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**

**Башева Владислава Игоревича**

Тема работы:

# ЯДРО WEB-IDE ПРОЦЕСС-ОРИЕНТИРОВАННОГО ЯЗЫКА POST

**«К защите допущена»**
Заведующий кафедрой,
д.т.н., доцент
Зюбин В. Е. /…………..
(ФИО) / (подпись)
«……» ……………… 2022г.

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

**Соруководитель ВКР**
к.т.н. доцент,
КафКТ ФИТ НГУ
Розов А. С. /…………..
(ФИО) / (подпись)
«……» ……………… 2022г.

Новосибирск, 2022

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.

………………….
(подпись)

«17» декабря 2020 г.

**ЗАДАНИЕ**
**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА**

Студент*у* Башеву Владиславу Игоревичу, группы 20222
(фамилия, имя, отчество, номер группы)

Тема Ядро Web-IDE процесс-ориентированного языка poST
(полное название темы выпускной квалификационной работы магистра)

утверждена распоряжением проректора по учебной работе от 17 декабря 2020 г. № 0446
Срок сдачи студентом готовой работы 20 мая 2022 г.
Исходные данные (или цель работы) Расширить функциональность процесс-ориентированного языка poST, разработать ядро языка и реализовать web IDE
Структурные части работы Анализ предметной области, разработка расширения языка poST, реализация ядра языка poST, разработка web IDE, практическая апробация
Консультанты по разделам ВКР (при необходимости, с указанием разделов)

Руководитель ВКР
Зав. КафКТ ФИТ НГУ
д.т.н., доцент
Зюбин В. Е. /…………..
(ФИ О) / (подпись)

«17» декабря 2020 г.

Задание принял к исполнению
Башев В. И. /…………..
(ФИО студента) / (подпись)

«17» декабря 2020 г.

Соруководитель ВКР
Ст. преподаватель КафКТ ФИТ НГУ
б/с
Розов А. С. /…………..
(ФИО) / (подпись)

«17» декабря 2020 г.

# СОДЕРЖАНИЕ

*ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ* ..... 5
*ВВЕДЕНИЕ* ..... 6
*1. Анализ предметной области* ..... 8
**1.1. Введение в специфику задачи** ..... 8
**1.2. Стандарт IEC 61131-3** ..... 8
**1.3. Язык Structured Text** ..... 10
**1.4. Процесс-ориентированный подход программирования** ..... 12
**1.5. Язык poST** ..... 13
**1.6. Анализ недостатков** ..... 14
**1.7. Анализ подходов реализации web IDE** ..... 15
**1.8. Требования к ядру языка poST** ..... 16
*2. Расширение языка poST* ..... 17
**2.1. Генератор в язык Structured Text** ..... 17
**2.2. Генератор в формат PLCopen XML Exchange** ..... 17
**2.3. Шаблонные процессы** ..... 18
**2.4. Библиотеки** ..... 19
*3. Разработка подходов к созданию web IDE* ..... 20
**3.1. Транслятор и web приложение трансляции языка poST** ..... 20
**3.2. Language Server и web IDE языка poST** ..... 21
*4. Определение трансформационной семантики языка poST в язык Structured Text* ..... 22
**4.1. Трансформационная семантика процессов** ..... 22
**4.2. Трансформационная семантика шаблонных процессов** ..... 24
**4.3. Трансформационная семантика библиотек** ..... 25
**4.4. Трансформационная семантика PLCopen XML Exchange** ..... 26
*5. Реализация инструментальных средств* ..... 28
**5.1. Реализация генератора в язык Structured Text** ..... 28
**5.2. Реализация генератора в формат PLCopen XML Exchange** ..... 29
**5.3. Реализация шаблонных процессов** ..... 30
**5.4. Реализация библиотек** ..... 33
**5.5. Реализация транслятора** ..... 34
**5.6. Выбор средств реализации web приложения** ..... 35
**5.7. Реализация web приложения трансляции языка poST** ..... 36
**5.8. Реализация Language Server** ..... 39

**5.9. Выбор платформы web IDE** ... **41**
**5.10. Реализация web IDE языка poST** ... **42**
***6. Практическая апробация лингвистических и инструментальных средств*** ... ***44***
**6.1. Описание алгоритмов тестирования** ... **44**
**6.2. Реализация алгоритма на языке poST** ... **44**
**6.3. Анализ полученных результатов** ... **46**
***ЗАКЛЮЧЕНИЕ*** ... ***47***
***СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ*** ... ***48***
***ПРИЛОЖЕНИЕ А*** ... ***52***
***ПРИЛОЖЕНИЕ Б*** ... ***64***
***ПРИЛОЖЕНИЕ В*** ... ***76***
***ПРИЛОЖЕНИЕ Г*** ... ***84***

# ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

СО РАН — сибирское отделение Российской академии наук.

ПО — програмное обеспечение.

ПЛК — программируемый логический контроллер.

ФБ — функциональный блок.

FBD — Function Block Diagram, язык из состава IEC 61131-3.

SFC — Sequential Function Chart, язык из состава IEC 61131-3.

LD — Ladder Diagram, язык из состава IEC 61131-3.

IL — Instruction List, язык из состава IEC 61131-3.

ST — Structured Text, язык из состава IEC 61131-3.

poST — process-oriented Structured Text.

БНФ — форма Бэкуса-Наура.

AST — Abstract Syntax Tree, абстрактное синтаксическое дерево.

DSL — Domain-Specific Language, проблемно-ориентированный язык.

DSM — Domain-Specific Module, проблемно-ориентированный модуль.

IDE — Integrated Development Environment, интегрированная среда разработки.

LSP — Language Server Protocol.

LS — Language Server.

# ВВЕДЕНИЕ

В настоящее время все большую актуальность приобретает задача разработки киберфизических систем для промышленной автоматизации. Ее решение требует особых методов разработки, языков программирования и инструментальных средств для реализации алгоритмов управления технических систем.

Так, на стыке технологий IEC 61131-3 [1] и процесс-ориентированного программирования [2], появился язык poST (process-oriented Structured Text) [3] – процесс-ориентированное расширения языка Structured Text, или ST [4], из семейства языков IEC 61131-3.

Семейство языков IEC 61131-3 является закрытой экосистемой языков систем промышленной автоматизации. Данные языки используют свои собственные принципы и методы программирования, основанные на использовании комбинаций графических и текстовых языков программирования. Но с момента создания стандарта IEC 61131-3, в 1993 году, ничего принципиально нового не было изобретено, и опыт последних лет показал, что методов разработки IEC 61131-3 не хватает для создания современных систем промышленной автоматизации [5].

В свою очередь, процесс-ориентированного принцип программирования основан на использовании расширенных конечных автоматов, что крайне удобно при написании алгоритмов автоматизации. Принцип разрабатываются и апробируются в институте автоматики и электрометрии СО РАН (ИАиЭ СО РАН), на примере языков программирования Reflex и IndustrialC [6, 7]. Данные языки программирования расширяют язык C, и предназначены для разработки систем промышленной автоматизации. Они уже доказали свою эффективность, простоту написания алгоритмов автоматизации и возможности верификации программ во многих реальных задачах, таких как станки с ЧПУ и системы управления выращивания монокристаллического кремния [8, 9, 10]. Именно поэтому у языка poST есть хорошие возможности стать простым и надежным инструментом разработки для закрытого семейства языков IEC 61131-3.

При распространении языка poST возникает задача обеспечить заинтересованным лицам возможность практического ознакомления с возможностями языка poST. При этом есть осознание, что процедура самостоятельного ознакомления должна быть очень простой, в противном случае, например, при необходимости установки на ПК пользователя дополнительного ПО, число попыток ознакомиться с языком poST будет невелико. Именно поэтому решено сделать web версию среды разработки для языка poST.

Таким образом, была поставлена следующая цель: расширение языка poST, разработка ядра языка для целей облачных технологий и реализация web IDE.

Для достижения данной цели были поставлены и решены следующие задачи:

1. Анализ специфики IEC 61131-3, языка poST и существующих подходов реализации web IDE.
2. Формулировка требований к ядру web IDE и расширению языка poST.
3. Проектирование архитектуры ядра web IDE.
4. Расширение синтаксиса и семантики языка poST.
5. Реализация ядра языка poST.
6. Практическая апробация разработанных лингвистических и инструментальных средств.

Разработанное ядро web IDE и расширение процесс-ориентированного языка poST упрощают использование и изучение методов процесс-ориентированного языка poST и позволяет интегрировать язык poST в системы IEC 61131-3, что сокращает трудоемкость разработки управляющих программ и облегчает сопровождение разрабатываемых алгоритмов.

Работа состоит из введения, шести глав и заключения. В первой главе анализируется предметная область и формируются требования для разрабатываемых инструментальных средств. Вторая глава посвящена вопросу расширения функциональности языка poST. В третьей главе рассматриваются подходы к реализации web IDE. Четвертая глава объясняет трансформационную семантику языка poST в язык ST. Пятая глава посвящена вопросу выбора инструментальных средств разработки и реализации ядра и расширений языка poST. Шестая глава описывает процесс практической апробации разработанных инструментальных средств.

# 1. Анализ предметной области

## 1.1. Введение в специфику задачи

В сфере киберфизических систем и промышленной автоматизации существуют большое количество всевозможных объектов управления, которые используются в разработке систем управления.

«Станки, поточные линии, роботы, гибкие производственные комплексы, технологические процессы, сложное диагностическое и экспериментальное оборудование – примеры объектов, при создании которых используются особые подходы, особые языки и особый понятийный аппарат» [11].

При этом сложность систем управления со временем только увеличивается, появляется все больше специализированных объектов управления и систем детектирования. Создавать управляющие блоки и алгоритмы для каждой отдельной части является достаточно трудной и дорогой задачей, для решения этой проблемы система управления процессами выносится в программируемые логические контроллеры или ПЛК [12], базовый элемент управляющего блока. Специализированные объекты управления соединяются с ПЛК посредствам каналов связи.

Выполняемый на ПЛК алгоритм управления должен следить не только за собственным состоянием, но и учитывать состояния внешних объектов киберфизической системы, корректно реагировать на входящие сигналы от сенсоров и, за определенное время, выдавать выходные сигналы объектам управления. Помимо это важным аспектом работы киберфизической системы является ее реакция на критические ситуации, поскольку любой простой или поломка ее элементов может привести к большим проблемам [13].

Из-за особенностей киберфизических систем и большого количества требований к управляющим алгоритмам, осложняющих разработку программ, зародилось множество различных решений для реализации задачи промышленной автоматизации.

## 1.2. Стандарт IEC 61131-3

Международный стандарт IEC 61131 — является стандартом промышленной автоматизации и состоит из нескольких частей, одна из которых IEC 61131-3, созданная в 1993 году. IEC 61131-3 включает в себя описание и рекомендует к использованию 5 языков программирования.

3 графических языка:

- Function Block Diagram, или FBD — алгоритмы конструируются посредствам использования функциональных блоков. Каждый ФБ имеет входы (слева) и выходы (справа). Программа создается путем соединения множества ФБ.

- Sequential Function Chart или SFC — основан на базе математического аппарата сетей Петри. Описывает последовательность состояний, блоков действий для каждого состояния и условий переходов.
- Ladder Diagram, или LD — представляет собой программную реализацию электрических схем на базе электромагнитных реле.

И 2 текстовых языка:

- Structured Text, или ST — язык программирования общего назначения с Паскале-подобным синтаксисом. Содержит конструкции ветвления, циклов, функций и функциональных блоков.
- Instruction List, или IL — низкоуровневый язык программирования, являющийся аналогом одно-регистрового аппаратно-независимый ассемблер.

Из-за того, что стандарт IEC 61131-3 включает в себя 5 языков программирования, его также называют семейство языков IEC 61131-3.

Поскольку разработка и проектирование систем промышленной автоматизации — это изначально инженерная задача, то часть языков стандарта IEC 61131-3 получили близкий к инженерным системам графический интерфейс. Обусловлено это удобством обучения программированию существующих специалистов и простотой написания алгоритмов для построенных систем [14].

Стандарт IEC 61131-3 дает описание графических или синтаксических конструкций языков и их семантическое объяснение. Но не описывает принципы реализации инструментальных средств. Таким образом зародилось множество компиляторов и сред разработок, использующие разные принципы обработки исходных кодов.

Во времена создания стандарта графические языки программирования имели большую актуальность, но со временем они начали вытесняться текстовыми языками программирования [15]. При этом, описанный в стандарте IEC 61131-3, текстовый язык IL также потерял свою актуальность из-за своей низкоуровневой основы, а оставшийся текстовый язык ST наоборот все больше набирал свою аудиторию.

В результате, IEC 61131-3 является семейством языков программирования, созданным для стандартизации разработки киберфизических систем в сфере промышленной автоматизации, а в последнее время позиционируются как основное средство для систем Industry 4.0 [16]. При этом, семейство языков IEC 61131-3 представляет собой закрытую экосистему языков программирования со своими библиотеками и принципами написания программ.

Помимо 3 части важно также упомянуть часть 10, а именно IEC 61131-10 [17]. Данная часть содержит описание специального формата, созданного для переносимости программ. Описанный формат разработан ассоциацией PLCopen и получил название PLCopen XML Exchange [18]. Первоначальная цель ассоциации заключалась в создании общего формата переносимости программ из одной среды разработки в другую, и пытался решить проблему кросс-брендовой переносимости программ. Данный формат получил большое распространение среди разработчиков и вскоре стал часть стандарта IEC 61131-10. На данный момент XML Exchange формат поддерживают все среды разработки промышленной автоматизации [19].

## 1.3. Язык Structured Text

Язык Structured Text из семейства языков IEC 61131-3 является текстовым императивным высокоуровневым языком программирования со статической типизацией и процедурами. Синтаксис и семантика языка ST является Паскале-подобными.

Данный язык является наиболее универсальным языком программирования в семействе языков IEC 61131-3, поскольку не задает особых правил написания кода, а также позволяет реализовывать как аппаратно-зависимые компоненты, так и аппаратно-независимый код.

В ассортимент операторов языка ST входят все необходимые для программирования конструкции общего назначения: оператор присваивания, условные операторы, циклы и вызовы подпрограмм.

Процедурная составляющая языка заключается в 3 основных конструкциях:

- Программа — исполняемый код, являющийся конечным алгоритмом и загружаемый на ПЛК.
- Функция — фрагмент программного кода, к которому можно обратиться из другого места программы.
- Функциональный блок — фрагмент программного кода с наличием постоянной памяти, к которому можно обратиться из другого места программы. Является наследием графических языков программирования.

Программа, функция и функциональный блок имеют общую внутреннюю структуру, идет разделение на объявление переменных и исполняющийся код.

Объявление переменных вынесены из общего кода в отдельные конструкции следующего вида:

```
VAR <Модификаторы переменных>
    <Объявление переменных>;
```

END_VAR

Ключевое слово VAR также может меняться и определяет тип всех переменных, описанных в данном блоке. Существуют следующие типы переменных:

- VAR — локальные переменные.
- VAR_INPUT — входные переменные.
- VAR_OUTPUT — выходные переменные.
- VAR_IN_OUT — входные и выходные переменные.
- VAR_GLOBAL — глобальные переменные.
- VAR_EXTERNAL — внешние переменные (использование глобальных переменных).
- VAR_TEMP — временные переменные (без сохранения состояния между вызовами).
- VAR_ACCESS — объявление пути доступа.
- VAR_CONFIG — специфическая для инициализации экземпляра и назначение его расположения.

