТАПЕ ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ: ОБЗОР*

А.А. ВОЕВОДА, С.В. КОРОТИКОВ, А.В. МАРКОВ

Приводится обзор работ, посвященных созданию программного обеспечения с помощью сетей Петри и UML-диаграмм. Подробно описывается этап проектирования. Проанализированы достоинства и недостатки сетей Петри, например, такие как параллелизм сетей, и UML-диаграмм – визуальное моделирование системы. Приведено описание совместного использования математического аппарата сетей Петри и языка UML, правила преобразования UML-диаграмм в сеть и наоборот. Описана предложенная ранее методика создания программного обеспечения с использованием сетей Петри и UML-диаграмм. Обсуждается возможность и необходимость автоматической трансляции UML-диаграмм в сети Петри.

Ключевые слова: инженерия программного обеспечения, UML-диаграммы, сети Петри, CPN Tools, параллельные процессы, автоматическая трансляция, слияние диаграмм.

ВВЕДЕНИЕ

В настоящее время трудно представить успешную, высокопроизводительную работу предприятий без качественных программных продуктов. Постоянно растущие требования заказчиков приводят к повышению функциональности и мощности разрабатываемых систем, что в свою очередь влечет 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 является использование различных средств визуализации.

Наибольшим успехом пользуются методы анализа, которые могут гарантировать высокую производительность и безотказность работы разрабатываемых программных продуктов. Таковыми являются язык описаний и спецификаций SDL, язык MSC (диаграммы последовательности и сообщений), ИВТ-сети, Workflow модели, алгоритмический язык ДРАКОН (Дружелюбный Русский Алгоритмический Язык, Который Обеспечивает Наглядность), графический язык UML, математический аппарат сетей Петри. Сюда же можно отнести также различные алгоритмы: машина Тьюринга, нормальный алгоритм Маркова, алгоритма Петерсона, алгоритм Лампорта.

Следует отдельно выделить методологии, базирующиеся на использовании диаграмм UML и сетей Петри (L. Baresi, M. Pezze [5, 8–11, 18, 19]). Сети Петри используются для обеспечения автоматизированного процесса тестирования проектируемого программного обеспечения, а UML-диаграммы для увеличения скорости, уменьшения стоимости разработки и «порога» необходимых разработчику знаний.

1. СЕТЬ ПЕТРИ

Сеть Петри – двудольный ориентированный граф, который состоит из вершин двух типов: позиций и переходов, соединенных между собой дугами, вершины одного типа не могут быть соединены непосредственно. В позициях могут размещаться метки (маркеры), способные перемещаться по сети.

Сети Петри приобрели широкое применение в разработке во многих сферах деятельности: от проектирования сетевых протоколов до разработки логики работы домашних кинотеатров. Большой ряд решаемых задач стал возможен благодаря интенсивному развитию сетей Петри, имеющих большое количество модификаций и разновидностей, основные из которых представлены на рис. 1.

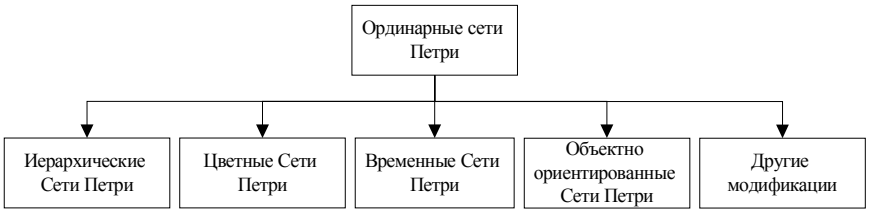


Рис. 1. Разновидности сети Петри

Параллелизм сетей Петри. Несмотря на разнообразие моделируемых систем, выделяется несколько общих

черт, которые должны быть отражены в особенностях используемой модели этих систем. Основная идея заключается в том, что системы состоят из отдельных взаимодействующих компонент. Каждая компонента сама может быть системой, но ее поведение можно описать независимо от других компонент системы, за исключением точно определенных взаимодействий с другими компонентами.

Действиям компонент системы присущи совмещенность или параллелизм. Действия одной компоненты системы могут производиться одновременно с действиями других компонент.

Совмещенная природа действий в системе создает некоторые трудности при моделировании. Поскольку компоненты системы взаимодействуют, необходимо установление синхронизации. Пересылка информации или материалов от одной компоненты к другой требует, чтобы действия включенных в обмен компонент были во время взаимодействия синхронизированы. Это может привести к тому, что одна компонента будет ждать другую компоненту. Согласование во времени действий различных компонент может быть очень сложным, а получающиеся в результате взаимодействия между компонентами трудны в описании.

Ряд современных вычислительных задач, например, таких как расчет высокодетализированной компьютерной графики, требуют весьма значительных аппаратных ресурсов.

Сети Петри разрабатывались специально для моделирования тех систем, которые содержат взаимодействующие параллельные компоненты, таким образом, алгоритмы параллельного программирования и гиперпоточности можно протестировать с помощью сетей Петри [5].

2. ПАРАЛЛЕЛИЗМ НА ПРИМЕРЕ РАСКРАШЕННОЙ СЕТИ ПЕТРИ: МЕТКА В ЛАБИРИНТЕ [3, 7]

Проиллюстрируем параллелизм в сетях Петри на примере лабиринта, по которому будет передвигаться метка. Лабиринт содержит проходы, стены, клетку входа и клетку выхода. Метка будет начинать движение с клетки *«Начало»* и заканчивать в клетке *«Конец»*. За один шаг метка передвигается на одну клетку.

Лабиринт (рис. 2) задан двухмерным массивом x[i][j]. Каждый элемент массива будет равен 1 или 0 (0 – проход, 1 – стена).

