МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра компьютерных технологий

Направление подготовки 09.04.01 Информатика и вычислительная техника
Направленность (профиль): Технология разработки программных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Абраменко Артема Андреевича

Тема работы:

РАЗРАБОТКА МОДУЛЯ ВИЗУАЛИЗАЦИИ СТРУКТУРЫ ПРОГРАММ ДЛЯ WEBIDE ЯЗЫКА POST

«К защите допущена»
Заведующий кафедрой КТ,
д.т.н., доцент
Зюбин В. Е. /……
(ФИО) / (подпись)
«20» мая 2023 г.

Руководитель ВКР
Д.т.н., доцент,
зав. каф. КТ ФИТ НГУ
Зюбин В. Е. /……
(ФИО) / (подпись)
«20» мая 2023 г.

Новосибирск, 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра компьютерных технологий
(название кафедры)

Направление подготовки: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА
Направленность (профиль): Технология разработки программных систем

УТВЕРЖДАЮ
Зав. кафедрой Зюбин В. Е.
(фамилия, И., О.)

……
(подпись)
«23» ноября 2022 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

Студенту Абраменко Артему Андреевичу, группы 21222
(фамилия, имя, отчество, номер группы)

Тема Разработка модуля визуализации структуры программ для webIDE языка poST
(полное название темы выпускной квалификационной работы магистра)

утверждена распоряжением проректора по учебной работе от 29 октября 2021 г. №0296, скорректирована распоряжением проректора по учебной работе от 23 ноября 2022 г. №0384.

Срок сдачи студентом готовой работы 20.05.2023 г.

Исходные данные (или цель работы):

Разработать модуль генерации диаграмм по исходному коду для webIDE языка poST.

Структурные части работы:

Содержание, введение, обзор предметной области, постановка задачи, описание типов генерируемых диаграмм, разработка и реализация необходимых инструментальных средств.

Руководитель ВКР
Зав. каф. КТ ФИТ НГУ,
д.т.н., доцент
Зюбин В. Е. /……
(ФИО) / (подпись)
«23» ноября 2022 г.

Задание принял к исполнению
Абраменко А. А. /……
(ФИО студента) / (подпись)
«23» ноября 2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Существует несколько подходов к разработке программного обеспечения, и один из наиболее распространенных – итерационный – предусматривает создание и регулярное изменение программной документации, в частности, графической. Для этого используется подход так называемого автодокументирования, т. е. автоматизированного или даже автоматического построения различных диаграмм по исходному коду программы [1]. Поскольку эти диаграммы наглядно представляют структуру программы, они также активно используются при реверсивном инжиниринге и создании новых версий того или иного программного продукта [2].

В рамках промышленной автоматизации, в том числе для процесс-ориентированного языка poST [3], актуален вопрос разработки средств анализа исходного кода программ. В текущих проектах анализ исходного кода делается вручную, и в силу этого обстоятельства создаваемая документация может содержать ошибки. В связи с этим задача автоматизации процесса создания графической документации на основе исходного кода, включая диаграммы, является для языка poST особенно актуальной.

Цель работы – разработка модуля генерации диаграмм по исходному коду для webIDE языка poST.

Для достижения цели были определены следующие задачи:

1. Анализ предметной области
2. Формирование списка требований к разрабатываемому модулю
3. Проектирование архитектуры разрабатываемого модуля
4. Определение типов генерируемых диаграмм и формата их представления
5. Реализация разрабатываемого модуля
6. Разработка транслятора Reflex-poST
7. Апробация модуля генерации диаграмм на модельных задачах

Новизна работы заключается в создании модуля генерации диаграмм для программ на языке poST, поддерживающего возможность встраивания в webIDE помимо независимой работы.

Практическая ценность работы заключается в автоматической генерации диаграмм, существенно сокращающей время создания графической документации, и следовательно упрощающей разработку и поддержку проектов на языке poST.

Работа изложена в трех главах. В первой главе производится анализ предметной области, и формулируются требования к разрабатываемому модулю. Во второй главе определяются и описываются типы генерируемых диаграмм, формат их представления, и архитектура разрабатываемого модуля. Третья глава посвящена вопросам реализации и тестирования модуля генерации диаграмм, а также транслятора Reflex-poST.

1 Анализ предметной области и формулировка требований к разрабатываемой системе

1.1 Анализ предметной области

1.1.1 Подходы к автоматической генерации диаграмм по исходному коду

Разработка систем, позволяющих автоматически генерировать графическую документацию управляющих алгоритмов на основе исходного кода, является важным направлением развития технологий программирования управляющих систем. Такие средства значительно упрощают процесс сопровождения программных продуктов, включая изучение и анализ кода.

Рассмотрим основные подходы к автоматической генерации диаграмм по исходному коду:

1. Генерация диаграмм классов.

   Подход заключается в том, что инструмент анализа исследует исходный код программы и автоматически создает диаграммы с информацией о найденных классах, интерфейсах, их методах и свойствах. Примеры инструментов: Visual Paradigm, Astah.

2. Генерация диаграмм последовательности вызовов.

   Подход используется для создания диаграмм, которые отображают порядок вызовов методов в программе. Такие диаграммы могут помочь при анализе производительности, выявлении узких мест и оптимизации работы приложения. Примеры инструментов: IntelliJ IDEA, Visual Studio.

3. Генерация диаграмм состояний.

   Подход используется для создания диаграмм, которые показывают поведение объектов в зависимости от их состояния. Такие диаграммы особенно полезны при разработке сложных систем, таких как автоматизированные системы управления производством или бизнес-

процессы. Примеры инструментов: Enterprise Architect, Visual Paradigm.

4. Генерация диаграмм потоков данных.

   Подход используется для создания диаграмм, которые иллюстрируют поток данных в приложении. Такие диаграммы могут помочь при анализе и оптимизации производительности, а также при разработке систем обработки данных. Примеры инструментов: IBM Rational Software Architect, Visual Paradigm.

Проведенный анализ, а также результаты работы [4], показывают, что средства автоматической генерации диаграмм по коду можно считать востребованными, и их наличие в среде разработки обеспечивает преимущества и упрощает процесс разработки и создания документации.

Также можно сделать вывод о том, что выбор конкретных подходов к генерации диаграмм зависит от задачи и предметной области, и для определения типов диаграмм, которые будут генерироваться в результате работы разрабатываемого модуля, необходимо сначала проанализировать специфику процесс-ориентированного подхода программирования в целом, и языка poST в частности.

1.1.2 Специфика процесс-ориентированного подхода программирования

В Институте автоматики и электрометрии СО РАН активно ведется разработка процесс-ориентированного подхода программирования. Основная концепция подхода – взаимодействие друг с другом процессов, представленных как расширенные конечные автоматы. Главное преимущество подхода заключается в том, что он позволяет описать алгоритм автоматизации максимально эффективно [5, 6, 7]. Это подтверждает опыт применения в разработке алгоритмов управления языков Reflex и IndustrialC, расширяющих язык Си процесс-ориентированными конструкциями.