Модификаторы блоков объявления переменных, являются опциональными:

- CONSTANT — модификатор константы.
- RETAIN — модификатор сохраняемых переменных.
- NON_RETAIN — модификатор не сохраняемых переменных.

В совокупности использование вышеописанного способа объявления переменных, 3 программных структур и императивный стиль программирования конструкциями общего назначения дают обширные возможности для написания кода.

Язык ST является достаточно универсальным средством разработки алгоритмов автоматизации, но на практике его используют только в 30% случаев [20]. В основном данный язык используется для описания подпрограмм в виде функций или функциональных блоков и вызывается в других языках программирования [21]. Объясняется это тем, что язык ST, описанный в стандарте IEC 61131-3, является языком программирования общего назначения, и для реализации больших киберфизических систем на одном языке ST необходимо написать достаточно много кода и реализовать хорошую программную архитектуру. Изменить ситуацию с ST поможет переход к другим принципам программирования и введение новых уровней абстракций в язык. Как вариант, переход к объектно-ориентированному стилю программирования [22], но киберфизические системы характеризуются наличием обширным множеством объектов и способов взаимодействия между собой. В таком контексте использование объектно-ориентированного стиля может негативно сказаться на количестве

кода и понимании системы, порождая ненужный код, необходимого только для поддержания архитектуры программы. Поэтому более целесообразно использование других принципов программирования, ориентированных на разработку алгоритмов автоматизации.

## 1.4. Процесс-ориентированный подход программирования

Процесс-ориентированная парадигма программирования — методология программирования предназначения для разработки алгоритмов автоматизации киберфизических систем. Основная идея данной парадигмы заключается в добавлении к языкам общего назначения нового уровня абстракции — процессов и состояний [23], что помогает наиболее эффективно описать систему управления в виде алгоритма.

Теоретическая и практическая основа парадигмы была заложена в Институте Автоматики и Электрометрии СО РАН (ИАиЭ СО РАН) на примере языков программирования Reflex и IndustrialC, которые расширяют синтаксис и семантику языка Си, и на языке poST, который расширяет синтаксис и семантику языка ST. Описанные языки программирования уже продолжительное время успешно используются в реализации различных систем управления и смогли доказать, что принцип процесс-ориентированного программирования позволяет удобно разрабатывать алгоритмы автоматизации, а его структура позволяет легко доказывать корректность написанных программ [24].

Процесс-ориентированная парадигма программирования начинается с определения гиперпроцесса — множество взаимодействующих друг с другом процессов, с организацией общей памяти и средствами синхронизации между процессами.

Процесс — расширенный конечный автомат с набором состояний и переходов. Расширение заключается в наличии остановочных состояний, наличии средствах синхронизации своего исполнения, и возможности контролировать состояние других процессов.

Состояние — состояние конечного автомата с наличием реакции, набором действий, описанный на языке программирования и следующих операторов:

- Перевод текущего процесса в другое состояние.
- Запуск / остановка / остановка по ошибке текущего или другого процесса.
- Проверка нахождения указанного процесса в следующих состояниях: активный, неактивный, остановлен и остановлен по ошибке.
- Таймаут — контроль нахождения текущего процесса в текущем состоянии.

Состояния могут быть как обычными и замкнутыми, не имеющие переходов в другие состояния. Данный модификатор процесса создан для целей упрощения понимания

программы. Модификатор имеет только синтаксическое отличие, отличия в семантике отсутствуют.

Программа в процесс-ориентированной парадигме имеет циклическую основу. На каждой итерации программы процессы выполняют одно из своих состояний, или могут быть в состоянии нормальной остановки или остановки по ошибке. Данное свойство реализуется с помощью добавления стандартных состояний каждому процессу: нормальная остановка — STOP и остановка по ошибке — ERROR. Данные состояния не имеют реакцию и вывести процесс из них могут только другие процессы.

Синхронизация своего состояния реализуется с помощью возможности манипуляции с временными интервалами и таймаутами. Реализуется посредствам сохранения времени начала работы каждого состояния и возможности сбросить это время.

В начале работы программы первый по порядку объявления процесс находится в своем начальном состоянии, чаще всего его называют инициализирующим, остальные процессы находятся в состоянии нормальной остановки и запускаются по мере выполнения программы.

Параллелизм гиперпроцессов в существующих решениях является логическим, то есть процессы выполняются поочередно в одном потоке.

## 1.5. Язык poST

Язык Structured Text, из семейства языков IEC 61131-3, является достаточно мощным инструментом реализации систем управления, но наличие только конструкций общего назначения и отсутствие более высокого уровня абстрагирования усложняет разработку алгоритмов автоматизации, что увеличивает стоимость разработки и затрудняет поддержку написанных программ.

В свою очередь, процесс-ориентированный подход программирования является более удобным текстовым средством разработки алгоритмов автоматизации. Обусловлено это наличием нового уровня абстрагирования — процессами.

Язык poST является объединением двух вышеупомянутых инструментов: он является расширением языка ST, что дает возможность интегрировать данный язык в существующие системы IEC 61131-3, а также он является процесс-ориентированным языком программирования, что, в свою очередь, позволяет упростить процесс разработки и поддержки управляющих программ.

Таким образом язык poST — процесс-ориентированное расширения языка Structured Text (ST) из семейства языков IEC 61131-3. Предназначен для создания программного обеспечения Программируемых Логических Контроллеров (ПЛК) и ориентирован на решение задач промышленной автоматизации.

Для языка poST сделано: базовое ядро, генератор в язык C, плагин для Eclipse IDE и отдельный Java JAR с транслятором в язык C. Ядро включает в себя: парсер, представление абстрактного семантического дерева или AST, синтаксические и семантические проверки, а также функционал необходимый для IDE в виде подсветки синтаксиса, выделения ошибок, автодополнения, навигации и другое. Разработка ядра производится с применением Фреймворка Xtext [25].

Инструментальные средства языка poST делает данный язык самостоятельным, без связи с системами IEC 61131-3. Также в языке отсутствуют библиотеки, а при практической апробации процесс-ориентированного программирования была обнаружено, что в языке возможно наличие повторяющегося кода при дублирующихся элементов в системе, что противоречит введению нового уровня абстракция для упрощения кода.

## 1.6. Анализ недостатков

Язык poST уже смог доказать свою эффективность в ряде задач. Упрощение процесса разработки алгоритмов, улучшение читаемости кода и упрощение поддержки. Но во время практической апробации было выявлено ряд проблем.

Поскольку язык poST является расширением языка ST из семейства IEC 61131-3 и рассчитан на системы поддерживающие данные языки программирования, то целесообразно переиспользовать системы сбора данных языков программирования. Но к сожалению язык poST является транслирующим в язык C.

Также в экосистеме IEC 61131-3 существуют форматы переносимости программ под названием PLCopen XML Exchange. Данный формат создан ассоциацией PLCopen и с недавних времен стал частью стандарта IEC 61131-10. В связи с этим, XML Exchange поддерживают все существующие специализированные среды разработки. Но язык poST не поддерживает данный формат.

Во время практической апробации языка poST также было выявлено, что, при создании систем с повторяющимися элементами, такие как лифт, появляется множество одинаковых процессов, где меняются только используемые переменные [26]. Приходится копировать код, меняя только часть код и не трогая логику процессов. Это негативно сказывается на эффективности языка.

Другой выявленной проблемой было отсутствие библиотек. Языки IEC 61131-3 являются закрытой экосистемой со своими библиотеками. Также существуют стандартная библиотека с обширным набором функций и функциональных блоков. Математические функции, функции работы со временем, обработка сигналов, преобразование типов. Всех этих

функций нет в языке poST, как и механизма библиотек, что может вызвать дискомфорт при разработке алгоритмов.

При распространении языка poST также встречается проблема легкого доступа к языку. Существующий плагин для Eclipse IDE с языком poST позволяет пользоваться всем его функционалом, но это требует дополнительной установки программного обеспечения на компьютеры пользователей, а это уменьшает количество заинтересованных пользователей.

Один из способов решения последней проблемы является использование облачных технологий и создание web сред разработки.

### 1.7. Анализ подходов реализации web IDE

При анализе подходов реализации web IDE было выделено 2 способа: реализация собственного протокола, что является крайне сложность задачей, или использование готового протокола работы IDE.

Использование первого способа позволяет собственноручно определять функциональность web IDE, что можно использовать для уменьшения сложности системы и дает возможность реализовать простое и универсальное web средство поддержки языка poST.

Второй способ является более сложным, поскольку готовые протоколов является унифицированными и требует полной реализации его функциональности. Основным протоколом для web IDE и легковесных локальных IDE является Language Server Protocol [27].

Language Server Protocol, или LSP — открытый протокол передачи информации между редактором кода и языковым сервером.

Language Server, или LS — языковой сервер, который передает информацию о языке через LSP. Вся информация передается только по протоколу, то есть LS не смотрит ни на файловую систему, ни на другой источник.

При запуске LS получает все необходимые файлы, далее в протоке передаются только изменения, введенные пользователем. В ответ LS отправляет информации о языке, включая выделение ошибок и предупреждений, автодополнение для текущего места, навигацию по коду, доступ к документации и многое другое. Подсветка синтаксиса не входит в решаемые задачи LSP и решается средствами редакторов, поскольку подсветка синтаксиса требует большого количества сообщений и нагружает соединение. Помимо передачи информации о языке, LSP поддерживает и другие запросы по типу вызова рефакторингов кода, поиска, а также возможно расширение видов запроса через LSP.

Использование LS с LSP позволяет абстрагироваться от платформы и операционной системы, позволяя создавать легковесные интегрированные среды разработки или IDE, где

интерфейс представляется в виде электронной web страничке, а вся обработка происходит на стороне LS.

В последнее время такие легковесные IDE получаются все большее распространение. На рынке присутствует множество примеров: VS Code, Atom, Vim, некоторые проекты Eclipse [28]. Также с применением технологий LSP появляется возможность создавать web IDE, поскольку данная технология является схожей с протоколами интернета.

В случае web IDE, с поддержкой LSP, ядром языка является LS, который общается с редактором кода и предоставляет всю функциональность IDE. При этом в глобальном проекте языка poST разрабатывается микросервисная архитектура web IDE с поддержкой проблемно-ориентированных модулей языка, данная архитектура не входит в данную работу, но требует поддержки ее реализации в ядре языка.

## 1.8. Требования к ядру языка poST

Для ядра языка poST были сформулированы следующие требования, охватывающие как развитие самого языка, так и инструментальные средства поддержки языка:

- Интеграция языка poST в системы IEC 61131-3
- Поддержка шаблонных процессов
- Поддержка библиотек IEC 61131-3
- Возможность создавать проблемно-ориентированные модули:
    - Библиотека для обработки AST
    - Сериализация и десериализация AST
- Поддержка создания:
    - Eclipse plug-in
    - Запускаемого JAR с транслятором языка
    - LS с основными функциями LSP

## 2. Расширение языка poST

### 2.1. Генератор в язык Structured Text

При практической апробации языка poST возникала проблема с отсутствием возможности переиспользования инструментальных средств IEC 61131-3, поскольку язык poST транслировался только в язык C, после чего использовались компиляторы языка C.

Для решения этой проблемы было предложено сделать транслятор в исходный язык ST и дальше использовать системы сборок IEC 61131-3. Данный подход дает возможность интеграции языка poST в уже существующую экосистему разработки алгоритмов управления, а не создавать альтернативное течение.

Процесс-ориентированный подход программирования добавляет новый уровень абстракции в виде процессов и состояний, которые реализуются с помощью конструкций общего назначения. Конечные автоматы реализуются с помощью операторов множественного ветвления "CASE" и язык ST имеет данный оператор.

Поскольку язык poST расширяет язык ST и не меняет основные конструкции, то трансляция данных конструкций производится идентично.

Проблема остается только с переменными, поскольку введение нового уровня абстракций в язык создает новые области видимостей для переменных. Появляется возможность создавать переменные с одинаковыми именами в разных процессах и с разными областями видимости. Проверка семантики новых переменных производится средствами языка poST, но существует возможность коллизий имен переменных в сгенерированном ST коде. В языке ST описание переменных является глобальным, и ST не позволяет инкапсулировать переменные для конкретной конструкции "CASE". Для решения вопроса коллизий предлагается ввести префикс для переменных, таким образом получается реализовать взаимно однозначное отношение для имен переменных в исходном коде на языке poST и в генерируемом коде на языке ST.

### 2.2. Генератор в формат PLCopen XML Exchange

Помимо создания транслятора в язык ST необходимо поддержать и другую альтернативу, а именно переиспользовать формат PLCopen XML Exchange. Данный формат создан для кросс-брендовой переносимости программ в сфере промышленной автоматизации. Использование данного формата дает возможность бесшовной интеграции кода на языке poST в специализированные средства IEC 61131-3.

PLCopen XML Exchange за основу использует язык разметки XML [29] и структурирует формат документа с помощью специальных тегов. Структура формата позволяет сохранять код на всех языках семейства IEC 61131-3. Формат XML был выбран с целью сохранения

графических языков с точностью до координат каждого элемента. Текстовые языки хранятся в своем родном текстовом формате. Это что дает возможность сохранять сгенерированный ST код в данном формате.

При этом в формате XML Exchange отдельное место выделяется для описания используемых переменных. Существуют специальные теги для описания переменных: их имен, типов и начальных значений. Тело же исполняемого кода на языке ST хранится в текстовом формате.

## 2.3. Шаблонные процессы

Другой проблемой, обнаруженной при практической апробации языка poST, было наличие одинаковых процессов. Если в системе есть дублирующие элементы, а когда речь идет о системах автоматизации, такие элементы встречаются достаточно часто, то приходится копировать процессы с изменением только используемых переменных. Это является достаточно большим минусом языка. В процесс-ориентированной парадигме нельзя вынести часть функционала в отдельную подпрограмму, как, например, в функцию в процедурной парадигме.

Для решения этой проблемы было предложено использовать шаблонные процессы. Шаблонные процессы, в отличие от обычных, не являются частью исполняемого кода, а существуют только для переопределения с использованием конкретных переменных.

В структуру шаблонного процесса были введены: входные / выходные переменные и переменные процессов, область видимости других переменных не распространяются на шаблонные процессы. Входные / выходные переменные являются обычными переменными IEC 61131-3, а переменные процессов является нововведением. Поскольку в процесс-ориентированной парадигме все процессы взаимодействуют друг с другом, то есть могут запускать, останавливать и проверять состояние друг друга, то для поддержания данной парадигмы было предложено добавить переменные процессов. Такие переменные переопределяются реальными процессами и далее взаимодействие процессов происходит именно через эти переменные.

Далее при конфигурировании программы, вместе с переопределением входных / выходных переменных программы, появилась возможность дополнительно определять процессы из шаблонных процессов. При определении процесса определяются входные / выходные переменные для текущего процесса и переменные других процессов.

Введение шаблонных процессов позволяет избавиться от дублирования кода и дает возможность менять переменные в дублирующих элементах.