Реализация алгоритма поиска выхода из лабиринта с помощью сетей Петри. Создается класс *«Робот»*, который обращается к массиву *«Лабиринт»*. Из него получаем информацию о своих координатах и возможных передвижениях. История передвижений будет сохраняться в метке.

При движении в самой метке сохраняется вся история передвижений, по которой можно отследить предыдущие перемещения. Также в метке должны указываться координаты местоположения. Авторам неизвестны работы, в которых бы предлагалось хранить информацию о системе в фишках. На *«Перекрестке»* метка делится на возможное количество фишек. Получившиеся метки будут двигаться в своем направлении асинхронно. Движение метки на предыдущее местоположение запрещено (в истории это последнее передвижение). При попадании метки в тупик (нет возможных движений) свободные метки, если они существуют, продолжают движение, если отсутствуют, то выход из лабиринта не найден или не существует.

При попадании на клетку, уже пройденную другой меткой, первая метка будет считать это тупиком (после обращения к элементу массива равному 0, т. е. проход, данный элемент станет равным 2 – пройденное поле).

Смысл реализация этого алгоритма в том, что фишка, которая была изъята из места *«Robot»,* сразу размножается на 4 метки, и в каждой из составных частей меняется координата (рис. 3). Переход *«Comprasion n»* сработает в том случае, если координата в структуре массива свободна (присутствует фишка и она имеет символьный элемент равный «0»).

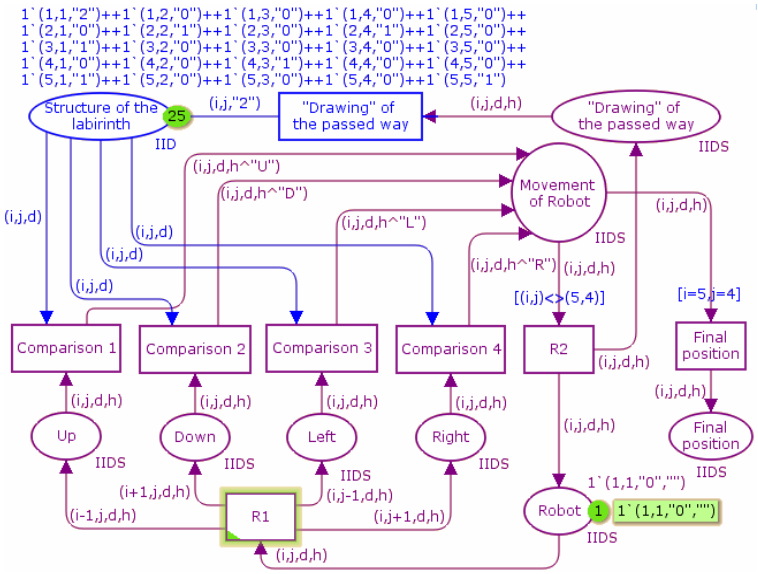


Рис. 3. Фрагмент сети Петри: логика передвижения по лабиринту

Лабиринт создан как двухмерный массив в месте *«Structure of the labyrinth»* с помощью составного множества цветов. Фишке *«Robot»* задаются начальные координаты, другими словами клетка входа. После срабатывания перехода *«Comprasion n»* фишка возвращается в место *«Robot»* и в место *«Structure of the labyrinth»* с измененным символьным типом на «2» для того, чтобы другие метки не пересекали пути, пройденные другими метками. Переход *«Final position»* сработает только в том случае, если к нему обратиться фишка с координатами поля выхода.

3. ПАРАЛЛЕЛИЗМ НА ПРИМЕРЕ РАСКРАШЕННОЙ СЕТИ ПЕТРИ: МАНИПУЛЯТОР В ЛАБИРИНТЕ [12, 13, 17]

Вышеприведенная система претерпела ряд изменений, а именно: робот теперь представляется как конструкция, состоящая из двух звеньев [12].

Структура и динамическая часть системы. Лабиринт также задается массивом данных. Помимо проходов и стенок лабиринт содержит клетку выхода (рис. 4).

Во избежание блокировки программы, во время перебора возможных перемещений робота, вокруг лабиринта добавлены клетки с типом данных *«Wall»*. Звено может находиться в 16 положениях и из каждого положения может перемещаться по следующим правилам:

1) каждый элемент звена за один такт имеет возможность переместиться на одну клетку;
2) длина конструкции может изменяться в пределах 0.20 части клетки;
3) передвижение *«головы»* и *«ног»* влево/вправо одновременно запрещено.

Изменения положения звена разрешено только вверх, вправо, влево, вниз. Движение наискосок, к примеру, влево/вверх или вправо/вниз, запрещено.

Спроектированная диаграмма активности содержит деятельность проверки близлежащих координат возможных положений *(Inspection),* определение отобранных перемещений *(IdentificationMovings),* проверка наложения *«головы»* и *«ног»* на стену (*Wall*), а также проверка существования роботов с одинаковыми координатами (*Contrast*). Манипулятор, который достиг данного положения за наименьшее количество тактов, остается в лабиринте, а тот, у которого история перемещения больше, удаляется. Детализированная диаграмма деятельности представлена на рис. 5.