Рассмотрим подробнее основные концепции подхода:

1. Гиперпроцесс [6] – основа процесс-ориентированного подхода, представляет собой упорядоченное множество процессов, которые могут взаимодействовать между собой путем запуска, остановки или контроля текущего состояния.
2. Процесс – описывается как конечный автомат с набором состояний и переходов, который был расширен средствами, обеспечивающими синхронизацию исполнения, такими как управление временными интервалами и таймауты. Такие расширенные возможности достигаются благодаря сохранению времени начала работы каждого состояния и возможности сброса этого времени [5].
3. Состояние – описанный на языке программирования набор действий, и может содержать следующие операторы:
    - Перевод текущего процесса из текущего состояния в другое.
    - Запуск, остановка или остановка по ошибке текущего или другого процесса.
    - Проверка на нахождение определенного процесса в одном из состояний: активен, неактивен, остановлен, остановлен по ошибке.
    - Таймаут – контроль продолжительности нахождения текущего процесса в текущем состоянии.

    Состояния также разделяются на обычные и замкнутые (замкнутое состояние всегда переходит само в себя). Важно отметить, что это разделение не влияет на работу программы напрямую, а служит лишь для обеспечения правильной разработки алгоритмов. Это значит, что замкнутое по смыслу состояние должно быть явно помечено как замкнутое чтобы избежать возможных ошибок.

При процесс-ориентированном подходе программа выполняется циклически, и на каждой итерации цикла программы процессы могут либо выполнять одно из своих состояний, либо пребывать в одном из специальных состояний: STOP – состояние нормальной остановки, ERROR – состояние

остановки по ошибке. Такие состояния не включают никаких действий, и выход процесса из них возможен только при помощи другого процесса [5, 6, 7]. В начале работы программы все ее процессы, кроме первого по порядку, пребывают в состоянии нормальной остановки, и в дальнейшем запускаются по мере выполнения программы.

1.1.3 Специфика языка poST

В 1993 г. был создан стандарт IEC 61131-3, который определяет набор стандартных языков программирования для промышленных контроллеров и систем автоматизации. Structured Text – один из пяти языков стандарта – был разработан как стандартный язык программирования для микропроцессорных контроллеров. В качестве основных преимуществ языка выделяют:

- простоту чтения и написания кода,
- удобство модификации и отладки программ,
- поддержку структурного программирования и объектно-ориентированного подхода,
- большое количество библиотек и функций, предназначенных для автоматизации процессов.

Данный язык представляет собой мощный инструмент для создания систем автоматизации, однако с течением времени разрабатываемые алгоритмы становятся все сложнее, что может приводить к определенным проблемам.

Для решения проблемы увеличения сложности кода и стоимости разработки в ИАиЭ СО РАН для языка Structured Text было разработано процесс-ориентированное расширение – poST [3]. При разработке в язык были добавлены процесс-ориентированные конструкции. Таким образом язык poST объединил в себе преимущества стандарта IEC 61131-3 и процесс-ориентированного подхода. Эти особенности языка poST способствуют уменьшению трудозатрат на разработку, и при этом позволяют снизить порог вхождения для программистов, облегчая изучение и модификацию программ.

Описание переменных в программе на языке poST выносится в отдельные блоки кода следующего вида:

VAR <Модификатор переменной>

<Объявление переменных>;

...

END_VAR

Ключевое слово VAR определяет тип всех переменных, объявленных внутри данного блока. Вместо него может быть использовано любое из следующих:

- VAR_INPUT – для обозначения входных переменных,
- VAR_OUTPUT – для обозначения выходных переменных,
- VAR_IN_OUT – для обозначения входных и выходных переменных,
- VAR_TEMP – для обозначения временных переменных, у которых не сохраняется состояние между вызовами.

Язык poST поддерживает как конструкции общего назначения (условные операторы и циклы), так и специальные конструкции, характерные для процесс-ориентированного подхода:

- PROCESS – процесс;
- STATE – состояние;
- START PROCESS <имя процесса> – запуск определенного процесса;
- RESTART – перезапуск процесса (переход процесса в начальное, т. е. первое по порядку, состояние);
- STOP – переход процесса в состояние нормальной остановки;
- ERROR – переход процесса в состояние остановки по ошибке;
- SET STATE <имя состояния> – переход процесса в определенное состояние;
- PROCESS <имя процесса> IN STATE <ACTIVE | INACTIVE | STOP | ERROR> – проверка определенного процесса на нахождение в одном из состояний (возвращает TRUE либо FALSE);

- RESET TIMER – сброс времени запуска текущего состояния;
- TIMEOUT – контроль продолжительности нахождения текущего процесса в текущем состоянии.

1.2 Требования к программному модулю

При анализе программы на poST наибольший интерес представляют процессы и их внутренняя структура, а также связи между процессами, а именно связь по управлению (запуск или остановка одним процессом другого) и связь по данным (использование тех или иных переменных теми или иными процессами). Таким образом, для генерации определяются три основных типа диаграмм:

1. диаграмма состояний процесса,
2. диаграмма связи процессов по управлению,
3. диаграмма использования переменных.

При этом, во время анализа, например, связей процессов по управлению, может понадобиться информация о внутренней структуре процесса. Следовательно, между разными типами диаграмм, относящихся к одной программе, должна обеспечиваться связь, например, путем сохранения внутри вершины-процесса ссылки на соответствующую диаграмму состояний этого процесса.

Для анализа диаграмм могут быть использованы средства визуализации по коду, также выполняющие автоматическую укладку элементов на плоскость, что значительно экономит время. Однако существующие средства не всегда выполняют размещение элементов в удобном виде с точки зрения их восприятия. Таким образом, файлы с диаграммами должны поддерживать возможность редактирования и укладки сторонними средствами.

Для языка poST создан и активно разрабатывается прототип webIDE (post.iae.nsk.su) на базе стека технологий Eclipse/Theia [2]. Для серверной части webIDE используется микросервисная архитектура [8], в которой базовая

функциональность ядра webIDE может быть дополнена через подключение модулей расширения.

На основе проведенного анализа были сформулированы следующие требования к разрабатываемому модулю:

1. Модуль должен получать на вход исходный код программы на языке poST.
2. Для полученного на вход текста должен производиться контроль принадлежности к языку poST.
3. Модуль должен выполнять генерацию диаграмм:
    - состояний процесса,
    - связи процессов по управлению,
    - использования переменных.
4. Модуль должен поддерживать возможность сохранения диаграммы в отдельный файл.
5. Формат файла с диаграммой должен поддерживать возможность редактирования и укладки полученных диаграмм сторонними средствами.
6. Между разными типами диаграмм, генерируемых из одного исходного файла, должна быть обеспечена связь с помощью ссылок.
7. Модуль должен быть реализован как модуль расширения существующего webIDE языка poST.

2 Проектирование архитектуры системы. Типы генерируемых диаграмм и форматы их представления

2.1 Архитектура разрабатываемого модуля

Архитектура системы была спроектирована на основе шаблона Модель-Представление-Контроллер (MVC). Такой подход был выбран, поскольку с его

помощью система разделяется на логические части, которые могут быть разработаны и изменены независимо друг от друга. Это позволяет как облегчить, так и ускорить изменение и корректировку компонентов системы. Схема взаимодействия компонентов представлена на рис. 1.