## 2.4. Библиотеки

При анализе IEC 61131-3 и средств его поддержки в виде средств разработки CodeSys Beremiz, 4diac [30, 31, 32], и производных от них, был выделен схожий способ реализации библиотек. Все библиотечные элементы в данных средах разработки хранятся в отдельных файлах. При этом CodeSys хранит в зашифрованных архивах, а Beremiz и 4diac в XML файлах в открытом доступе [33].

Также был выделен список стандартных функций и функциональных блоков. Список содержит 331 функцию и 21 функциональный блок [34]. В него входят функции: арифметические, числовые, операции работы с битами, операции работы с текстовой строкой, нахождения границы сдвига сигнала, сравнивания, преобразования. При этом большую часть занимают функции преобразования типов. Также существуют generic функции, которые могут принимать разные типы данных, но выполняют одинаковые действия. В список также входят функциональные блоки: таймеры, счетчики и определения изменения сигнала.

При анализе возможных решений добавления данного списка в язык poST, было решено сделать как в средстве разработки Beremiz и 4diac. Библиотеки хранятся отдельно от ядра и подключаются посредством обработчика библиотек. Каждая отдельная функция и функциональный блок помещается в xml файл, в котором содержится интерфейсная информация. Для удобства, все файлы сгруппированы по операциям, которые они выполняют и разделены по директориям. Xml файлы созданы в формате PLCopen XML exchange. В каждом файле находится один тег <pou> с именем и типом. Тип может быть или "function" или "functionBlock". Внутри тега <pou> находятся входные и выходные функции. В случае функции добавляется дополнительный тег возвращаемого типа. Тег <body> не используется.

Данное решение было принято по причине использования других библиотек, например, библиотек CodeSys. Чтобы добавить в poST программу стороннюю библиотеку, достаточно добавить в папку с библиотеками xml файлы в формате PLCopen XML Exchange.

# 3. Разработка подходов к созданию web IDE

## 3.1. Транслятор и web приложение трансляции языка poST

При анализе вариантов создания средств поддержки языка poST с помощью облачных технологий было предложено 2 способа.

Первый способ — это использовать запускаемый Java архив, или JAR [35], с транслятором языка в другом программном обеспечении. Например, использовать Фреймворки для разработки web приложений. Реализовать не сложную web страничку с окном для ввода poST кода, передавать введенный текст на вход JAR транслятору и получать логи транслятора, а в случае успешной трансляции также получать сгенерированный ST код и PLCopen XML Exchange файл.

Данный способ является легким в реализации и позволяет создать простое web приложение трансляции языка poST. Способ не поддерживает функциональность полноценных интегрированных сред разработки такие как подсветка синтаксиса, выделение ошибок, автодополнение и другое, но позволяет достаточно удобно и быстро транслировать код на языке poST в язык ST и формат PLCopen XML Exchange, тем самым упрощая процесс использования языка poST и ознакомления с языком poST.

Для web приложение трансляции языка poST зарегистрировано доменное имя: http://post2st.iae.nsk.su [36].

Сформулирован ряд требований к интерфейсу и функциональности данного web приложения:

- Функциональные возможности приложения должны быть доступны пользователю только через веб-браузер с активным сетевым подключением.
- Независимость веб-браузера от веб-приложения.
- Временное хранение результатов работы пользователя и восстановление состояния после перезагрузки web страницы.
- Пользовательский интерфейс должен обеспечивать:
    - Размещать примеры и шаблоны.
    - Окно редактора для poST программы.
    - Загрузка poST программы с локального ПК пользователя в окно редактора.
    - Загрузка poST программы, сгенерированной ST программы и файла PLCopen XML Exchange.
    - Окно для сообщений об ошибках и предупреждениях транслятора.
    - Форма обратной связи с пользователем.

## 3.2. Language Server и web IDE языка poST

Второй способ заключается в использовании Language Server Protocol. Реализовать Language Server для языка poST и интегрировать в существующие платформы web IDE.

Данный способ имеет большой плюс в виде полноценной поддержки функциональности интегрированной среды разработки. Подсветка синтаксиса, выделение ошибок и предупреждений в коде, автодополнение, навигация по коду и многое другое.

Вся информация о языке передается по средствам LSP, что дает возможность инкапсулировать данный протокол в протоколы интернета.

Для web IDE языка poST зарегистрировано доменное имя: http://post.iae.nsk.su.

# 4. Определение трансформационной семантики языка poST в язык Structured Text

## 4.1. Трансформационная семантика процессов

Для интеграции языка poST в системы IEC 61131-3 было предложено реализовать генератор в исходный язык ST и переиспользовать сгенерированный код в данных системах. За основу был взят генератор в язык C, поскольку данный язык имеет схожую структуру с языком ST. Для реализации генератора в язык ST была предложена следующая трансформационная семантика.

Основой гиперпроцесса, из процесс-ориентированного подхода программирования, стала программа и функциональные блоки. Функции остались без изменений, то есть описываются на исходном языке ST и транслируется эквивалентно. В зависимости от необходимости использовать функциональность процессов в системах IEC 61131-3 можно реализовать программу, как финальный алгоритм системы, а также функциональный блок, как подпрограмму, используемую в других частях алгоритма. Данный подход позволяет использовать язык poST как самостоятельный язык, так и как часть семейства IEC 61131-3.

Далее программы и функциональные блоки будут упоминаться как программные элементы.

Гиперпроцесс реализуется в рамках одного программного элемента. Изначально гиперпроцесс является множеством параллельно исполняющихся процессов, но в данном случае параллелизм является логическим, то есть преобразуется в поочередное исполнение процессов в порядке их написания в исходном коде на языке poST. Данный принцип использовался и в трансформационной семантике языка C.

Процесс является конечным автоматом с набором состояний, переходов и 2 дополнительными неактивными состояниями. Трансформационная семантика в язык ST представляет собой использование оператора множественного выбора "CASE" над целочисленными значениями. Каждому состоянию конкретного процесса определяется своя численная константа в локальных переменных программного элемента, имя для константы определяется в верхнем регистре следующем образом: "_P_<Имя процесса>_S_<Имя состояние>". Для каждого процесса нумерация состояний начинается с 0 и инкрементируется на 1 с каждым новым состоянием процесса в порядке написания в исходном коде. Операндом оператора "CASE" является локальная не константная числовая переменная, для каждого процесса соответственно, и хранит числовое значение с текущем состоянием процесса.. Имя операнда определяется следующем образом: "_g_p_<Имя процесса>_state", с сохранением регистра имени процесса в исходном коде. Дополнительные неактивные состояния выносятся

в отдельные константы "_STOP" и "_ERROR" с числовыми значениями 254 и 255 соответственно. Таким образом накладывается верхнее ограничение на количество состояний в процессе со значением не более 253 состояния.

Состояния представляют из себя блок кода на языке ST внутри оператора "CASE", определённого после числового значения "_P_<Имя процесса>_S_<Имя состояние>:".

Дополнительно вводится локальная переменная с именем "_global_time" и типом "TIME", предназначенная для сохранения времени начала работы текущей итерации программного элемента. В начале исполняемого кода программного элемента данная переменная инициализируется временным значением, полученное вызовом функции "TIME()" из стандартной библиотеки IEC 61131-3 и возвращающая прошедшее время с момента запуска ПЛК.

В случае если одно или несколько состояний процесса используют временной контроль, то для процесса вводится дополнительная временная переменная с типом "TIME" и именем определенным следующем образом: "_g_p_<Имя процесса>_time", с сохранением регистра имени процесса в исходном коде. Данные переменные используются для сохранения времени запуска состояний процесса. Для сохранения используется время активации текущей итерации программного элемента из переменной "_global_time".

Типы переменных, описанные на языке poST, преобразуются эквивалентно.

Типы блоков переменных, описанные на языке poST, преобразуются эквивалентно.

Инициализация переменных, описанные на языке poST, преобразуются эквивалентно.

Имена переменных программных элементов, описанных на языке poST, преобразуются эквивалентно.

Имена переменных процессов, описанных на языке poST, преобразуются с изменениями. Вводится система префиксов, для исключения варианта коллизии имен. Именам переменных, описанных в процессах, дополнительно вводится префикс и имя определяется следующим образом: "_p_<Имя процесса>_v_<Имя переменной процесса>", с сохранением регистра имени процесса и имени переменной в исходном коде.

Операторы RESTART / START PROCESS <Имя процесса> преобразуются в выставление начального состояния переменной текущего или указанного процесса. В случае если начальное состояние текущего или указанного процесса обладает временным контролем, то дополнительно в переменную времени запуска процесса записывается значение "_global_time".

Операторы STOP/STOP PROCESS <Имя процесса>, ERROR/ERROR PROCESS <Имя процесса> преобразуются в выставление переменной состояния, текущего или указанного процесса, значения "_STOP" или "_ERROR" соответственно.

Операторы SET STATE <Имя состояния> преобразуются в выставление соответствующего значения соответствующей переменной состояния текущего процесса. Операторы SET NEXT определяют следующее по порядку состояние, выставляют определенное значение соответствующей переменной состояния текущего процесса. В случае если указанное состояние текущего процесса обладает временным контролем, то дополнительно в переменную времени запуска текущего процесса записывается значение "_global_time".

Операторы PROCESS <Имя процесса> IN STATE <Ключевое слово> преобразуются в логическое выражение сравнения соответствующей переменной состояния указанного процесса с выделенными состояниями нормальной остановки STOP и остановки по ошибке ERROR в зависимости от ключевого слова:

- ACTIVE — не в состоянии STOP и не в состоянии ERROR.
- INACTIVE — в состоянии STOP или в состоянии ERROR.
- STOP — в состоянии STOP.
- ERROR — в состоянии ERROR.

Операторы RESET TIMER обновляют переменную страта состояния текущего процесса значением "_global_time".

Операторы TIMEOUT преобразуются в условный оператор "IF", условием которого становится логические выражение: глобальное время ("_global_time") минус время старта состояние больше условия оператора TIMEOUT. Тело данного оператора переносится в тело условного оператора "IF".

Оставшиеся конструкции общего назначения языка ST преобразуются эквивалентно, с учетом добавления префиксов переменным процессов.

Для сохранения результата генерации используется текстовый файл с расширением ".st".

## 4.2. Трансформационная семантика шаблонных процессов

Для решения проблемы повторяющегося кода у дублирующих элементов было предложено ввести механизм шаблонных процессов. Шаблонные процессы не преобразуются напрямую в ST код, а только используются для объявления новых процессов в конфигурации. Объявление шаблонных процессов происходит в конфигурации в разделе привязки программ к задачам. Для шаблонных процессов была предложена следующая трансформационная семантика.

Структура конфигурации, описанная на языке poST, преобразуются эквивалентно.

Структура глобальных переменных, описанных на языке poST, преобразуются эквивалентно.

Задачи, описанные на языке poST, преобразуются эквивалентно.

Привязка программ, описанные на языке poST, преобразуется с изменениями. Привязка входных / выходных переменных программных элементов преобразуется эквивалентно. В случае если в привязке программы используются инстанцирование шаблонных процессов, происходят изменения в исходном коде программного элемента.

Исходный программного элемента копируется, имя программного элемента изменяется на имя, описанное в привязке программы. Шаблонные процессы заменяются на инстанцирование процессы с полным сохранением логики, при этом имя процесса берется из его инстанцирования.

Предопределенные переменные программ и процессов заменяются в исходном коде программных элементов. Входные / выходные переменные и переменные процессов убираются из сигнатуры программного элемента, на их места подставляются значения из конфигурации. Контакты поставляются напрямую, использование обычных переменных заменяется на использование глобальных переменных, использование переменных процессов заменяются на инстансы процессов, а далее на переменные состояния процессов.

### 4.3. Трансформационная семантика библиотек

Для интеграции библиотек IEC 61131-3 в язык poST было предложено реализовать механизм библиотек и поддержать стандартную библиотеку IEC 61131-3. Для библиотек была предложена следующая трансформационная семантика.

Библиотечные элементы хранятся в XML файлах в формате PLCopen XML Exchange, для поддержания общего стандарта и простоты импортирования библиотек в язык poST. Ядро языка считывает библиотечные элементы из файлов в начале работы и при изменении файлов, и использует считанную информацию при работе. Считывается только интерфейсная информация в виде имени и типе программного элемента, а также опциональную информацию о входных / выходных переменных и возвращаемом типе. Файлы хранятся по соседству от ядра в директории с именем "Tool Library". В случае Eclipse IDE, в одном poST проекте, в случае web IDE, в одном рабочем пространстве, в случае отдельного транслятора, рядом с JAR файлом.

Объявление функциональных блоках, описанных на языке poST, в поле локальных переменных преобразуются эквивалентно. В случае если функциональный блок объявлен в локальных переменных процесса, в силу вступает правило префиксов для имен переменных. Вызов функциональных блоков приводится к унифицированному виду, где переопределение

входных / выходных переменных функционального блока происходит после вызова самого функционального блока, с учетом добавления префиксов переменным процессов.

Вызов функций преобразуется эквивалентно, с учетом добавления префиксов переменным процессов.

### 4.4. Трансформационная семантика PLCopen XML Exchange

Для бесшовной интеграции языка poST в системы IEC 61131-3 было предложено реализовать генератор в формат PLCopen XML Exchange с применением языка ST, данный способ позволяет автоматически импортировать кода в специализированные системы IEC 61131-3. За основу был взят генератор в язык ST. Для реализации генератора в формат PLCopen XML Exchange была предложена следующая трансформационная семантика.

Основными элементами трансформационной семантики являются теги и атрибуты XML разметки, стандартизированные форматом PLCopen XML Exchange.

Все программные элементы исходного кода на языке poST инкапсулируются тегом "<pous>" без атрибутов.

Каждый отдельный программный элемент исходного кода на языке poST инкапсулируется тегом "<pou ...>", дополнительно указываются атрибут "name", для указания имени программного элемента, а также атрибут "pouType", для указания типа программного элемента. Типом может быть программа, функциональный блок или функция.

Внутри тега "<pou>" описывается тег "<interface>", для описания интерфейса программного элемента. В интерфейс включаются опциональные теги: входные переменные "<inputVars>", выходные переменные "<outputVars>", локальные переменные"<localVars>" и временные переменные "<tempVars>". Для тегов переменных существует опциональный атрибут "constant" для указания, что блок переменных является константным. Интерфейс дополнительно может содержать опциональный тег "<returnType>", используемый для функций и обозначающий возвращаемый тип.

Также тег "<pou>" содержит тег "<body>", содержащий тело исполняемого кода программного элемента на языках IEC 61131-3. В случае ST внутри тега "<body>" используется тег "<ST>". В данной секции описывается исполняемый код, полученный в результате трансляции языка poST в язык ST, в текстовом формате. Для поддержания текстовой разметки XML в сгенерированном ST коде заменяются символ "<" на "&lv;", а символ ">" на ">"

Снаружи тега "<pous>" возможно объявление опционального тега "<addData>", который содержит дополнительную информацию. Дополнительная информация содержит опциональный тег "<globalVars ...>", в котором описываются глобальные переменные

исходного кода на языке poST. Для глобальных переменных используется атрибут "name" для указания имени файла для хранения глобальных переменных.