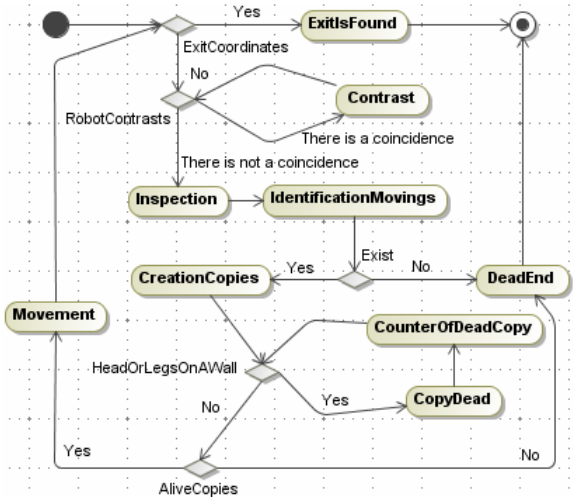


Рис. 5. Проработанная диаграмма деятельности

Проектирование раскрашенной сети Петри. Для проверки корректности построения и более глубокого изучения логики работы системы полученная диаграмма деятельности транслировалась в цветную иерархическую сеть Петри (рис. 6, 7) по правилам, предложенным в [9, 10].

Как видно, место *Robot* содержит одну метку с множеством типов данных. Первые два типа данных показывают координаты *«головы»,* третий и четвертый местоположение *«ног»* манипулятора, пятый необходим для движения по лабиринту, а в последнем хранится вся история движения робота.

Работа сети начинается с проверки типов данных близлежащих координат возможных положений манипулятора. Опытным путем стало известно, что для манипулятора длиной в четыре клетки нужно проверять тип данных шести координат. Данные координаты отображаются в местах *Left1, Left2, Left3, Right1, Right2, Right3*. После проверки отбираются положения, на которые может переместиться робот.

На следующем этапе проверяется, попали ли *«голова»* и *«ноги»* робота на стенку. При нахождении манипулятора на свободных координатах происходит передвижение по сети. Если происходит наложение на стенки лабиринта, то данная копия манипулятора удаляется.

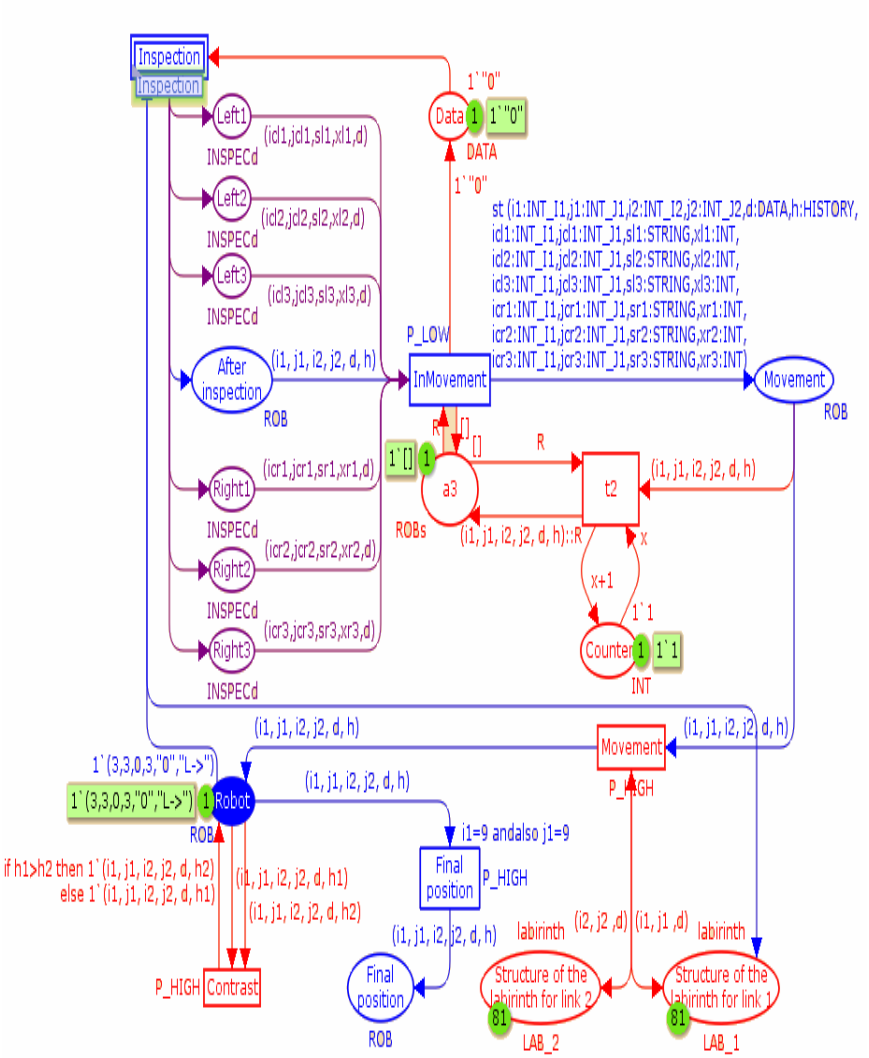
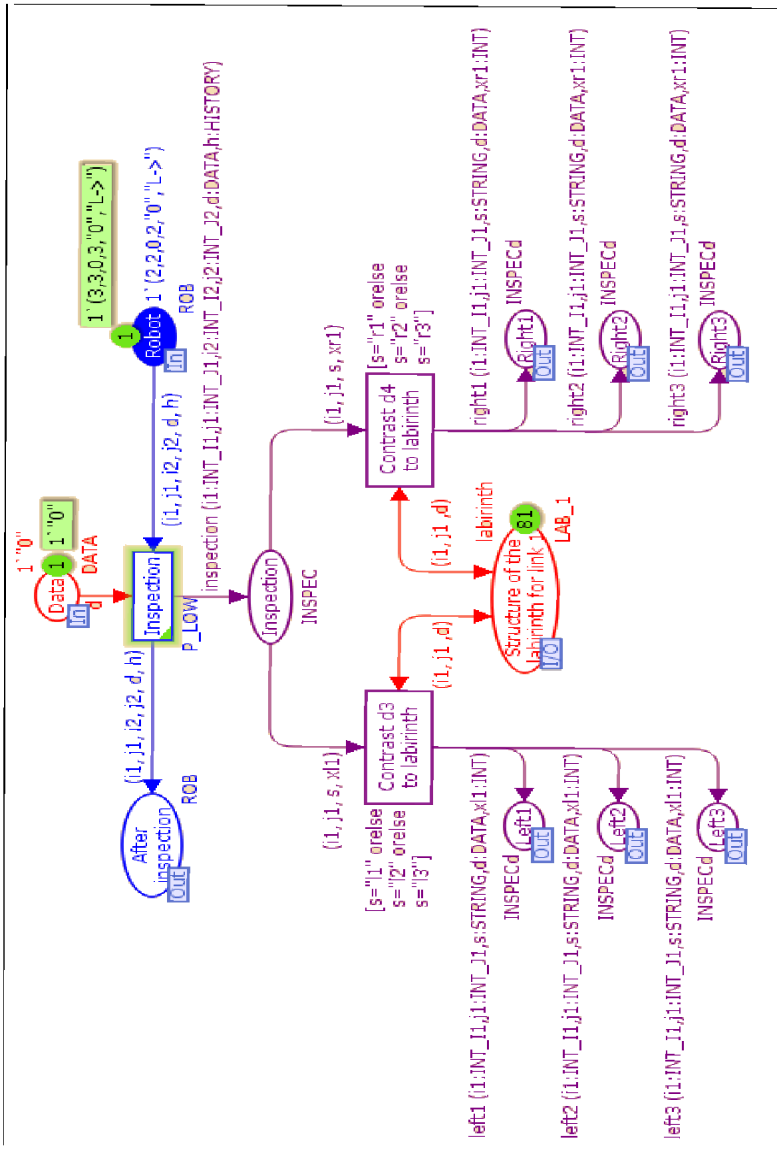