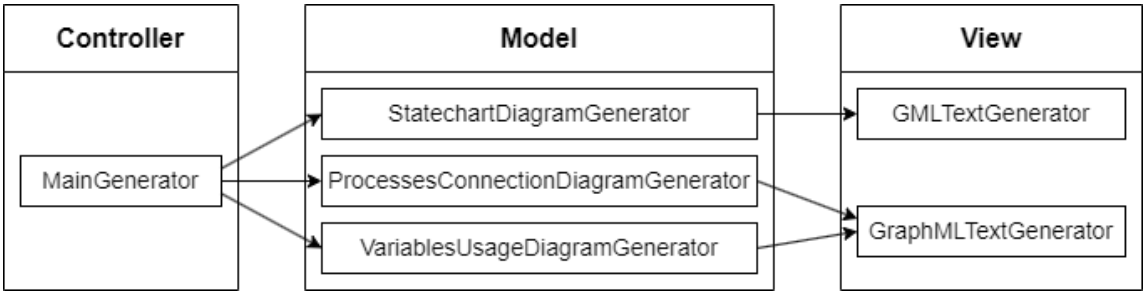

Рисунок 1 – Схема взаимодействия компонентов модуля генерации диаграмм

Класс MainGenerator контролирует работу всего модуля. Классы StatechartDiagramGenerator, ProcessesConnectionDiagramGenerator и VariablesUsageDiagramGenerator отвечают за анализ абстрактного синтаксического дерева (AST) программы и запуск процесса генерации соответствующих типов диаграмм. Классы GMLTextGenerator и GraphMLTextGenerator выполняют генерацию текстового представления диаграмм в форматах GML и GraphML соответственно.

2.2 Описание типов генерируемых диаграмм

2.2.1 Диаграмма состояний процесса

Внутри вершин указываются имена состояний. Над стрелками перехода, соединяющими вершины, указывается ограждающее выражение, т. е. условие перехода из одного состояния в другое. Если при переходе в состояние совершаются какие-либо дополнительные действия, они указываются после ограждающего выражения и отделяются от него косой чертой. Пример

диаграммы для процесса MainLoop системы наполнения бутылок (см. приложение А) представлен на рис. 2.

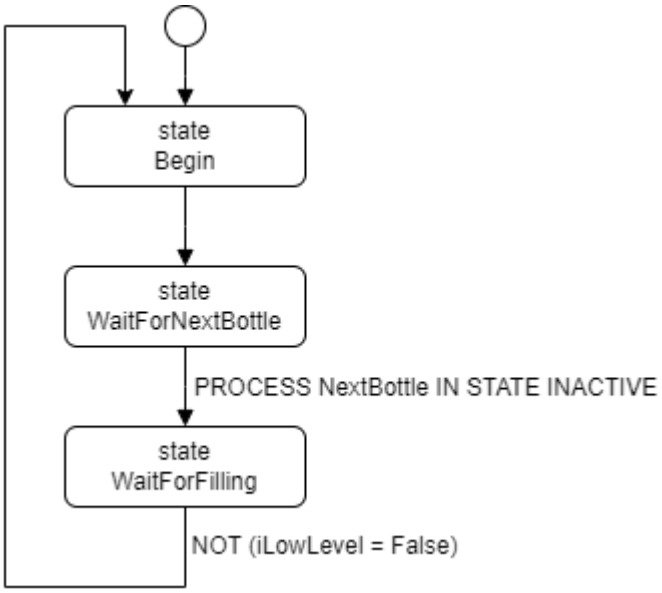

Рисунок 2 – схематическое представление диаграммы состояний процесса

2.2.2 Диаграмма связи процессов по управлению

Внутри вершин указываются имена процессов. Вершины соединяются стрелками, отражающими связь по управлению, и над стрелками указывается тип связи – запуск (start) или остановка (stop). Пример диаграммы представлен на рис. 3.

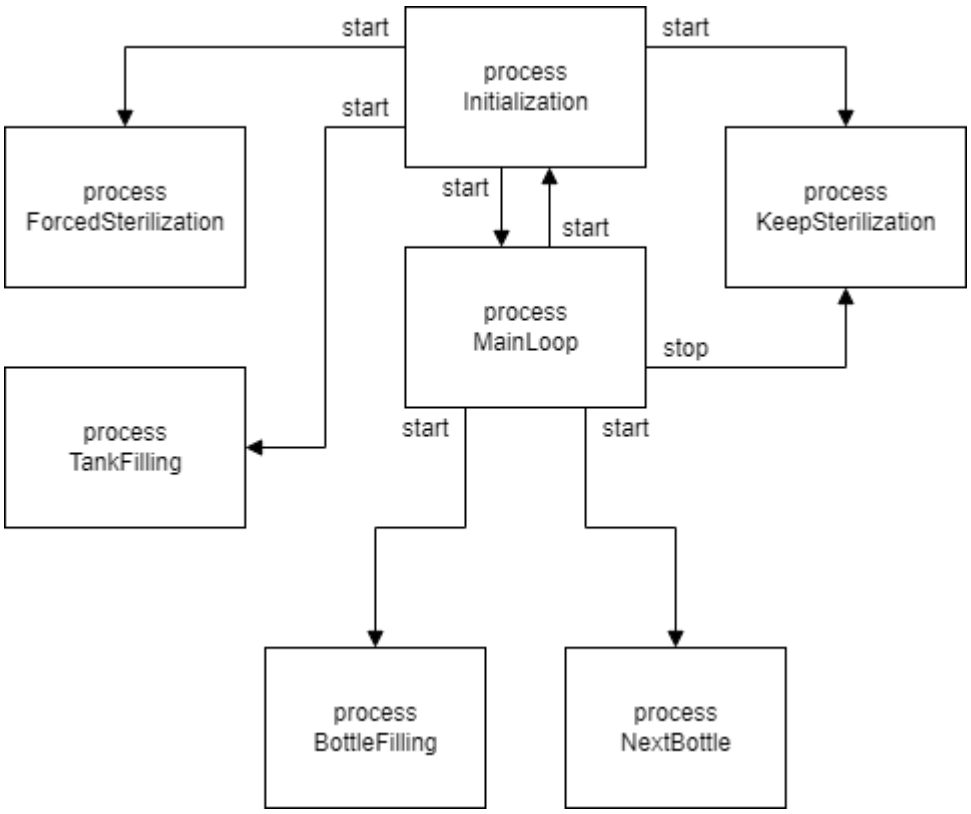

Рисунок 3 – схематическое представление диаграммы связи процессов по управлению

2.2.3 Диаграмма использования переменных

Внутри прямоугольных вершин указываются имена процессов, внутри овальных вершин – имена переменных. Каждая прямоугольная вершина-процесс соединяется с теми овальными вершинами, которые соответствуют переменным, использующимся в данном процессе. Пример диаграммы представлен на рис. 4.