Для описания переменных в блоках используется тег "<variable>" с атрибутом "name", для указания имени переменной. Тег "<variable>" содержит обязательный тег "<type>", для указания типа переменной и опциональный тег "<initialValue>", для описания начального значения. Тег "<type>" содержит внутри себя тег с именем типа переменной. Тег "<initialValue>" содержит тег "<simpleValue>" с атрибутом "value", в котором описывается константа или выражение.

Помимо описанных тегов используются также теги необходимые для поддержания общей структуры формата PLCopen XML Exchange. Тег "<project>" атрибутом "xmlns" и значением "http://www.plcopen.org/xml/tc6_0200". Тег "<fileHeader>" с атрибутами "companyName" с пустым значением, "productName" со значением "poSTIDE", "productVersion" с пустым значением и "creationDateTime" со значением даты и времени генерации XML файла. И стандартные обязательные теги с указанием размеров графических языков.

Для сохранения результата генерации используется текстовый файл с расширением ".xml".

## 5. Реализация инструментальных средств

### 5.1. Реализация генератора в язык Structured Text

Генераторация ST кода является последним этапом трансляции языка poST. На вход генератору подается обработанное абстрактное синтаксическое дерево, или AST, прошедшее этапы синтаксических проверок, линковки поддеревьев и семантического контроля [37]. AST подается обернутое в специальный формат Eclipse Modeling Framework, или EMF [38], предоставляемый Фреймворками Eclipse. В таком состоянии AST является деревом семантического разбора и содержит всю необходимую информацию о синтаксисе отдельного узла, связанной с узлом семантической информацией, а также информацию о местоположении узла в исходном коде на языке poST. На выходе ожидаются артефакты в виде файлов с расширением ".st" с сгенерированным ST кодом. Для удобства изложения, далее семантическое дерево разбора в формате EMF будет упоминаться как AST.

Для реализации генератора в исходный язык ST был взят генератор в язык C и модифицирован. Архитектура генератора была спроектирована на основе представления AST, для упрощения процесса генерирования кода по дереву. Все узлы AST были скопированы в классы генератора и к их именам добавлен постфикс "Generator", конструктор таких классов принимает объекты оригинальных узлов, конструктор извлекает информацию из узлов и создает детей в виде классов генератора, в точности, как и в оригинальном AST. Каждому классу генератора добавлен метод "generate", который по полученной информации из оригинального AST генерирует необходимый текст. Таким образом, генератор является копией исходного AST, с наличием метода генерации текста.

В случае, когда необходимо добавить новые узлы дерева генератора, например, при добавлении дополнительных переменных процесса, происходит модификация дерева генератора, без модификации исходного AST, на этапе обработки информации узлов исходного AST. При обработке процессов логика алгоритма поднимается выше по узлам генератора до генератора программного элемента и добавляет в переменные программного элемента локальные переменные процесса и дополнительные переменные процесса. Также автоматически добавляются общие константы "_STOP" и "_ERROR" и переменная "_global_time". Данное действие нарушает концепцию полной копии исходного AST, но необходимо для реализации трансформационной семантики языка poST в язык ST.

Генерация ST кода происходит в 2 этапа. На первом этапе происходит инстанцирование дерева генератора по исходному AST программы на языке poST. На втором этапе происходит генерация ST кода: вызывается метод "generate" у корня дерева генератора, который инициирует обход дерева генератора в глубину с применением модифицированного метода

обратного обхода. При посещении узла дерева генератора происходит генерация текста для текущего узла, при переходе к родителям сгенерированный текст детей соединяется, добавляя дополнительный текст при необходимости. Таким способом, обход дерева генератора методом обратного обхода позволяет генерировать корректный ST код по исходному poST коду без лишних трудозатрат.

Конструкции языка poST, унаследованные из языка ST, генерируются эквивалентно. Процесс-ориентированные конструкции и операторы генерируются по трансформационной семантике. Объявление и использование переменных генерируются по трансформационной семантике с добавлением дополнительных переменных.

Для реализации системы префиксов переменных дополнительно реализована карта преобразований имен переменных. При объявлении переменных процессов автоматически добавляется префикс к имени переменной. При обработке узлов, где происходит обращение к переменным, логика алгоритма поднимается выше по узлам генератора до генератора процесса, в случае если используемая переменная находится в выше расположенном процессе, к ней добавляется префикс по трансформационной семантике, при альтернативном варианте имя используемой переменной сохраняется. Область видимости переменных, трансформационная семантика и семантический контроль языка poST, проведенный до этапа генерации кода, гарантирует корректность данного преобразования.

Генерация кода гарантирует корректное форматирование ST кода, с сохранением всех табуляций.

На последнем этапе генератора создается файл с расширением ".st" в который записывается сгенерированный ST код.

## 5.2. Реализация генератора в формат PLCopen XML Exchange

Генераторация PLCopen XML Exchange файла аналогично является последним этапом трансляции языка poST и является альтернативным генератором. На вход генератору подается семантическое дерево разбора в формате EMF или AST. На выходе ожидаются артефакты в виде файлов с расширением ".xml" с сгенерированной XML разметкой.

Для реализации генератора в формат PLCopen XML Exchange был взят и пере использован генератор в исходный язык ST. Архитектура генератора была спроектирована аналогично генератору в ST код. Изменен генерируемый текст для объявления переменных и оболочки программных элементов. Для тела исполняемого кода был переиспользовать код для генерации ST кода, поскольку исполняемый код хранится в текстовом формате, с заменой символов "<" на "&lv;" и символ ">" на ">" после генерации текста.

Программные элементы генерируются по трансформационной семантике. Объявление переменных генерируются по трансформационной семантике.

Для поддержания формата и структуры PLCopen XML Exchange добавляется дополнительный неизменяемый текст с тегами, необходимый для указания размеров графических языков, выбраны единичные размеры:

```
<contentHeader name="poST.project">
    <coordinateInfo>
        <fbd>
            <scaling x="1" y="1" />
        </fbd>
        <ld>
            <scaling x="1" y="1" />
        </ld>
        <sfc>
            <scaling x="1" y="1" />
        </sfc>
    </coordinateInfo>
</contentHeader>
```

Для секции с указанием времени генерации файла берется актуальное время системы в формате "ГГГГ-ММ-ДДТчч-мм-сс-лллллл", где Г — год, М — месяц, Д — день, ч — часы, м — минуты, с — секунды, л — миллисекунды.

Генерация кода гарантирует корректное форматирование XML файла, с сохранением всех табуляций.

На последнем этапе генератора создается файл с расширением ".xml" в который записывается сгенерированная XML разметка.

### 5.3. Реализация шаблонных процессов

Для реализации конфигурирования poST программ было сделано:

Грамматика языка poST была доработана. Добавлены правила и терминалы для элементов конфигурации, изменена грамматика для процессов, добавлен новый тип переменных — переменные процессов.

Доработана грамматика для конфигурации, в элементы привязки программ добавлено инстанцирование шаблонных процессов:

```
ProgramConfiguration:
    'PROGRAM' ID ('WITH' Task)? ':' Program ('(' ProgramConfElements ')')?
```

ProgramConfElements:

ProgramConfElement (',' ProgramConfElement)*

ProgramConfElement:

AttachVariableConfElement | TemplateProcessConfElement

Разработаны грамматические правила и новые терминальные символы для описания инстанцирования шаблонных процессов:

TemplateProcessConfElement:

'PROCESS' ('ACTIVE')? ID ':' Process ('(' TemplateProcessElements ')')?

TemplateProcessElements:

TemplateProcessAttachConfElement (',' TemplateProcessAttachConfElement)*

TemplateProcessAttachConfElement:

Variable AssignmentType (Variable Constant)

Изменено правило процесса, добавлена возможность использования входных / выходных переменных и переменных процессов:

Process:

'PROCESS' ID

(

nputVarDeclaration |

OutputVarDeclaration |

InputOutputVarDeclaration |

ProcessVarDeclaration |

VarDeclaration |

TempVarDeclaration

)*

(State)*

'END_PROCESS';

Для переменных процессов добавлены правила объявления данных переменных:

ProcessVariable:

ID

ProcessVarList:

ProcessVariable (',' ProcessVariable)*

ProcessVarInitDeclaration:

ProcessVarList ':' Process

ProcessVarDeclaration:

'VAR_PROCESS'

(ProcessVarInitDeclaration ';')*

'END_VAR'

Синтаксические проверки и проверки линковки были автоматически сгенерированы Фреймворком Xtext. Для более глубокого семантического анализа новых элементов грамматики были созданы дополнительные семантические проверки:

- Проверка на коллизии имен инстанцированых процессов
- Проверка на регистр первой буквы имени инстанции рованного процесса
- Проверка на привязку переменных при инстанцировании процесса
  - Проверка на тип блока переменной
  - Проверка на тип переменной
- Проверка на используемые шаблонные процессы
- Проверка коллизии имен в области видимости переменных процессов
- При реализации элементов конфигурации было выявлено, что стандартный линковщик Фреймворка Xtext не может определить объявление и использование элементов в разных ветках AST. Например, при инстанцировании процесса переменные процесса не определялись.

Стандартные средства Xtext создают для каждого узла AST уникальное имя. Имена генерируются посредствам перехода от текущего узла к его родителю и так до корня AST. Иначе говоря, имена всех родителей и текущего узла формируют уникальное имя. Когда в AST встречается ссылка на другой объект, то используется линковщик. Линковщик получает список потенциальных объектов по контексту, генерирует уникальное имя по схожему принципу, но в текущем узле, и сравнивает полученное имя с именами всех элементов полученного списка. Поскольку инстанцирование процесса и определение процесса находятся в разных ветвях AST, то уникальные имена для определений и инстанцирований генерировались разные и линковщик не находил нужный элемент в полученном списке претендентов. Для решения данной проблемы было сделано следующее:

Система генерации уникальных идентификаторов узлов AST была откорректирована: ранее для генерации использовалась стандартная функция Xtext, была введена новая функция, которая генерирует одинаковые идентификаторы для узлов AST. В случае определения переменной в программе, в шаблоне процесса и в случае объявления (привязке, инстанцировании) переменной в конфигурации. Это позволяет в дальнейшем связывать (линковать) узлы при синтаксических-семантических проверках и кодогенерации.

Откорректирован механизм получения списка объектов по контексту (области видимости). Ранее по запросу выдавался список всех переменных, формирование списка была

переработана с учетом контекста. Например, при параметризации процесса в конфигурации, для формирования списка входных / выходных переменных: сначала находится узел в AST с декларацией (объявлением) процесса с текущим именем процесса, и формируем список из его переменных.

Откорректирована система линковки (установления связи между узлами с одинаковыми идентификаторами) с использованием новой системы генерации уникального имени для ссылки, получения списка претендентов и сравнения уникального имени для текущего узла и списка претендентов. Например, в узле с объявлением (инстанцированием) добавляются ссылки на определения переменных.

Для целей кодогенерации, было решено изменять AST. После парсинга, линковки, синтаксических и семантических проверок, AST изменятся. Программные элементы с шаблонными процессами удаляются из AST и создаются новые ветви с программными элементами. Создание новых ветвей происходит по инстанцированным программным элементам в конфигурации. Все переопределенные входные / выходные переменные программного элемента заменяются в AST, в соответствии с трансформационной семантикой. Например, если в коде программы используется входная переменная, то она заменяется на переопределенную переменную в конфигурации. Все шаблонные процессы убираются из ветви AST программного элемента. Если в программе инстанцируются новые процессы, то в ветвь AST добавляются новые ветви процессов с переопределенными именем.

Изменение AST позволяет не модифицировать разработанные генераторы.

Полная грамматика языка poST находится в приложении А.

## 5.4. Реализация библиотек

Библиотеки являются неотъемлемой часть любой системы разработки, это утверждение касается и промышленной автоматизации. Системы IEC 61131-3 обладают обширной базой библиотек, которые хотелось переиспользовать в языке poST. Для решения данной проблемы было разработано следующее.

По трансформационной семантике, библиотечные элементы хранятся в XML файлах рядом с ядром языка. В связи с этим, архитектура библиотечного модуля была разбита на 2 логические части: обработка файлов с библиотечными модулями и использование полученной информации в процессе парсинга.

Для реализации обработчика библиотек в ядро языка было добавлен модуль ответственный за считывание XML файлов с файловой системы. Данный обработчик, при запуске ядра, отрывает библиотечную директорию, обходя все поддиректории и файлам и обрабатывает XML файлы. Для разбора XML файлов используется стандартная библиотека

Java. Из библиотечных элементов считывается только интерфейсная информация, по трансформационной семантике, и сохраняются в памяти через структуру множества. Интерфейсная информация преобразуется в ветку AST языка poST, ветка содержит узлы только с интерфейсной информацией и не имеют ссылки на ресурсы и исходный код языка poST. Добавлен наблюдатель файловой системы, реализованный с помощью стандартных средств Java, который детектирует изменение директории с библиотечными элементами и в случае изменения файлов запускает обработку заново.

Переработано формирование списка претендентов, который используется при линковке и автодополнении. Возвращаемый список претендентов, при определенном контексте вызова библиотечных элементов, содержат поддеревья AST, которые корректно используются в автодополнении и на этапе линковки подставляются в AST программы, что дает возможность проводить корректной синтаксический анализ и семантический контроль.

Поддержана стандартная библиотека IEC 61131-3. За основу была взята библиотека из среды разработки 4diac, в которой библиотеки хранятся в отдельных XML файлах с близким к PLCopen XML Exchange форматом. Был разработан транслятор XML файлов 4diac в XML файлы PLCopen XML Exchange, с помощью которого перенесена вся стандартная IEC 61131-3 в язык poST. Каждый библиотечный элемент хранится в своем XML файле, с логическим разбиением файлов на поддиректории. В плагине Eclipse IDE для языка poST была добавлена возможность создавать проект языка poST, в который загружается шаблон кода и стандартная библиотека IEC 61131-3.

## 5.5. Реализация транслятора

Транслятор это Java программа, представленная в виде запускаемого Java архива, или JAR, с точкой старта программы. Транслятор включает все необходимые библиотеки по технологии FAT JAR [39] и позволяет запускать программу на любом компьютере при наличии виртуальной машины Java. В структуру JAR транслятора входят: лексер, парсер, линковщик, синтаксические проверки, семантический контроль и один или несколько генераторов, также необходимы библиотеки в виде antler парсера, EMF и библиотек, необходимых для обработки AST. Модули для поддержания IDE функциональности не входит в состав JAR транслятора.

Ядро языка poST содержит все необходимые составляющие кроме генераторов. Также в ядре находится точка старта программы, или main. Точка старта спроектирована работать в 2 режимах: единичная трансляция и сервис. В режиме единичной трансляции программа принимает на вход, в виде аргументов программы, файл с расширением ".post", на выход создает новые файлы и завершает работу. В режиме сервиса транслятор начинает работу в

непрерывном режиме для фонового исполнения, транслятор занимает порт операционной системы и ожидает подключения. При подключении клиента запускается задача трансляции кода на тредпуле, что позволяет вести работу в многопоточном режиме. Подключение ожидает передачу через сокет строки с указанием полного пути до файла с расширением ".post", после чего происходит трансляция кода в директории исходного файла.