Рис. 6. Главная страница раскрашенной сети Петри



4. UML – UNIFIED MODELLING LANGUAGE

UML – язык графического описания для объектного моделирования в области разработки программного обеспечения, фактически является стандартом визуального моделирования. UML-диаграммы условно разделяют на три категории:

1. Структурные диаграммы описывают статическую структуру системы.
2. Диаграммы поведения описывают различные состояния элементов системы и их изменения в зависимости от различных условий.
3. Диаграммы взаимодействия – показывают взаимодействие элементов системы во времени.

Наглядным примером визуального представления алгоритмов и процессов является стандарт построения блок-схем (ГОСТ 19.701-90), а также соответствующая ему UML-диаграмма деятельности (*activity diagram*).

Диаграмма деятельности – ориентированный граф, вершины которого обозначают действия, а дуги являются переходами от одного действия к другому.

Кризис UML. UML – широко распространенный и используемый стандарт, но несмотря на это его часто критикуют из-за наличия следующих недостатков:

Избыточность языка. UML включает большое количество избыточных или практически неиспользуемых диаграмм и конструкций.

Неточная семантика. Только код отражает код. UML полезен при подходах, с помощью которых

компилируют модели для генерирования исходного или выполнимого кода. Возможно, это недостаточно, т.к. в UML отсутствует свойство полноты по Тьюрингу, таким образом, любой сгенерированный код будет ограничен тем, что может разглядеть или предположить интерпретирующий UML инструмент.

Кумулятивная нагрузка/Рассогласование нагрузки. (*CumulativeImpedance/Impedancemismatch*). Рассогласование нагрузки – термин, применяемый в теории системного анализа, который обозначает неспособность входа одной системы воспринять выход другой.

Как в любой системе обозначений UML может представить одни системы более кратко и эффективно, чем другие. Следовательно, разработчик принимает решения, которые более комфортно подходят к синтезу сильных сторон UML и языков программирования. Проблема становится более очевидной, если язык разработки не придерживается принципов ортодоксальной объектно-ориентированной доктрины (не старается соответствовать традиционным принципам ООП).

Пытается быть всем для всех. UML является языком моделирования общего назначения, с помощью которого возникает возможность совместимости со всеми возможными языками разработки. Для конкретного проекта при достижении проектировщиками определенной цели, должны быть выбраны применимые возможности UML. Кроме того, пути ограничения области применения UML в конкретной работе проходят через формализм, который не полностью сформулирован и который сам является объектом критики.

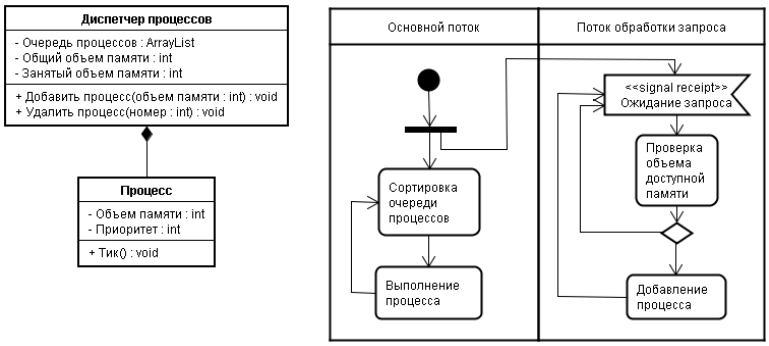
5. ПОРЯДОК ПРОЕКТИРОВАНИЯ [15, 16]

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. 8).

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, в свою очередь, отражает 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.

