Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э.Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Компьютерные системы автоматизации производства

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ДИПЛОМНОМУ ПРОЕКТУ

НА ТЕМУ:

Процессный подход в системе имитационного моделирования Rao X

Студент РК9-Д1 (Группа)	_____ (Подпись, дата)	О.В. Зудина (И.О.Фамилия)
Руководитель дипломного проекта	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по научно-исследовательской части	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по конструкторской части	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по технологической части	_____ (Подпись, дата)	А.В. Урусов (И.О.Фамилия)
Консультант по организационно-экономической части	_____ (Подпись, дата)	Л.А. Силаева (И.О.Фамилия)
Консультант по охране труда и экологии	_____ (Подпись, дата)	А.С. Козодаев (И.О.Фамилия)
Нормоконтролер	_____ (Подпись, дата)	М.Н. Святкина (И.О.Фамилия)

2016 г.

СОДЕРЖАНИЕ

РЕФЕРАТ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, СИМВОЛОВ И СПЕЦИАЛЬНЫХ ТЕРМИНОВ

ИМ — Имитационное моделирование

СДС — Сложная дискретная система

СМО — Система массового обслуживания

ПО — Программное обеспечение

ПП — Программный продукт

ПЭВМ — Персональная электронно-вычислительная машина

ТЗ — Техническое задание

ЭП — Эскизный проект

ТП — Технический проект

РП — Рабочий проект

GPSS — General Purpose Simulation System — Система моделирования общего назначения

FIFO — First In First Out — дисциплина очереди "первым пришел — первым вышел"

LIFO — Last In First Out — дисциплина очереди "последним пришел — первым вышел"

IDE — Integrated Development Environment — Интегрированная среда разработки

OpenGL — Open Graphics Library — Открытая графическая библиотека

API — Application Programming Interface — Интерфейс прикладного программирования

AWT — Abstract Window Toolkit — Оконная библиотека графического интерфейса (языка Java)

SWT — Standard Widget Toolkit — Стандартная библиотека графического интерфейса (языка Java)

Средство визуального программирования — Графический редактор — программа, позволяющая создавать, просматривать и редактировать имитационные модели посредством палитры инструментов и графических блоков.

Плагин — независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей.

Транзакт — объект, обслуживание которого производится в имитационной модели процессно-ориентированного подхода.

Блок — объект, который обрабатывает транзакты в имитационной модели процессно-ориентированного подхода.

Кроссплатформенность — возможность программного обеспечения работать более, чем на одной аппаратной платформе и/или операционной системе.

Конвертация — преобразование данных из одного формата в другой.

Скрипт — программа, работающая с готовыми программными компонентами.

ВВЕДЕНИЕ

Имитационное моделирование (ИМ)[1] на ЭВМ находит широкое применение при исследовании и управлении сложными дискретными системами (СДС) и процессами, в них протекающими. К таким системам можно отнести экономические и производственные объекты, морские порты, аэропорты, комплексы перекачки нефти и газа, ирригационные системы, программное обеспечение сложных систем управления, вычислительные сети и многие другие. Широкое использование ИМ объясняется тем, что размерность решаемых задач и неформализуемость сложных систем не позволяют использовать строгие методы оптимизации. Эти классы задач определяются тем, что при их решении необходимо одновременно учитывать факторы неопределенности, динамическую взаимную обусловленность текущих решений и последующих событий, комплексную взаимозависимость между управляемыми переменными исследуемой системы, а часто и строго дискретную и четко определенную последовательность интервалов времени. Указанные особенности свойственны всем сложным системам.

Проведение имитационного эксперимента позволяет:

- Сделать выводы о поведении СДС и ее особенностях:
  - без ее построения, если это проектируемая система;
  - без вмешательства в ее функционирование, если это действующая система, проведение экспериментов над которой или слишком дорого, или небезопасно;
  - без ее разрушения, если цель эксперимента состоит в определении пределов воздействия на систему.
- Синтезировать и исследовать стратегии управления.
- Прогнозировать и планировать функционирование системы в будущем.
- Обучать и тренировать управленческий персонал и т.д.

ИМ является эффективным, но и не лишенным недостатков, методом. Трудности использования ИМ, связаны с обеспечением адекватности описания системы, интерпретацией результатов, обеспечением стохастической сходимости процесса моделирования, решением проблемы размерности и т.п. К проблемам применения ИМ следует отнести также и большую трудоемкость данного метода.

Интеллектуальное ИМ, характеризующиеся возможностью использования методов искусственного интеллекта и прежде всего знаний, при принятии решений в процессе имитации, при управлении имитационным экспериментом, при реализации интерфейса пользователя, создании информационных банков ИМ, использовании нечётких данных, снимает часть проблем использования ИМ.

1 Предпроектное исследование

1.1 Основные положения языка РДО

Основные положения языка РДО[2] могут быть сформулированы следующим образом[1]:

- Все элементы СДС представлены как ресурсы, описываемые некоторыми параметрами. Ресурсы могут быть разбиты на несколько типов; каждый ресурс определенного типа описывается одними и теми же параметрами.
- Состояние ресурса определяется вектором значений всех его параметров; состояние СДС - значением всех параметров всех ресурсов.
- Процесс, протекающий в СДС, описывается как последовательность целенаправленных действий и нерегулярных событий, изменяющих определенным образом состояние ресурсов; действия ограничены во времени двумя событиями: событиями начала и событиями конца.
- Нерегулярные события описывают изменения состояния СДС, непредсказуемые в рамках продукционной модели системы (влияние внешних по отношению к СДС факторов либо факторов, внутренних по отношению к ресурсам СДС). Моменты наступления нерегулярных событий случайны.
- Действия описываются операциями, которые представляют собой модифицированные продукционные правила, учитывающие временные связи. Операция описывает предусловия, которым должно удовлетворять состояние участвующих в операции ресурсов, и правила изменения состояния ресурсов в начале и в конце соответствующего действия.

- Множество ресурсов R и множество операций O образуют модель СДС.

1.2 Процессно-ориентированный подход дискретного имитационного моделирования

Процессно-ориентированный подход предназначен для описания систем массового обслуживания. Имитационные модели, описывающие СМО, содержат последовательности компонентов, которые возникают в них по определенной схеме, например очередь, в которой клиенты ожидают обслуживания. Логика возникновения компонентов по требуемой схеме может быть обобщена и задана в одном блоке. Имитационный процессно-ориентированный язык транслирует блоки в соответствующую последовательность событий, происходящих с компонентами модели. Блоки определяют последовательность событий, которые автоматически выполняются имитационным языком, по мере того как транзакты продвигаются через систему.

Процессно-ориентированный подход обеспечивает описание прохождения компонентов через процесс, содержащий ресурсы. Простота этого подхода состоит в том, что определяемая блоками логика событий заложена в самом имитационным языке. Однако, так как набор стандартных блоков языка ограничен, процессно-ориентированный подход является менее гибким, чем событийный. Кроме того, требуется постоянный анализ ресурсов после их использования.

1.3 Система имитационного моделирования Rao X

Система имитационного моделирования Rao X представляет собой плагин для интегрированной среды разработки Eclipse [13], позволяющий вести

разработку имитационных моделей на языке РДО. Система написана на языке Java[3] и состоит из четырех основных компонентов:

- rao – компонент, производящий преобразование кода на языке РДО в код на языке Java.
- rao.lib – библиотека системы. Этот компонент реализует ядро системы имитационного моделирования.
- rao.ui – компонент, реализующий графический интерфейс системы с помощью библиотеки SWT.
- rao.tests – компонент, реализующий тестирование системы посредством Unit-тестов.

2 Формирование ТЗ

2.1 Введение

Программный комплекс Rao X предназначен для разработки и отладки имитационных моделей на языке РДО. Основные цели данного комплекса - обеспечение пользователя легким в обращении, но достаточно мощным средством разработки текстов моделей на языке РДО, обладающим большинством функций по работе с текстами программ, характерных для сред программирования, а также средствами проведения и обработки результатов имитационных экспериментов.

Программный комплекс Rao X позволяет разрабатывать имитационные модели на основе двух подходов: событийного и подхода сканирования активностей.

Требуется реализовать возможность проведения имитационных экспериментов на основе процессно-ориентированного подхода дискретного имитационного моделирования.

Техническое задание на разрабатываемую подсистему оформлено в соответствии с ГОСТ 34.602-89.

2.2 Общие сведения

Основание для разработки: задание на дипломный проект.

Заказчик: Кафедра «Компьютерные системы автоматизации производства» МГТУ им. Н.Э. Баумана

Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Зудина О.В.

Наименование темы разработки: «Процессный подход в системе имитационного моделирования Rao X»

2.3 Назначение подсистемы

Реализовать подсистему процессно-ориентированного подхода и средство визуального программирования (графический редактор) моделей процессного подхода в системе имитационного моделирования Rao X.

2.4 Требования к программному продукту

2.4.1 Требования к функциональным характеристикам

Подсистема процессно-ориентированного подхода должна удовлетворять следующим требованиям:

- Интеграция в Rao X;
- Реализация следующих блоков:
  - создания транзактов;
  - удаления транзактов;
  - очереди;
  - захвата ресурса;
  - освобождения ресурса;
  - задержки транзакта;
  - ветвления модели;
- Создание графических блоков модели процессного подхода;
- Задание параметров блоков модели процессного подхода;
- Создание связей между графическими блоками модели;
- Отображение ошибок и предупреждений;
- Отображение результатов прогона модели в виде трассировки.

2.4.2 Требования к надежности

Основное требование к надежности направлено на поддержание в исправном и работоспособном состоянии ЭВМ, на которой происходит использование программного комплекса Rao X и подсистемы процессно-ориентированного подхода.

2.4.3 Условия эксплуатации

Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости

Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением

2.4.4 Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

- объем ОЗУ не менее 2 Гб;
- объем жесткого диска не менее 50 Гб;
- микропроцессор с тактовой частотой не менее 2ГГц;
- монитор с разрешением от 1366*768 и выше.

2.4.5 Требования к информационной и программной совместимости

Система должна работать под управлением следующих ОС: Windows 7, Windows 8, Ubuntu 15.10.

2.4.6 Требования к маркировке и упаковке

Требования к маркировке и упаковке не предъявляются.

2.4.7 Требования к транспортированию и хранению

Требования к транспортированию и хранению не предъявляются.

2.4.8 Требования к программной документации

Требования к программной документации не предъявляются.

2.5 Технико-экономические показатели

2.5.1 Стадии и этапы разработки

Состав, содержание и сроки выполнения работ по созданию системы в соответствии с календарным планом на выполнение дипломного проекта.

2.5.2 Порядок контроля и приемки

Контроль и приемка приложения должны состоять из следующих этапов:

- Запуск средства визуального программирования моделей процессного подхода системы имитационного моделирования Rao

X в операционной системе Linux и проверка основного функционала системы на тестовых имитационных моделях;

- Запуск средства визуального программирования моделей процессного подхода системы имитационного моделирования Rao X в операционной системе Windows и проверка основного функционала системы на тестовых имитационных моделях.

3 Концептуальный этап проектирования

На концептуальном этапе проектирования требовалось:

- определить взаимосвязь процессно-ориентированного подхода с реализованными подходами в системе имитационного моделирования Rao X;
- разработать концепцию продвижения транзактов по блокам имитационной модели;
- разработать набор блоков, необходимых для моделирования;
- разработать средство визуального программирования (графический редактор) имитационных моделей процессного подхода.

3.1 Процессно-ориентированный подход в Rao X

Процессно-ориентированный подход должен быть способен работать в Rao X как независимо, так и взаимодействуя с другими подходами (событийным и сканирования активностей). Процессная часть имитационной модели должна состоять из последовательности стандартных блоков, каждый из которых выполняет определенное действие над транзактом. При этом процессная часть имитационной модели не должна завершать свою работу, если остался хотя бы один блок, все еще способный совершить действие над транзактом.

3.2 Продвижение транзактов

Для организации продвижения транзактов по блокам имитационной модели процессно-ориентированного подхода необходимы узлы для каждого входа и выхода. После обработки блоком транзакта, он временно хранится в выходном узле блока. Когда следующий блок оказывается готов принять

транзакт, он забирает его через входной узел из выходного узла текущего блока.

3.3 Блоки подсистемы

Процессный подход ориентирован на моделирование СМО. В соответствии с этим определены требования к алгоритмам и параметрам блоков.

3.3.1 Создание и удаление транзактов

Для обеспечения потока транзактов через систему необходимо наличие блока создания транзактов, который будет генерировать новые транзакты с заданной периодичностью. Периодичность транзакта должна быть конфигурируемой. Блок создания транзактов должен иметь один выход и не иметь входов.

Для обеспечения возможности удаления транзактов из системы должен существовать отдельный блок удаления транзактов, имеющий один вход и не имеющий выходов.

3.3.2 Очередь транзактов

Любая СМО включает в себя очередь.

Блок, реализующий очередь транзактов, должен иметь один вход и один выход. Необходимо иметь возможность задания и реализации параметров очереди: ёмкость и дисциплину (FIFO, LIFO).

3.3.3 Захват и освобождение ресурсов

Любая СМО имеет ограничение на количество доступных ресурсов. В связи с этим необходима реализация двух блоков: захвата и освобождения ресурса. Разработчик модели должен иметь возможность связывать ресурс с блоками захвата и освобождения, указывать параметр ресурса, отвечающий за состояния "свободен-занят", а также конкретные значения этого параметра, соответствующие этим состояниям. Каждый из блоков должен иметь один вход и один выход.

3.3.4 Задержка транзактов

Обслуживание транзактов в СМО – процесс, занимающий определенное время. Для моделирования этого процесса необходим блок, способный задерживать транзакт, пока не завершится его обслуживание. Длительность обслуживания должна быть конфигурируемой. Блок задержки транзактов должен иметь один вход и один выход.

3.3.5 Ветвление модели

Для реализации сложных СМО возникает потребность в блоке ветвления модели. Условие ветвления или вероятность прохода транзакта через определенный выход должны задаваться разработчиком модели. Блок ветвления должен иметь один вход и несколько выходов.

3.3.6 Средство визуального программирования ИМ

Современные средства визуального программирования (графические редакторы) предоставляют пользователю палитру, которая содержит инструменты для создания доступных блоков, связей и выделения созданных

объектов. С помощью этих инструментов происходит работа с имитационной моделью. Для реализации подобного функционала целесообразно использовать специализированные графические библиотеки, отвечающие следующим требованиям:

- графическое отображение объектов;
- создание палитры инструментов;
- создание, удаление и изменение объектов;
- задание параметров объектов;
- создание, удаление и изменение связей;
- отображение ошибок и предупреждений модели.

4 Технический этап

На техническом этапе проектирования требовалось:

- разработать алгоритм работы подсистемы процессно-ориентированного подхода;
- разработать алгоритмы работы блоков процессно-ориентированного подхода;
- выбрать графическую библиотеку для реализации средства визуального программирования (графический редактор) имитационных моделей.

4.1 Алгоритм работы подсистемы процессно-ориентированного подхода

Вызов подсистемы, отвечающий за реализацию процессно-ориентированного подхода, должен быть встроен в общий алгоритм прогона имитационных моделей симулятора Rao X. При старте модели или после того, как произошло очередное продвижение модельного времени, происходит сканирование всех активностей модели. Если ни одна из активностей не была выполнена, симулятор запускает процессную часть имитационной модели.

Функция работы процессной части имитационной модели возвращает один из трех возможных статусов:

- SUCCESS – сканирование процесса завершено успешно, модель изменила свое состояние. Выполняется оповещение о том, что модель изменила свое состояние, и цикл сканирования активностей запускается заново.
- NOTHING_TO_DO – сканирование процесса завершено успешно, модель не изменила свое состояние. Процессная часть имитационной модели завершила свою работу, не выполнив ни

одного действия. Симулятор, если это возможно, продвигает модельное время, выполняет следующее событие и запускает заново цикл сканирования активностей.

FAILURE – сканирование процесса завершено с ошибкой.

- Симулятор не может больше продолжать работу, и прогон модели завершается со статусом ошибки (RUNTIME_ERROR).

В процессной части имитационной модели происходит поочередная проверка всех блоков и выполнение их действий, если это представляется возможным. Функция проверки блоков может вернуть три вида статусов:

- SUCCESS – блок успешно выполнил свои действия. Процессная часть модели завершает свою работу со статусом SUCCESS.
- NOTHING_TO_DO - нет действий, которые блок мог бы выполнить. Происходит переход к следующему блоку.
- CHECK_AGAIN - блок не может выполнить требуемое действие. Если в процессе проверки остальных блоков, ни один из них не завершится со статусом SUCCESS, т.е. состояние модели больше не будет изменено, то вся функция выполнения процессной части имитационной модели должна вернуть статус FAILURE.

4.2 Блоки подсистемы

4.2.1 Блок Generate

Блок Generate имеет статус готовности создавать транзакт. Если статус готовности выставлен в значение *false*, то блок при его проверке не производит никаких действий. При отсутствии транзакта в выходном узле блок создает новый транзакт. Для обеспечения генерации следующего транзакта в нужный момент времени, блок Generate выставляет свой статус готовности в значение *false* и планирует событие, которое выставит блоку статус готовности *true*. После чего функция проверки блока возвращает статус SUCCESS.

Интервал времени, с которым блок Generate создает транзакты, задается при его создании и хранится в нем в виде функции (*Supplier<Integer>*), возвращающей некоторое значение при ее вызове. На рис. 1 представлена диаграмма активности блока Generate.