Из-за особенностей Java, и наличия just in time компиляции [40], режим единичной трансляции работает достаточно долго, а режим сервиса ускоряется со временем и, после нескольких итерация, трансляция происходит мгновенно.

Разработка генераторов для языка poST происходит следующем образом: ядро языка poST является группой проектов, реализованный с помощью Фреймворка Xtext, генераторы реализуются как отдельные проекты и используют ядро как библиотеку. Разработка производится средствами Eclipse IDE и для отдельных генераторов существует функционал для бесшовного связывания генератора с ядром.

Для создания JAR транслятора с ядром и генератором была разработана стратегия и написаны инструкции по созданию. Исходный код генератора копируется в проект ядра и вызывается явным образом. Далее создание исполняемого JAR происходит средствами Eclipse IDE, вызовом экспорта runnable JAR методом FAT JAR. Данный метод гарантирует создание JAR со всеми необходимыми библиотека и без лишних модулей.

Далее сгенерированный JAR файла запускается виртуальной машиной Java с помощью команды "java -jar" или может быть использован стороннем программном обеспечении.

Создан JAR транслятор языка poST в язык ST и формат PLCopen XML Exchange.

### 5.6. Выбор средств реализации web приложения

Для реализации web приложения трансляции языка poST были проанализированы популярные web фреймворки и метода разработки. За основу используется JAR транслятор языка poST в язык ST и формат PLCopen XML Exchange, было выявлено 2 способа использования транслятора: как отдельного процесса операционной системы, в режиме единичной трансляции или в режиме сервиса, или использовать как библиотеку в фреймворках Java-подобных языков программирования.

Были определены требования для web приложения:

- Web приложение должно быть легковесным и не использовать много памяти.
- Процесс разработки должен быть простым.
- Поддержка сессий пользователей, для временного сохранения результатов работы пользователей.

Требования были сформулированы исходя из ограничений сервера, времени на разработку и функциональности. На сервере, где развернуто приложение, планируется запуск нескольких систем, и технические ограничения не позволяют использовать много памяти. Для распространения языка poST необходимо было быстро поднять web приложение. Для удобства пользователей, результаты их работы необходимо временно сохранять.

Web фреймворки на Java-подобных языках, с методом использования JAR транслятора как библиотеки, решено было отложить исходя из того, что для работы такого приложения виртуальная Java машина занимает слишком много памяти.

Было предложено использовать JAR транслятор как отдельный процесс операционной системы, а web приложение реализовывать на легковесном Фреймворке. При этом существует возможность запускать JAR транслятор в 2 режимах: единичной трансляции и сервиса, тем самым балансировать между скоростью работы и потреблением ресурсов.

Из альтернативных язык программирования был выбран Python, а web Фреймворк был выбран Flask [41], из-за простоты использования и нативной поддержки сессий пользователей.

Flask используется для реализации серверной части, которая запускает или общается с JAR транслятором языка poST. Для пользовательской части web приложения был выбран нативный язык разметки HTML с применением стилей CSS и языком JavaScript для реализации логики на стороне пользователя [42].

Для дизайна используется стили библиотеки Bootstrap [43].

## 5.7. Реализация web приложения трансляции языка poST

Реализация web приложения трансляции языка poST разбита на 2 части: пользовательский интерфейс и серверная часть.

Пользовательский интерфейс состоит из 3 частей: главная страница, страница с информацией и логика для отображения номера строк на стороне пользователя.

Главная страница web приложения реализована с помощью HTML разметки с применением стилей Bootstrap CSS. Интерфейс разбит на 4 логические части: Слева находится поле для ввода кода на языке poST, справа находится не редактируемое поля для вывода кода на языке ST, в нижней левой части находится не редактируемое поля для вывода информации о трансляции, в нижней правой части находится блок управления. Верхняя и нижняя части интерфейса находятся в соотношении 70% на 30% в отношении высоты. В верхней правой части интерфейса находится логотип института автоматики и электрометрии СО РАН и кнопка "About" для перехода на страницу с информацией. Интерфейс реализован масштабируемым, с сохранением общего вида и разделения на логические части.

Поля для ввода и вывода текста реализованы стандартной HTML формой для ввода с поддержкой множественных строк и прокруткой текста в случае если текст не помещается в форму. Блок управления содержит набор кнопок, разделенный на 3 логические секции: трансляция и загрузка на сервер poST файла, скачивание с сервера артефактов, загрузка примеров. Имеется следующий набор кнопок: трансляция, открытие файла с компьютера пользователя, скачивание poST кода, скачивание ST кода, скачивание PLCopen XML Exchange файла, открытие шаблона poST программы, открытие примера сушилки для рук, открытие примера 3-х этажного лифта. Для каждой кнопки реализована отправка запроса с соответствующим идентификатором команды.

Пользовательский интерфейс web приложения трансляции языка poST в язык ST и формат PLCopen XML Exchange представлен на рисунке 1.

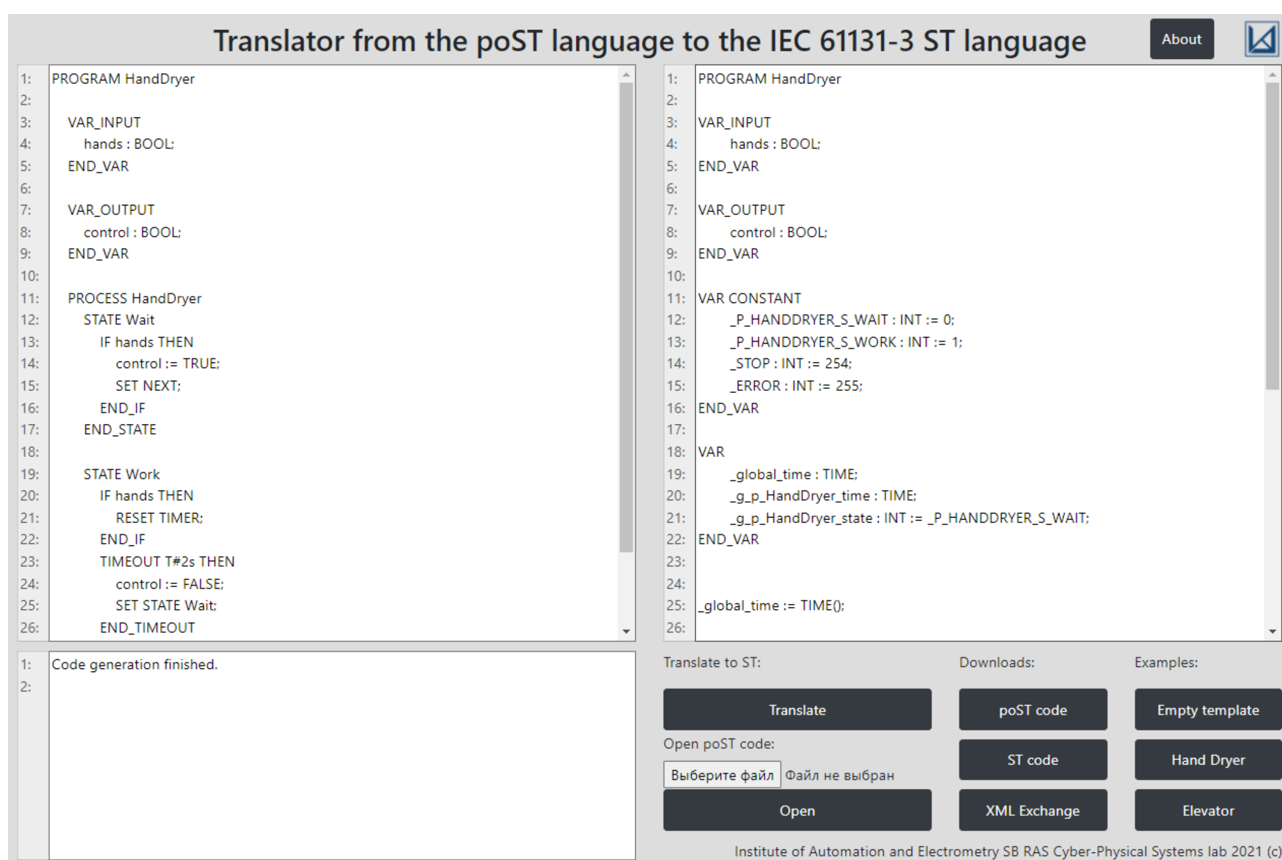


Рисунок 1 — пользовательский интерфейс web приложения трансляции языка poST

Страница с информацией содержит текстовую информацию в центре, кнопку возвращения на главную страницу и статистику посещения web приложения.

Для удобства работы с кодом реализована нумерация строк в 3 текстовых формах. Логика нумерации реализована на языке JavaScript и отрабатывает на стороне пользователя в браузере. Нумерация синхронизируется с текстом в формах, нумерация идет до последней строки текста, в случае прокрутки формы, нумерация также синхронно прокручивается.

Во время отправки запроса на сервер отправляется идентификатор команды и текст из поля для ввода кода на языке poST, в ответ на все запросы принимается новая web страница, которая инициирует перерисовку интерфейса. Сервер может посылать web страницу с заполненными текстовыми формами, в таком случае форма сразу отображается с текстом.

Серверная часть реализуется web Фреймворка Flask на языке программирования Python. Сервер принимает запросы на открытие главной страницы и страницы с информацией, а также обрабатывает запросы из блока управления пользовательского интерфейса.

Алгоритм работы серверной части следующий. При получении запроса на открытие страницы, пользователю отправляется web страницы. При получении команды на трансляцию, код на языке poST, полученный из запроса, поля копируется в файл на сервере, после этого данный файл подается на вход исполняемому JAR файлу с транслятором, алгоритм работы транслятора будет описан далее. При получении команды на скачивание poST кода или ST кода, или PLCopen XML Exchange соответствующий файла отправляется клиенту по протоколу передачи файлов в сети интернет. При получении команды на открытие шаблона или примеров, создается новая web страничка, куда в форму с poST кодом загружается текст из файлов на сервере, и отправляется пользователю.

Для поддержания требования временного сохранения результатов работы пользователя и восстановление состояния после перезагрузки web страницы была реализована система с клиентскими сессиями. Фреймворк Flask позволяет реализовать клиентские сессии, в рамках которой на сервере будет временно сохраняться информация о подключенных клиентах, а также возможность использовать cookies клиента. Алгоритм работы клиентских сессий следующий: для каждого нового пользователя на сервере открывается новая сессия, данной сессии генерируется universally unique identifier, или UUID, который сохраняется в cookies [44] клиента в зашифрованном виде, выделяется дисковое пространство на серверной части в виде директории с название UUID пользователя. Далее вся работа пользователя ведется через данную директорию. В данных директориях хранится код на языке poST, сгенерированных ST код, при наличии, сгенерированный файл PLCopen XML Exchange, при наличии и вывод транслятора, при наличии. Таким образом получается сохранять прогресс пользователя, и при перезагрузке web страничке пользователя, весь его прогресс автоматически загружается. UUID в cookies клиента хранится временно, в зависимости от настроек браузера пользователя, в среднем 2 недели. Дисковое пространство на сервере очищается в ручном режиме, для возможности анализа действий пользователей и сбора статистики использования языка poST.

Реализовано 2 варианта трансляции poST кода: единичная трансляция по запросу и транслятор в виде сервиса. Вариант работы определяется ключом при запуске Flask сервера. В случае единичной трансляции, при получении запроса на трансляцию от пользователя,

порождается новый процесс в операционной системе, на вход подается файл с poST кодом в директории пользователя, по завершению процесса, клиенту отправляется ответ. В случае сервиса, транслятор запускается вместе с Flask сервером, Flask сервер подсоединяется к транслятору посредством сокетов и держит соединение открытым на протяжении всего цикла работы. При получении запроса от клиента, Flask сервер отправляет транслятору путь до файла с poST кодом и ожидает ответа от транслятора, при получении ответа от транслятора Flask сервер отправляет ответ на запрос пользователя. В случае если соединение с транслятором было прервано, сервер пересоздает его и запускает единичную трансляцию. В случае если произошла неизвестная ошибка на стороне транслятора, сервер запускает единичную трансляцию.

Вариант работы с единичной трансляции работает значительно медленнее чем вариант работы с сервисом. Обусловлено это особенностью работы виртуальной машины Java и just in time компиляцией. Но при этом вариант единичной трансляции потребляет значительно меньше ресурсов сервера. Ручной контроль работы сервера позволяет балансировать между производительностью и потреблением ресурсом, в зависимости от актуальности web приложения. Контрольные замеры времени работы показали: на задаче трансляции алгоритма 3-х этажного лифта вариант единичной трансляции производит задержку примерно в 8 секунд, вариант с сервисом, после "прогрева" виртуальной машины Java, срабатывает мгновенно.

Web приложение трансляции языка poST в языке ST позволяет без трудозатрат ознакомится с особенностями языка poST, не требуя установки дополнительного ПО на компьютеры пользователей, а также с легкостью работать с языком poST, транслируя и запуская алгоритмы в специализированных средствах IEC 61131-3.

Web приложение трансляции языка poST была развернуто на сервере института по адресу http://post2st.iae.nsk.su.

### 5.8. Реализация Language Server

Для целей создания web IDE, необходимо реализовать Language Server языка poST. При анализе подходов реализации была найдена реализация протокола средствами Eclipse под названием lsp4j [45]. Данная реализация является самой популярной и позволяет реализовывать LS на Java-подобных языках программирования. При этом ядро языка poST и генераторы языка реализуются средствами Фреймворка Xtext, на языке программирования Xtend, который является транслирующим в язык Java, и выходные артефакты проектов языка poST имеют Java формат, что позволяет переиспользовать lsp4j для языка poST.

При выборе способов реализации LS языка poST было выделено 2 метода: реализовать LS самостоятельно с применением lsp4j или использовать готовые варианты реализации из проекта Xtext.

Вариант с самостоятельным создание LS языка poST позволяет более гибко переиспользовать LSP, реализовывать новые команды для IDE, новую функциональность взаимодействия с другими сущностями, управлять производительностью и собирать статистику. Вариант с использованием готовых реализации LS для Фреймворка Xtext позволяет за короткие сроки реализовать готовый LS языка poST, но данный способ будет поддерживать только базовую функциональность LSP. Было принято решение использовать 2 вариант реализации, поскольку базовой функциональности LSP будет достаточно для языка poST.

В рамках глобального проекта языка poST, была разработана архитектура с проблемно-ориентированными модулями, или DSM, которая не входит в данную работу. Архитектура позволяет запускать генераторы в виде DSM как отдельные процессы операционной системы и передавать AST через сокеты. Данная архитектура разработана для уменьшения порога вхождения студентов в проекты языка poST. При этом, в данной архитектуре, LS является изолированной сущностью и работает только с IDE, для предоставления ее функциональности. DSM не взаимодействуют с LS и не требуют дополнительной логики внутри LS.

Готовый вариант LS языка poST должен быть представлен в виде Java архива, или JAR, собранный по технологии FAT JAR. Данный метод гарантирует наличие всех зависимостей LS и позволяет запускать LS на всех компьютерах с установленной виртуальной машиной Java.