Взаимосвязь диаграмм. Следует выделить связь между структурными и поведенческими диаграммами. Эти два вида диаграмм описывают одну и ту же систему только с разных ракурсов, но 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.

Проектирование 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. 9, *а*.

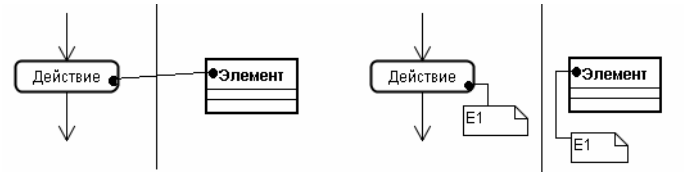


а *б*

Рис. 9. Диаграмма классов (*а*) и деятельности (*б*) [15, 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. 9,*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.

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. 10, *а*).



а *б*

Рис. 10. Варианты представления связи диаграмм [15, рис. 3]

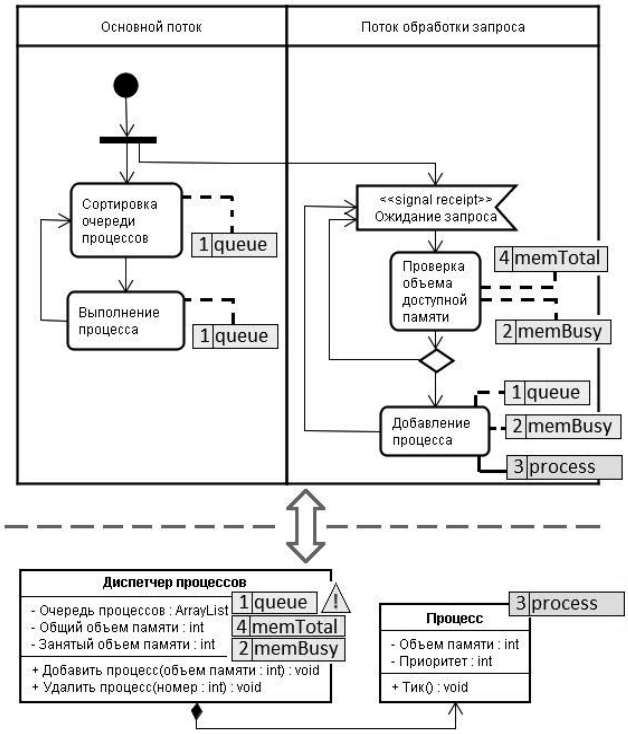


Рис. 11. Синтез структурной и динамической диаграмм [15, рис. 4]

Второй способ (рис. 10, *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. 11. 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.

6. 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 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.

Правила преобразования UML-диаграмм в сеть Петри и обратно. Трансляция UML-диаграмм в раскрашенную иерархическую сеть Петри и обратно должна осуществляться по формальным правилам, которые были предложены в [9, 10]. Перечислим их.

R1. Состояние ожидания трансформируется в место, а состояние действия преобразуется в переход, который начинает действие; в место, отражающее состояние выполнения действия; в 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. 12 [9].

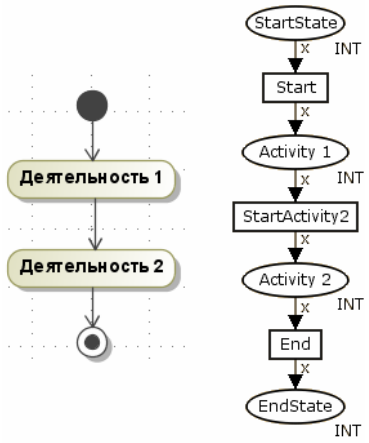


Рис. 12. Пример преобразования последовательности состояний

R2. Разделение и слияние параллельных потоков управления UML-диаграмм преобразуется в соответствующий эквивалент сети Петри, представленный на рис. 13 [9].

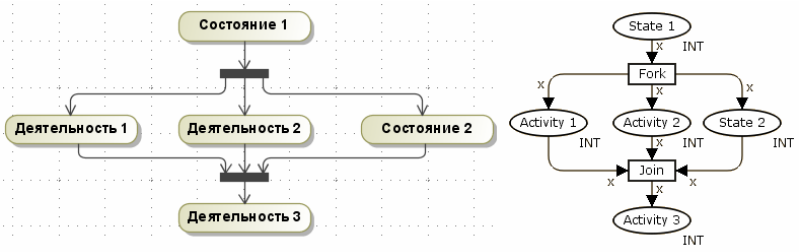


Рис. 13. Пример преобразования элементов разделение и слияни

R3. Ветвление на диаграмме деятельности, обозначаемое символом решения, преобразуется в соответствующий эквивалент сети Петри, представленный на рис. 14 [9].

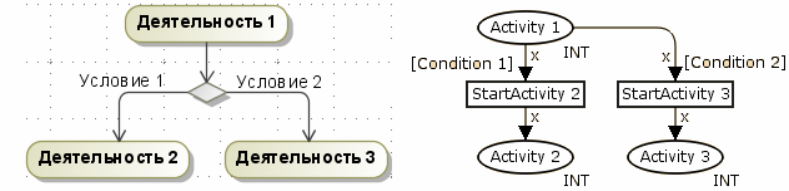


Рис. 14. Пример преобразования ветвления и условий переходов

R4. Условия ограничения показывают с помощью условий ограничения переходов, как показано на рис. 14: *«Условие 1»* транслируется в «защитное выражение» (*guard*) «*Condition1*», а *«Условие 2»* в «*Condition2*».

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. 15 [9]. 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.