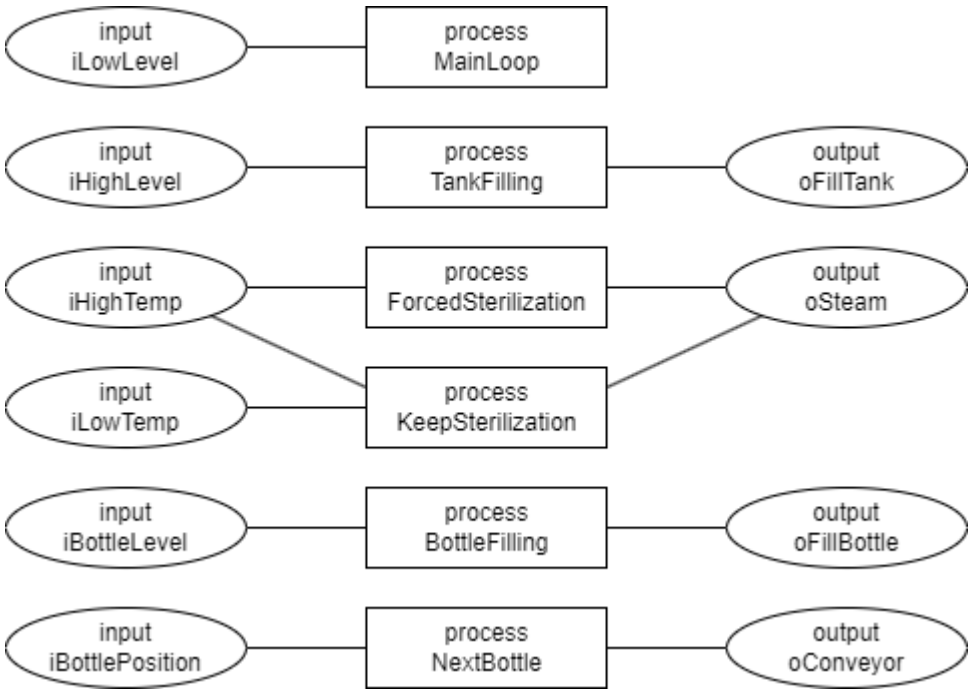

Рисунок 4 – схематическое представление диаграммы использования переменных

2.3 Форматы представления диаграмм

Для представления диаграмм были выбраны два языка описания графов: GML и GraphML. Оба данных формата имеют свои преимущества: файл в формате GML меньше, чем GraphML, однако GraphML поддерживает возможность хранения ссылок в вершинах графа.

В качестве формата для диаграммы связи процессов по управлению и диаграммы использования переменных был выбран GraphML, так как он позволяет хранить ссылки. Таким образом, в каждой вершине графа, соответствующей тому или иному процессу, сохраняется ссылка на соответствующую диаграмму состояний этого процесса.

В качестве формата для диаграммы состояний процесса был выбран GML, так как анализируемая программа может содержать очень большое число процессов, и для каждого из них должен генерироваться отдельный файл с диаграммой состояний. В таком случае целесообразнее использовать GML, поскольку файлы в этом формате занимают меньше места на диске.

3 Реализация разрабатываемых инструментальных средств

3.1 Реализация модуля генерации диаграмм

3.1.1 Детали реализации модуля

Модуль генерации диаграмм было решено реализовать в виде Eclipse-плагина, с применением Xtext [9] – инструментария, позволяющего создавать предметно-ориентированные языки (DSL). Основой Xtext является механизм парсинга ANTLR, который позволяет определить синтаксис языка, а также его семантику в виде грамматики. На основе описания грамматики Xtext позволяет автоматически генерировать полную инфраструктуру для редактирования и анализа текста на целевом языке, включающую классы, необходимые для хранения абстрактного синтаксического дерева (AST) программы. Помимо этого Xtext генерирует такие компоненты системы, как лексический и семантический анализаторы, а также готовые редактор кода и парсер. Это позволяет не тратить время на их разработку, и сконцентрироваться на реализации компонента, отвечающего за генерацию текста.

При реализации модуля использовался язык Xtend [10] – статически типизированный язык программирования, транслирующийся в Java. В качестве наиболее полезной для поставленной задачи особенности языка можно выделить шаблонные выражения. Шаблон (template) представляет собой набор из одной или более строк, и может содержать внутри себя вложенные выражения, такие как идентификатор переменной или вызов метода. Внутри шаблона также можно определить условное выражение (IF) и цикл (FOR). Таким образом, шаблонные выражения значительно упрощают, а также расширяют возможности работы со строками, что позволяет быстрее и удобнее

реализовывать методы, отвечающие за генерацию текстового представления диаграмм.

3.1.2 Алгоритм работы модуля

Алгоритм работы модуля описывается следующим образом: сначала в главном классе-контроллере MainGenerator сохраняется AST программы, получаемой на вход в редактор кода.

Затем контроллер проходит по списку процессов, и для каждого из них сперва запускается создание модели диаграммы состояний на основе сохраненного дерева программы, после чего запускается основной метод класса StatechartDiagramGenerator, отвечающий за анализ дерева и генерацию текста соответствующей диаграммы в формате GML с помощью методов класса GMLTextGenerator.

Затем контроллер запускает создание моделей диаграммы связи процессов по управлению и диаграммы использования переменных на основе сохраненного дерева программы. Далее запускаются основные методы классов ProcessesConnectionDiagramGenerator и VariablesUsageDiagramGenerator, отвечающие за анализ дерева и генерацию текста вышеназванных диаграмм в формате GraphML с помощью методов класса GraphMLTextGenerator. При этом в каждую вершину, представляющую определенный процесс, помещается ссылка на соответствующую диаграмму состояний этого процесса, сгенерированную ранее.

В результате работы модуля генерируются по одному GML-файлу с диаграммой состояний для каждого процесса, GraphML-файл с диаграммой связи процессов по управлению и GraphML-файл с диаграммой использования переменных.

Для просмотра сгенерированных файлов с диаграммами рекомендуется использовать программы визуализации графов, которые обеспечивают

возможность автоматической укладки элементов диаграммы и позволяют ее редактировать (например, десктоп-версия редактора диаграмм yEd [11]).

3.2 Реализация транслятора Reflex-poST

В целях тестирования модуля генерации диаграмм был разработан и апробирован транслятор, выполняющий перевод с языка Reflex 1.0 на язык poST. Транслятор позволяет автоматизировать и ускорить трудоемкий процесс перевода и в результате получить тексты больших программ на языке poST из уже существующих Reflex-программ (на языке poST пока не было реализовано подобных программ, и ручной перевод кода с Reflex 1.0 на poST занял бы неоправданно много времени, не исключая риск возникновения ошибок).

3.2.1 Различия языков Reflex 1.0 и poST

Был проведен сравнительный анализ грамматик языков Reflex 1.0 и poST, и по его результатам были сделаны следующие выводы относительно различий рассматриваемых языков:

1. Перечисления в языке poST отличаются по функциональности от перечислений в языке Reflex 1.0, в связи с чем было решено при переводе на poST для каждого перечисления выносить все его элементы в отдельный блок VAR с модификатором CONSTANT.
2. Reflex 1.0 поддерживает конструкции, отвечающие за разделение переменных между процессами, вида <FOR ALL | FOR PROC <список имен процессов>>. При переводе на poST удобнее всего вынести подобные переменные в отдельный блок VAR, таким образом они становятся доступны всем процессам.
3. Конструкция Reflex 1.0 вида FROM PROC <имя процесса> <список импортируемых переменных> при переводе на poST опускается,

поскольку разделяемые переменные становятся доступны всем процессам.

4. Объявления переменных, зависимых от физических входных и выходных портов, при переводе на poST выносятся в отдельный блок VAR_INPUT, если порт входной, и VAR_OUTPUT, если порт выходной.
5. В конструкции РАЗБОР в Reflex 1.0, в отличие от CASE в poST, допустимо после условия как использовать пустые выражения, так и не указывать выражений вообще. Для обработки подобных случаев при переводе на poST в начале программы определяется переменная _VOID_. Она позволяет использовать внутри конструкции CASE фиктивное выражение, которое выглядит следующим образом: _VOID_ := _VOID_;

На основе анализа грамматик и сделанных выводов была составлена таблица правил перевода языковых конструкций с Reflex 1.0 на poST, представленная в приложении Б.