Было выделено 2 способа создания LS. Первый способ от стороннего разработчика с применением стороннего Gradle [46] скрипта и кодовой базой из стороннего репозитория. Второй способ реализован через средства Фреймворка Xtext, с помощью которого разрабатывается ядро языка. Оба способа имеют минусы, первый способ поддерживает старую версию протокола LSP, второй создает LS специфичный для проектов Eclipse, но также поддерживает основной протокол, а специфика заключается только в возможности дебага. Было выявлено, что второй способ не собирает необходимые зависимости, что является большим минусом. Также созданные LS, с помощью обоих методов, поддерживают все языковые функции, кроме подсветки синтаксиса, что не входит в функциональность LSP и реализуется средствами редакторов кода.

Поскольку для языка poST необходимы только базовые функции LSP и возможность запуска LS как отдельного Java процесса, со всеми необходимыми зависимостями, было

решено выбрать первый способ с использованием Gradle скриптов от сторонних разработчиков.

Для создания LS через Gradle была изменена структура проекта ядра. Ранее сборка проекта производилась с помощью средств maven и Eclipse плагинов, в результате которой создавался плагин для Eclipse IDE. Структура проекта также была особенной, поскольку для разработки ядра используется язык Xtend, который на лету транслируется в язык Java. Это порождает в проекте одновременное наличие 2 версии исходных кодов, которые находятся в разных поддиректориях проекта. Для того чтобы Gradle распознал структуру проекта и не путался в копиях исходных кодов, необходимо было создать отдельный проект с другой структурой директорий. Таким образом появилось 2 проекта с разными системами сборки и структурой проектов.

Для создания LS необходимо в Gradle проекте вызвать соответствующую задачу. Скрипт автоматически подгружает все необходимые зависимости, собирает исходные коды и формирует LS в виде jar файла, который можно запускать и использовать.

### 5.9. Выбор платформы web IDE

На этапе анализа средств разработки web IDE были проанализированы платформы Eclipse Theia [47] и VS Code [48]. Обе платформы поддерживают создание локальных версий IDE, устанавливаемых на локальные компьютеры пользователей, а также web интеграцию, с возможностью создания удаленного сервера и использования обычного браузера на стороне пользователей. Основой данных платформ является технология LS, схожий по принципу работы с обычным сервером, но работающий на технологиях LSP. Таким образом для поддержки языка poST в платформах Eclipse Theia и VS Code достаточно сделать LS и минимально взаимодействовать с кодом данных платформ. При сравнении Eclipse Theia и VS Code, было выявлено, что VS Code является более популярной платформой, но при этом частично закрытой. Eclipse Theia, в свою очередь, является полностью открытой платформой и заимствует интерфейс VS Code в виде редактора Monaco [49]. Для конечного пользователя, в случае использования web версии IDE, разницы будут минимальны, но со стороны разработки и апробации критерий открытости платформы является ключевым. Для создания web IDE языка poST на первое место становится платформа Eclipse Theia, из-за своей открытости.

Eclipse Theia — бесплатная интегрированная среда разработки (IDE) с открытым исходным кодом для настольных и веб-приложений. Реализован на TypeScript, основан на Visual Studio Code и поддерживает расширяемость. Расширяемость реализуется через создание расширений на языке программирования TypeScript.

Также стоит отметить, что платформа Eclipse Theia может стать потенциально очень хорошей версии локальной IDE для языка poST. Поскольку язык изначально предназначен для специалистов промышленной автоматизации, то существует вероятность отсутствия интернета у пользователя данного языка, например, при работе в отдаленной местности. В связи с этим, наличие локальной офлайн IDE для языка poST тоже необходимо. Платформа Eclipse Theia позволяет без лишних трудозатрат создавать локальные версии IDE с применением технологии Electron.

## 5.10. Реализация web IDE языка poST

Для реализации web IDE языка poST была выбрана платформа Eclipse Theia, для нее было создано расширение с помощью средств платформы. Расширение является серверной логикой на языке TypeScript, с возможностью создавать логику на пользовательской стороне.

Eclipse Theia использует в качестве редактора кода редактор Monaco с поддержкой просмотра файловой системы, вызова контекстного меню и множества других окон. Пользовательский интерфейс web IDE языка poST представлен на рисунке 2.

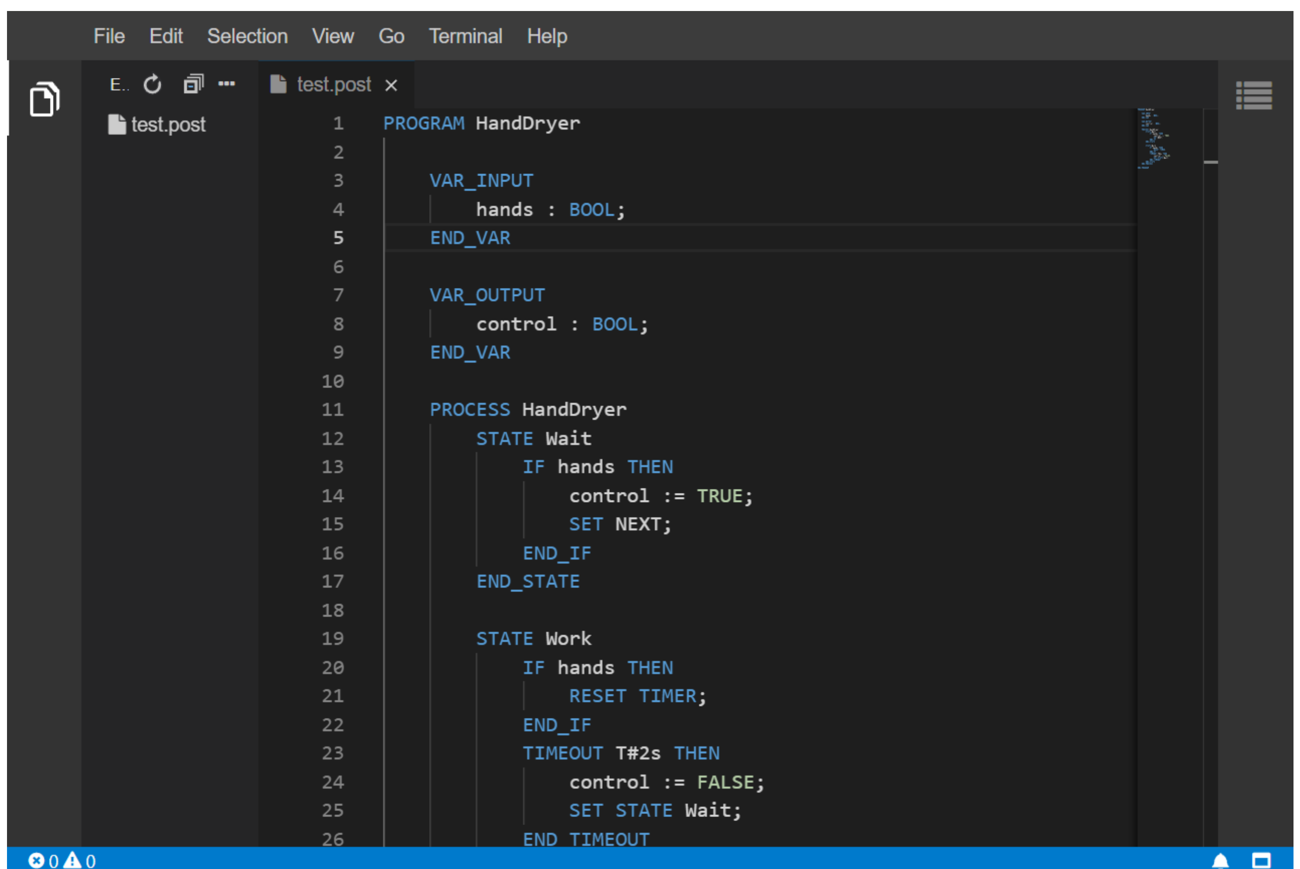


Рисунок 2 — пользовательский интерфейс web IDE языка poST

Разработано расширение с поддержкой poST LS, расширение получило имя "poST DSL" с поддержкой файлов в расширении ".post". Реализована логика для запуска LS и подключение запущенного LS к редактору кода Monaco.

LS запускается как отдельный Java процесс и общается через экстеншен с пользовательским интерфейсом, предоставляя информацию о языке: синтаксические и семантические ошибки, автодополнения, поиск по контексту и другое. Кроме подсветки синтаксиса. Отображение описанной функциональности производится на стороне клиента средствами текстового редактора Monaco, который встроен в Eclipse Theia.

Коммуникация редактора и сервера происходит в сети интернет по HTTP протоколу. Коммуникация серверной части и LS происходит при помощи сокетов операционной системы по LSP протоколу. При взаимодействии пользователя с редактором IDE, на пользовательской стороне создается LSP запрос, далее данный запрос оборачивается HTTP протоколом и отправляется на сервер. Сервер распаковывает HTTP запрос и отправляет, хранившийся в нем, LSP запрос на LS. LS получает LSP запрос, выполняет определенную логику и отправляет на сервер LSP ответ, сервер оборачивает полученный от LS ответ HTTP протоколом и отправляет на сторону пользователя. Данная логика полностью реализуется платформой Eclipse Theia.

Подсветка синтаксиса не входит в функциональность LSP, для решения этой проблемы была расширена функциональность Monaco. Разработана новая грамматика, в которой описаны все ключевые слова языка poST в TextMate нотации [50], специальная грамматика для одноименной библиотеки. Для многострочных комментариев добавлены текстовые интервалы. Далее описанная грамматика отправляется пользователю и уже на стороне пользователя грамматика сопоставляется с написанным в редакторе текстом и подсвечивает ключевые слова. Таким образом подсветка синтаксиса осуществляется на машине пользователя, что почти не сказывается на скорости работы IDE. Данная библиотека поддерживается всеми основными браузерами.

Реализовано расширение для web IDE и для локальной IDE языка poST с поддержкой poST LS и подсветкой синтаксиса.

Написаны инструкции по сборке и запуску web IDE языка poST.

Web IDE языка poST была развернуто на сервере института по адресу http://post.iae.nsk.su.

Для поддержания архитектуры с DSM, вне рамок данной работы, дополнительно был создан сервис AST в виде запускаемого JAR файла, который содержит в себе ядро языка poST, сервис ожидает на вход файл, а на выход передает сериализованное AST. Ядро языка содержит средства для сериализации и десериализации AST, что используемся в DSM для получения AST из текстового вида.

# 6. Практическая апробация лингвистических и инструментальных средств

## 6.1. Описание алгоритмов тестирования

Для практической апробации разработанных лингвистических и инструментальных средств была выбрана задача 3-х этажного лифта.

Система содержит 6 сенсоров: сенсоры наличия лифта на 3-х этажах и сенсоры детектирующие закрытые дверей на 3-х этажах.

В наличие также имеются 6 кнопок: снаружи и внутри лифта для 3-х этажей.

Из выходных сигналов имеется: сигналы для включения LED индикаторов 6 кнопок, сигналы для движения лифта вверх и вниз и целочисленный сигнал отвечающий за отображение текущего этажа.

Система имеет несколько дублирующих элементов в виде кнопок. Алгоритм должен быть масштабируемым для простого увеличения числа этажей в системе.

Сформулированы требования к алгоритму:

1. Лифт должен продолжать движение в том же направлении, пока в этом направлении остаются запросы.
2. Если нет дальнейших запросов, лифт должен остановиться и стать бездействующим. Если присутствуют только запросы на противоположное направление, лифт должен переключиться на предпочтительный направлении и начать обслуживать запросы.
3. Двери всегда должны быть закрыты при движении лифта.
4. Когда кабина достигла нужного места на требуемый этаж, лифт должен открыть дверь и затем закрыть их после трехсекундной паузы.

Для тестирования разработанного алгоритма используется среда CodeSys, в которой существует возможность создания динамической визуализации. Помимо алгоритма реализуется визуализация и симуляция на языке poST.

## 6.2. Реализация алгоритма на языке poST

За основу алгоритма 3-х этажного лифта был взят алгоритм на языке ST из открытых источников [51], в котором применялись конечные автоматы на основе конструкции "CASE", что позволило с легкостью переписать код на язык poST и увеличить читаемость алгоритма. После этого код был изменен с применением принципов процесс-ориентированного программирования и особенностей языка poST. На языке poST реализован как сам алгоритм лифта, с именем Controller, так и модель симуляции лифта, с именем Plant, для использования визуализации в средстве CodeSys.

Алгоритм на языке poST имеет ряд преимуществ, в отличии от алгоритма на языке ST. Структуризация по процессам делает код легко-читаемым, а применение принципов процесс-ориентированного программирования делает код легко-расширяемым. Алгоритм без проблем транслируется в язык ST и запускается в средстве CodeSys с применением визуализации.

При реализации алгоритма применялись все новые разработки лингвистические и инструментальные средства. Применялся механизм шаблонных процессов для дублирующихся элементов.

Алгоритм имеет следующую структуру по процессам:

Controller (Алгоритм лифта):

- Call0Latch — процесс обработки нажатия кнопки снаружи лифта на 0 этаже.
- Call1Latch — процесс обработки нажатия кнопки снаружи лифта на 1 этаже.
- Call2Latch — процесс обработки нажатия кнопки снаружи лифта на 2 этаже.
- Button0Latch — процесс обработки нажатия кнопки внутри лифта на 0 этаже.
- Button1Latch — процесс обработки нажатия кнопки внутри лифта на 1 этаже.
- Button2Latch — процесс обработки нажатия кнопки внутри лифта на 2 этаже.
- CheckCurFloor — процесс проверки текущего этажа.
- UpControl — процесс проверки движения лифта с приоритетом вверх.
- UpMotion — процесс контроля движения лифта вверх.
- DownControl — процесс проверки движения лифта с приоритетом вниз.
- DownMotion — процесс контроля движения лифта вниз.
- DoorCycle — процесс контроля открытия дверей.

Plant (Модель симуляции лифта):

- Door0Sim — процесс симуляции двери на 0 этаже.
- Door1Sim — процесс симуляции двери на 1 этаже.
- Door2Sim — процесс симуляции двери на 2 этаже.
- Floor0SensorSim — процесс симуляции сенсора на 0 этаже.
- Floor1SensorSim — процесс симуляции сенсора на 1 этаже.
- Floor2SensorSim — процесс симуляции сенсора на 2 этаже.
- ElevatorSim — процесс симуляции движения лифта.

Программа Controller реализует сам алгоритм работы лифта. Состоит из 5 шаблонных процессов: Latch, CheckCurFloor, DoorCycle, Motion и Control. Процесс Latch реализует зажатие кнопки, сохранение состояние при нажатии на кнопку и сбрасывание состояния при приезде лифта на нужный этаж, в случае 3-х этажного лифта реализуются для 6 кнопок, внутренних и внешних. Процесс CheckCurFloor проверяет текущий этаж и инициализирует

все внутренние переменные программы. Процесс DoorCycle отвечает за открытие дверей, удержание в открытом состоянии и закрытии дверей, в случае 3-х этажного лифта реализуются для 3 дверей. Процесс Motion отвечает за движение лифта в одну сторону, реализуются для движения вверх и вниз. Процесс Control отвечает выбор направления движения с приоритетом текущего движения, реализуются для выбора направления с приоритетом вверх и вниз.