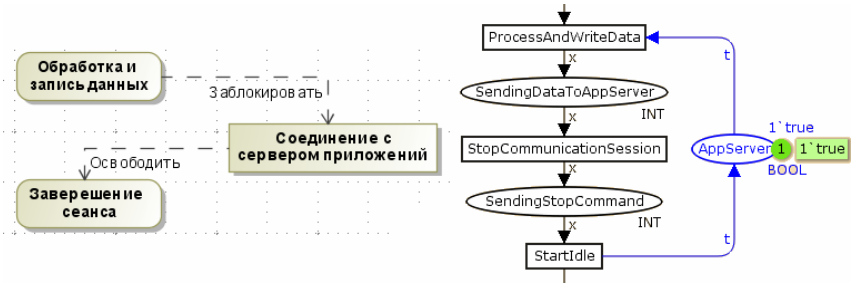


Рис. 15. Пример преобразования критической секции

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. 16 [9]. 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. При необходимости (поток канала для передачи команды и получения ответа) 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.

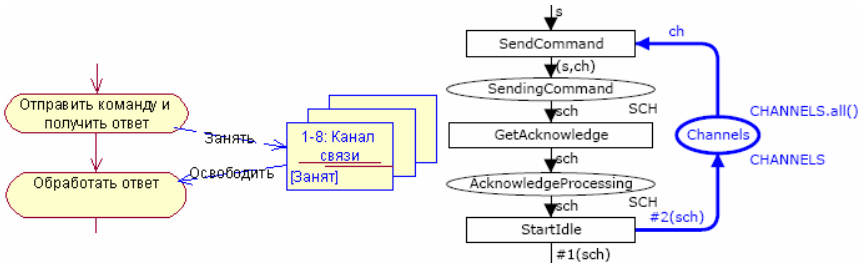


Рис. 16. Пример преобразования семафора [9, рис. 3.10]

R7. Дорожка (*swimlane*) – область на диаграмме деятельности, содержащая элементы модели, которые выполняет отдельная подсистема, отражаются в виде отдельной подсети (страницы). Пример преобразования приведен на рис. 17 [9]. *«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 странице, упрощают восприятие модели и отображают функциональность.

Предложенные выше правила преобразования UML-диаграмм в сеть Петри были дополнены правилами для сетей, критичных ко времени исполнения [6].

R8. 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. 18).

R9. При взаимном переходе из/в иерархическую сеть Петри необходимо добавить пустые места и соответствующие им переходы, которые соединены последовательно [10] (рис. 19).

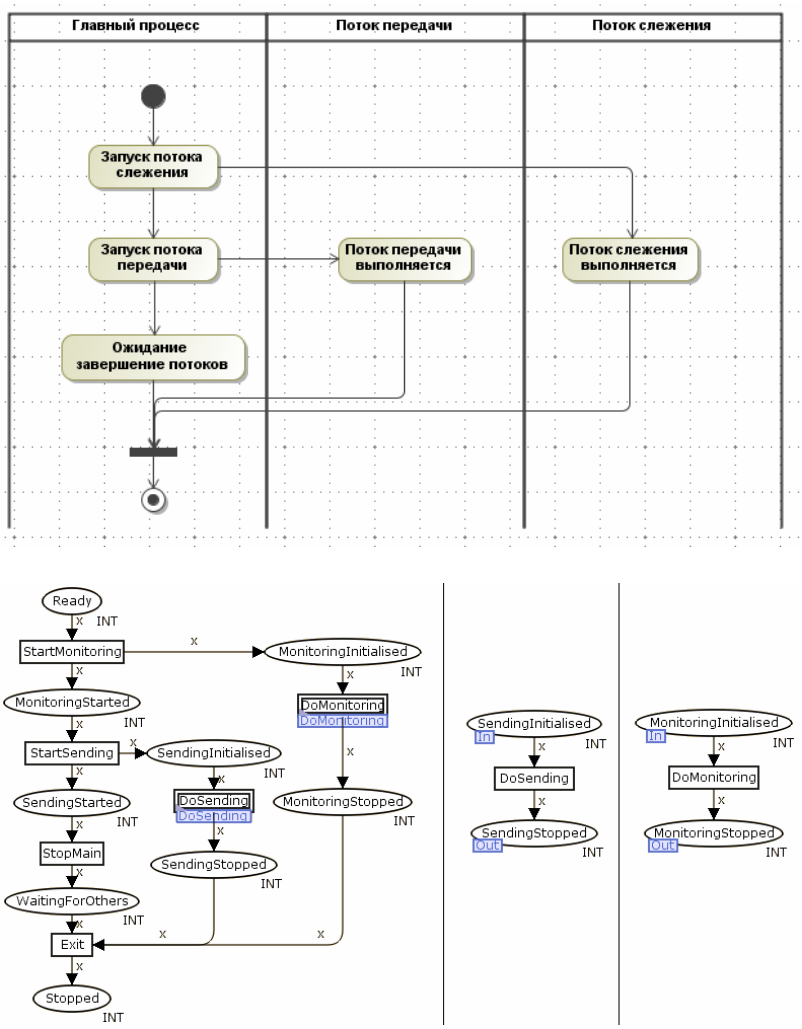


Рис. 17. Пример преобразования *«дорожек»*

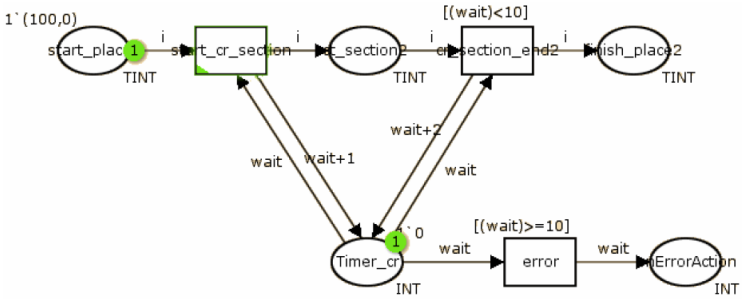


Рис. 18. Реализация защиты в критической секции в сетях Петри [10]