3.2.2 Детали реализации транслятора

Реализация транслятора была также произведена с использованием технологий Eclipse/Xtext и языка Xtend. В качестве основы для генерации инфраструктуры, редактора кода и парсера было использовано описание грамматики Reflex 1.0 из работы [12], составленное на специализированном языке описания грамматик, также предоставляемом Xtext.

Транслятор состоит из 3 основных частей: редактора кода, парсера и кодогенератора. Взаимодействие частей транслятора происходит следующим образом: сперва редактор кода принимает на вход текст программы на языке Reflex 1.0. Затем этот текст из редактора передается в парсер, который осуществляет синтаксический разбор программы и создает на ее основе абстрактное синтаксическое дерево, в котором операторы языка Reflex 1.0

представлены внутренними вершинами, а операнды - листьями. После этого дерево передается в кодогенератор, который проводит обход узлов дерева, и создает файл с переведенным на poST текстом программы.

3.2.3 Тестирование транслятора

Для тестирования транслятора использовался текст программы управления установкой для выращивания монокристаллического кремния [13], состоящей из нескольких десятков тысяч строк кода.

Текст программы был передан в разработанный транслятор через редактор кода, транслятор успешно распознал язык Reflex 1.0 и выполнил его перевод на язык poST за несколько секунд.

Затем полученный текст был передан в транслятор poST-ST (post2st.iae.nsk.su), который успешно определил принадлежность текста к языку poST, не обнаружив в нем синтаксических и семантических ошибок.

3.3 Тестирование модуля на модельных задачах

3.3.1 Система управления установкой для наполнения бутылок

В качестве первой модельной задачи рассматривается система наполнения бутылок, описанная в работе [14] (см. рис. 5).

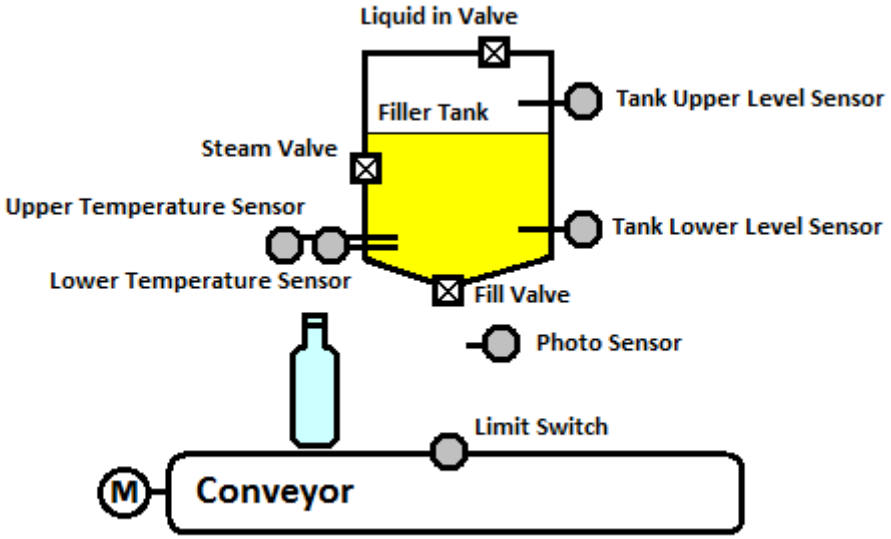

Рисунок 5 – схематический вид системы наполнения бутылок

Рассматриваемая система, предназначенная для транспортировки бутылок, состоит из конвейера и наливного бака с датчиками и клапанами. Алгоритм работы системы заключается в том, чтобы обеспечивать подачу бутылок по конвейеру, контролировать температуру и заполнение этих бутылок стерилизованной жидкостью. Для этого используются два датчика температуры, которые контролируют нагрев наливного бака до 100 градусов Цельсия, а также паровой клапан, который обеспечивает подачу перегретого пара. Концевой выключатель на конвейере обнаруживает наличие бутылки под баком, клапан в нижней части бака используется для наливания жидкости в бутылку. Уровень жидкости в бутылке определяется фотодатчиком, два датчика уровня контролируют степень заполненности бака жидкостью. Клапан в верхней части бака служит для пополнения его жидкостью.

3.3.2 Система управления установкой для выращивания монокристаллического кремния

В качестве второй модельной задачи рассматривается система выращивания кристаллов кремния, описанная в работе [13] (см. рис. 6).

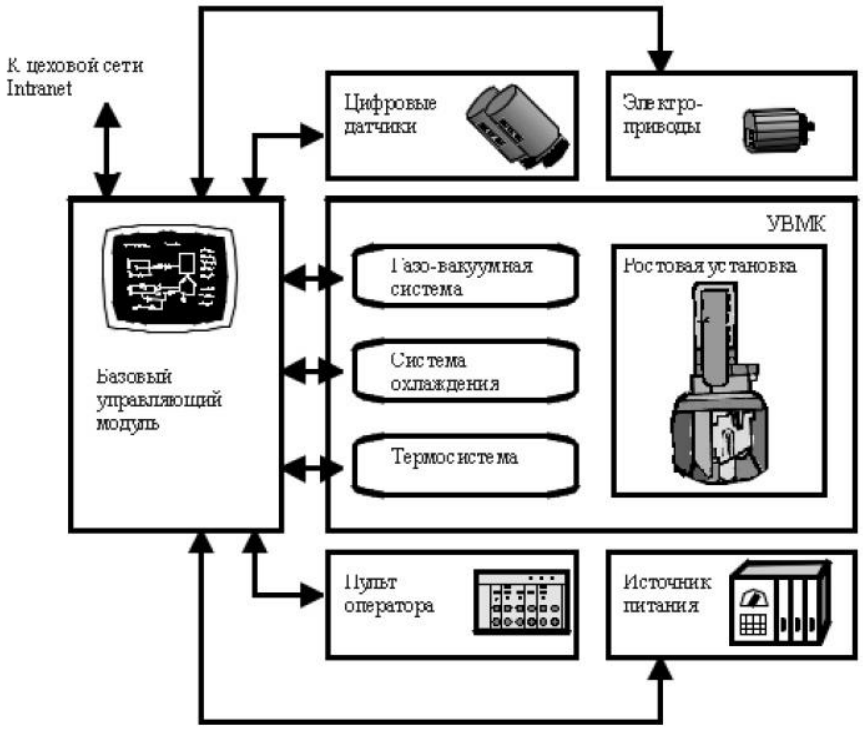

Рисунок 6 – схематический вид системы выращивания кристаллов кремния

Рассматриваемая система (УВМК), предназначенная для получения монокристаллов кремния, состоит из ростовой камеры, газовакуумной системы, термосистемы и системы охлаждения. Ростовая камера имеет размеры около 3 м в высоту и 1,5 м в диаметре. Газовакуумная система включает в себя четыре вакуумных насоса, линию подачи аргона и множество клапанов, которые определяют протекание процесса вакуумирования. Температура до 1700 °C поддерживается с помощью расположенного внутри камеры нагревателя, который совместно с электропитанием образует термосистему. Система охлаждения включает магистрали, проложенные в стенках ростовой камеры. УВМК содержит несколько десятков разнородных входных и выходных аналоговых сигналов, и около сотни входных и выходных цифровых сигналов.