Программа Simulator реализует имитацию движения лифта для целей симуляции. Состоит из 3 шаблонных процессов: DoorSim, FloorSensorSim и ElevatorSim. Процесс DoorSim симулирует открытие и закрытие дверей по координатам, в случае 3-х этажного лифта реализуются для 3 дверей. Процесс FloorSensorSim симулирует датчик нахождения лифта на этаже, в случае 3-х этажного лифта реализуются для 3 этажей. Процесс ElevatorSim симулирует движение кабины лифта.

Полный исходный код алгоритма 3-х этажного лифта на языке poST находится в приложении Б.

### 6.3. Анализ полученных результатов

Разработанный алгоритм на языке poST был транслирован в язык ST с помощью web приложения трансляции языка poST. ST-код был загружен в среду CodeSys и корректность программы была проверена с помощью визуализации методом ручного тестирования.

При запуске визуализации и нажатии на произвольную кнопку начинается движение дверей или лифта по алгоритму.

Сформулированные требования к алгоритму выполняются.

При анализе исходных кодов на языке poST и на сгенерированном ST коде была получена следующая статистика.

Программа Controller: 13 процессов, 183 строк на языке poST, 311 строк на языке ST.

Программа Simulator: 7 процессов, 98 строк на языке poST, 148 строк на языке ST.

# ЗАКЛЮЧЕНИЕ

В результате работы был проанализирован стандарт IEC 61131-3, язык poST и его недостатки, выделены подходы к разработке web IDE, разработана архитектура ядра. Описана трансформационная семантика языка poST в язык ST и формат PLCopen XML Exchange. Реализован генератор в язык ST и формат PLCopen XML Exchange, добавлена поддержка шаблонных процессов и библиотек. Разработано ядро с инструментами для создания проблемно-ориентированных модулей, создано web приложение трансляции языка poST и web IDE языка poST. Проведена практическая апробация разработанных лингвистических и инструментальных средств на задаче 3-х этажного лифта. Поставленные задачи были выполнены.

В дальнейшем планируется продолжить поддержку и развитие языка poST. Создать проблемно-ориентированные модули и реализовать новые алгоритмы на языке poST.

Язык poST был представлен в статье на международной конференции RusAutoCon [3], рецензируемой в Scopus и IEEE Xplore. Генераторы в язык ST и формат PLCopen XML Exchange были представлены в тезисе на 59-й международной научной студенческой конференции (МНСК) [52]. Механизм библиотек был представлен в тезисе на 60-й международной научной студенческой конференции (МНСК) [53]. Web приложение трансляции языка poST было представлено в статье на международной конференции EDM [36], рецензируемой в Scopus и IEEE Xplore.

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».

____________________________ ____________________________

*ФИО студента* *Подпись студента*

« ____ »___________ 2022 г.

*(заполняется от руки)*

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ


1. IEC 61131-3 Programmable controllers. Part 3: Programming languages // International Electrotechnic Commission. 2013.
2. Zyubin V. E. Hyper-automaton: A model of control algorithms //2007 Siberian Conference on Control and Communications. – IEEE, 2007. – С. 51-57.
3. Bashev V., Anureev I., Zyubin V. The post language: process-oriented extension for IEC 61131-3 structured text //2020 International Russian Automation Conference (RusAutoCon). – IEEE, 2020. – С. 994-999.
4. Antonsen T. M. PLC Controls with Structured Text (ST), V3: IEC 61131-3 and best practice ST programming. – BoD–Books on Demand, 2020.
5. Зюбин В. Е. Программирование ПЛК: языки МЭК 61131-3 и возможные альтернативы //Промышленные АСУ и контроллеры. – 2005. – №. 11. – С. 31-35.
6. Anureev I. et al. Towards safe cyber-physical systems: the Reflex language and its transformational semantics //2019 International Siberian Conference on Control and Communications (SIBCON). – IEEE, 2019. – С. 1-6.
7. Rozov A. S., Zyubin V. E. Adaptation of the process-oriented approach to the development of embedded microcontroller systems //Optoelectronics, Instrumentation and Data Processing. – 2019. – Т. 55. – №. 2. – С. 198-204.
8. Краснов Д. В. и др. Практическая апробация процесс-ориентированной технологии программирования на открытых микроконтроллерных платформах //Bulletin of the East Siberian State University of Technology/Vestnik VSGTU. – 2017. – Т. 66. – №. 3.
9. Розов А. С. и др. Практическая апробация языка IndustrialC на примере автоматизации установки термовакуумного напыления //Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2017. – Т. 15. – №. 3. – С. 90-99.
10. Зюбин В. Е. и др. Базовый модуль, управляющий установкой для выращивания монокристаллов кремния //Датчики и системы. – 2004. – №. 12. – С. 17-22.
11. Зюбин В. Е. Процесс-ориентированное программирование: Учеб. пособие // Новосибирск. Новосиб. гос. ун-т. 2011. 194 с.
12. Dierks H. PLC-automata: A new class of implementable real-time automata //International AMAST Workshop on Aspects of Real-Time Systems and Concurrent and Distributed Software. – Springer, Berlin, Heidelberg, 1997. – С. 111-125.
13. Riera B. et al. Complementary usage of real and virtual manufacturing systems for safe PLC training //IFAC Proceedings Volumes. – 2010. – Т. 42. – №. 24. – С. 89-94.

14. Tiegelkamp M., John K. H. IEC 61131-3: Programming industrial automation systems. – Springer, 2010.
15. Crater K. C. When Technology Standards Become Counterproductive //Control Technology Corporation. – 1992.
16. Lasi H. et al. Industry 4.0 //Business & information systems engineering. – 2014. – Т. 6. – №. 4. – С. 239-242.
17. IEC 61131-10 Programmable controllers. Part 10: PLC open XML exchange format // International Electrotechnic Commission. 2019.
18. Marcos M. et al. XML exchange of control programs //IEEE Industrial Electronics Magazine. – 2009. – Т. 3. – №. 4. – С. 32-35.
19. Simros M., Wollschlaeger M., Theurich S. Programming embedded devices in IEC 61131-languages with industrial PLC tools using PLCopen XML //CONTROLO’2012. – 2012.
20. Enoiu E. P. Programming languages popularity and implications to testing programmable logic controllers. – PeerJ PrePrints, 2015. – №. e1085.
21. Roos N. Programming plcs using structured text //International Multiconference on Computer Science and Information Technology. – 2008. – С. 20-22.
22. Werner B. Object-oriented extensions for IEC 61131-3 //IEEE Industrial Electronics Magazine. – 2009. – Т. 3. – №. 4. – С. 36-39.
23. Wagner F. Modeling software with finite state machines: a practical approach. – Auerbach Publications, 2006.
24. Staroletov S. et al. Modeling and Verification using Different Notations for CPSs: The One-Water-Tank Case Study //2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS). – IEEE, 2021. – С. 485-488.
25. Bettini L. Implementing domain-specific languages with Xtext and Xtend. – Packt Publishing Ltd, 2016.
26. Zyubin V. E. et al. poST: A Process-Oriented Extension of the IEC 61131-3 Structured Text Language //IEEE Access. – 2022. – Т. 10. – С. 35238-35250.
27. Gunasinghe N., Marcus N. Language Server Protocol and Implementation.
28. Language Server Protocol. Implementations. Tools supporting the LSP. // Microsoft [сайт]. URL: https://microsoft.github.io/language-server-protocol/implementors/tools (дата обращения: 30.04.2022)
29. Siméon J., Wadler P. The essence of XML //ACM Sigplan Notices. – 2003. – Т. 38. – №. 1. – С. 1-13.
30. Hanssen D. H. Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS. – John Wiley & Sons, 2015.

31. de Sousa M. The Beremiz PLC: Adding Support for Industrial Communication Protocols //2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). – IEEE, 2019. – C. 232-239.
32. Strasser T. et al. Framework for distributed industrial automation and control (4diac) //2008 6th IEEE international conference on industrial informatics. – IEEE, 2008. – C. 283-288.
33. Zoitl A., Strasser T., Ebenhofer G. Developing modular reusable IEC 61499 control applications with 4DIAC //2013 11th IEEE International Conference on Industrial Informatics (INDIN). – IEEE, 2013. – C. 358-363.
34. Wenger M., Zoitl A. Re-use of IEC 61131-3 structured text for IEC 61499 //2012 ieee international conference on industrial technology. – IEEE, 2012. – C. 78-83.
35. Pugh W. Compressing Java class files //ACM SIGPLAN Notices. – 1999. – T. 34. – №. 5. – C. 247-258.
36. Bashev V., Rozov A., Zyubin V. PoST2ST: a Web Service for Translating poST Programs to the IEC 61131-3 Structured Text //2021 IEEE 22nd International Conference of Young Professionals in Electron Devices and Materials (EDM). – IEEE, 2021. – C. 520-523.
37. Aho A. V., Ullman J. D. The theory of parsing, translation, and compiling. – Englewood Cliffs, NJ : Prentice-Hall, 1973. – T. 1. – C. 309.
38. Steinberg D. et al. EMF: eclipse modeling framework. – Pearson Education, 2008.
39. Bryant D., Marín-Pérez A. Continuous Delivery in Java: Essential Tools and Best Practices for Deploying Code to Production. – O'Reilly Media, 2018.
40. Krall A. Efficient JavaVM just-in-time compilation //Proceedings. 1998 International Conference on Parallel Architectures and Compilation Techniques (Cat. No. 98EX192). – IEEE, 1998. – C. 205-212.
41. Grinberg M. Flask web development: developing web applications with python. – " O'Reilly Media, Inc.", 2018.
42. Robbins J. N. Learning web design: A beginner's guide to HTML, CSS, JavaScript, and web graphics. – " O'Reilly Media, Inc.", 2012.
43. Spurlock J. Bootstrap: responsive web development. – " O'Reilly Media, Inc.", 2013.
44. Park J. S., Sandhu R. Secure cookies on the Web //IEEE internet computing. – 2000. – T. 4. – №. 4. – C. 36-44.
45. Siemund F., Tovesson D. Language Server Protocol for ExtendJ. – 2018.
46. Ikkink H. K. Gradle Dependency Management. – Packt Publishing Ltd, 2015.
47. Theia — Cloud and Desktop IDE Platform. // Theia IDE [сайт]. URL: https://theia-ide.org (дата обращения: 30.04.2022)

48. Visual Studio Code. Documentation. // Visual Studio Code [сайт]. URL: https://code.visualstudio.com/docs (дата обращения: 30.04.2022)
49. Monaco Editor — code editor. // Microsoft [сайт]. URL: https://microsoft.github.io/monaco-editor (дата обращения: 30.04.2022)
50. Language Grammars — TextMate 1.x Manual. // MacroMates [сайт]. URL: https://macromates.com (дата обращения: 30.04.2022)
51. PLC programming an elevator with Structured Text in Codesys. // Gallop Automation Blog [сайт]. URL: https://blog-gallopautomation.com/plc-programming-an-elevator-with-structured-text-in-codesys_part1modular-approach (дата обращения: 30.04.2022)
52. Башев В. И. Бесшовная интеграция языка poST в среду CoDeSys //МНСК-2021. – 2021. – С. 124-124.
53. Башев В. И. Разработка механизма библиотек для языка poST //МНСК-2022. – 2022.

# ПРИЛОЖЕНИЕ А

## Описание формата грамматики

Для описания грамматики языка poST используется расширенная форма Бэкуса-Наура, или РБНФ, с применением терминальных и нетерминальных правил.

Правила вывода имеют следующую формат:

NonTerminalSymbol:

ExtendedStructure

Расширенные структуры формируются по следующими правилами:

Любой терминальный символ — расширенная структура.

Любой нетерминальный символ — расширенная структура.

Если S — расширенная структура, то следующие выражения являются расширенными структурами:

(S) — одно вхождение S.

(S)* — ноль или несколько сцеплений S.

(S)+ — одно или несколько сцеплений S.

(S)? — ноль или одно вхождений S.

Если S1 и S2 — это расширенные структуры, то следующие выражения являются расширенными структурами:

S1 | S2 — выбор, S1 или S2.

S1 S2 — сцепление, за S1 следует S2.

S1 | S2 S3 — эквивалентно S1 | ( S2 S3 ).

S1 S2 | S3 — эквивалентно ( S1 S2 ) | S3.

## Модель языка

Model:

(Configuration)? |

(GlobalVarDeclaration)* |

(Program)* |

(FunctionBlock)* |

(Function)*

Variable:

SymbolicVariable | ProcessVariable | Process | TemplateProcessConfElement

## Конфигурация

Configuration:

'CONFIGURATION' ID

(GlobalVarDeclaration | Resource)*

'END_CONFIGURATION'

Resource:

'RESOURCE' ID 'ON' ID

(GlobalVarDeclaration)*

SingleResource

'END_RESOURCE'

SingleResource:

((Task ';') | (ProgramConfiguration ';'))*

Task:

'TASK' ID '(' TaskInitialization ')'

TaskInitialization:

(SINGLE_DECLARATION ':=' Constant) | (INTERVAL_DECLARATION ':=' Constant)

',' PRIORITY_DECLARATION ':=' INTEGER

SINGLE_DECLARATION:

'SINGLE'

INTERVAL_DECLARATION:

'INTERVAL'

PRIORITY_DECLARATION:

'PRIORITY'

ProgramConfiguration:

'PROGRAM' ID ('WITH' Task)? ':' Program ('(' ProgramConfElements ')')?

ProgramConfElements:

ProgramConfElement (',' ProgramConfElement)*

ProgramConfElement:

AttachVariableConfElement | TemplateProcessConfElement

AttachVariableConfElement:

SymbolicVariable AssignmentType (SymbolicVariable | Constant)

AssignmentType:

':=' | '=>'

**Шаблонные процессы**

TemplateProcessConfElement:

'PROCESS' ('ACTIVE')? ID ':' Process ('('TemplateProcessElements ')')?