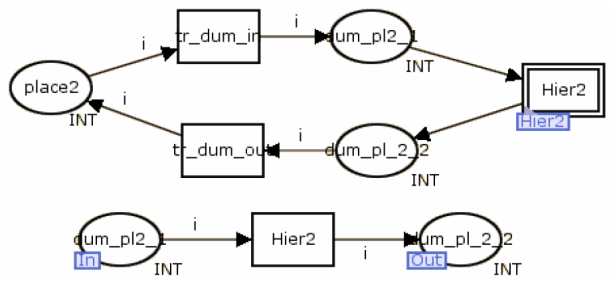


Рис. 19. Взаимный переход в иерархической сети Пери [10]

Методика построения и проверки модели проекта. В [91] предлагается методика 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.

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 преобразуется 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 и 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. 20.

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 соответствия требованиям к проектируемой СИСТЕМЕ.

Предложенная в данном разделе методика позволяет разработчику системы получить набор UML-диаграмм проекта, правильность и согласованность которых доказана с помощью формального метода. Представленная методика развивает и дополняет этап проектирования технологического цикла разработки, принятого в качестве стандарта в инженерии программного обеспечения, усовершенствуя его применением раскрашенных иерархических сетей Петри для проверки правильности и согласованности набора UML-диаграмм проекта.

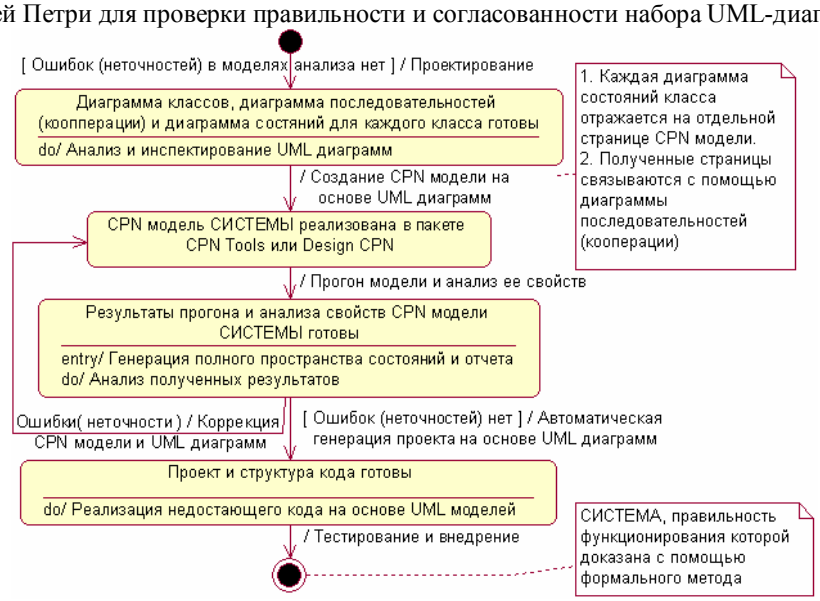


Рис. 20. Последовательность разработки СИСТЕМЫ (проверка проекта)
[9, рис. 4.1]

О возможности автоматической трансляции [6]. Зачастую на диаграммах деятельности встречаются такие элементы, как внешние сигналы и задержки времени. Если на диаграмме в виде свойств (стереотипов, тегов) объекта заданы параметры объекта, то при трансляции в сеть Петри открываются широкие возможности по реализации данных конструкций. Внешний сигнал есть появление в определенный момент времени в позиции маркера, а условие его появления (точнее перехода из буферной позиции), в свою очередь, можно задать как в виде обычных условных выражений, так и при помощи встроенного скриптового языка ML (программная среда CPN Tools). Категория временных сетей Петри позволяет работать со временем. Сам механизм учета времени легко реализовать с помощью маркеров, хранящих время, с его изменением на переходах.

Автоматизация трансляции. Процесс автоматического преобразования UML-диаграммы деятельности в сеть Петри можно представить в виде следующей последовательности.

1. Построение диаграммы с учетом рекомендаций и соглашений.
2. Преобразование основных элементов (действий и объектов – раздел «базовые элементы»), ветвлений и объединений, условий, критических секций и семафоров, разделов.
3. Преобразование merge-слияний.
4. Преобразование статических объектов.
5. Определение типов данных по стереотипам объектов.
6. Добавление элементов генерации внешних сигналов.
7. Добавление элементов работы со временем.

Реализация экспериментальной проверки, дальнейшего развития методики автоматической трансляции нуждается в разработке специализированного программного обеспечения. Формальные соглашения о трактовке назначения элементов диаграмм деятельности и правил их сочетания облегчает возможность автоматической трансляции диаграмм в сеть Петри.

ЗАКЛЮЧЕНИЕ

В данной работе подробно излагаются подходы к проектированию программного обеспечения с использованием UML и аппарата сетей Петри. Непосредственно описан язык UML с присущими ему положительными свойствами и недостатками, показано преимущество сетей Петри – свойство параллелизма.