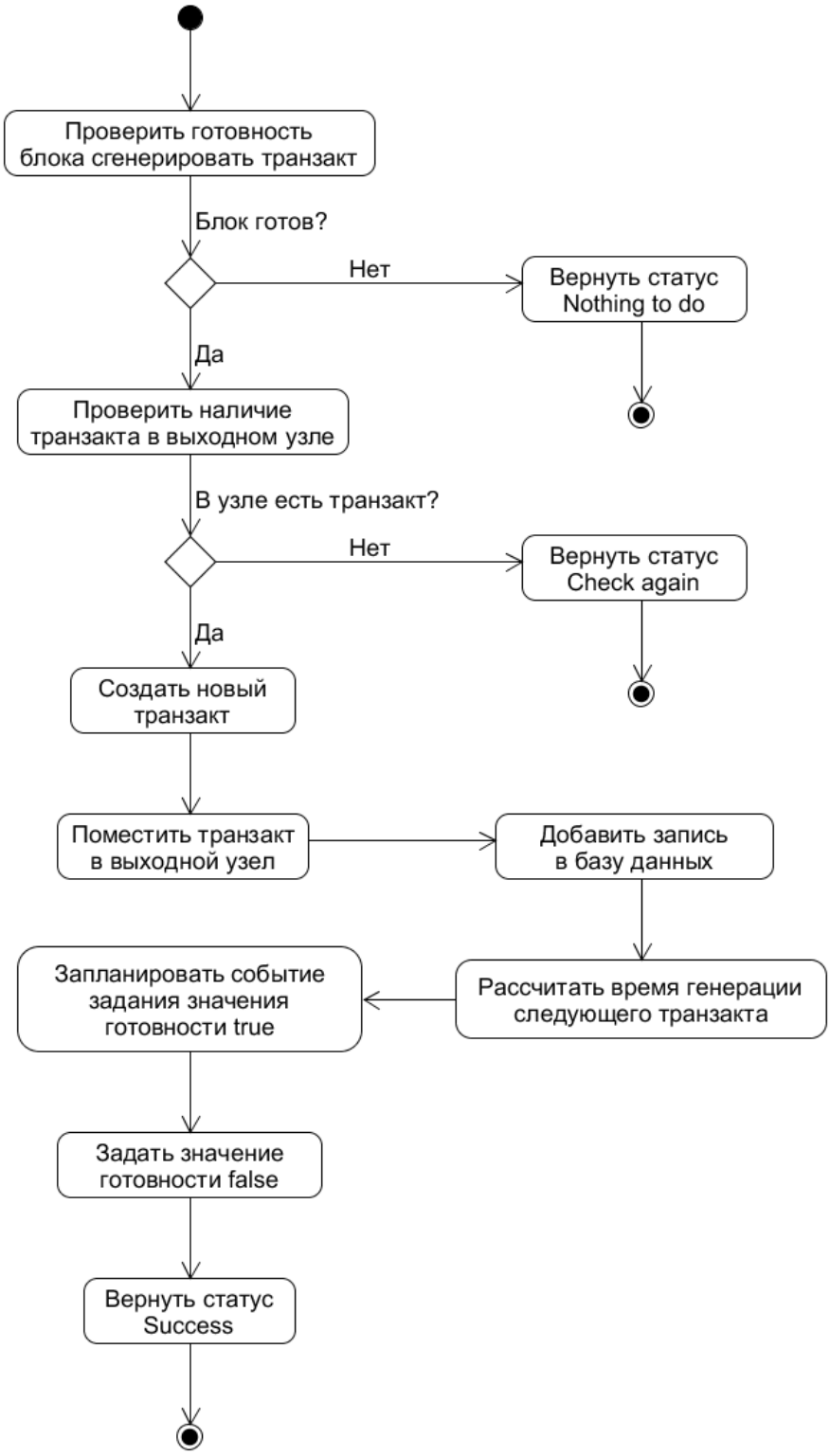

Рис. 1. Диаграмма активности блока Generate

4.2.2 Блок Terminate

В начале своей работы блок Terminate получает транзакт из входного узла. В случае, если транзакт не был получен, блок возвращает статус NOTHING_TO_DO. В противном случае полученный транзакт удаляется, возвращается статус SUCCESS. На рис. 2 представлена диаграмма активности блока Terminate.

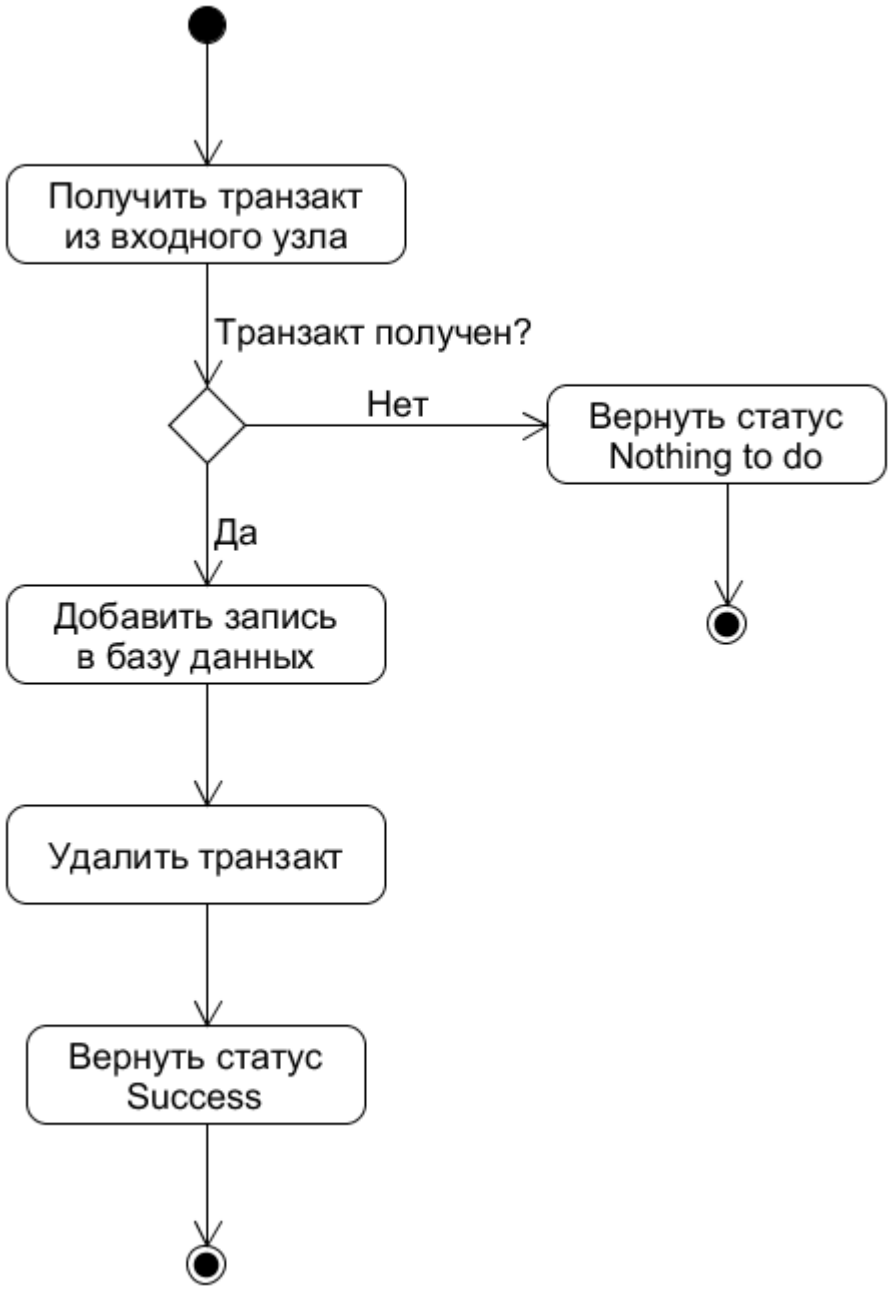

Рис. 2. Диаграмма активности блока Terminate

4.2.3 Блок Queue

В начале своей работы блок Queue получает транзакт из входного узла. Он добавляется в очередь, если заданная разработчиком емкость больше, чем текущее количество транзактов в очереди. После добавления транзакта в очередь блок возвращает статус SUCCESS. На рис. 3 представлена диаграмма активности блока Queue.

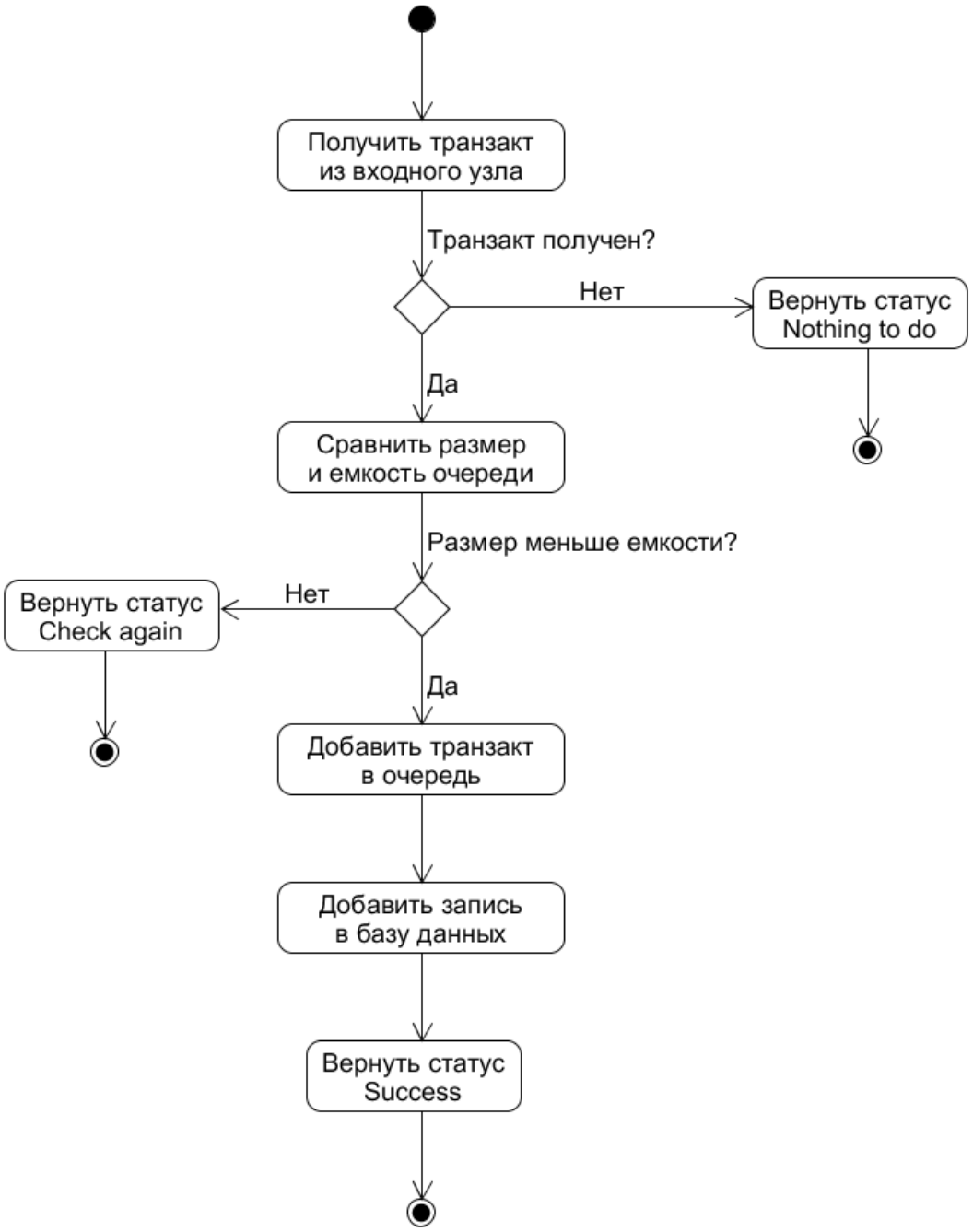

Рис. 3. Диаграмма активности блока Queue

4.2.4 Блок Seize

Работа блока начинается с проверки занятости ресурса: если ресурс занят, блок возвращает статус NOTHING_TO_DO. При свободном выходном узле происходит попытка получения транзакта из входного узла. При получении транзакта состояние ресурса изменяется на "занят", возвращается статус SUCCESS. Необходимые параметры для блока: ресурс, состояния ресурса.

На рис. 4. представлена диаграмма активности блока Seize.

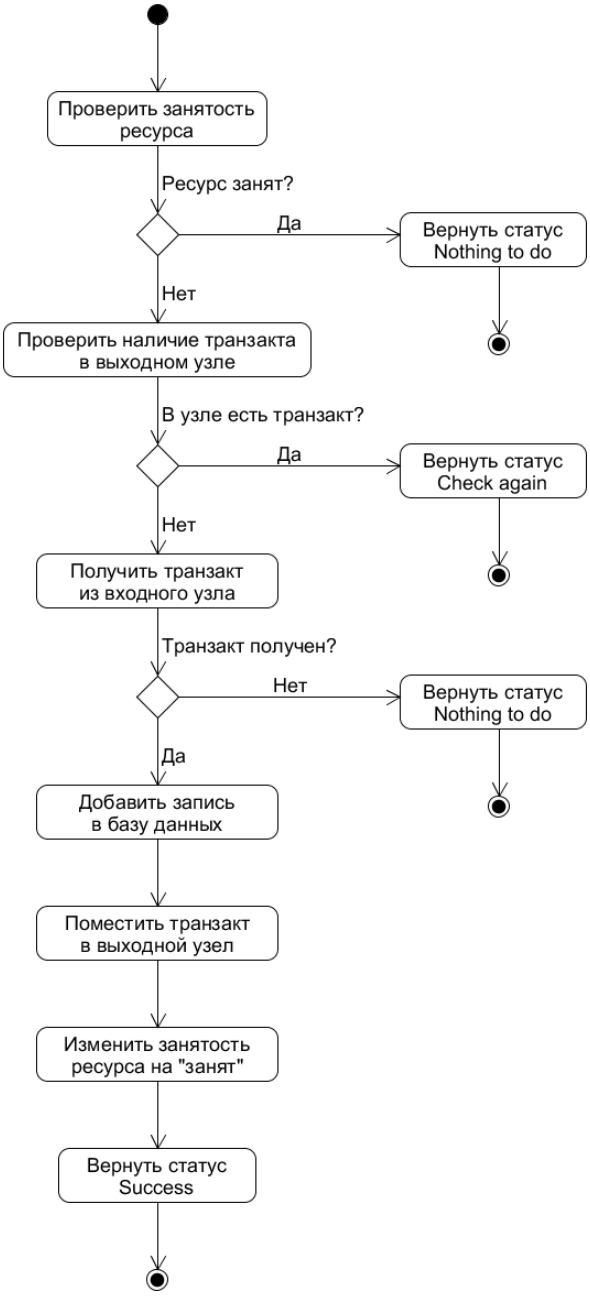

Рис. 4. Диаграмма активности блока Seize

4.2.5 Блок Release

Работа блока Release происходит по аналогии с блоком Seize. Отличия наблюдаются в последовательности проверок и в реакции на проверку занятости ресурса: если ресурс свободен, то следует сообщение об ошибке. Иначе, блок продолжает свою работу. Необходимые параметры для блока: ресурс, состояния ресурса.

На рис. 5. представлена диаграмма активности блока Release.

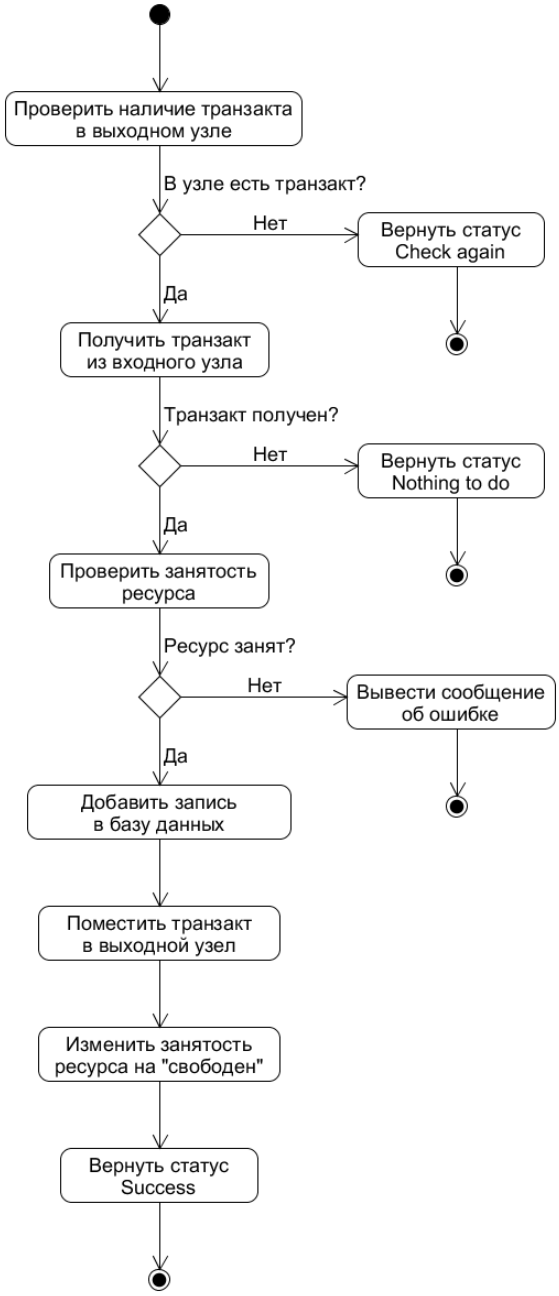

Рис. 5. Диаграмма активности блока Release

4.2.6 Блок Hold

В первую очередь блок Hold проверяет наличие временного транзакта, который закончил моделируемое обслуживание. Временный транзакт перемещается в выходной узел. Далее блок получает транзакт из входного узла и планирует событие окончания его обслуживания с заданной блоку длительностью. После чего возвращается статус SUCCESS.

Интервал времени, на который блок задерживает транзакты задается функцией, как и в случае с блоком Generate.

Блок-схема, описывающая алгоритм блока Hold представлена на рис. 6.