3.3.3 Результаты тестирования

Первичное тестирование модуля проводилось на тексте программы управления системой наполнения бутылок. Модуль успешно определил принадлежность текста к языку poST, не выявив в нем синтаксических и семантических ошибок, и сгенерировал все необходимые диаграммы (каждого процесса, связи процессов по управлению, использования переменных).

Затем модуль был запущен на тексте программы управления установкой по выращиванию монокристаллического кремния, полученной для языка poST в результате работы транслятора Reflex-poST. Модуль так же успешно определил язык poST и произвел генерацию всех необходимых диаграмм.

В обоих случаях диаграммы были затем открыты через редактор графов yEd, и была задействована функция автоматической укладки диаграмм. Сравнительный анализ составленной вручную диаграммы связи процессов по управлению для программы управления системой наполнения бутылок и аналогичной диаграммы, сгенерированной модулем, показал, что сгенерированная диаграмма соответствует исходной, составленной вручную, и в каждой вершине сохранена ссылка на диаграмму соответствующего процесса. Текст сгенерированной диаграммы представлен в приложении В.

Результаты тестирования подтверждают работоспособность модуля, в том числе и на большой программе, содержащей более 43000 строк кода. Вследствие автоматизации процесса создания диаграмм применение модуля исключает возможность внесения ошибок, вызванных человеческим фактором, а также позволяет значительно сократить время, затрачиваемое на создание диаграмм, до порядка 1 секунды в случае программы среднего размера (первая модельная задача – 150 строк кода), и 1 минуты в случае программы большого размера (вторая модельная задача – более 43000 строк кода).

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы был разработан и апробирован модуль генерации диаграмм по исходному коду для webIDE языка poST. Модуль позволяет автоматически генерировать диаграммы, значительно сокращая время создания графической документации, и следовательно упрощая разработку и поддержку проектов на языке poST.

Также был разработан и апробирован транслятор Reflex-poST, позволяющий получить тексты больших poST-программ, что обеспечивает возможность проверки разрабатываемых для poST средств реверсивного инжиниринга и статического анализа.

Работа была представлена на Международной научной студенческой конференции 2022 в секции «Информационные технологии», подсекции «Инструментальные и прикладные программные системы», представленные тезисы опубликованы в сборнике материалов конференции.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Абраменко Артем Андреевич _____

ФИО студента *Подпись студента*

« ____ »_____20 __г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Zhang C., Wang J., Zhou Q., Xu T., Tang K., Gui H., Liu F. A survey of automatic source code summarization //Symmetry. – 2022. – Т. 14. – №. 3. – С. 471.
2. Bastrykina A., Zyubin V., Rozov A. Developing reflex IDE kernel with Xtext framework //2021 IEEE 22nd International Conference of Young Professionals in Electron Devices and Materials (EDM). – IEEE, 2021. – С. 511-514.
3. Zyubin, V. E., Rozov, A. S., Anureev, I. S., Garanina, N. O., Vyatkin, V. poST: A process-oriented extension of the IEC 61131-3 structured text language // IEEE Access. – 2022. – Т. 10. – С. 35238-35250.
4. Беленькая С. Е. Разработка программного модуля визуализации диаграмм процессов по спецификации на языке Reflex: диплом. работа бак. / Беленькая С. Е. – Новосибирский государственный университет. – 2020. – 73 с.
5. Зюбин В. Е. Процесс-ориентированное программирование: Учеб. пособие // Новосибирск. Новосиб. гос. ун-т. 2011. 194 с.
6. Розов А. С., Зюбин В. Е. Расширенная модель гиперпроцесса для программирования микроконтроллеров // Промышленные АСУ и контроллеры, №9, 2016 г., Стр. 34-38.
7. Розов А. С., Зюбин В. Е., Нефедов Д. В. Программирование встраиваемых микроконтроллерных систем на основе гиперпроцессов // Вестн. НГУ. Серия: информационные технологии. 2017. Т. 15, № 4. С. 64–73.
8. Gornev I., Liakh T. Ride: Theia-based web ide for the Reflex language //2021 IEEE 22nd International Conference of Young Professionals in Electron Devices and Materials (EDM). – IEEE, 2021. – С. 503-506.
9. Bettini L. An Eclipse-based IDE for Featherweight Java implemented in Xtext //The Fifth Workshop of the Italian Eclipse Community, Proceedings. – Eclipse Italian community, 2010. – С. 14-28.
10. Bettini L. Implementing domain-specific languages with Xtext and Xtend. – Packt Publishing Ltd, 2016.
11. YWorks. yEd graph editor. – 2023, URL: https://www.yworks.com/products/yed (Дата обращения 20.03.2023)
12. Абраменко А. А. Разработка межверсионного конвертера для программ на языке Reflex 1.0: диплом. работа бак. / Абраменко А. А. – Новосибирский государственный университет. – 2021. – 48 с.
13. Зюбин В. Е. и др. Базовый модуль, управляющий установкой для выращивания монокристаллов кремния //Датчики и системы. – 2004. – №. 12. – С. 17-22.
14. Anureev, I. S., Zyubin, V. E., Garanina, N. O., Staroletov, S. M. Developing Distributed Control Software with the Reflex Language: Bottle-filling System Case Study //2022 International Russian Automation Conference (RusAutoCon). – IEEE, 2022. – С. 683-688.