Параллелизм сетей наглядно продемонстрирован на примере системы нахождения выхода из лабиринта меткой – манипулятором. Основными достоинствами спроектированной системы является следующее: структура лабиринта может быть сколь угодно большой, но конструкция сети останется прежней при изменении структуры лабиринта, все свойства закладываются в метке; возможность задавать размерность, структуру лабиринта и всевозможного рода условия в текстовом файле. При необходимости изменить структуру лабиринта нужно лишь изменить массив типов данных, с помощью которого задается лабиринт.

Описана методология проектирования при совместном использовании UML и сетей Петри. Приведены правила для трансляции и преобразования UML в сеть Петри.

[1] *Питерсон Дж.* Теория сетей Петри и моделирование систем: Пер. с англ. – М.: Мир, 1984. – 264 с.

[2] *Воевода А.А., Марков А.В.* О Тестирование UML-диаграмм с помощью аппарата сетей Петри на примере разработки ПО для игры «Змейка» // Сборник научных трудов НГТУ. – 2010. – № 3(61). – С. 51–60.

[3] *Марков А.В.* Моделирование процесса поиска пути в лабиринте при помощи сетей Петри // Сборник научных трудов НГТУ. – 2010. – № 4(62). – С. 133–140.

[4] *Романников Д.О., Марков А. В., Зимаев И.В.* Обзор работ посвященным разработке ПО с использованием UML и сетей Петри // Сборник научных трудов НГТУ. – 2011. – № 1(63). – С. 91–104.

[5] *Воевода А.А., Романников Д.О.* Особенности проектирования систем реального времения при помощи UML и сетей Петри // Сборник научных трудов НГТУ. – 2009. – № 1(55). – С. 57–62.

[6] *Зимаев И.В.* О возможности автоматической трансляции UML-диаграмм деятельности с сети Петри // Сборник научных трудов НГТУ. – 2010. – № 1(59). – С. 149–156.

[7] *Прытков Д.В.* О применении сетей Петри для исполнения алгоритмов на примере решения задач о кратчайших путях с единственным источником // Сборник научных трудов НГТУ. – 2010(61). – № 3. – С. 91–99.

[8] *Марков А.В.* Автоматизация разработки программного обеспечения с использованием сетей Петри: магистерская диссертация. – Новосибисрк, НГТУ. – 2011.

[9] *Коротиков С.В.* Применение сетей Петри в разработке программного обеспечения центров контроля и

управления: дис. канд. техн. наук. – Новосибирск: Изд-во НГТУ, 2007.

[10] *Романников Д.О.* Использование UML-диаграмм и сетей Петри в разработке программного обеспечения систем распределенной обработки информации, критических ко времени исполнения: магистерская диссертация. – Новосибирск, НГТУ. – 2011.

[11] *Марков А.В. Романников Д.О.* Совокупное использование сетей петри и UML-диаграмм при разработке программного обеспечения // Сборник научных трудов НГТУ. – 2011. – № 2(64). – С. 85–94.

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

[13] *Марков А.В., Воевода А.А.* Описание разрабатываемой системы «Поиск манипулятором кратчайшего пути в лабиринте» // Сборник научных трудов НГТУ. – 2011. – № 3(65). – С.105–112.

[14] *Марков А.В., Прытков Д.В.* Описание работы двухсимочных мобильных телефонов с помощью сетей Петри. Камбиев метод // Сборник научных трудов НГТУ. – 2011. – № 3(65). – С. 113–118.

[15] *Зимаев И.В.* Интеграция структурных и динамических UML-моделей // Сборник научных трудов НГТУ. – 2010. – № 3(61). – С. 77–84.

[16] *Зимаев И.В.* Блоки анализирующей сети Петри // Сборник научных трудов НГТУ. – 2010. – № 3(61). – С. 169–172.

[17] *Марков А.В.* Поиск манипулятором кратчайшего пути в лабиринте // Сборник научных трудов НГТУ. – 2011. – № 4(66). – С. 75–90.

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

[19] *Романников Д.О., Воевода А.А.* Использование UML и временных сетей Петри в методе разработке ПО. Ч.2 // Сборник научных трудов НГТУ. – 2010. – № 4(62). – С. 117–126.

Воевода Александр Александрович – доктор технических наук, профессор кафедры автоматики Новосибирского государственного технического университета. Основное научное направление научных исследований – управление многомерными объектами, исследование свойств UML-диаграмм и сетей Петри. Имеет 200 публикаций. E-mail: voevoda@ucit.ru.

Коротиков Сергей Викторович, кандидат технических наук, ст. преподаватель кафедры автоматики Новосибирского государственного технического университета. Основное направление научных исследований – автоматизация написания программного обеспечения для технических систем. Имеет 19 публикаций. E-mail: nsk_nstu@mail.ru.

Марков Александр Владимирович – аспирант кафедры автоматики НГТУ по специальности 05.13.01 «Системный анализ, управление и обработка информации (в промышленности)». Имеет 16 публикаций. E-mail: muviton3@mail.ru.

A.A. Voevoda, S.V. Korotikov, A.V. Markov

The review of works describing sharing UML diagrams and Petri nets at a design stage of software

In given article the review of works devoted to creation of software by means of Petri nets and UML of diagrams is resulted. The design stage is in detail described. In offered article the basic aspects, the positive moments and lacks of Petri nets are allocated. Such as parallelism of networks, on an example of movement of a label and the manipulator on a labyrinth, and UML diagrams – visual modeling of system. The description of sharing of mathematical apparatus of Petri nets and language UML, rules of transformation UML in a network and on the contrary is resulted. The technique of creation of the software offered earlier with use of Petri nets and UML diagrams is described. It is told about possibility and necessity of automatic translation UML of diagrams for a Petri nets.

Key words: software engineering, UML-diagrams, Petri nets, CPN Tools, automatic translation, merge of diagrams.