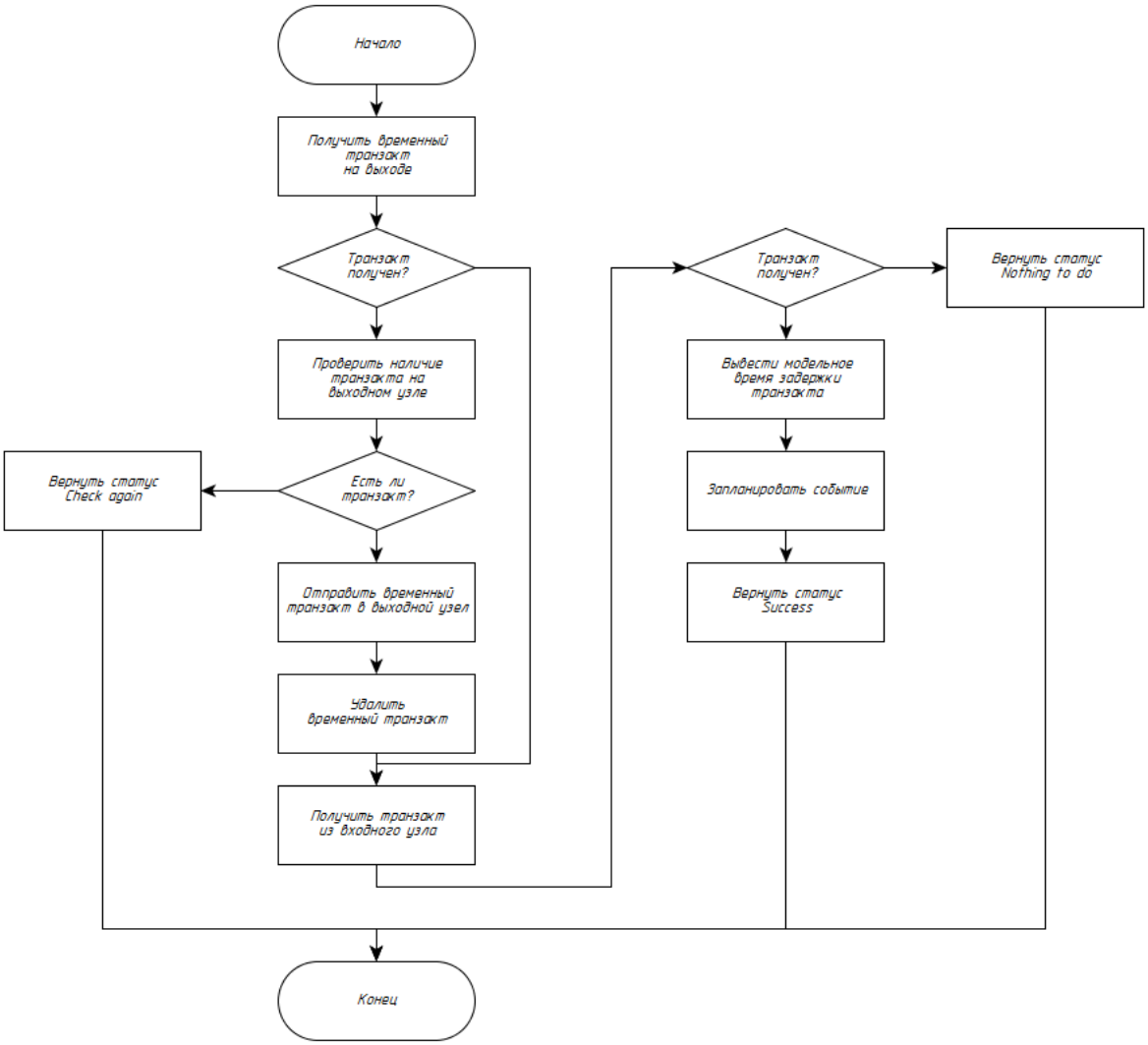

Рис. 6. Блок-схема алгоритма блока Hold

4.2.7 Блок SelectPath

Алгоритм блока SelectPath начинается с получения транзакта из входного узла. Далее выполняется проверка условия, заданного в блоке SelectPath, и назначаются соответствующие условию выходные узлы.

Параметр, необходимый для реализации блока SelectPath: режим работы блока, имя функции (при работе с условием), значение вероятности (при работе с вероятностью).

На рис. 7 представлена диаграмма активности блока SelectPath.

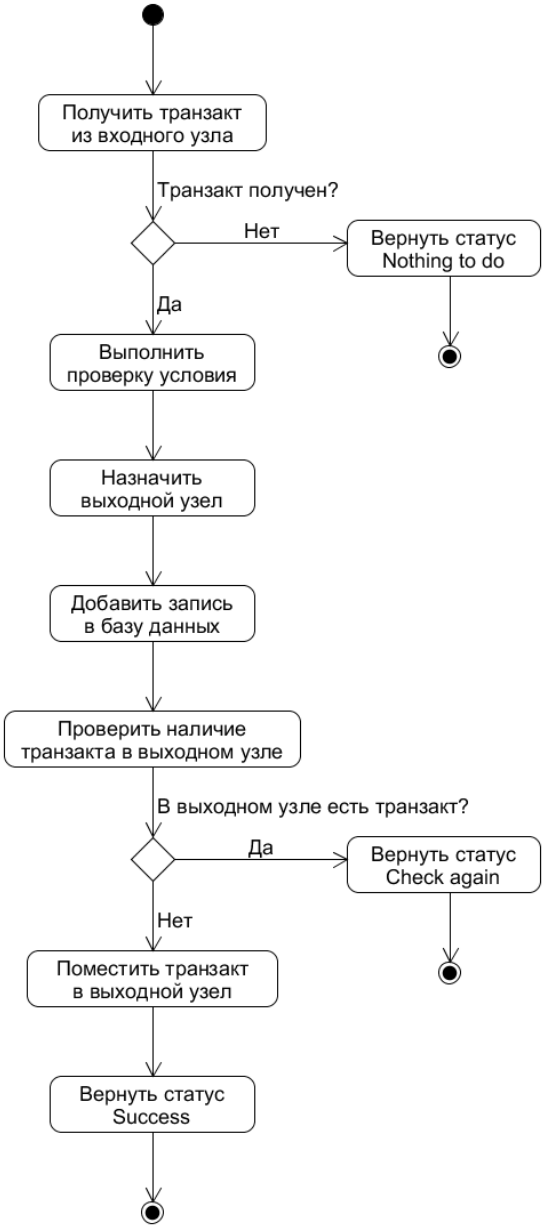

Рис. 7. Диаграмма активности блока SelectPath

4.3 Выбор графической библиотеки

Были рассмотрены следующие графические Java-библиотеки:

- GEF;
- JGraphX;
- LibGDX;
- Mica.

4.3.1 Библиотека GEF

Кроссплатформенная библиотека GEF [10] предоставляет возможность создавать модель произвольной иерархии, что идеально подходит для описания графов. Является разработкой Eclipse, поэтому предоставляет возможности для глубокой интеграции в приложения на платформе Eclipse[13]. Специализируется на создании средств визуального программирования, имеет встроенные классы для создания поля редактирования, палитры и инструментов. Поддерживает использование слоев и координатной сетки в поле редактирования.

4.3.2 Библиотека JGraphX

Библиотека JGraphX [9] предназначена для отрисовки графов (вершин и связей), используется в подсистеме графов системы Rao X. Поддерживает автоматическое выстраивание и выравнивание вершин графа по координатной сетке.

Разработана на базе AWT, что затрудняет встраивание в среду разработки Eclipse[13]. Не имеет возможности реализации графического редактора, палитры и моделей, имеющих более сложное представление, чем граф.

4.3.3 Библиотека LibGDX

LibGDX [12] является современной кроссплатформенной библиотекой, предоставляющей мощные инструменты для работы с графикой. Использует OpenGL для отрисовки на всех платформах, и предоставляет высокоуровневое API как для 2D, так и для 3D отрисовки. Имеет встроенные возможности для моделирования математических и физических процессов.

Функциональные возможности, которые предоставляет данная библиотека, могут быть применены на этапе реализации анимации прогона, однако готовых решений для реализации графического редактора LibGDX не предоставляет.

4.3.4 Библиотека Mica

Mica [11] имеет встроенные классы палитры инструментов, необходимые для создания средства визуального программирования. Кроме того, в библиотеке реализованы следующие возможности: масштабирование, перемещение по области редактирования, создание связей, поддержка слоев и координатной сетки, построение графов, отмена и повторение операций, экспорт в различные файловые форматы.

4.3.5 Сравнение графических библиотек

При сравнении графических библиотек было рассмотрено соответствие каждой библиотеки ряду требований. Наиболее важными требованиями являются следующие: кроссплатформенность, совместимость с Java 8 [3], встраивание в среду Eclipse, реализация графического редактора и палитры инструментов, поддержка слоев и координатной сетки, поддержка библиотеки разработчиками.

Сравнение рассмотренных графических библиотек подробно представлено в таблице 1.

Таблица 1. Сравнение графических библиотек

	GEF	JGraphX	LibGDX	Mica
Кроссплатформенность	+	+	+	+
Совместимость с Java 8	+	+	+	+
Встраивание в среду Eclipse	+	-	+	-
Реализация графического редактора	+	-	-	+
Реализация палитры инструментов	+	-	-	+
Поддержка слоев	+	-	+	+
Поддержка координатной сетки	+	+	+	+
Поддержка 2D	+	+	+	+
Поддержка 3D	-	-	+	-
Поддержка анимации	-	-	+	-
Дата последнего обновления	25/09/15	02/06/16	16/05/16	2004

На основании проведенного сравнения была выбрана графическая Java-библиотека GEF, так как удовлетворяет всем необходимым требованиям.

5 Рабочий этап

На рабочем этапе проектирования подсистемы были реализованы разработанные на предыдущих этапах схемы и концепции:

- алгоритм подсистемы процессно-ориентированного подхода;
- алгоритмы блоков процессно-ориентированного подхода;
- средство визуального программирования моделей процессного подхода;
- графические блоки подсистемы;
- алгоритм конвертации графических блоков;
- трассировка прогона модели процессного подхода.

5.1 Алгоритм подсистемы процессно-ориентированного подхода

Для реализации работы процессного подхода в рамках общего алгоритма симулятора Rao X в основной цикл моделирования был добавлен код выполнения подсистемы процессного подхода:

```
ProcessStatus processStatus = INSTANCE.processManager.scan();
if (processStatus == ProcessStatus.SUCCESS) {
     notifyChange(ExecutionState.STATE_CHANGED);
     continue;
} else if (processStatus == ProcessStatus.FAILURE) {
     return stop(SimulationStopCode.RUNTIME_ERROR);
               }
```

Реализация основного алгоритма подсистемы была выполнена в методе *scan()* класса *Process.java*:

```
public ProcessStatus scan() {
     boolean needCheckAgain = false;
     for (Block block : blocks) {
          BlockStatus blockStatus = block.check();
          if (blockStatus == BlockStatus.SUCCESS)
```

```
                return ProcessStatus.SUCCESS;
            if (blockStatus == BlockStatus.CHECK_AGAIN)
                needCheckAgain = true;
        }
        return needCheckAgain ? ProcessStatus.FAILURE :
ProcessStatus.NOTHING_TO_DO;
    }
```

В зависимости от возвращаемых статусов алгоритм может сообщить об ошибке, продолжить или завершить сканирование.

Полный код класса *Process.java* представлен в приложении А.

5.2 Алгоритмы блоков подсистемы

Все классы блоков подсистемы реализуют интерфейс *Block*:

```
public interface Block {
    public BlockStatus check();
}
```

В методе *check()* описывается алгоритм работы блока.

5.2.1 Блок Generate

При создании блока Generate в конструкторе требуется задать интервал создания транзактов и запланировать событие создания первого транзакта:

```
public Generate(Supplier<Double> interval) {
    this.interval = interval;
    Simulator.pushEvent(new GenerateEvent(interval.get()));
}
```

Для реализации алгоритма блока Generate вызывается метод *pushEvent()*, с помощью которого планируется событие типа *GenerateEvent*, реализующее интерфейс *Event:*

```
private class GenerateEvent extends Event {
        public GenerateEvent(double time) {
```

```
            this.time = time;
        }
        @Override
        public String getName() {
            return null;
        }
        @Override
        public void execute() {
            ready = true;
        }
    }
```

При выполнении запланированного события типа *GenerateEvent* статусу готовности присваивается значение *true*.

5.2.2 Блок Terminate

В методе *check()* блока Terminate происходит вызов метода *eraseTransact*, который удаляет текущий транзакт из системы:

```
    @Override
    public BlockStatus check() {
        Transact currentTransact = inputDock.pullTransact();
        if (currentTransact == null)
            return BlockStatus.NOTHING_TO_DO;
        Simulator.getDatabase().addProcessEntry(ProcessEntryType.
TERMINATE, currentTransact.getNumber(), null);
        Transact.eraseTransact(currentTransact);
        return BlockStatus.SUCCESS;
    }
```

5.2.3 Блок Queue

Для создания блока Queue в конструкторе необходимо задать емкость и дисциплину очереди. В зависимости от заданной разработчиком дисциплины создается объект очереди транзактов.

```
public Queue(int capacity, Queueing queueing) {
     this.capacity = capacity;
     switch (queueing) {
     case FIFO:
          queue = new FIFOQueue();
          break;
     case LIFO:
          queue = new LIFOQueue();
          break;
     }
}
```

Пример определения методов объекта очереди, реализующего дисциплину LIFO:

```
@Override
public boolean offer(Transact transact) {
     return queue.offerLast(transact);
}
@Override
public Transact remove() {
     return queue.removeLast();
}
@Override
public Transact peek() {
     return queue.peekLast();
}
@Override
public int size() {
     return queue.size();
}
```

5.2.4 Блоки Seize и Release

Так как пользователю предоставляется возможность выбрать, какой параметр ресурса и какие значения этого параметра отвечают за состояние "свободен-занят", то на этапе создания блока формируются объекты типов *Runnable* и *Supplier<Boolean>*. При вызове объекта *Runnable* происходит изменение состояния ресурса "свободен-занят". *Supplier<Boolean>* при вызове возвращает текущее значение состояния ресурса.

Конструктор блока Seize:

```
public Seize(Resource resource, Runnable seizeResource,
Supplier<Boolean> checkResourceFree) {
     this.resource = resource;
     this.seizeResource = seizeResource;
     this.checkResourceState = checkResourceFree;
}
```

Использование объекта *Runnable* в блоке Seize*:*

```
seizeResource.run();
```

Использование объекта *Supplier<Boolean>* в блоке Seize:

```
if (!checkResourceState.get())
     return BlockStatus.NOTHING_TO_DO;
```

Использование объекта *Supplier<Boolean>* в блоке Release:

```
if (!checkResourceState.get())
          throw new ProcessException("Attempting to release
unlocked resource");
```

5.2.5 Блок Hold

При создании блока Hold необходимо задать длительность обработки транзакта:

```
public Hold(Supplier<Double> duration) {
     this.duration = duration;
}
```

Для реализации алгоритма блока Hold необходимо создавать события типа *HoldEvent*. При выполнении события *HoldEvent* происходит проверка выходного узла с последующим выводом сообщения об ошибке, если узел занят предыдущим транзактом:

```
@Override
public void execute() {
     if (!transactStorage.pushTransact(transact))
          throw new ProcessException("Transact collision in
Hold block");
     addHoldEntryToDatabase(transact, HoldAction.OUT);
}
```

5.2.6 Блок SelectPath

Особенностью алгоритма блока SelectPath является назначение выходных узлов в зависимости от условия, заданного разработчиком модели:

```
if (condition.get()) {
     storage = trueOutputTransactStorage;
     data.put((byte) SelectPathOutputs.TRUE.ordinal());
} else {
     storage = falseOutputTransactStorage;
     data.put((byte) SelectPathOutputs.FALSE.ordinal());
}
```

В конструкторе блока SelectPath задается условие ветвления модели. При работе с вероятностью объектом условия является объект типа double, при работе с функцией, написанной разработчиком модели — объект типа *Supplier<Boolean>*.

Конструктор с заданием вероятности:

```
public SelectPath(Supplier<Boolean> condition) {
     this.condition = condition;
}
```

5.3 Графический редактор моделей

Графический редактор моделей процессного подхода представляет собой средство визуального программирования, описанное классом *ProcessEditor.java*, в котором реализованы следующие функции:

- создание и конфигурация палитры инструментов;
- сохранение и чтение файла;
- вызов валидации модели;
- конфигурация слоев модели.

Полный код класса *ProcessEditor.java* представлен в приложении Б.

5.3.1 Палитра инструментов

Палитра инструментов представляет собой объект типа *PaletteRoot*, для конфигурации которого необходимы объекты следующих типов:

- *PaletteGroup* — определяет группы инструментов;
- *PaletteSeparator* — разделитель палитры;
- *Entry* — инструменты палитры: связи, блоки, инструменты выделения.

Добавление в палитру инструмента создания связей:

```
PaletteGroup connectionGroup = new
PaletteGroup("Connection");
root.add(connectionGroup);
ConnectionCreationToolEntry connections = new
ConnectionCreationToolEntry("Connection", "Create Connections",
        new ConnectionCreationFactory(), null, null);
             connectionGroup.add(connections);
```

5.3.2 Сохранение и чтение файла

При сохранении объектов модели в файл вызывается метод *writeModelToFile()*, в котором происходит работа с объектами типа *ByteArrayOutputStream*:

```
ByteArrayOutputStream outputStream = new
ByteArrayOutputStream();
writeToOutputStream(outputStream);
IFile file = ((IFileEditorInput) getEditorInput()).getFile();
file.setContents(new
ByteArrayInputStream(outputStream.toByteArray()), true, false,
monitor);                getCommandStack().markSaveLocation();
```

В вызываемом методе *writeToOutputStream()* происходит запись модели в объект *ObjectOutputStream:*

```
ObjectOutputStream objectOutputStream = new
ObjectOutputStream(outputStream);
objectOutputStream.writeObject(getModel());
objectOutputStream.close();
```

При чтении модели из файла происходит считывание *ProcessModelNode* из объекта типа *ObjectInputStream*:

```
InputStream inputStream = file.getContents(false);
ObjectInputStream objectInputStream = new
ObjectInputStream(inputStream);
ProcessModelNode model = (ProcessModelNode)
objectInputStream.readObject();
objectInputStream.close();
```

5.3.3 Вызов валидации модели

При вызове валидации модели в графическом редакторе происходит итерация по блокам модели:

```
for (Node node : model.getChildren()) {
```

```
        if (node instanceof BlockNode) {
             ((BlockNode) node).validateProperty(file);
        }
```

Метод *validateProperty()* является уникальным для каждого типа графических блоков модели.

5.3.4 Конфигурация слоев модели

В объекте типа *LayeredPane* хранятся все необходимые слои модели:

- фоновый слой;
- слой координатной сетки;
- слой блоков модели;
- слой связей модели.

Создание слоя связей модели:

```
LayeredPane layers = new LayeredPane();
ConnectionLayer connectionLayer = new ConnectionLayer();
          connectionLayer.setAntialias(SWT.ON);
layers.add(connectionLayer, CONNECTION_LAYER);
```

5.4 Графические блоки

Каждый из графических блоков подсистемы процессного подхода представлен тремя классами:

- *Node* — параметры, валидация и особенности конвертации блока;
- *Figure* — отображение блока;
- *EditPart* — связующее звено между *Node* и *Figure*.

Для всех блоков подсистемы класс *EditPart* имеет одинаковое содержание:

```
IFigure figure = new BlockFigure();
     return figure;
}
```

5.4.1 Блок Generate

Блок *Generate* имеет один выход и не имеет входа, что отображено в создании узлов графического блока:

```
ConnectionAnchor outputConnectionAnchor = new
ConnectionAnchor(getShape());
outputConnectionAnchors.add(outputConnectionAnchor);
connectionAnchors.put(GenerateNode.DOCK_OUT,
outputConnectionAnchor);
```

Параметром блока является интервал модельного времени между событием создания транзактов. При валидации данного параметра осуществляется проверка соответствия типа значения типу *double*:

```
try {
     Double.valueOf(interval);
} catch (NumberFormatException e) {
     createProblemMarker(file, "Wrong interval",
IMarker.SEVERITY_ERROR);
}
```

Внешний вид блока представлен треугольником, в вершине которого расположен выходной узел.

5.4.2 Блок Terminate

Для блока *Terminate* в классе *TerminateFigure.java* создается один выходной узел в соответствии с требованиями алгоритма работы блока:

```
ConnectionAnchor inputConnectionAnchor = new
ConnectionAnchor(getShape());
inputConnectionAnchors.add(inputConnectionAnchor);
connectionAnchors.put(TerminateNode.DOCK_IN,
inputConnectionAnchor);
```

Визуализацией блока *Terminate* является треугольник, в вершине которого расположен входной узел, симметрично блоку *Generate*.