ПРИЛОЖЕНИЕ А

Программа управления установкой для наполнения бутылок

```
PROGRAM BottleFillingController

/*===== P1 VARIABLES =====*/
    VAR_INPUT
        iLowLevel  : BOOL;
        iHighLevel  : BOOL;
    END_VAR
    VAR_OUTPUT
        oFillTank : BOOL;
    END_VAR

/*===== P2 VARIABLES =====*/
    VAR_INPUT
        iLowTemp  : BOOL;
        iHighTemp  : BOOL;
    END_VAR
    VAR_OUTPUT
        oSteam : BOOL;
    END_VAR

/*===== P3 VARIABLES =====*/
    VAR_INPUT
        iBottleLevel  : BOOL;
    END_VAR
    VAR_OUTPUT
        oFillBottle : BOOL;
    END_VAR

/*===== P4 VARIABLES =====*/
    VAR_INPUT
        iBottlePosition  : BOOL;
    END_VAR

    VAR_OUTPUT
        oConveyor : BOOL;
    END_VAR

/*====== CONSTANTS =========*/

//    VAR CONSTANT
//        OFF : BOOL := FALSE;
//        ON : BOOL := TRUE;
//    END_VAR



/* === Controller 1 ================ */
PROCESS Initialization
    STATE Begin
        START PROCESS TankFilling;
        SET NEXT;
    END_STATE
    STATE WaitForFilling
        IF (PROCESS TankFilling IN STATE INACTIVE) THEN
            START PROCESS ForcedSterilization;
            SET NEXT;
```

```
        END_IF
    END_STATE
    STATE WaitForSterilization
        IF (PROCESS ForcedSterilization IN STATE INACTIVE) THEN
            START PROCESS KeepSterilization;
            START PROCESS MainLoop;
            STOP;
        END_IF
    END_STATE
END_PROCESS /* END OF PROCESS */

PROCESS MainLoop
    STATE Begin
        START PROCESS NextBottle;
        SET NEXT;
    END_STATE
    STATE WaitForNextBottle
        IF (PROCESS NextBottle IN STATE INACTIVE) THEN
            START PROCESS BottleFilling;
            SET NEXT;
        END_IF
    END_STATE
    STATE WaitForFilling
        IF (PROCESS BottleFilling IN STATE INACTIVE) THEN
            IF (iLowLevel =  FALSE) THEN // no liquid
                STOP PROCESS KeepSterilization;
                START PROCESS Initialization;
                STOP;
            ELSE
                RESTART;
            END_IF
        END_IF
    END_STATE
END_PROCESS

PROCESS TankFilling
    STATE Begin
        IF (iHighLevel <> TRUE) THEN // is not filled
            oFillTank := TRUE;
        END_IF
        SET NEXT;
    END_STATE
    STATE TankFilled
        IF (iHighLevel = TRUE) THEN
            oFillTank :=  FALSE;
            STOP;
        END_IF
    END_STATE
END_PROCESS /* END OF PROCESS */

/* === Controller 2 ================ */
PROCESS ForcedSterilization
    STATE HeatUp
        oSteam := TRUE;
        IF (iHighTemp = TRUE) THEN
            SET NEXT;
        END_IF
    END_STATE
    STATE SterilizationFor1min
        TIMEOUT T#1m THEN
            oSteam :=  FALSE;
            STOP;
```

```
        END_TIMEOUT
    END_STATE
END_PROCESS /* END OF PROCESS */

PROCESS KeepSterilization
    STATE WaitLowTemp
        IF (iLowTemp <> TRUE) THEN
            oSteam := TRUE;
            SET NEXT;
        END_IF
    END_STATE
    STATE WaitHighTemp
        IF (iHighTemp = TRUE) THEN
            oSteam :=  FALSE;
            RESTART;
        END_IF
    END_STATE
END_PROCESS /* END OF PROCESS */

/* === Controller 3 ====== */
PROCESS BottleFilling
    STATE Begin
        oFillBottle := TRUE;
        IF (iBottleLevel = TRUE) THEN
            oFillBottle :=  FALSE;
            STOP;
        END_IF
    END_STATE
END_PROCESS /* END OF PROCESS */

/* === Controller 4 ================ */
PROCESS NextBottle
    STATE Begin
        oConveyor := TRUE;
        IF (iBottlePosition <> TRUE) THEN // push current bottle
            SET NEXT;
        END_IF
    END_STATE
    STATE WaitBottlePosition
        IF (iBottlePosition = TRUE) THEN
            oConveyor :=  FALSE;
            STOP;
        END_IF
    END_STATE
END_PROCESS /* END OF PROCESS */
END_PROGRAM
```

ПРИЛОЖЕНИЕ Б

Правила перевода языковых конструкций с языка Reflex 1.0 на язык poST

Конструкция в Reflex 1.0	Правило перевода на poST
PROGR <имя программы> { <тело программы> }	PROGRAM <имя программы> <тело программы> END_PROGRAM
CONST <имя константы> <значение>; CONST <имя константы> <значение>; …	VAR CONSTANT <имя> : <тип> := <значение>; <имя> : <тип> := <значение>; … END_VAR
ENUM { <список имен констант> };	*Для каждого перечисления все константы выносятся в отдельный блок VAR CONSTANT, аналогично описанному выше.*
PROC <имя процесса> { <тело процесса> }	PROCESS <имя процесса> <тело процесса> END_PROCESS
<тип переменной> <имя переменной> LOCAL; <тип переменной> <имя переменной> LOCAL; …	VAR <имя> : <тип> := <значение>; <имя> : <тип> := <значение>; … END_VAR
<тип переменной> <имя переменной> <FOR ALL | FOR PROC <список имен процессов>>;	*Переменные выносятся в отдельный блок VAR, доступный всем процессам.*
FROM PROC <имя процесса> <список импортируемых переменных>;	*Поскольку вышеописанные вынесенные из процессов переменные теперь доступны всем процессам, конструкция опускается.*
STATE <имя состояния> { <тело состояния> }	STATE <имя состояния> <тело состояния> END_STATE
START PROC <имя процесса>;	START PROCESS <имя процесса>;
RESTART PROC;	RESTART;
STOP PROC <имя процесса>;	STOP PROCESS <имя процесса>;
STOP;	STOP;
ERROR PROC <имя процесса>;	ERROR PROCESS <имя процесса>;
ERROR;	ERROR;
PROC <имя процесса> IN STATE <ACTIVE | PASSIVE | STOP | ERROR>	PROCESS <имя процесса> IN STATE <ACTIVE | INACTIVE |STOP | ERROR>

TIMEOUT <интервал> { <последовательность выражений> }	TIMEOUT <интервал> THEN <последовательность выражений> END_TIMEOUT
SET STATE <имя состояния>;	SET STATE <имя состояния>;
SET NEXT;	SET NEXT;
RESTART TIMER;	RESET TIMER;
STATE <имя зацикленного состояния> { <тело зацикленного состояния> LOOP; }	STATE <имя зацикленного состояния> LOOPED <тело зацикленного состояния> END_STATE

ПРИЛОЖЕНИЕ В

GraphML-представление сгенерированной диаграммы связи процессов по управлению для системы наполнения бутылок

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graphml                                xmlns="http://graphml.graphdrawing.org/xmlns"
xmlns:java="http://www.yworks.com/xml/yfiles-common/1.0/java"
xmlns:sys="http://www.yworks.com/xml/yfiles-common/markup/primitives/2.0"
xmlns:x="http://www.yworks.com/xml/yfiles-common/markup/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:y="http://www.yworks.com/xml/graphml"
xmlns:yed="http://www.yworks.com/xml/yed/3"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
http://www.yworks.com/xml/schema/graphml/1.1/ygraphml.xsd">
  <key for="port" id="d0" yfiles.type="portgraphics"/>
  <key for="port" id="d1" yfiles.type="portgeometry"/>
  <key for="port" id="d2" yfiles.type="portuserdata"/>
  <key attr.name="url" attr.type="string" for="node" id="d3"/>
  <key for="node" id="d5" yfiles.type="nodegraphics"/>
  <key for="graphml" id="d6" yfiles.type="resources"/>
  <key for="edge" id="d9" yfiles.type="edgegraphics"/>0
<graph edgedefault="directed" id="G">
   <node id="n0">
     <data                                                key="d3"><![CDATA[generated-
code/Initialization_statechart_diagram.gml]]></data>
     <data key="d5">
       <y:ShapeNode>
         <y:Geometry height="48.0" width="140"/>
         <y:Fill color="#FFFFFF" transparent="false"/>
         <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
         <y:NodeLabel    alignment="center"    fontFamily="Dialog"    fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
height="18.701171875"        horizontalTextPosition="center"        iconTextGap="4"
modelName="internal"             modelPosition="c"             textColor="#000000"
verticalTextPosition="bottom"                                       visible="true"
width="26.6640625">Initialization</y:NodeLabel>
         <y:Shape type="roundrectangle"/>
       </y:ShapeNode>
     </data>
   </node>
   <node id="n4">
     <data                                                key="d3"><![CDATA[generated-
code/KeepSterilization_statechart_diagram.gml]]></data>
     <data key="d5">
       <y:ShapeNode>
         <y:Geometry height="48.0" width="170"/>
         <y:Fill color="#FFFFFF" transparent="false"/>
         <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
         <y:NodeLabel    alignment="center"    fontFamily="Dialog"    fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
height="18.701171875"        horizontalTextPosition="center"        iconTextGap="4"
modelName="internal"             modelPosition="c"             textColor="#000000"
verticalTextPosition="bottom"                                       visible="true"
width="26.6640625">KeepSterilization</y:NodeLabel>
         <y:Shape type="roundrectangle"/>
       </y:ShapeNode>
     </data>
```