5.4.3 Блок Queue

Разработчик имитационной модели для блока *Queue* задает два параметра: емкость и дисциплину очереди. Дисциплина очереди имеет стандартное значение FIFO:

```
private Queueing queueing = Queueing.FIFO;
```

Для значения емкости очереди предусмотрена проверка:

```
try {
     Double.valueOf(capacity);
} catch (NumberFormatException e) {
     createProblemMarker(file, "Wrong capacity",
IMarker.SEVERITY_ERROR);
}
```

Если же поле ввода параметра осталось пустым, емкость принимает значение *Integer.MAX_VALUE.*

В качестве графического отображения блока было использовано обозначение очереди, использующееся в описании СМО. Визуализация блока *Queue*:

```
final int partitionWidth = border;
int partitionIndex = 0;
while (true) {
     final int left = bounds.right() - partitionWidth -
partitionIndex * partitionWidth * 2;
     if (left < bounds.x + bounds.width / 5)
          break;
     graphics.fillRectangle(left, bounds.y + border,
partitionWidth, bounds.height - border * 2);
     ++partitionIndex;
}
```

5.4.4 Блоки Seize и Release

Блоки захвата и освобождения ресурса имеют одинаковые параметры, реализованные в классе *BlockNodeWithResource.java*. Разработчику модели необходимо выбрать из выпадающих списков имя и состояние ресурса. Выпадающие списки формируются на основе текстового файла имитационной модели:

```
     for (IResource resource :
BuildUtil.getAllFilesInProject(project, "rao")) {
     String raoFileName = resource.getName();
     raoFileName = raoFileName.substring(0,
raoFileName.lastIndexOf("."));
     String modelClassName = project.getName() + "." +
raoFileName;
     URI uri = BuildUtil.getURI(resource);
     Resource loadedResource = resourceSet.getResource(uri, true);
     if (loadedResource == null)
          continue;
     List<ResourceDeclaration> resourceDeclarations =
SerializationConfigurator
               .filterAllContents(loadedResource.getAllContents(),
ResourceDeclaration.class);
     for (ResourceDeclaration resourceDeclaration :
resourceDeclarations)
          resources.put(modelClassName + "." +
resourceDeclaration.getName(), resourceDeclaration);
     List<ResourceType> types =
SerializationConfigurator.filterAllContents(loadedResource.getAllC
ontents(),ResourceType.class);
     for (ResourceType type : types)
          resourceTypes.put(modelClassName + "." + type.getName(),
type);
     }
```

При валидации блоков осуществляется проверка наличия ресурса с заданным именем:

```
if (!getResourcePropertyInfos().contains(resourceInfo))
     createProblemMarker(file, "Wrong resource",
IMarker.SEVERITY_ERROR);
```

Блоки *Seize* и *Release* отображены взаимно симметричными усеченными окружностям.

5.4.5 Блок Hold

Для блока задержки транзактов необходимо создание одного входа и одного выхода:

```
ConnectionAnchor inputConnectionAnchor = new
ConnectionAnchor(getShape());
inputConnectionAnchors.add(inputConnectionAnchor);
connectionAnchors.put(HoldNode.DOCK_IN,
inputConnectionAnchor);

ConnectionAnchor outputConnectionAnchor = new
ConnectionAnchor(getShape());
outputConnectionAnchors.add(outputConnectionAnchor);
connectionAnchors.put(HoldNode.DOCK_OUT,
outputConnectionAnchor);
```

Задаваемый параметр, отображающий интервал задержки транзакта, должен соответствовать типу *double*:

```
try {
     Double.valueOf(duration);
} catch (NumberFormatException e) {
     createProblemMarker(file, "Wrong duration",
IMarker.SEVERITY_ERROR);
}
```

Внешний вид блока представлен циферблатом на зеленом фоне.

5.4.6 Блок SelectPath

Особенностью блока *SelectPath* является наличие двух выходов:

```
ConnectionAnchor trueOutputConnectionAnchor = new
ConnectionAnchor(getShape());
outputConnectionAnchors.add(trueOutputConnectionAnchor);
connectionAnchors.put(SelectPathNode.DOCK_TRUE_OUT,
trueOutputConnectionAnchor);

ConnectionAnchor falseOutputConnectionAnchor = new
ConnectionAnchor(getShape());
outputConnectionAnchors.add(falseOutputConnectionAnchor);
connectionAnchors.put(SelectPathNode.DOCK_FALSE_OUT,
falseOutputConnectionAnchor);
```

Параметрами блока являются режим работы, вероятность и имя функции. Режим работы имеет стандартное значение *PROBABILITY*.

Для вероятности необходима проверка типа значения и попадание в интервал от 0 до 1:

```
boolean valid = false;
try {
     double probability = Double.valueOf(this.probability);
     valid = 0 <= probability && probability <= 1;
} catch (NumberFormatException e) {
}
if (!valid)
     createProblemMarker(file, "Wrong probability",
IMarker.SEVERITY_ERROR);
```

Имя функции разработчик модели выбирает из выпадающего списка, который формируется на основании *rao*-файла модели по аналогии со списком ресурсов.

5.5 Конвертация блоков

Алгоритм конвертации блоков из графического представления в Java-объекты описан в классе *BlockConverter.java*. Перевод происходит с помощью объектов типа *BlockConverterInfo*, которые хранят в себе входные и выходные узлы блока.

При конвертации происходит итерация по всем блокам модели. На каждом шаге создаются блоки, проверяется наличие связей у текущего блока с последующим созданием связей.

Все созданные блоки добавляются в список, чтобы избежать повторного создания:

```
blocks.add(sourceBlockInfo.block);
```

Если блок не может быть создан, происходит возврат сообщения об ошибке пользователю:

```
if (!sourceBlockInfo.isSuccessful)
     throw new
ProcessParsingException(sourceBlockInfo.errorMessage);
```

Полный код класса *ProcessEditor.java* представлен в приложении В.

5.6 Трассировка процессного подхода

Трассировка прогона имитационной модели процессного подхода является отображением результатов моделирования.

При попадании транзакта в блок формируется строка вывода состояния, уникальная для каждого типа блока. Общая часть строки представлена значением модельного времени, именем блока и номером транзакта. Для блоков *Queue*, *SelectPath*, *Seize* и *Release* в строку попадают значения их состояний или параметров.

Формирование строки вывода:

```
     new TraceOutput(TraceType.PROCESS, new
StringJoiner(delimiter).add(TraceType.PROCESS.toString()).add(time
).add(blockName).add(transactIndex).add(parseProcessData(data,
entryType)).getString());
```

Формирование строки вывода в зависимости от типа блока:

```
     switch (entryType) {
          case GENERATE:
          case TERMINATE:
               return "";
          case SELECT_PATH:
               SelectPathOutputs output =
SelectPathOutputs.values()[data.get()];
               String outputName = output.getString();
               return outputName;
          case HOLD:
               HoldAction holdAction =
HoldAction.values()[data.get()];
               String holdActionDescription =
holdAction.getString();
               return holdActionDescription;
          case QUEUE:
               QueueAction queueAction =
QueueAction.values()[data.get()];
               String queueActionDescription =
queueAction.getString();
               return queueActionDescription;
          case SEIZE:
          case RELEASE:
               int resourceTypeIndex = data.getInt();
               int resourceIndex = data.getInt();
               String resourceName =
Simulator.getStaticModelData().getResourceName(resourceTypeIndex,
resourceIndex);
               return resourceName;
          default:
```

```
                throw new TracerException("Unexpected process entry
type: " + entryType);
            }
```

Полный алгоритм формирования строк трассировки представлен на рисунке 8.

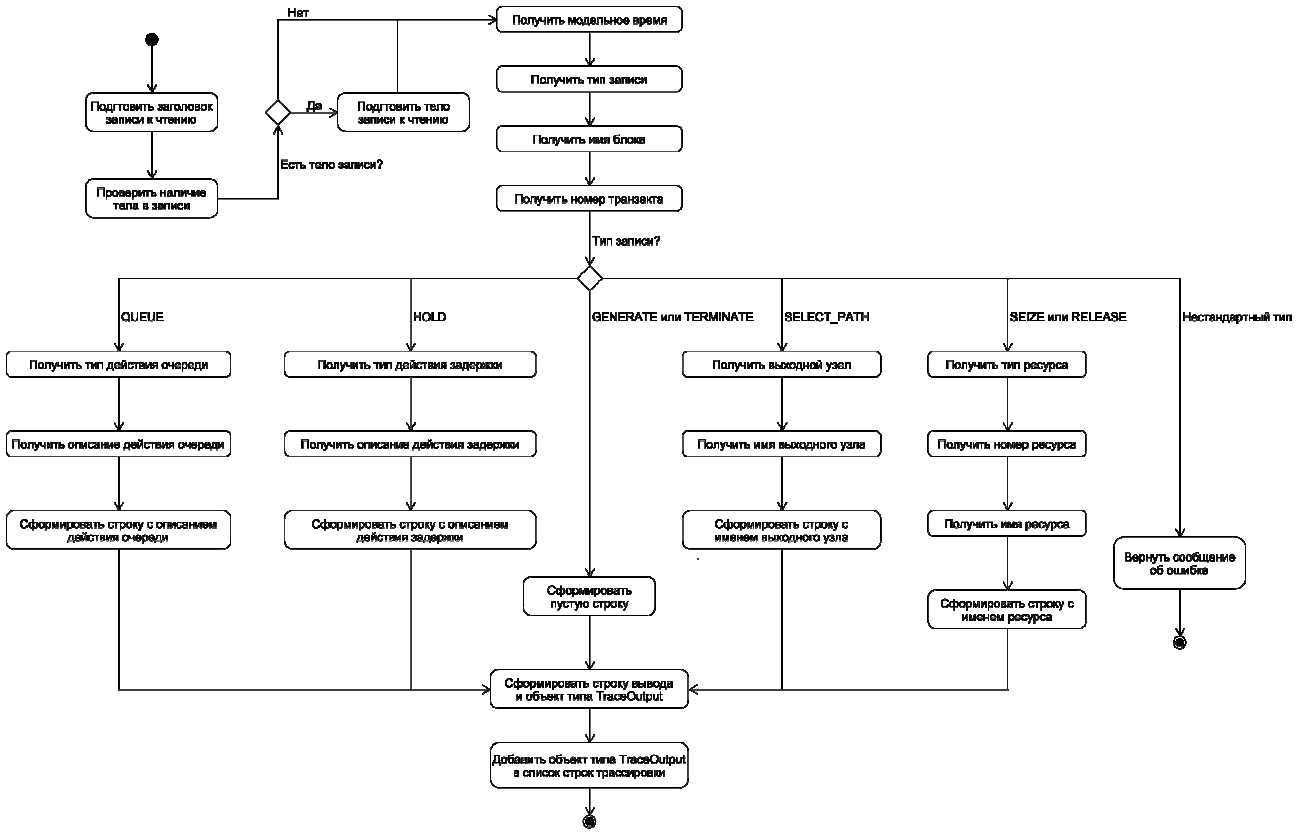

Рис. 8. Алгоритм формирования строк трассировки

6 Исследовательская часть

В рамках исследовательской части дипломного проекта были рассмотрены операторы языка GPSS[14] в реализации среды разработки GPSS World[15].

6.1 Операторы языка GPSS

Была поставлена задача оценить частоту использования различных блоков в текстах моделей процессно-ориентированного подхода языка GPSS[14]. В качестве исходных данных были использованы 87 моделей, включенные в среду GPSS World[15] в качестве примеров. Проводились замеры, на основании которых была построена гистограмма. Были выявлены наиболее часто используемые блоки и сделаны выводы о целесообразности реализации каждого из блоков в подсистеме процессного подхода Rao X.

6.1.1 Измерение частоты использования

На первом этапе измерения частоты использования операторов файлы моделей были переведены из бинарного в текстовое представление. Для перевода из бинарного в текстовое представление был написан скрипт для программы AutoHotkey[16]. Программа-скрипт итерирует по всем файлам с расширением *.gps* в указанной директории, открывает их с помощью среды GPSS World[15], копирует текст модели и сохраняет в текстовый файл.

На втором этапе в полученных текстовых файлах моделей процессного подхода с помощью программ grep[17], uniq[18] и sort[19] было подсчитано суммарное количество упоминаний операторов языка GPSS[14].

На рисунке 9 представлена гистограмма частоты использования операторов языка GPSS.

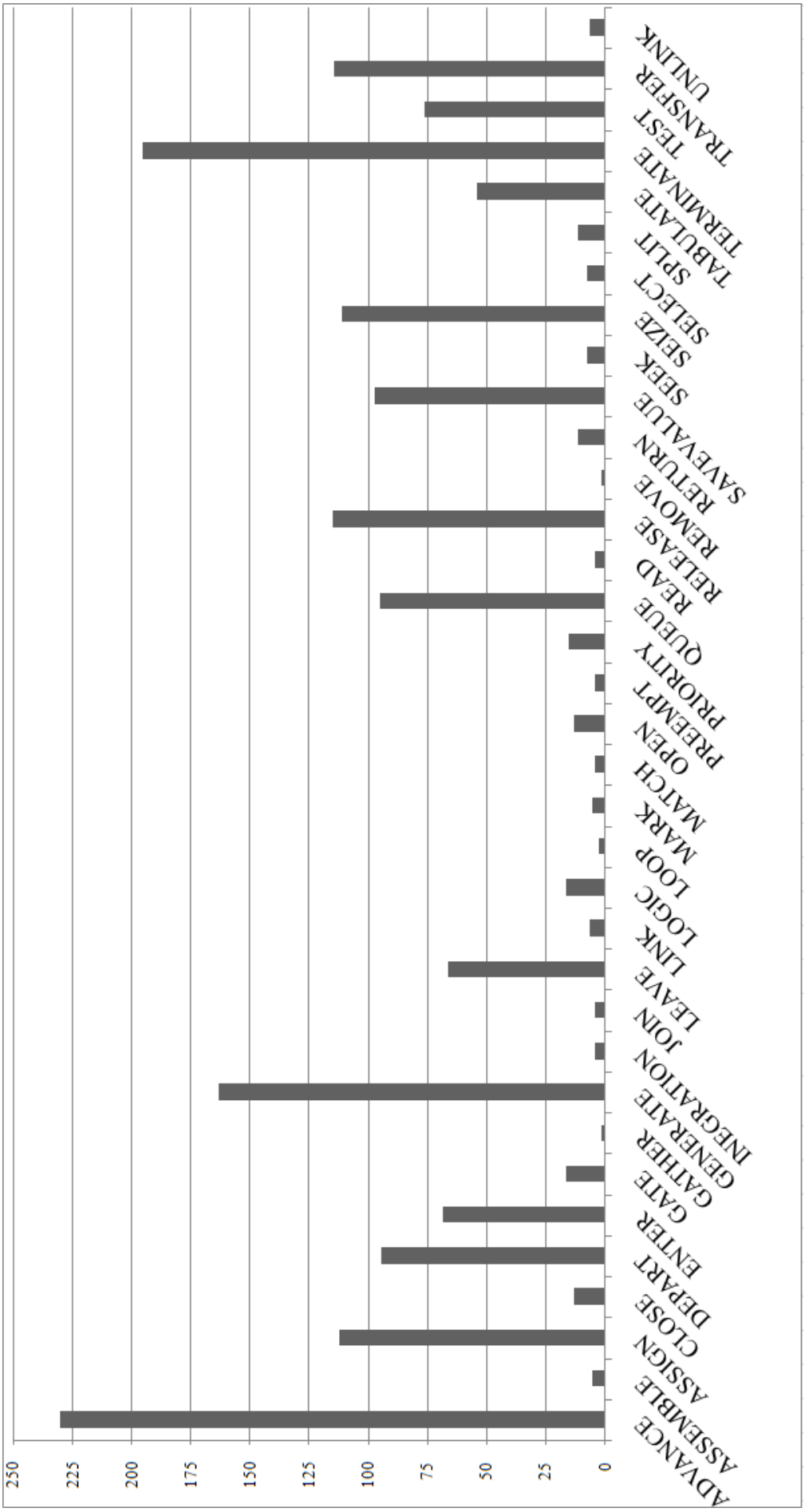

Рис. 9. Гистограмма частоты использования операторов языка GPSS

Из 53 операторов, представленных в среде разработки GPSS World, в гистограмме отмечены 25, как самые часто используемые в примерах моделей процессно-ориентированного подхода.

6.1.2 Результаты

Подробно были рассмотрены операторы, число вхождений которых суммарно составило более 75.

Операторы, которые не были реализованы:

- ASSIGN работает с параметрами транзактов, которые не были реализованы в рамках данного дипломного проекта;
- DEPART, QUEUE и SAVEVALUE являются операторами, собирающими статистику;
- TRANSFER работает с текстовыми метками, которые не могут быть реализованы в графическом подходе.

Операторы, которые были реализованы:

- ADVANCE — блок задержки транзактов Hold;
- GENERATE — блок создания транзактов Generate;
- TERMINATE — блок удаления транзактов Terminate;
- SEIZE — представлен блоками очереди Queue и захвата ресурсов Seize;
- RELEASE — блок освобождения ресурса Release;
- TEST — блок ветвления модели SelectPath.

.

7 Организационно-экономическая часть

7.1 Введение

В организационно-экономической части дипломного проекта был выполнен расчет затрат на разработку подсистемы процессного подхода системы имитационного моделирования Rao X. Произведены следующие расчеты:

- Оценка трудоемкости разработки подсистемы процессного подхода системы имитационного моделирования Rao X;
- Оценка продолжительности всех стадий разработки подсистемы процессного подхода;
- Оценка стоимости разработки подсистемы процессного подхода.

7.2 Организация и планирование процесса разработки ПП

Организация и планирование процесса разработки ПП предусматривает выполнение следующих работ:

- формирование состава выполняемых работ и группировка их по стадиям разработки;
- расчет трудоемкости выполнения работ;
- установление профессионального состава и расчет количества исполнителей;
- определение продолжительности выполнения отдельных этапов разработки;
- построение календарного графика выполнения разработки;
- контроль выполнения календарного графика.

Разработка программной продукции является сложным и длительным процессом, требующим выполнения большого числа разнообразных операций. Перечень стадий и состав работ при создании ПП может быть определен при

помощи ГОСТов ЕСПД. В таблице 2 приведен укрупненный состав работ при создании ПП.

Таблица 2. Укрупненный состав работ

Стадия разработки ПП	Состав выполняемых работ
Техническое задание	Постановка задач, выбор критериев эффективности. Разработка технико-экономического обоснования разработки. Выбор языков программирования. Предварительный выбор методов выполнения работы. Разработка календарного плана выполнения работ.
Эскизный проект	Предварительная разработка структуры входных и выходных данных. Разработка общего описания алгоритмов реализации решения задач. Разработка пояснительной записки. Консультации разработчиков постановки задач. Согласование и утверждение эскизного проекта.
Технический проект	Разработка алгоритмов решения задач. Разработка пояснительной записки. Согласование и утверждение технического проекта. Разработка структуры программы. Разработка программной документации и передача ее для включения в технический проект. Уточнение структуры, анализ и определение формы представления входных и выходных данных. Выбор конфигурации технических средств.
Рабочий проект	Комплексная отладка задач и сдача в опытную эксплуатацию. Разработка проектной документации. Программирование и отладка программ. Описание контрольного примера. Разработка программной документации. Разработка, согласование программы и методики испытаний. Предварительное проведение всех методов испытаний.
Внедрение	Подготовка и передача программной документации для сопровождения с оформлением соответствующего Акта. Передача программной продукции в фонд алгоритмов и программ. Проверка алгоритмов, корректировка документации после опытной эксплуатации ПП.

Трудоемкость разработки программной продукции зависит от ряда факторов, основными из которых являются следующие:

- степень новизны разрабатываемого программного комплекса;
- сложность алгоритма его функционирования;
- объем используемой информации;
- вид ее представления и способ обработки;
- уровень используемого алгоритмического языка программирования.

Более подробно вышеперечисленные факторы рассмотрены в таблице 3.

Таблица 3. Факторы трудоемкости программного продукта

Фактор	Группа
Степень новизны разрабатываемого программного комплекса	Группа В — разработка программной продукции, имеющей аналоги
Степени сложности алгоритма функционирования	1 группа сложности — программная продукция, реализующая оптимизационные и моделирующие алгоритмы
Вид представления исходной информации, способ ее контроля и структура выходных документов	Группа 11 — исходная информация представлена в форме документов, имеющих различный формат и структуру. Требуется учитывать взаимовлияние показателей в различных документах Группа 22 — требуется вывод на печать одинаковых документов, вывод информационных массивов на машинные носители

Трудоемкость разработки программной продукции $\tau_{пп}$ может быть определена как сумма величин трудоемкости выполнения отдельных стадий разработки ПП:

$$\tau_{пп} = \tau_{тз} + \tau_{эп} + \tau_{тп} + \tau_{рп} + \tau_{в}, \quad (1)$$

где $\tau_{тз}$ — трудоемкость разработки технического задания.

$\tau_{эп}$ — трудоемкость разработки эскизного проекта.

$\tau_{тп}$ — трудоемкость разработки технического проекта.

$\tau_{рп}$ — трудоемкость разработки рабочего проекта.

$\tau_{в}$ — трудоемкость внедрения разработанного ПП.

7.2.1 Расчет трудоемкости разработки технического задания

Трудоемкость разработки технического задания рассчитывается по формуле:

$$\tau_{тз} = T_{рз}^{з} + T_{рп}^{з}, \quad (2)$$

где $T_{рз}^{з}$ — затраты времени разработчика постановки задач на разработку ТЗ, чел.-дни;

$T_{рп}^{з}$ — затраты времени разработчика программного обеспечения на разработку ТЗ, чел.-дни.

Значения величин $T_{рз}^{з}$ и $T_{рп}^{з}$ рассчитывают по формулам:

$$T_{рз}^{з} = t_{з} \cdot K_{рз}^{з}, \quad (3)$$

$$T_{рп}^{з} = t_{з} \cdot K_{рп}^{з}, \quad (4)$$

где $t_{з}$ — норма времени на разработку ТЗ на программный продукт в зависимости от функционального назначения и степени новизны разрабатываемого ПП, чел.-дни. Принимаем $t_{з} = 37$чел.-дни (Перспективное, технико-экономическое, оперативное планирование, диспетчеризация; группа новизны В) [5, Табл. 2].

$K_{рз}^{з}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задач на стадии ТЗ.

$K_{рп}^{з}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком программного обеспечения на стадии ТЗ. Принимаем $K_{рз}^{з} = 0{,}65$ и $K_{рп}^{з} = 0{,}35$ — разработчик постановки задач и разработчик программного обеспечения являются одним лицом, поэтому считаем, что разработка проводится совместно [5, стр. 8].

Таким образом, трудоемкость разработки технического задания $\tau_{тз}$, чел.-дни:

$$\tau_{тз} = 37 \cdot (0{,}65 + 0{,}35) = \ 37 \text{ чел.-дни}$$

7.2.2 Расчет трудоемкости разработки эскизного проекта

Трудоемкость разработки эскизного проекта ПП $\tau_{эп}$ рассчитывают по формуле:

$$\tau_{эп} = \ T_{рз}^{Э} + T_{рп}^{Э}, \quad (5)$$

где $T_{рз}^{Э}$ — затраты времени разработчика постановки задач на разработку ЭП, чел.-дни;

$T_{рп}^{Э}$ — затраты времени разработчика программного обеспечения на разработку ЭП, чел.-дни.

Значения величин $T_{рз}^{Э}$ и $T_{рп}^{Э}$ рассчитывают по формулам:

$$T_{рп}^{Э} = t_{э} \cdot K_{рз}^{Э}, \quad (6)$$

$$T_{рп}^{Э} = t_{э} \cdot K_{рп}^{Э}, \quad (7)$$

где $t_{э}$ — норма времени на разработку ЭП программного продукта в зависимости от его функционального назначения и степени новизны, чел.-дни. Принимаем $t_{э} = 77$ чел.-дни (Перспективное, технико-экономическое, оперативное планирование, диспетчеризация; группа новизны В) [5, Табл. 2].

$K_{рз}^{Э}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задач на стадии ЭП.

$K_{рп}^{З}$ — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком программного обеспечения на стадии ЭП. Принимаем $K_{рз}^{Э} = 0{,}7$ и $K_{рп}^{Э} = 0{,}3$ — разработчик постановки задач и разработчик программного обеспечения являются одним лицом, поэтому считаем, что разработка проводится совместно [5, стр. 8].

$$\tau_{эп} = 77 \cdot (0{,}7 + 0{,}3) = \ 77 \text{ чел.-дни}$$

7.2.3 Расчет трудоемкости разработки технического проекта

Трудоемкость разработки технического проекта $\tau_{тп}$ зависит от функционального назначения ПП, количества разновидностей форм входной и выходной информации и определяется как сумма времени, затраченного разработчиком постановки задач и разработчиком программного обеспечения:

$$\tau_{тп} = \left(t_{рз}^{T} + t_{рп}^{T}\right) \cdot К_{в} \cdot К_{р}, \quad (8)$$

где $t_{рз}^{T}$, $t_{рп}^{T}$ — норма времени, затрачиваемого на разработку ТП разработчиком постановки задач и разработчиком программного обеспечения соответственно, чел.-дни. Принимаем $t_{рз}^{T} = 46$ чел.-дни, $t_{рп}^{T} = 10$ чел.-дни (Перспективное, технико-экономическое, оперативное планирование, диспетчеризация; количество разновидностей форм входной информации — 1, количество разновидностей форм выходной информации — 3) [5, Табл. 4].

$К_{в}$ — коэффициент учета вида используемой информации. Значение коэффициента $К_{в}$ определяют из выражения:

$$К_{в} = \frac{К_{п} \cdot n_{п} + К_{нс} \cdot n_{нс} + К_{б} \cdot n_{б}}{n_{п} + n_{нс} + n_{б}}, \quad (9)$$

где $К_{п}$, $К_{нс}$, $К_{б}$ — значения коэффициентов учета вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно. Принимаем $К_{п} = 1{,}00$; $К_{нс} = 0{,}72$; $К_{б} = 2{,}08$ (группа новизны — В) [5, Табл. 18].

$n_{п}$, $n_{нс}$, $n_{б}$ — количество наборов данных переменной, нормативно-справочной информации и баз данных соответственно. Принимаем $n_{п} = 1$; $n_{нс} = 0$; $n_{б} = 0$.

$$К_{в} = \frac{1{,}00 \cdot 1 + 0{,}72 \cdot 0 + 2{,}08 \cdot 0}{1 + 0 + 0} = 1$$

$К_{р}$ — коэффициент учета режима обработки информации. Принимаем $К_{р} = 1{,}26$ (РВ, группа новизны — В) [5, Табл. 17].

$$\tau_{тп} = (46 + 10) \cdot 1 \cdot 1{,}26 = 58{,}6 \text{ чел.} - \text{дни}$$

7.2.4 Расчет трудоемкости разработки рабочего проекта

Трудоемкость разработки рабочего проекта $\tau_{рп}$ зависит от функционального назначения ПП, количества разновидностей форм входной и выходной информации, сложности алгоритма функционирования, сложности контроля информации, степени использования готовых программных модулей, уровня алгоритмического языка программирования:

$$\tau_{рп} = К_к \cdot К_р \cdot К_я \cdot К_з \cdot К_{иа} \cdot \left(t^{Р}_{рз} + t^{Р}_{рп}\right), \qquad (10)$$

где $К_к$ — коэффициент учета сложности контроля информации. Принимаем $К_к = 1{,}07$ [5, Табл. 19].

$К_я$ — коэффициент учета уровня используемого алгоритмического языка программирования. Принимаем $К_я = 1{,}00$ [5, Табл. 20].

$К_з$ — коэффициент учета степени использования готовых программных модулей. Принимаем $К_з = 0{,}7$ [5, Табл. 21].

$К_{иа}$ — коэффициент учета вида используемой информации и сложности алгоритма ПП. Значение коэффициента $К_{иа}$ определяют из выражения:

$$К_{иа} = \frac{К'_п \cdot n_п + К'_{нс} \cdot n_{нс} + К'_б \cdot n_б}{n_п + n_{нс} + n_б}, \qquad (11)$$

где $К'_п$, $К'_{нс}$, $К'_б$ — значения коэффициентов учета сложности алгоритма ПП и вида используемой информации для переменной, нормативно-справочной информации и баз данных соответственно. Принимаем $К'_п = 1{,}20$; $К'_{нс} = 0{,}65$; $К'_б = 0{,}54$ (1 группа сложности алгоритма, группа новизны — В) [5, Табл. 22].

$$К_{иа} = \frac{1{,}20 \cdot 1 + 0{,}65 \cdot 0 + 0{,}54 \cdot 0}{1 + 0 + 0} = 1{,}20$$

$t^{Р}_{рз}$, $t^{Р}_{рп}$ — норма времени, затраченного на разработку РП на алгоритмическом языке высокого уровня разработчиком постановки задач и разработчиком программного обеспечения соответственно, чел.-дни. Принимаем $t^{Р}_{рз} = 14$ чел.-дни, $t^{Р}_{рп} = 86$ чел.-дни. [5, Табл. 23].

Таким образом, трудоемкость разработки рабочего проекта $\tau_{рп}$, чел.-дни:

$$\tau_{рп} = 1{,}07 \cdot 1{,}26 \cdot 1{,}00 \cdot 0{,}7 \cdot 1{,}20 \cdot (14 + 86) = 113{,}249 \text{ чел.} - \text{дни}$$

7.2.5 Расчет трудоемкости выполнения стадии "Внедрение"

Трудоемкость выполнения стадии "Внедрение" $\tau_в$ может быть рассчитана по формуле:

$$\tau_в = К_к \cdot К_р \cdot К_з \cdot \left(t_{рз}^{В} + t_{рп}^{В}\right), \qquad (12)$$

где $t_{рз}^{В}$, $t_{рп}^{В}$ — норма времени, затрачиваемого разработчиком постановки задач и разработчиком программного обеспечения соответственно на выполнение процедур внедрения ПП, чел.-дни. Принимаем $t_{рз}^{В} = 16$ чел.-дни, $t_{рз}^{В} = 19$ чел.-дни [5, Табл. 36].

Трудоемкость выполнения стадии "Внедрение" $\tau_в$, чел.-дни:

$$\tau_в = 1{,}07 \cdot 1{,}26 \cdot 0{,}7 \cdot (16 + 19) = 33{,}03 \text{ чел.} - \text{дни}$$

7.2.6 Расчет общей трудоемкости

Общая суммарная трудоемкость разработки программной продукции:

$$\tau_{пп} = 37 + 77 + 58{,}6 + 113{,}249 + 33{,}03 = 318{,}879 \text{ чел.} - \text{дни}$$

Принимаем:

$$\tau_{пп} = 319 \text{ чел.} - \text{дни}$$

Планирование и контроль хода выполнения разработки проводят по календарному графику выполнения работ. Продолжительность выполнения всех работ по этапам разработки ПП определяют из выражения:

$$T_i = \frac{\tau_i + Q}{n_i},$$

где τ_i — трудоемкость i-й работы, чел.-дни;

Q — трудоемкость дополнительных работ, выполняемых исполнителем, чел.-дни; n_i — количество исполнителей, выполняющих i-ю работу, чел.

Пусть разработка системы на всех стадиях ведется одним специалистом:

$$T_{\text{тз}} = \frac{37}{1} = 37 \text{ дн} \qquad T_{\text{рп}} = \frac{113{,}249}{1} = 113{,}249 \text{ дн}$$

$$T_{\text{эп}} = \frac{77}{1} = 77 \text{ дн} \qquad T_{\text{в}} = \frac{33{,}03}{1} = 33{,}03 \text{ дн}$$

$$T_{\text{тп}} = \frac{58{,}6}{1} = 58{,}6 \text{ дн} \qquad T = \frac{319}{1} = 319 \text{ дн}$$

7.3 Определение цены ПП

Процесс разработки сложной программной продукции сопровождается необходимостью решения социальных и экономических проблем. Одна из серьезных экономических проблем — определение стоимости ПП.

Если ПП рассматривается и создается как продукция производственно-технического назначения, допускающая многократное тиражирование и отчуждение от непосредственных разработчиков, то ее цена определяется по формуле:

$$\text{Ц} = \text{К} \cdot \text{С} + \text{П}_\text{р}, \qquad (13)$$

где С — затраты на разработку программной продукции (сметная себестоимость);

К — коэффициент учета затрат на изготовление опытного образца ПП как продукции производственно-технического назначения, $\text{К} = 1{,}1$;

$\text{П}_\text{р}$ — нормативная прибыль. Принимаем $\text{П}_\text{р} = 0$, так как ПП разрабатывается для внутренних нужд кафедры.

Затраты на разработку программной продукции могут быть представлены в виде сметы затрат, включающей в себя следующие статьи:

- амортизация специального оборудования;
- основная заработная плата;
- дополнительная заработная плата;
- отчисления на социальное страхование;

- накладные расходы.

7.3.1 Амортизация специального оборудования

В статье о затратах на специальное оборудование учитываются суммарные затраты на приобретение оборудования, необходимого для разработки данного ПП.

$$С_{со} = \sum_{i} \frac{Ц_{Бi} \cdot \alpha_i}{100 \cdot F_д} \cdot t_i, \qquad (14)$$

где $Ц_{Бi}$ — балансовая цена i-го вида оборудования, руб.

α_i — норма годовых амортизационных отчислений для оборудования i-го вида, %. Принимаем $\alpha_i = 20\%$ — требуемое оборудование представляет собой электронные цифровые вычислительные машины общего назначения [5, Табл. 50].

$F_д$ — действительный годовой фонд времени, ч. Принимаем $F_д = 2\,080$ ч — 5-ти дневная неделя, 8-и часовой рабочий день.

t_i — время использования i-го вида оборудования при выполнении данной разработки, ч.

Стоимость оборудования приведена в таблице 4.

Таблица 4. Стоимость оборудования

п/п	Наименование	Единица измерения	Кол-во	Цена за единицу, руб.	Сумма, руб.
1	ПЭВМ	шт	1	40 000	40 000

Цены указаны по состоянию на май 2016 года.

Затраты на оборудование, руб.:

$$С_{со} = \frac{40\,000 \cdot 20}{100 \cdot 2\,080} \cdot 2\,080 = 8000 \text{ руб.}$$

7.3.2 Основная заработная плата

В статью о заработной плате включается основная зарплата всех исполнителей, непосредственно занятых разработкой данного ПП, с учетом их должностного оклада и времени участия в разработке.

$$С_{зо} = \sum_i \frac{З_i}{d} \cdot \tau_i, \quad (15)$$

где $З_i$ — среднемесячный оклад i-го исполнителя, руб.

d — среднее количеств рабочих дней в месяце. Принимаем d = 21.

τ_i — трудоемкость работ, выполняемых i-ым исполнителем, чел.-дни. $\tau_i = 319$ чел.-дни.

Основная заработная плата:

$$С_{зо} = \frac{38\,000}{21} \cdot 319 = 577\,238 \text{ руб.}$$

7.3.3 Дополнительная заработная плата

В статье учитываются выплаты непосредственным исполнителям за время, непроработанное на производстве, в том числе: оплата очередных отпусков, компенсация за недоиспользованный отпуск, оплата льготных часов подросткам и др.

$$С_{зд} = С_{зо} \cdot \alpha_д, \quad (16)$$

где $\alpha_д$ — коэффициент отчислений на дополнительную зарплату, $\alpha_д = 0{,}2$.

Дополнительная заработная плата, руб.:

$$С_{зд} = 577\,238 \cdot 0{,}2 = 115\,448 \text{ руб.}$$

7.3.4 Отчисления на социальное страхование

В статье учитываются отчисления в бюджет социального страхования по установленному законодательством тарифу от суммы основной и дополнительной заработной платы.

$$С_{зд} = \alpha_{сс} \cdot (С_{зо} + С_{зд}), \quad (17)$$

где $\alpha_{сс}$ — коэффициент отчислений на социальное страхование. $\alpha_{сс} = 0{,}302$.

$$С_{зд} = 0{,}302 \cdot (577\,238 + 115\,448) = 209\,191 \text{ руб.}$$

7.3.5 Накладные расходы

В статье учитываются затраты на общехозяйственные расходы, непроизводительные расходы и расходы на управление.

$$С_{зд} = \alpha_{н} \cdot С_{зо}, \quad (18)$$

где $\alpha_{н}$ — коэффициент накладных расходов. $\alpha_{н} = 1{,}8$.

$$С_{зд} = 1{,}8 \cdot 577\,238 = 1\,039\,028 \text{ руб.}$$

7.3.6 Результаты расчетов затрат

Результаты расчетов затрат представлены в таблице 5.

Таблица 5. Результаты расчетов затрат на разработку ПП

№ п/п	Наименование статьи	Сметная стоимость, руб.
1	Амортизация специального оборудования и ПО	8 000
2	Основная заработная плата	577 238
3	Дополнительная заработная плата	115 448
4	Отчисления на социальное страхование	209 191
5	Накладные расходы	1 039 028
	Итого:	1 948 905

Цена программной продукции:

$$Ц = 1{,}1 \cdot 1\,948\,905 + 0 = 2\,143\,795.5 \text{ руб.}$$

7.4 Вывод

Результаты произведенного расчета:

- Суммарная трудоемкость разработки ПП составляет 319 чел.-дней.
- Продолжительность выполнения работ всех стадий составляет 319 раб.дней.
- Общие расходы на разработку программного продукта составляют 2 143 795.5 руб.