```
   </node>
   <node id="n3">
     <data                                                    key="d3"><![CDATA[generated-
code/ForcedSterilization_statechart_diagram.gml]]></data>
     <data key="d5">
       <y:ShapeNode>
         <y:Geometry height="48.0" width="190"/>
         <y:Fill color="#FFFFFF" transparent="false"/>
         <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
         <y:NodeLabel    alignment="center"    fontFamily="Dialog"    fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
height="18.701171875"        horizontalTextPosition="center"        iconTextGap="4"
modelName="internal"             modelPosition="c"             textColor="#000000"
verticalTextPosition="bottom"                                       visible="true"
width="26.6640625">ForcedSterilization</y:NodeLabel>
         <y:Shape type="roundrectangle"/>
       </y:ShapeNode>
     </data>
   </node>
   <node id="n5">
     <data                                                    key="d3"><![CDATA[generated-
code/BottleFilling_statechart_diagram.gml]]></data>
     <data key="d5">
       <y:ShapeNode>
         <y:Geometry height="48.0" width="130"/>
         <y:Fill color="#FFFFFF" transparent="false"/>
         <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
         <y:NodeLabel    alignment="center"    fontFamily="Dialog"    fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
height="18.701171875"        horizontalTextPosition="center"        iconTextGap="4"
modelName="internal"             modelPosition="c"             textColor="#000000"
verticalTextPosition="bottom"                                       visible="true"
width="26.6640625">BottleFilling</y:NodeLabel>
         <y:Shape type="roundrectangle"/>
       </y:ShapeNode>
     </data>
   </node>
   <node id="n1">
     <data                                                    key="d3"><![CDATA[generated-
code/MainLoop_statechart_diagram.gml]]></data>
     <data key="d5">
       <y:ShapeNode>
         <y:Geometry height="48.0" width="80"/>
         <y:Fill color="#FFFFFF" transparent="false"/>
         <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
         <y:NodeLabel    alignment="center"    fontFamily="Dialog"    fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
height="18.701171875"        horizontalTextPosition="center"        iconTextGap="4"
modelName="internal"             modelPosition="c"             textColor="#000000"
verticalTextPosition="bottom"                                       visible="true"
width="26.6640625">MainLoop</y:NodeLabel>
         <y:Shape type="roundrectangle"/>
       </y:ShapeNode>
     </data>
   </node>
   <node id="n6">
     <data                                                    key="d3"><![CDATA[generated-
code/NextBottle_statechart_diagram.gml]]></data>
     <data key="d5">
       <y:ShapeNode>
         <y:Geometry height="48.0" width="100"/>
         <y:Fill color="#FFFFFF" transparent="false"/>
```

```
        <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
        <y:NodeLabel    alignment="center"    fontFamily="Dialog"    fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
height="18.701171875"         horizontalTextPosition="center"         iconTextGap="4"
modelName="internal"              modelPosition="c"              textColor="#000000"
verticalTextPosition="bottom"                                          visible="true"
width="26.6640625">NextBottle</y:NodeLabel>
        <y:Shape type="roundrectangle"/>
      </y:ShapeNode>
    </data>
  </node>
  <node id="n2">
    <data                                                  key="d3"><![CDATA[generated-
code/TankFilling_statechart_diagram.gml]]></data>
    <data key="d5">
      <y:ShapeNode>
        <y:Geometry height="48.0" width="110"/>
        <y:Fill color="#FFFFFF" transparent="false"/>
        <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
        <y:NodeLabel    alignment="center"    fontFamily="Dialog"    fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
height="18.701171875"         horizontalTextPosition="center"         iconTextGap="4"
modelName="internal"              modelPosition="c"              textColor="#000000"
verticalTextPosition="bottom"                                          visible="true"
width="26.6640625">TankFilling</y:NodeLabel>
        <y:Shape type="roundrectangle"/>
      </y:ShapeNode>
    </data>
  </node>
  <edge id="e0" source="n0" target="n2">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"    iconTextGap="4"    preferredPlacement="anywhere"
textColor="#000000"                                   verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
  </edge>
  <edge id="e1" source="n0" target="n3">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"    iconTextGap="4"    preferredPlacement="anywhere"
textColor="#000000"                                   verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
  </edge>
  <edge id="e2" source="n0" target="n4">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
```

```
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e3" source="n0" target="n1">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e4" source="n0" target="n0">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">stop<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e5" source="n4" target="n4">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e6" source="n3" target="n3">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">stop<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
```

```
          </y:PolyLineEdge>
        </data>
    </edge>
    <edge id="e7" source="n1" target="n6">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"    iconTextGap="4"    preferredPlacement="anywhere"
textColor="#000000"                                   verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
    </edge>
    <edge id="e8" source="n1" target="n5">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"    iconTextGap="4"    preferredPlacement="anywhere"
textColor="#000000"                                   verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
    </edge>
    <edge id="e9" source="n1" target="n4">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"    iconTextGap="4"    preferredPlacement="anywhere"
textColor="#000000"                                   verticalTextPosition="bottom"
visible="true">stop<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
    </edge>
    <edge id="e10" source="n1" target="n0">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"    iconTextGap="4"    preferredPlacement="anywhere"
textColor="#000000"                                   verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
    </edge>
    <edge id="e11" source="n1" target="n1">
        <data key="d9">
          <y:PolyLineEdge>
```

```
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">stop<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e12" source="n1" target="n1">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">start<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e13" source="n2" target="n2">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">stop<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e14" source="n5" target="n5">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
textColor="#000000"                                     verticalTextPosition="bottom"
visible="true">stop<y:PreferredPlacementDescriptor/>
            </y:EdgeLabel>
          </y:PolyLineEdge>
        </data>
   </edge>
   <edge id="e15" source="n6" target="n6">
        <data key="d9">
          <y:PolyLineEdge>
            <y:LineStyle color="#000000" type="line" width="1.0"/>
            <y:Arrows source="none" target="standard"/>
            <y:EdgeLabel   alignment="center"   fontFamily="Dialog"   fontSize="12"
fontStyle="plain"          hasBackgroundColor="false"          hasLineColor="false"
horizontalTextPosition="center"     iconTextGap="4"     preferredPlacement="anywhere"
```

```
textColor="#000000"                                    verticalTextPosition="bottom"
visible="true">stop<y:PreferredPlacementDescriptor/>
             </y:EdgeLabel>
           </y:PolyLineEdge>
         </data>
   </edge>
  </graph>
    <data key="d6">
     <y:Resources/>
   </data>
</graphml>
```