8 Мероприятия по охране труда и технике безопасности

8.1 Введение

В данном разделе дипломного проекта рассматриваются опасные и вредные факторы, действующие на разработчика моделей процессного подхода в системе имитационного моделирования Rao X. Описываются мероприятия по обеспечению безопасности, приводятся рекомендации по ограничению действия вредных и исключению воздействия опасных факторов.

8.2 Анализ опасных и вредных факторов

Опасные и вредные факторы, действующие на разработчика моделей, приведены в таблице 6.

Таблица 6. Опасные и вредные факторы согласно ГОСТ 12.0.003-74

№	Фактор	Опасный	Вредный
	Физические:		
1.	Повышенная запыленность воздуха рабочей зоны		+
2.	Повышенный уровень шума на рабочем месте		+
3.	Повышенный уровень вибрации		+
4.	Повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека	+	
5.	Повышенный уровень статического электричества		+
6.	Повышенный уровень электромагнитных излучений		+
7.	Отсутствие или недостаток естественного света		+
8.	Недостаточная освещенность рабочей зоны		+
	Психофизиологические:		
9.	Умственное перенапряжение		+
10.	Монотонность труда		+
11.	Эмоциональные перегрузки		+

8.2.1 Повышенная запыленность воздуха рабочей зоны

Пыль может оказывать на организм различные действия: фиброгенное, токсическое, раздражающее и т.д. Также может вызывать профессиональные заболевания легких — пневмокониозы, пылевые бронхиты и другие хронические заболевания органов дыхания.

Попадая в органы дыхания, пыль вызывает атрофию или гипертрофию слизистой верхних дыхательных путей, а задерживаясь в легких, приводит к развитию соединительной ткани в воздухообменной зоне и рубцеванию легких. Профессиональные заболевания, связанные с воздействием аэрозолей, пневмокониозы и пневмосклерозы, хронический пылевой бронхит занимают второе место по частоте среди профессиональных заболеваний в России.

В зависимости от природы пыли пневмокониозы могут быть различных видов: например, силикоз — наиболее частая и характерная форма пневмокониоза, развивающаяся при действии свободного диоксида кремния; силикатоз может развиваться при попадании в легкие аэрозолей солей кремниевой кислоты; асбестоз — одна из агрессивных форм силикатоза, сопровождающаяся фиброзом легких и нарушениями функций нервной и сердечнососудистой систем. [6, стр. 159]

Регламентирующие документы:

- ГОСТ 12.1.005-88 "Общие санитарно-гигиенические требования к воздуху рабочей зоны"
- ГН 2.2.5.1313-03 "Предельно допустимые концентрации (ПДК) вредных веществ в воздухе рабочей зоны. Гигиенические нормативы"

Предлагаемые меры по обеспечению безопасности:

- Проведение регулярного медицинского осмотра;
- Оборудование помещения системой вентиляции.

8.2.2 Повышенный уровень шума на рабочем месте

Причины возникновения:

- Работа вентиляторов ПЭВМ;
- Работа вентиляторов системы кондиционирования.

Влияние на человека:

Интенсивный шум способствует снижению внимания и увеличению числа ошибок при выполнении работы, исключительно сильное влияние оказывает шум на быстроту реакций, сбор информации и аналитические процессы, из-за шума снижается производительность труда и ухудшается качество работы.

Шум является заметным стрессовым фактором, способным вызвать срыв приспособительных реакций. Акустический стресс может приводить к разным проявлениям: от функциональных нарушений регуляции ЦНС до морфологически обозначенных дегенеративных деструктивных процессов в разных органах и тканях.

Шум оказывает влияние на весь организм человека: угнетает ЦНС, вызывает изменение скорости дыхания и пульса, способствует нарушению обмена веществ, возникновению сердечнососудистых заболеваний, гипертонической болезни, может приводить к профессиональным заболеваниям.

Регламентирующие документы:

- СН 2.2.4/2.1.8.562-96 "Шум на рабочих местах, в помещениях жилых, общественных зданий и на территории жилой застройки"
- ГОСТ 12.1.003—83* с дополнениями 1989г..

Предлагаемые меры по обеспечению безопасности:

- Акустическая обработка помещений;
- Своевременная замена неисправных узлов, создающих повышенные звуковые нагрузки;

- Контроль шумовой нагрузки;
- Размещение вне помещений с ПЭВМ шумящего оборудования (серверы, печатающие устройства и т.д.), уровни шума которого превышают нормативные.

8.2.3 Повышенный уровень вибрации

Причины возникновения:

- Работа с ПЭВМ.

Влияние на человека:

Воздействие вибрации ухудшает самочувствие, снижает производительность труда и приводит к тяжелому профессиональному заболеванию — виброболезни.

Особенно опасно совпадение частоты вибрации с собственной частотой колебания тела человека или отдельных органов.

В зависимости от способа передачи колебаний человеку вибрацию подразделяют на общую, передающуюся через опорные поверхности на тело сидящего или стоящего человека, и локальную, передающуюся через руки человека. Вибрация, воздействующая на ноги сидящего человека, на предплечья, контактирующие с вибрирующими поверхностями рабочих столов, также относится к локальной.

Для стоящего человека резонансными являются частоты 5-15 Гц, для сидящего — 4-6 Гц.

Собственные частоты:

- Желудок — 2 Гц
- Сердце — 4 Гц
- Печень — 4Гц
- Мозг — 6-7 Гц.

Вредность фактора зависит от частоты воздействующей вибрации:

- Низкочастотные — до 35 Гц. Поражаются нервы, мышцы, костный аппарат;
- Высокочастотные — 100-250 Гц. Поражаются кровеносные сосуды. Следствие: боль в органах, тошнота, рвота, нарушение сна, головокружение из-за нарушения работы вестибулярного аппарата.

При действии на организм общей вибрации страдает в первую очередь нервная система и анализаторы: вестибулярный, зрительный, тактильный. Вибрация является специфическим раздражителем для вестибулярного анализатора. [6, стр. 170]

Регламентирующие документы:

- ГОСТ 12.1.012-90 "ССБТ. Вибрационная безопасность. Общие требования"
- СН 2.2.4/2.1.8.566-96 "Производственная вибрация, вибрация в помещениях жилых и общественных зданий"

Предлагаемые меры по обеспечению безопасности:

- Виброакустические обработки помещений;
- Проведение инструктажа и обучения;
- Обеспечение медикаментозными средствами защиты.

8.2.4 Повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека

Причины возникновения:

- В помещении установлены устройства, требующие электропитания от сети переменного тока с номинальным напряжением 220В, к ним приложена необходимая проводка. Максимальное опасное напряжение в цепи, замыкание которой может пройти через тело человека, составляет 220В.

- Устройства имеют металлические корпуса, которые являются потенциально опасными местами, в результате соприкосновения с которыми может произойти замыкание электрической цепи через тело человека.

Влияние на человека:

Электрический ток, протекающий через организм человека, воздействует на него термически, электролитически и биологически.

- Термическое действие характеризуется нагревом тканей, вплоть до ожогов.
- Электролитическое — разложением органических жидкостей.
- Биологическое действие электрического тока проявляется в нарушении биоэлектрических процессов и сопровождается раздражением и возбуждением живых тканей и сокращением мышц, в том числе — сердечной и мышц, ответственных за дыхание.

Регламентирующие документы:

При гигиеническом нормировании ГОСТ 12.11.038—82* устанавливает предельно допустимые напряжения прикосновения и токи, протекающие через тело человека при аварийном режиме работы электроустановок постоянного и переменного тока частотой 50 и 400Гц.

Предлагаемые меры по обеспечение безопасности:

- Оборудование защитного заземления;
- Установка устройств защитного отключения;
- Инструктаж сотрудников по организации безопасной работы с электроприборами;
- Создание регламента по обслуживанию электроприборов с целью своевременного определения ненадежных соединений и возможных мест замыкания на корпус;
- Размещение предупреждающих наклеек.

8.2.5 Повышенный уровень статического электричества

Статическое электричество — совокупность явления, связанных с возникновением, сохранением и релаксацией свободного электрического заряда на поверхности и в объеме диэлектрических и полупроводниковых веществ, материалов изделий или на изолированных проводниках. Широкое использование в промышленности диэлектрических и полупроводниковых материалов значительно расширило область проявления статического электричества.

Влияние на человека:

Воздействие статического электричества на человека может проявляться в виде слабого длительно протекающего тока или в форме кратковременного разряда. Такой разряд вызывает у человека рефлекторное движение, что в ряде случаев может привести к попаданию работающего в опасную зону оборудования и закончиться несчастным случаем. Кроме того, электростатическое поле повышенной напряженности отрицательно влияет на организм человека, вызывая функциональные изменения со стороны центральной нервной, сердечнососудистой и других систем организма. Для ограничения вредного воздействия электростатического поля проводится его нормирование.

Регламентирующие документы:

- СанПиН 2.2.4.1191-03 "Электромагнитные поля в производственных условиях"
- СанПиН 2.2.2/2.4.1340-03 "Гигиенические требования к персональным электронно-вычислительным машинам и организацию работы"

8.2.6 Повышенный уровень электромагнитных излучений

Причина возникновения:

- Электропроводка и большое количество электроприборов на малой площади (несколько рабочих мест ПЭВМ).

Влияние на человека:

В электрическом поле атомы и молекулы, из которых состоит человеческое тело, поляризуются, а полярные молекулы ориентируются по направлению распространения электромагнитного поля. В электролитах, которыми являются жидкие составляющие тканей, крови, межклеточной жидкости и т.п., после приложения внешнего поля появляются ионные токи. Переменное электрическое поле вызывает нагрев тканей тела человека: переменная поляризация диэлектриков и появление токов проводимости.

Тепловой эффект является следствием поглощения энергии электромагнитного поля. Имеет место отражение электромагнитных волн от поверхности человеческого тела из-за изменения на этой границе волнового сопротивления среды. Поглощение энергии и возникновение ионных токов сопровождается специфическим воздействием на биологические ткани, поскольку нарушается тонкая структура электрических потенциалов и циркуляции жидкости в клетках и внутренних органах.

Переменное магнитное поле приводит к изменению ориентации магнитных моментов атомов и молекул. Этот эффект слабее вызываемого внешним электрическим полем, но он также небезразличен для организма.

Чем больше напряженность поля и чем больше время воздействия, тем описанные эффекты проявляются сильнее.

Регламентирующие документы:

- СанПиН 2.2.4.1191-03 "Электромагнитные поля в производственных условиях"

- СанПиН 2.2.2/2.4.1340-03 "Гигиенические требования к персональным электронно-вычислительным машинам и организацию работы.

Предлагаемые меры по обеспечению безопасности:

- Установка экранирующих защитных конструкций;
- Рациональное расположение излучающих устройств;
- Проведение инструктажа и обучения;
- Размещение различных плакатов, предупредительных сигналов, маркировки и краски, знаков безопасности.

8.2.7 Отсутствие или недостаток естественного света

Влияние на человека:

Отсутствие естественного света влияет на психику, эмоциональное состояние. Освещенность помещения, значительно отличающаяся от естественного, плохо влияет на функционирование зрительного аппарата человека вплоть до нарушения зрительной функции.

Естественный свет имеет высокую биологическую и гигиеническую ценность, так как обладает благоприятным для зрения человека спектральным составом и оказывает положительное воздействие на психологическое состояние человека - создает ощущение связи его с окружающим миром

Регламентирующие документы:

- СНиП 23-05-95 "Естественное и искусственное освещение"

Предлагаемые меры по обеспечению безопасности:

- Оборудование помещения искусственным освещением;
- Проведение инструктажа персонала.

8.2.8 Недостаточная освещенность рабочей зоны

Причины возникновения:

- Рабочее место располагается в помещении с искусственным освещением.

Влияние на человека:

При организации производственного освещения необходимо обеспечить равномерное распределение яркости на рабочей поверхности и окружающих предметах. Перевод взгляда с ярко освещенной на слабо освещенную поверхность вынуждает глаз переадаптироваться, что ведет к утомлению зрения и соответственно к снижению производительности труда. Для повышения равномерности естественного освещения больших помещений осуществляется комбинированное освещение. Светлая окраска потолка, стен и оборудования способствует равномерному распределению яркостей в поле зрения работающего.

Производственное освещение должно обеспечивать отсутствие в поле зрения работающего резких теней. Наличие резких теней искажает размеры и формы объектов различения и тем самым повышает утомляемость, снижает производительность труда. Особенно вредны движущиеся тени, которые могут привести к травмам. Тени необходимо смягчать.

Колебания освещенности на рабочем месте, вызванные резким изменением напряжения в сети, обусловливают переадаптацию глаза, приводя к значительному утомлению.

Недостаток освещенности влияет на функционирование зрительного аппарата человека вплоть до нарушения зрительной функции.

Регламентирующие документы:

- СанПиН 2.2.2/2.4.1340-03 "Гигиенические требования к персональным электронно-вычислительным машинам и организации работы"

Предлагаемые меры по обеспечению безопасности:

- Произвести расчет освещения в помещениях, скорректировать существующее освещение в соответствии с результатами;
- Проведение инструктажа персонала.

8.2.9 Умственное перенапряжение

Влияние на человека:

Длительная работа, требующая высоких интеллектуальных нагрузок может привести к утомлению или переутомлению. При умственном утомлении отмечается расстройство внимания, ухудшение памяти и мышления, ослабляется точность и координированность движений.

Регламентирующие документы:

- Р 2.2.2006-05 "Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда".

Предлагаемые меры по обеспечению безопасности:

- Регламентированные перерывы в работе;
- Проведение курсов по повышению квалификации работников;
- Проведение инструктажа персонала.

8.2.10 Монотонность труда

Причины возникновения:

- Однообразные, повторяющиеся монотонные действия.

Влияние на человека:

Монотонный труд вызывает изменения в функциональном состоянии ЦНС, что проявляется в удлинении латентного периода простой и сложной зрительно-моторной реакции, увеличении процента расторможенных

дифференцировок, замедлении способности к переключению внимания, снижению подвижности основных нервных процессов.

Регламентирующие документы:

- Р 2.2.2006-05 "Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда".

Предлагаемые меры по обеспечению безопасности:

- Регламентированные перерывы в работе;
- Проведение инструктажа персонала.

8.2.11 Эмоциональные перегрузки

Причины возникновения:

- Большой объем информации;
- Недостаток времени.

Влияние на человека:

Обостряет восприимчивость к воздействию вредных факторов окружающей среды. Нарушает обмен веществ, ведет к изнашиванию и старению организма.

Регламентирующие документы:

- Р 2.2.2006-05 "Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда".

Предлагаемые меры по обеспечению безопасности:

- Создание комфортного рабочего места;
- Регламентированные перерывы в работе;
- Проведение инструктажа персонала.

8.3 Расчет освещенности в помещении

В настоящем дипломном проекте был проведен расчет освещенности в помещении методом светового потока, учитывающий световой поток, отраженный от потолка и стен. [7, стр. 127] Целью расчета является определение потребной мощности электрической осветительной установки для создания в помещении заданной освещенности или, при известном числе и мощности ламп, определение ожидаемой освещенности на рабочей поверхности.

Параметры помещения:

Длина A = 10м, ширина B = 5м, высота H = 2,75м

Высота рабочей поверхности над полом h = 0,8м

Высота подвеса светильников T = 0,08м

Требования стандарта СанПиН 2.2.2/2.4.1340-03 "Гигиенические требования к персональным электронно-вычислительным машинам и организации работы" представлены в таблице 7.

Таблица 7. Требования к освещению на рабочих местах, оборудованных ПЭВМ

Наименование	Норма
Освещенность на рабочем столе	300-500лк
Освещенность на экране ПЭВМ	не выше 300лк
Блики на экране	не выше 40кд/м^2
Прямая блесткость источника света	200кд/м^2
Показатель ослепленности	не более 20
Показатель дискомфорта	не более 15
Отношение яркости:	
- между рабочими поверхностями	3:1-5:1
- между поверхностями стен и оборудования	10:1
Коэффициент пульсации	не более 5%

При выборе типа светильников предпочтение было отдано люминесцентным лампам, при учете следующих факторов:

- Обеспечивают высокий уровень освещенности при низком, по сравнению с лампами накаливания, энергопотреблении;
- Имеют больший срок службы, по сравнению с лампами накаливания.

Для снижения пульсаций используются светильники с установленными электронными пускорегулирующими аппаратами (ЭПРА).

Для расчета общего равномерного освещения при горизонтальной рабочей поверхности основным является метод светового потока (коэффициента использования), учитывающий световой поток, отраженный от потолка и стен. Световой поток лампы Fл (лм) группы ламп светильника при люминесцентных лампах рассчитывают по формуле:

$$Fл = \frac{Eн S z k}{N \eta}, \qquad (19)$$

где $Eн$ — нормированная минимальная освещенность, лк;

S — площадь освещаемого помещения, м^2;

Z — коэффициент минимальной освещенности, равный отношению $Eср/Eмин$, значения которого обычно находятся в пределах 1,1-1,5 (в среднем 1,2), для люминесцентных ламп принимаем 1,1;

k — коэффициент запаса, принимаемый в соответствии с [7, Табл. 3], принимаем 1,5;

N — число светильников в помещении;

η — коэффициент использования светового потока ламп. Значения коэффициента η определяют по таблицам, в зависимости от коэффициентов отражения светового потока от потолка и стен и показателя помещения i, определяемого из отношения:

$$i = \frac{AB}{Hp(A+B)}, \qquad (20)$$

где A и B — два характерных размера помещения;

Hp — высота светильников над расчетной поверхностью:

$$Hp = H - h - T, \quad (21)$$

Таким образом:

$$Hp = 2{,}75 - 0{,}8 - 0{,}08 = 1{,}87 \text{ м}$$

$$i = \frac{10 * 5}{1{,}87 * (10 + 5)} = 1{,}783$$

Потолок и стены помещения бетонные, представляют собой поверхность средней светлости. Принимаем коэффициент отражения 30%. Тогда для светильника типа ЛСП-01 коэффициент использования составит 57% [7, Табл. 4].

Примем количество светильников 6, однако, учитывая, что в каждом светильнике расположены две лампы, рассчитаем световой поток лампы как:

$$Fл = \frac{300 * 10 * 5 * 1{,}1 * 1{,}5}{6 * 2 * 0{,}57} = 3620 \text{ лм}$$

По [7, Табл. 5] подбираем лампу ЛДЦ80 со световым потоком в 3740лм. Отклонение от рассчитанного значения может составлять от -10% до +20%, в данном случае эта величина составляет 3,31%, что является допустимым.

Суммарная потребляемая осветительной системой мощность составляет 6*2*80 = 960 Вт. Учитывая оборудование светильников ЭПРА с КПД 95%, получим результирующую мощность в 1010 Вт.

Результаты расчета:

- Оборудование светильников ЭПРА;
- 6 светильников типа ЛСП-01;
- Лампа ЛДЦ80;
- Потребляемая мощность 1010 Вт.

8.4 Утилизация ПЭВМ

Общая технологическая схема обработки отходов ПЭВМ представлена на рис. 10.

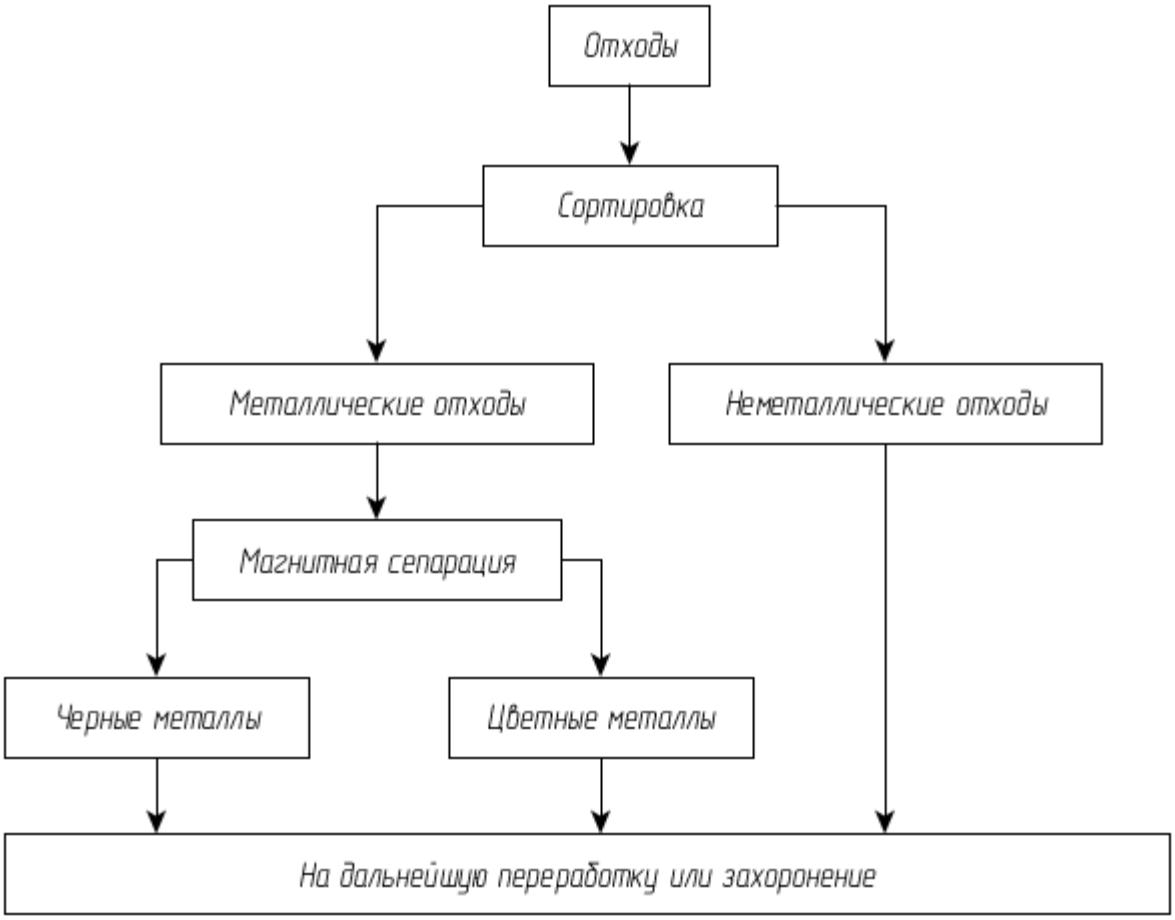

Рис. 10. Общая технологическая схема обработки отходов ПЭВМ

8.4.1 Утилизация неметаллических отходов ПЭВМ

Рассмотрим утилизацию неметаллических отходов, в данном случае, пластмасс. [8]

Из пластмассовых материалов изготавливаются следующие части ПЭВМ, периферийных устройств и расходных материалов:

- лицевые панели системного блока;

- корпуса и основания мониторов;
- клавиатуры, манипуляторы;
- носители информации.

Можно выделить следующие направления в разработке процессов и методов утилизации пластмассовых отходов ПЭВМ:

- прямая переработка отходов пластмасс ПЭВМ во вторичное сырье, материалы, изделия, включая и использование в различных композициях;
- термическое разложение с получением целевых продуктов;
- термическое обезвреживание с регенерацией выделяемой теплоты;
- разработка биоразрушаемых масс.

В наиболее типичном виде процесс переработки пластмасс, используемых в ПЭВМ, включает в себя следующие этапы, представленные на рис. 11.

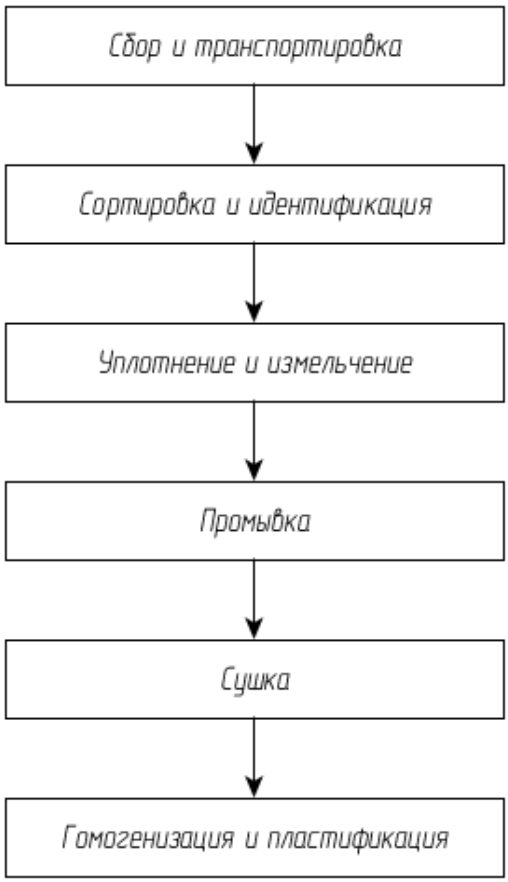

Рис. 11. Этапы процесса утилизации пластмассовых отходов ПЭВМ

На этапе сортировки и идентификации пластмассовые отходы ПЭВМ сортируют по видам термопласт — поливинилхлорид, полистирол, пенопластовая упаковка, полиэтилен — и подвергают механической переработке.

На этапе гомогенизации и пластификации пластмассовые отходы ПЭВМ подвергаются термическим методам обработки, которые считаются эффективными и перспективными: выход продукта составляет 85 % и более при расходуемых энергетических затратах менее 10 %.

Следует отметить, что до 80 % энергозатрат расходуется на предыдущих этапах (сбор – сортировка – измельчение – промывка – сушка).

К термическим методам утилизации пластмассовых отходов ПЭВМ относятся следующие:

- Пиролиз – термическое разложение органических продуктов при наличии или отсутствии кислорода;
- Гидролиз – метод, обратный пиролизу;
- Гликолиз – деструкция при высокой температуре и давлении в присутствии этиленгликоля и с участием катализаторов;
- Метаполиз – метод расщепления пластмасс с помощью металона.

Термические методы позволяют перерабатывать отходы ПЭВМ с относительно высоким уровнем загрязняющих веществ.

Степень загрязнения имеет два критерия:

- степень смешения с другими материалами;
- различия в составе собранного материала.

Например, в кабельной продукции содержание различных добавок колеблется от 50% до 60% с различными используемыми смесями и композициями.

Важнейшей характеристикой отходов пластмасс (как, впрочем, и самих пластмассовых изделий) является их энергетическая ценность. И по

химическому составу и по теплоте сгорания пластмассы подобны основным ископаемым топливам, природному газу, нефти, углям.

Однако прямая утилизация отходов пластмасс путем сжигания в энергетических установках, как правило, невозможна в связи с присутствием в них примесей, приводящих к образованию при сжигании токсичных соединений.

Одной из перспективных технологий переработки отходов пластмасс является их использование в металлургическом производстве в качестве источника энергии и восстановителей, прежде всего – в доменных печах.

Данный способ, во-первых, исключает выбросы суперэкотоксикантов, а во-вторых, позволяет даже в доменных печах среднего объема полностью утилизировать отходы крупных промышленных регионов. [8]

При изготовлении ПЭВМ, периферийных устройств, расходных и упаковочных материалов, разнообразие используемых пластмассовых изделий затрудняет отличие одного вида пластика от другого, особенно если процесс рециклирования проводится через длительное время (2-5 лет) после производства изделия.

Для решения данной проблемы были разработаны международные стандарты маркировки пластмассовых изделий, используемых в ПЭВМ. Сейчас наиболее широко применяется стандарт Международной организации по стандартизации – ISO.

8.4.2 Утилизация металлических отходов ПЭВМ

После окончания срока эксплуатации компьютерной техники образуется компьютерный лом, который подвергается подготовке к переработке с целью разделения по фракциям и извлечения различными способами черных, цветных и драгоценных металлов. [8]

Эффективность использования лома и отходов металла зависят от их качества. Загрязнение и засорение металлических отходов приводят к большим

потерям при переработке. Поэтому сбор, хранение и сдача металлических отходов регламентируются специальными стандартами:

- ГОСТ 2787-75*. "Лом и отходы черных металлов. Шихтовые. Классификация и технические требования"
- ГОСТ 1639-78*. "Лом и отходы цветных металлов. Общие требования"

Основными операциями первичной обработки металлических отходов являются сортировка, разделка и механическая обработка.

Сортировка заключается в разделении лома и отходов по видам металлов. Разделка состоит в удалении из лома неметаллических включений. Механическая обработка включает в себя рубку, резку, пакетирование и брикетирование (окусковывание механическим уплотнением на прессах, под молотком и других механизмах). [8]

На предприятиях, где образуется большое количество металлических отходов, организуются специальные цеха для утилизации вторичных металлов. Чистые однородные отходы с паспортом, удостоверяющим их химический состав, используют без предварительного металлургического передела.

Общая технологическая схема первичной обработки лома и отходов цветных металлов приведена на рис. 12.

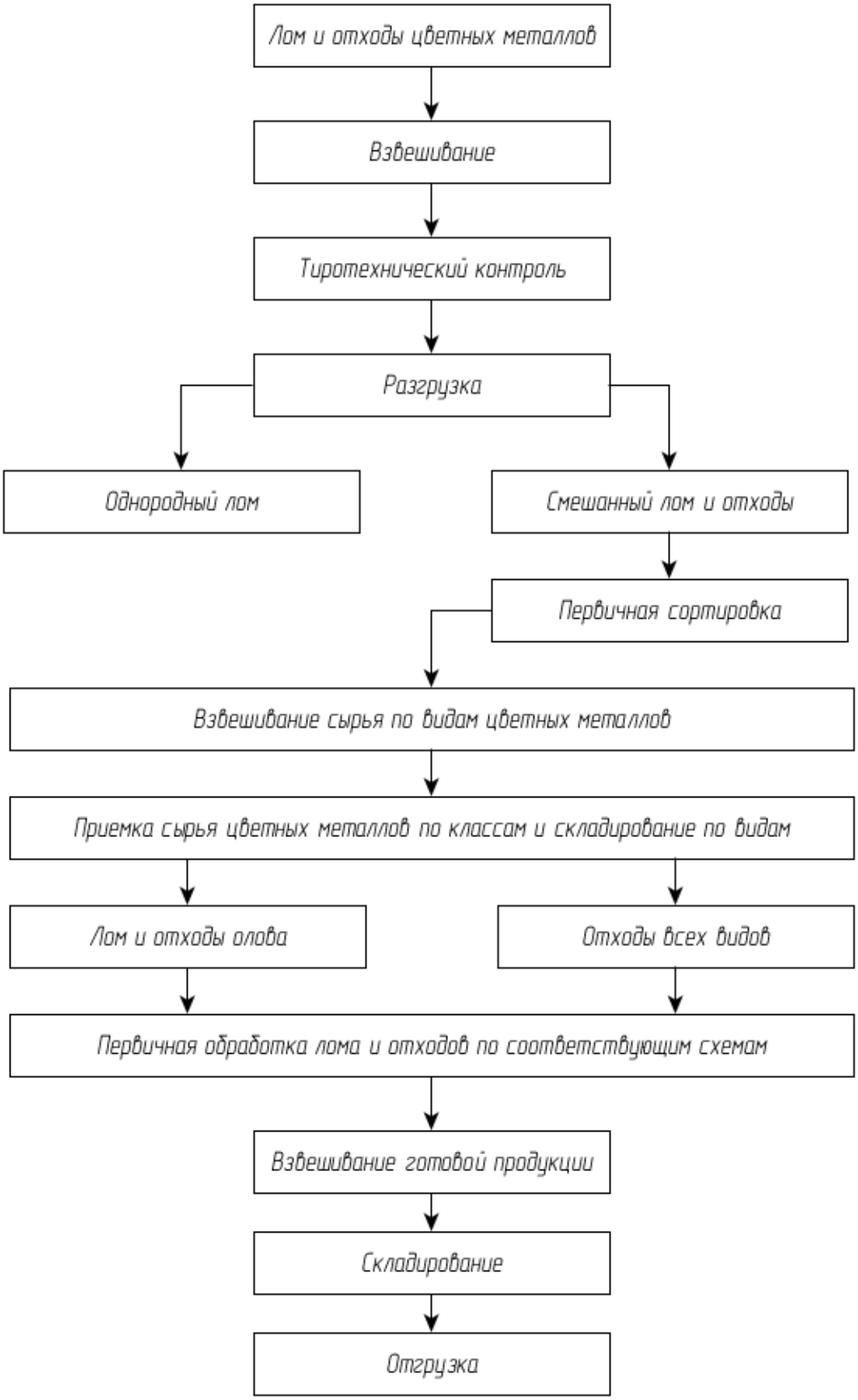

Рис. 12. Общая технологическая схема первичной обработки лома и отходов цветных металлов

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Емельянов В.В., Ясиновский С.И. Введение в интеллектуальное имитационное моделирование сложных дискретных систем и процессов. Язык РДО. — М.: Анвик, 1998. - 427 с., ил. 136.
2. Документация по языку РДО [Электронный ресурс] — http://raox.ru/docs/reference/base_types_and_functions.html
3. Прицкер А. Введение в имитационное моделирование и язык СЛАМ II: Пер. с англ. — М.: Мир, 1987. — 646 с., ил.
4. Java Platform, Standart Edition 7. API Specification [Электронный ресурс] — http://docs.oracle.com/javase/7/docs/api/
5. Арсеньев В.В., Сажин Ю.Б. Методические указания к выполнению организационно-экономической части дипломных проектов по созданию программной продукции. — М.: Изд-во МГТУ, 1994. — 52 с.
6. Безопасность жизнедеятельности: Учебник для вузов/С.В. Белов, А.В. Ильницкая, А.Ф. Козьяков и др.; Под общ. ред. С.В. Белова. 7-е изд., стер. — М.: Высш.шк., 2007. — 616 с.: ил.
7. Охрана труда в машиностроении: Ученик для машиностроительных вузов/Е.Я. Юдин, С.В. Белов, С.К. Баланцев и др.; Под ред. Е.Я. Юдина, С.В. Белова — 2-е изд., перераб. и доп. — М.: Машиностроение. — 1983, 432 с., ил.
8. Утилизация пластмассовых отходов ПЭВМ [Электронный ресурс] — http://studopedia.ru/14_131224_utilizatsiya-plastmassovih-othodov-pevm.html
9. Графическая библиотека JGraphX [Электронный ресурс] — https://github.com/jgraph/jgraphx
10. Графическая библиотека GEF [Электронный ресурс] — https://eclipse.org/gef/

11. Графическая библиотека Mica [Электронный ресурс] — http://gui.net/swfm/mica-classic/index.htm
12. Графическая библиотека LibGDX [Электронный ресурс] — https://libgdx.badlogicgames.com/
13. Среда разработки Eclipse [Электронный ресурс] - http://www.eclipse.org/
14. Язык моделирования GPSS [Электронный ресурс] - http://www.gpss.ru/
15. Среда разработки GPSS World [Электронный ресурс] — http://www.minutemansoftware.com/
16. Программа AutoHotkey [Электронный ресурс] - https://autohotkey.com/
17. Программа grep [Электронный ресурс] — http://www.gnu.org/software/grep/manual/grep.html
18. Программа uniq [Электронный ресурс] — https://www.gnu.org/software/coreutils/manual/html_node/uniq-invocation.html
19. Программа sort [Электронный ресурс] — https://www.gnu.org/software/coreutils/manual/html_node/sort-invocation.html

СПИСОК ИСПОЛЬЗОВАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Eclipse IDE for Java Developers Mars.1 Release (4.5.1)
2. openjdk version "1.8.0_40-internal"
3. UMLet v14.2
4. Inkscape v0.48.4
5. AutoCAD 2012
6. yEd Graph Editor v3.14.4
7. Microsoft® Office Word 2010
8. Microsoft® Office Excel 2010
9. Microsoft® Office Visio 2010
10. AutoHotkey v1.1.24.00
11. GNU grep v2.25
12. GNU uniq
13. GNU sort

ПРИЛОЖЕНИЕ А

Класс общего алгоритма процессного подхода

```
package ru.bmstu.rk9.rao.lib.process;

import java.util.ArrayList;
import java.util.List;

public class Process {

	public Process(List<Block> blocks) {
		this.blocks.addAll(blocks);
	}

	private final List<Block> blocks = new ArrayList<Block>();

	public ProcessStatus scan() {
		boolean needCheckAgain = false;
		for (Block block : blocks) {
			BlockStatus blockStatus = block.check();
			if (blockStatus == BlockStatus.SUCCESS)
				return ProcessStatus.SUCCESS;
			if (blockStatus == BlockStatus.CHECK_AGAIN)
				needCheckAgain = true;
		}

		return needCheckAgain ? ProcessStatus.FAILURE :
ProcessStatus.NOTHING_TO_DO;
	}

	public enum ProcessStatus {
		SUCCESS, FAILURE, NOTHING_TO_DO
	};

	public enum BlockStatus {
		SUCCESS, CHECK_AGAIN, NOTHING_TO_DO
	}
}
```

ПРИЛОЖЕНИЕ Б

Класс средства визуального программирования моделей процессного подхода

```
package ru.bmstu.rk9.rao.ui.process;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.EventObject;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.draw2d.ConnectionLayer;
import org.eclipse.draw2d.Figure;
import org.eclipse.draw2d.IFigure;
import org.eclipse.draw2d.Layer;
import org.eclipse.draw2d.LayeredPane;
import org.eclipse.draw2d.ScalableLayeredPane;
import org.eclipse.draw2d.StackLayout;
import org.eclipse.draw2d.geometry.Dimension;
import org.eclipse.draw2d.geometry.Rectangle;
import org.eclipse.gef.DefaultEditDomain;
import org.eclipse.gef.GraphicalViewer;
import org.eclipse.gef.KeyHandler;
import org.eclipse.gef.KeyStroke;
import org.eclipse.gef.dnd.TemplateTransferDragSourceListener;
import org.eclipse.gef.editparts.GridLayer;
import org.eclipse.gef.editparts.GuideLayer;
import org.eclipse.gef.editparts.ScalableRootEditPart;
import org.eclipse.gef.palette.CombinedTemplateCreationEntry;
import org.eclipse.gef.palette.ConnectionCreationToolEntry;
import org.eclipse.gef.palette.MarqueeToolEntry;
import org.eclipse.gef.palette.PaletteGroup;
import org.eclipse.gef.palette.PaletteRoot;
import org.eclipse.gef.palette.PaletteSeparator;
import org.eclipse.gef.palette.PanningSelectionToolEntry;
import org.eclipse.gef.ui.palette.FlyoutPaletteComposite;
import org.eclipse.gef.ui.palette.PaletteViewer;
import org.eclipse.gef.ui.palette.PaletteViewerProvider;
import org.eclipse.gef.ui.parts.GraphicalEditorWithFlyoutPalette;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.util.SafeRunnable;
import org.eclipse.jface.viewers.ISelection;
```

```
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.swt.SWT;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorPart;
import org.eclipse.ui.IFileEditorInput;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.ui.IViewSite;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.actions.ActionFactory;
import org.eclipse.ui.views.markers.MarkerItem;
import org.eclipse.xtext.ui.resource.IResourceSetProvider;
import org.eclipse.xtext.xbase.typesystem.IBatchTypeResolver;

import com.google.inject.Inject;

import ru.bmstu.rk9.rao.ui.gef.AntialiasedLayer;
import ru.bmstu.rk9.rao.ui.gef.EditPart;
import ru.bmstu.rk9.rao.ui.gef.Node;
import ru.bmstu.rk9.rao.ui.gef.NodeInfo;
import ru.bmstu.rk9.rao.ui.gef.model.ModelBackgroundLayer;
import ru.bmstu.rk9.rao.ui.gef.model.ModelLayer;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockNode;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockNodeFactory;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockTitleEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockTitleFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockTitleNode;
import ru.bmstu.rk9.rao.ui.process.blocks.generate.GenerateEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.generate.GenerateFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.generate.GenerateNode;
import ru.bmstu.rk9.rao.ui.process.blocks.hold.HoldEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.hold.HoldFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.hold.HoldNode;
import ru.bmstu.rk9.rao.ui.process.blocks.queue.QueueEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.queue.QueueFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.queue.QueueNode;
import ru.bmstu.rk9.rao.ui.process.blocks.release.ReleaseEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.release.ReleaseFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.release.ReleaseNode;
import ru.bmstu.rk9.rao.ui.process.blocks.seize.SeizeEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.seize.SeizeFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.seize.SeizeNode;
import ru.bmstu.rk9.rao.ui.process.blocks.selectpath.SelectPathEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.selectpath.SelectPathFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.selectpath.SelectPathNode;
import ru.bmstu.rk9.rao.ui.process.blocks.terminate.TerminateEditPart;
import ru.bmstu.rk9.rao.ui.process.blocks.terminate.TerminateFigure;
import ru.bmstu.rk9.rao.ui.process.blocks.terminate.TerminateNode;
import ru.bmstu.rk9.rao.ui.process.connection.ConnectionCreationFactory;
import ru.bmstu.rk9.rao.ui.process.model.ProcessModelEditPart;
import ru.bmstu.rk9.rao.ui.process.model.ProcessModelNode;

@SuppressWarnings("restriction")
public class ProcessEditor extends GraphicalEditorWithFlyoutPalette {

	class FeedbackLayer extends AntialiasedLayer {
		FeedbackLayer() {
			setEnabled(false);
		}
```

```
            @Override
            public Dimension getPreferredSize(int wHint, int hHint) {
                  Rectangle rectangle = new Rectangle();
                  for (int i = 0; i < getChildren().size(); i++)
                        rectangle.union(((IFigure)
getChildren().get(i)).getBounds());
                  return rectangle.getSize();
            }
      }

      public ProcessEditor() {
            setEditDomain(new DefaultEditDomain(this));
      }

      public static final String ID = "ru.bmstu.rk9.rao.ui.process.editor";
      private static final Map<Class<? extends Node>, NodeInfo> nodesInfo = new
LinkedHashMap<>();
      private static final Map<Class<? extends EditPart>, NodeInfo>
nodesInfoByEditPart = new LinkedHashMap<>();
      private ProcessModelNode model;

      @Inject
      IResourceSetProvider resourceSetProvider;

      @Inject
      IBatchTypeResolver typeResolver;

      static {
            addNodeInfo(ProcessModelNode.class, ProcessModelEditPart.class,
ModelLayer.class);
            addNodeInfo(GenerateNode.class, GenerateEditPart.class,
GenerateFigure.class);
            addNodeInfo(TerminateNode.class, TerminateEditPart.class,
TerminateFigure.class);
            addNodeInfo(SeizeNode.class, SeizeEditPart.class,
SeizeFigure.class);
            addNodeInfo(ReleaseNode.class, ReleaseEditPart.class,
ReleaseFigure.class);
            addNodeInfo(HoldNode.class, HoldEditPart.class, HoldFigure.class);
            addNodeInfo(QueueNode.class, QueueEditPart.class,
QueueFigure.class);
            addNodeInfo(SelectPathNode.class, SelectPathEditPart.class,
SelectPathFigure.class);
            addNodeInfo(BlockTitleNode.class, BlockTitleEditPart.class,
BlockTitleFigure.class);
      }

      public static boolean hasNodeInfo(Class<? extends Node> node) {
            return getNodeInfo(node) != null;
      }

      public static NodeInfo getNodeInfo(Class<? extends Node> node) {
            return nodesInfo.get(node);
      }

      public static NodeInfo getNodeInfoByEditPart(Class<? extends EditPart>
editPart) {
            return nodesInfoByEditPart.get(editPart);
      }

      private static void addNodeInfo(Class<? extends Node> node, Class<?
extends EditPart> editPart,
                  Class<? extends Figure> figure) {
```

```
            try {
                  NodeInfo nodeInfo = new
NodeInfo(node.getField("name").get(null).toString(), () -> createObject(node),
                              () -> createObject(editPart), () ->
createObject(figure));
                  nodesInfo.put(node, nodeInfo);
                  nodesInfoByEditPart.put(editPart, nodeInfo);
            } catch (NoSuchFieldException | SecurityException |
IllegalArgumentException | IllegalAccessException e) {
                  e.printStackTrace();
            }
      }

      private static <T> T createObject(Class<T> node) {
            try {
                  Constructor<T> constructor = node.getConstructor();
                  return constructor.newInstance();
            } catch (NoSuchMethodException | SecurityException |
InstantiationException | IllegalAccessException
                        | IllegalArgumentException | InvocationTargetException
e) {
                  e.printStackTrace();
            }
            return null;
      }

      @Override
      protected PaletteRoot getPaletteRoot() {
            PaletteRoot root = new PaletteRoot();

            PaletteGroup paletteGroup = new PaletteGroup("Selection");
            root.add(paletteGroup);

            PanningSelectionToolEntry panningSelectionToolEntry = new
PanningSelectionToolEntry();
            panningSelectionToolEntry.setToolClass(ProcessSelectionTool.class);
            paletteGroup.add(panningSelectionToolEntry);
            paletteGroup.add(new MarqueeToolEntry());

            PaletteSeparator separator = new PaletteSeparator();
            root.add(separator);

            PaletteGroup connectionGroup = new PaletteGroup("Connection");
            root.add(connectionGroup);
            ConnectionCreationToolEntry connections = new
ConnectionCreationToolEntry("Connection", "Create Connections",
                        new ConnectionCreationFactory(), null, null);
            connectionGroup.add(connections);
            root.add(separator);

            PaletteGroup processGroup = new PaletteGroup("Process");
            root.add(processGroup);

            for (Map.Entry<Class<? extends Node>, NodeInfo> node :
nodesInfo.entrySet()) {
                  if (!BlockNode.class.isAssignableFrom(node.getKey()))
                        continue;

                  final String nodeName = node.getValue().getName();
                  BlockFigure blockFigure = (BlockFigure)
node.getValue().getFigureFactory().get();
                  processGroup.add(new CombinedTemplateCreationEntry(nodeName,
nodeName, new BlockNodeFactory(node.getKey()),
```

```
      ImageDescriptor.createFromImageData(blockFigure.getSmallPreviewImageData()
),

      ImageDescriptor.createFromImageData(blockFigure.getLargePreviewImageData()
)));
            }
            root.setDefaultEntry(panningSelectionToolEntry);

      getPalettePreferences().setPaletteState(FlyoutPaletteComposite.STATE_PINNE
D_OPEN);
            return root;
      }

      private void setModel(ProcessModelNode model) {
            this.model = model;
            model.setResourceRetriever(new
EResourceRetriever(resourceSetProvider,
                        ((IFileEditorInput)
getEditorInput()).getFile().getProject(), typeResolver));
      }

      public ProcessModelNode getModel() {
            return model;
      }

      @Override
      protected void setInput(IEditorInput input) {
            super.setInput(input);
            IFile file = ((IFileEditorInput) input).getFile();
            setPartName(file.getName());
            try {
                  setModel(readModelFromFile(file));
                  if (getGraphicalViewer() != null)
                        getGraphicalViewer().setContents(getModel());
            } catch (EOFException e) {
                  // http://stackoverflow.com/a/18451336
            } catch (Exception e) {

      MessageDialog.openError(PlatformUI.getWorkbench().getActiveWorkbenchWindow
().getShell(), "Failed to open",
                              "Invalid file format");
            }
      };

      protected void writeToOutputStream(OutputStream outputStream) throws
IOException {
            ObjectOutputStream objectOutputStream = new
ObjectOutputStream(outputStream);
            objectOutputStream.writeObject(getModel());
            objectOutputStream.close();
      }

      @Override
      public void doSave(IProgressMonitor monitor) {
            writeModelToFile(monitor);
            validateModel();
      }

      @Override
      protected void configureGraphicalViewer() {
            super.configureGraphicalViewer();
            GraphicalViewer viewer = getGraphicalViewer();
```

```
		viewer.setEditPartFactory(new ProcessEditPartFactory());

		KeyHandler keyHandler = new KeyHandler();

		keyHandler.put(KeyStroke.getPressed(SWT.DEL, 127, 0),

	getActionRegistry().getAction(ActionFactory.DELETE.getId()));
		viewer.setKeyHandler(keyHandler);

		getGraphicalViewer().setRootEditPart(new ScalableRootEditPart() {

			@Override
			protected void createLayers(LayeredPane layeredPane) {
				layeredPane.add(getScaledLayers(), SCALABLE_LAYERS);
				layeredPane.add(new Layer() {
					@Override
					public Dimension getPreferredSize(int wHint, int
hHint) {
						return new Dimension();
					}
				}, HANDLE_LAYER);
				layeredPane.add(new FeedbackLayer(), FEEDBACK_LAYER);
				layeredPane.add(new GuideLayer(), GUIDE_LAYER);
			}

			@Override
			protected ScalableLayeredPane createScaledLayers() {
				ScalableLayeredPane layers = new ScalableLayeredPane();
				layers.add(new ModelBackgroundLayer(),
ModelBackgroundLayer.MODEL_BACKGROUND_LAYER);
				layers.add(createGridLayer(), GRID_LAYER);
				layers.add(getPrintableLayers(), PRINTABLE_LAYERS);
				return layers;
			}

			@Override
			protected GridLayer createGridLayer() {
				return new ProcessGridLayer();
			}

			@Override
			protected LayeredPane createPrintableLayers() {
				LayeredPane layers = new LayeredPane();

				ConnectionLayer connectionLayer = new ConnectionLayer();
				connectionLayer.setAntialias(SWT.ON);
				layers.add(connectionLayer, CONNECTION_LAYER);

				Layer primaryLayer = new AntialiasedLayer();
				primaryLayer.setLayoutManager(new StackLayout());
				layers.add(primaryLayer, PRIMARY_LAYER);

				return layers;
			}
		});
	}

	@Override
	protected void initializeGraphicalViewer() {
		GraphicalViewer viewer = getGraphicalViewer();
		if (model == null)
			setModel(new ProcessModelNode());
		viewer.setContents(model);
```

```
            viewer.addDropTargetListener(new ProcessDropTargetListener(viewer));
            IViewSite site = null;
            try {
                  site =
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage()

      .showView("org.eclipse.ui.views.ProblemView").getViewSite();
            } catch (PartInitException e) {

      MessageDialog.openError(PlatformUI.getWorkbench().getActiveWorkbenchWindow
().getShell(), "Failed to open",
                              "Failed to open problem view");
                  e.printStackTrace();
                  return;
            }

      site.getWorkbenchWindow().getSelectionService().addSelectionListener(new
ISelectionListener() {
                  @SuppressWarnings("unchecked")
                  @Override
                  public void selectionChanged(IWorkbenchPart part, ISelection
selection) {
                        if (!(selection instanceof IStructuredSelection))
                              return;

                        IStructuredSelection structedSelection =
(IStructuredSelection) selection;
                        if (!(structedSelection.getFirstElement() instanceof
MarkerItem))
                              return;

                        MarkerItem marker = (MarkerItem)
structedSelection.getFirstElement();
                        if (marker == null || marker.getMarker() == null)
                              return;

                        if (!marker.getMarker().getResource()
                                    .equals(((IFileEditorInput)
ProcessEditor.this.getEditorInput()).getFile()))
                              return;

                        int blockNodeID =
marker.getAttributeValue(BlockNode.BLOCK_NODE_MARKER, 0);
                        ProcessModelEditPart modelPart = (ProcessModelEditPart)
getGraphicalViewer().getRootEditPart()
                                    .getChildren().get(0);
                        List<EditPart> editParts = modelPart.getChildren();
                        for (EditPart editPart : editParts) {
                              if (!(editPart instanceof BlockEditPart))
                                    continue;

                              if (((BlockEditPart) editPart).getID() ==
blockNodeID) {
                                    viewer.select(editPart);
                                    viewer.reveal(editPart);
                                    break;
                              }
                        }

                  }
            });
      }
```

```
	@Override
	protected PaletteViewerProvider createPaletteViewerProvider() {
		return new PaletteViewerProvider(getEditDomain()) {
			@Override
			protected void configurePaletteViewer(PaletteViewer viewer) {
				super.configurePaletteViewer(viewer);
				viewer.addDragSourceListener(new
TemplateTransferDragSourceListener(viewer));
			}
		};
	}

	@Override
	public void commandStackChanged(EventObject event) {
		firePropertyChange(IEditorPart.PROP_DIRTY);
		super.commandStackChanged(event);
	}

	public static ProcessModelNode readModelFromFile(IFile file)
			throws ClassNotFoundException, IOException, CoreException {
		InputStream inputStream = file.getContents(false);
		ObjectInputStream objectInputStream = new
ObjectInputStream(inputStream);
		ProcessModelNode model = (ProcessModelNode)
objectInputStream.readObject();
		objectInputStream.close();
		return model;
	}

	private void writeModelToFile(IProgressMonitor monitor) {
		SafeRunnable.run(new SafeRunnable() {
			@Override
			public void run() throws Exception {
				ByteArrayOutputStream outputStream = new
ByteArrayOutputStream();
				writeToOutputStream(outputStream);
				IFile file = ((IFileEditorInput)
getEditorInput()).getFile();
				file.setContents(new
ByteArrayInputStream(outputStream.toByteArray()), true, false, monitor);
				getCommandStack().markSaveLocation();
			}
		});
	}

	private void validateModel() {
		IFile file = ((IFileEditorInput) getEditorInput()).getFile();
		try {
			file.deleteMarkers(BlockNode.PROCESS_PROBLEM_MARKER, true,
IResource.DEPTH_ZERO);
			for (Node node : model.getChildren()) {
				if (node instanceof BlockNode) {
					((BlockNode) node).validateProperty(file);
				}
			}
		} catch (CoreException e) {

	MessageDialog.openError(PlatformUI.getWorkbench().getActiveWorkbenchWindow
().getShell(), "Internal error",
					"Internal error during problem markers creation");
		}
	}
}
```

ПРИЛОЖЕНИЕ В

Класс конвертера блоков

```
package ru.bmstu.rk9.rao.ui.process;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import ru.bmstu.rk9.rao.lib.process.Block;
import ru.bmstu.rk9.rao.ui.execution.ModelContentsInfo;
import ru.bmstu.rk9.rao.ui.gef.Node;
import ru.bmstu.rk9.rao.ui.process.blocks.BlockNode;
import ru.bmstu.rk9.rao.ui.process.connection.Connection;

public class BlockConverter {

	public static List<Block> convertModelToBlocks(Node model,
ModelContentsInfo modelContentsInfo) {
		List<Node> children = model.getChildren();
		List<Block> blocks = new ArrayList<Block>();
		Map<BlockNode, BlockConverterInfo> blockNodes = new HashMap<>();
		for (Node node : children) {
			if (!(node instanceof BlockNode))
				continue;

			BlockNode sourceBlockNode = (BlockNode) node;
			BlockConverterInfo sourceBlockInfo;
			if (blockNodes.containsKey(sourceBlockNode)) {
				sourceBlockInfo = blockNodes.get(sourceBlockNode);
			} else {
				sourceBlockInfo =
sourceBlockNode.createBlock(modelContentsInfo);
				if (!sourceBlockInfo.isSuccessful)
					throw new
ProcessParsingException(sourceBlockInfo.errorMessage);
				blocks.add(sourceBlockInfo.block);
				blockNodes.put(sourceBlockNode, sourceBlockInfo);
			}

			for (Connection sourceConnection :
sourceBlockNode.getSourceConnections()) {
				BlockNode targetBlockNode =
sourceConnection.getTargetBlockNode();
				BlockConverterInfo targetBlockInfo;
				if (blockNodes.containsKey(targetBlockNode)) {
					targetBlockInfo = blockNodes.get(targetBlockNode);
				} else {
					targetBlockInfo =
targetBlockNode.createBlock(modelContentsInfo);
					if (!targetBlockInfo.isSuccessful)
						throw new
ProcessParsingException(targetBlockInfo.errorMessage);
					blocks.add(targetBlockInfo.block);
					blockNodes.put(targetBlockNode, targetBlockInfo);
				}

				ru.bmstu.rk9.rao.lib.process.Connection.linkDocks(
```

```
        sourceBlockInfo.outputDocks.get(sourceConnection.getSourceDockName()),

        targetBlockInfo.inputDocks.get(sourceConnection.getTargetDockName()));
                    }
              }
              return blocks;
        }
}
```