TemplateProcessElements:

TemplateProcessAttachVariableConfElement
(',' TemplateProcessAttachVariableConfElement)*
TemplateProcessAttachVariableConfElement:
Variable AssignmentType (Variable | Constant)

### Программы

Program:
'PROGRAM' ID
(InputVarDeclaration |
OutputVarDeclaration |
InputOutputVarDeclaration |
VarDeclaration |
TempVarDeclaration |
ExternalVarDeclaration)*
(Process)*
'END_PROGRAM'

### Функциональные блоки

FunctionBlock:
'FUNCTION_BLOCK' ID
(InputVarDeclaration |
OutputVarDeclaration |
InputOutputVarDeclaration |
VarDeclaration |
TempVarDeclaration |
ExternalVarDeclaration)*
(Process )*
'END_FUNCTION_BLOCK'

### Функции

Function:
'FUNCTION' ID ':' DataTypeName
(InputVarDeclaration |
OutputVarDeclaration |
InputOutputVarDeclaration |
VarDeclaration)*
StatementList

```
'END_FUNCTION'
```

### Процессы и состояния

```
Process:
    'PROCESS' ID
        (InputVarDeclaration |
        OutputVarDeclaration |
        InputOutputVarDeclaration |
        ProcessVarDeclaration |
        VarDeclaration |
        TempVarDeclaration)*
        (State)*
    'END_PROCESS'
State:
    'STATE' ID ('LOOPED')?
        StatementList
        TimeoutStatement?
    'END_STATE'
```

### Операторы poST

```
SetStateStatement:
    'SET' ('STATE' State | 'NEXT')
ProcessStatements:
    StartProcessStatement | StopProcessStatement | ErrorProcessStatement
ProcessStatusExpression:
    'PROCESS' Variable 'IN' 'STATE' ('ACTIVE' | ''INACTIVE'' | 'STOP' | 'ERROR')
StartProcessStatement:
    ('START' 'PROCESS' Variable) | 'RESTART'
StopProcessStatement:
    'STOP' ('PROCESS' Variable)?
ErrorProcessStatement:
    'ERROR' ('PROCESS' Variable)?
TimeoutStatement:
    'TIMEOUT' (Constant | SymbolicVariable) 'THEN'
        StatementList
    'END_TIMEOUT'
```

ResetTimerStatement:

'RESET' 'TIMER'

### Вызов подпрограмм

FBInvocation:

SymbolicVariable '('(ParamAssignmentElements)? ')'

FunctionCall:

Function '('(ParamAssignmentElements)? ')'

ParamAssignmentElements:

elements+=ParamAssignment (',' ParamAssignment)*

ParamAssignment:

SymbolicVariable AssignmentType Expression

### Выражения ST

Expression:

XorExpression (OR_OPERATOR XorExpression)*

OR_OPERATOR:

'OR'

XorExpression:

AndExpression (XOR_OPERATOR AndExpression)*

XOR_OPERATOR:

'XOR'

AndExpression:

CompExpression (AND_OPERATOR CompExpression)*

AND_OPERATOR:

'&' | 'AND'

CompExpression:

EquExpression (CompOperator EquExpression)*

CompOperator:

'=' | '<>'

EquExpression:

AddExpression (EquOperator AddExpression)*

EquOperator:

'<' | '>' | '<=' | '>='

AddExpression:

MulExpression (AddOperator MulExpression)*

AddOperator:

'+' | '-'

MulExpression:

PowerExpression (MulOperator PowerExpression)*

MulOperator:

'*' | '/' | 'MOD'

PowerExpression:

UnaryExpression (POWER_OPERATOR UnaryExpression)*

POWER_OPERATOR:

'**'

UnaryExpression:

PrimaryExpression | (UnaryOperator PrimaryExpression)

UnaryOperator:

'NOT' | '-'

PrimaryExpression:

Constant | SymbolicVariable | ArrayVariable | ProcessStatusExpression | FunctionCall | '('Expression ')';

## Операторы ST

StatementList:

(Statement)*

Statement:

(AssignmentStatement ';') |

SelectionStatement |

IterationStatement |

(FBInvocation ';') |

(SubprogramControlStatement ';') |

(ExitStatement ';') |

(ProcessStatements ';') |

(SetStateStatement ';') |

(ResetTimerStatement ';')

AssignmentStatement:

(SymbolicVariable | ArrayVariable) ':=' Expression

SelectionStatement:

IfStatement | CaseStatement

```
IfStatement:
        'IF' Expression 'THEN'
                StatementList
        ('ELSIF' Expression 'THEN'
                StatementList)*
        ('ELSE'
                StatementList)?
        'END_IF'
CaseStatement:
        'CASE' Expression 'OF'
                (CaseElement)+
        ('ELSE'
                StatementList)?
        'END_CASE'
CaseElement:
        CaseList ':' StatementList
CaseList:
        SignedInteger (',' SignedInteger)*
IterationStatement:
        ForStatement | WhileStatement | RepeatStatement
ForStatement:
        'FOR' SymbolicVariable ':=' ForList 'DO'
                StatementList
        'END_FOR'
ForList:
        Expression 'TO' Expression ('BY' Expression)?
WhileStatement:
        'WHILE' Expression 'DO'
                StatementList
        'END_WHILE'
RepeatStatement:
        'REPEAT'
                StatementList
        'UNTIL' Expression 'END_REPEAT'
SubprogramControlStatement:
```

'RETURN'

ExitStatement:

'EXIT'

### Переменные

SymbolicVariable:

ID

SimpleSpecificationInit:

DataTypeName (':=' Expression)?

VarList:

SymbolicVariable (',' SymbolicVariable)*

VarInitDeclaration:

VarList ':' (SimpleSpecificationInit | ArraySpecificationInit | FunctionBlock)

InputVarDeclaration:

'VAR_INPUT'

(VarInitDeclaration ';')*

'END_VAR'

OutputVarDeclaration:

'VAR_OUTPUT'

(VarInitDeclaration ';')*

'END_VAR'

InputOutputVarDeclaration:

'VAR_IN_OUT'

(VarInitDeclaration ';')*

'END_VAR'

VarDeclaration:

'VAR' ('CONSTANT')?

(VarInitDeclaration ';')*

'END_VAR'

TempVarDeclaration:

'VAR_TEMP'

(VarInitDeclaration ';')*

'END_VAR'

ExternalVarInitDeclaration:

VarList ':' DataTypeName

```
ExternalVarDeclaration:
    'VAR_EXTERNAL' ('CONSTANT')?
        (ExternalVarInitDeclaration ';')*
    'END_VAR'
GlobalVarInitDeclaration:
    VarList 'AT' DIRECT_VARIABLE ':' DataTypeName
GlobalVarDeclaration:
    'VAR_GLOBAL' ('CONSTANT')?
        ((VarInitDeclaration ';') |
        (GlobalVarInitDeclaration ';'))*
    'END_VAR'
```

### Переменные процессов

```
ProcessVariable:
    ID
ProcessVarList:
    ProcessVariable (',' ProcessVariable)*
ProcessVarInitDeclaration:
    ProcessVarList ':' Process
ProcessVarDeclaration:
    'VAR_PROCESS'
        (ProcessVarInitDeclaration ';')*
    'END_VAR'
```

### Массивы

```
ArrayVariable:
    SymbolicVariable '[' Expression ']'
ArraySpecificationInit:
    ArraySpecification (':=' ArrayInitialization)?
ArraySpecification:
    'ARRAY' '[' (ArrayInterval | '*') ']' 'OF'DataTypeName
ArrayInterval:
    Expression '..' Expression
ArrayInitialization:
    '[' Expression (',' Expression)* ']'
```

### Временные литералы

TimeLiteral:

TIME_PREF_LITERAL '#' '-'? interval=INTERVAL

TIME_PREF_LITERAL:

'T'

INTERVAL:

(INTEGER 'd')? (INTEGER 'h')? (INTEGER 'm')? (INTEGER 's')? (INTEGER 'ms')?

### Отображение на физические типы данных

DIRECT_VARIABLE:

'%' DIRECT_TYPE_PREFIX DIRECT_SIZE_PREFIX INTEGER ('.' INTEGER)*

DIRECT_TYPE_PREFIX:

'I' | 'Q' | 'M'

DIRECT_SIZE_PREFIX:

'X' | 'B' | 'W' | 'D' | 'L'

### Типы данных

DataTypeName:

NumericTypeName | BIT_STRING_TYPE_NAME | TIME_TYPE_NAME |
STRING_TYPE_NAME

NumericTypeName:

IntegerTypeName | REAL_TYPE_NAME

IntegerTypeName:

SIGNED_INTEGER_TYPE_NAME | UNSIGNED_INTEGER_TYPE_NAME;

SIGNED_INTEGER_TYPE_NAME:

'SINT' | 'INT' | 'DINT' | 'LINT'

UNSIGNED_INTEGER_TYPE_NAME:

'USINT' | 'UINT' | 'UDINT' | 'ULINT'

REAL_TYPE_NAME:

'REAL' | 'LREAL'

BIT_STRING_TYPE_NAME:

'BOOL' | 'BYTE' | 'WORD' | 'DWORD' | 'LWORD'

TIME_TYPE_NAME:

'TIME'

STRING_TYPE_NAME:

'STRING' | 'WSTRING'

**Числовые литералы**

Constant:

NumericLiteral | TimeLiteral | BINARY_INTEGER | OCTAL_INTEGER | HEX_INTEGER | BOOLEAN_LITERAL

INTEGER:

DIGIT+

SignedInteger:

('-')? INTEGER

REAL:

INTEGER '.' INTEGER

BINARY_INTEGER:

'2#' BIT+

OCTAL_INTEGER:

'8#' OCTAL_DIGIT+

HEX_INTEGER:

'16#' HEX_DIGIT+

NumericLiteral:

IntegerLiteral | RealLiteral

IntegerLiteral:

(IntegerTypeName '#')? SignedInteger

RealLiteral:

(REAL_TYPE_NAME '#')? ('-')? REAL

BOOLEAN_LITERAL:

'TRUE' | 'FALSE'

**Индетификаторы**

LETTER:

'A'..'Z' | 'a'..'z' | '_'

BIT:

'0' | '1'

OCTAL_DIGIT:

'0'..'7'

DIGIT:

'0'..'9'

HEX_DIGIT:

DIGIT | 'A'..'F'

ID:

LETTER (LETTER | DIGIT)*

# ПРИЛОЖЕНИЕ Б

## Описание конфигурации

```
CONFIGURATION Elevator
	VAR_GLOBAL
	(* floor sensors *)
		onfloor0 : BOOL;
		onfloor1 : BOOL;
		onfloor2 : BOOL;
	(* floor call buttons *)
		call0 : BOOL;
		call1 : BOOL;
		call2 : BOOL;
	(* inside car call buttons *)
		button0 : BOOL;
		button1 : BOOL;
		button2 : BOOL;
	(* doors sensors *)
		door0closed : BOOL;
		door1closed : BOOL;
		door2closed : BOOL;
	(* elevator *)
		up : BOOL;
		down : BOOL;
	(* doors *)
		open0 : BOOL;
		open1 : BOOL;
		open2 : BOOL;
	(* LEDs for inside/outside call buttons *)
		call0_LED : BOOL;
		call1_LED : BOOL;
		call2_LED : BOOL;
		button0_LED : BOOL;
		button1_LED : BOOL;
		button2_LED : BOOL;
```

```
(* cur floor LEDs *)
        floor0_LED : BOOL;
        floor1_LED : BOOL;
        floor2_LED : BOOL;
        (* cur floor num*)
        cur : INT;
END_VAR

RESOURCE r1 ON TestCPU
        VAR_GLOBAL CONSTANT
                NUMBER_OF_FLOORS: INT := 3;

                ELEV_DOWN_COORD_CONSTANT : REAL := 440.0;
                DELTA : REAL := 0.5;
                FLOOR_HIGHT : REAL := 225.0;

                FLOOR0_COORD : REAL := ELEV_DOWN_COORD_CONSTANT;
                MAX_FLOOR0_COORD : REAL := FLOOR0_COORD + DELTA;
                MIN_FLOOR0_COORD : REAL := FLOOR0_COORD - DELTA;

                FLOOR1_COORD : REAL := FLOOR0_COORD + FLOOR_HIGHT;
                MAX_FLOOR1_COORD : REAL := FLOOR1_COORD + DELTA;
                MIN_FLOOR1_COORD : REAL := FLOOR1_COORD - DELTA;

                FLOOR2_COORD : REAL := FLOOR1_COORD + FLOOR_HIGHT;
                MAX_FLOOR2_COORD : REAL := FLOOR2_COORD + DELTA;
                MIN_FLOOR2_COORD : REAL := FLOOR2_COORD - DELTA;
        END_VAR
        VAR_GLOBAL
                onfloorArray : ARRAY [0 .. NUMBER_OF_FLOORS] OF BOOL :=
                        [onfloor0, onfloor1, onfloor2];
                openArray : ARRAY [0 .. NUMBER_OF_FLOORS] OF BOOL :=
                        [open0, open1, open2];
                doorclosedArray : ARRAY [0 .. NUMBER_OF_FLOORS] OF BOOL :=
                        [door0closed, door1closed, door2closed];
```

```
        floorLEDs : ARRAY [0 .. NUMBER_OF_FLOORS] OF BOOL :=
                [floor0_LED, floor1_LED, floor2_LED];
        callLEDs : ARRAY [0 .. NUMBER_OF_FLOORS] OF BOOL :=
                [call0_LED, call1_LED, call2_LED];
        buttonLEDs : ARRAY [0 .. NUMBER_OF_FLOORS] OF BOOL :=
                [button0_LED, button1_LED, button2_LED];
END_VAR

TASK T1 (INTERVAL := T#100ms, PRIORITY :=1);

PROGRAM simulator WITH T1: Simulator (
        PROCESS ACTIVE door0Sim : DoorSim (open := open0,
                        doorclosed => door0closed),
        PROCESS ACTIVE door1Sim: DoorSim (open := open1,
                        doorclosed => door1closed),
        PROCESS ACTIVE door2Sim: DoorSim (open := open2,
                        doorclosed => door2closed),
        PROCESS ACTIVE elevatorSim: ElevatorSim (
                        ELEV_DOWN_COORD :=
                        ELEV_DOWN_COORD_CONSTANT),
                        PROCESS ACTIVE floor0SensorSim: FloorSensorSim (
                        UPPER_LIMIT := MAX_FLOOR0_COORD,
                        LOWER_LIMIT := MIN_FLOOR0_COORD,
                        onfloor => onfloor0),
        PROCESS ACTIVE floor1SensorSim: FloorSensorSim (
                        UPPER_LIMIT := MAX_FLOOR1_COORD,
                        LOWER_LIMIT := MIN_FLOOR1_COORD,
                        onfloor => onfloor1),
        PROCESS ACTIVE floor2SensorSim: FloorSensorSim (
                        UPPER_LIMIT := MAX_FLOOR2_COORD,
                        LOWER_LIMIT := MIN_FLOOR2_COORD,
                        onfloor => onfloor2)
);
PROGRAM controller WITH T1: Controller(
        numberOfFloors := NUMBER_OF_FLOORS,
```

```
PROCESS ACTIVE call0Latch: Latch (call := call0,
                open := open0,
                LED => call0_LED),
PROCESS ACTIVE call1Latch: Latch (call := call1,
                open := open1,
                LED => call1_LED),
PROCESS ACTIVE call2Latch: Latch (call := call2,
                open := open2,
                LED => call2_LED),
PROCESS ACTIVE button0Latch: Latch (call := button0,
                open := open0,
                LED => button0_LED),
PROCESS ACTIVE button1Latch: Latch (call := button1,
                open := open1,
                LED => button1_LED),
PROCESS ACTIVE button2Latch: Latch (call := button2,
                open := open2,
                LED => button2_LED),
PROCESS ACTIVE checkCurFloor: CheckCurFloor (
                rOnFloors:=onfloorArray,
                rFloorLEDs:=floorLEDs),
PROCESS doorCycle: DoorCycle (rOpenArray := openArray,
                rDoorClosedArray := doorclosedArray),
                PROCESS upMotion: Motion (motion => up,
                rCallLEDs := callLEDs,
                rButtonLEDs := buttonLEDs),
PROCESS downMotion: Motion (motion => down,
                rCallLEDs := callLEDs,
                rButtonLEDs := buttonLEDs),
PROCESS ACTIVE upControl: Control (rCallLEDs := callLEDs,
                rButtonLEDs := buttonLEDs,
                pReverseControl := downControl,
                pSameMotion := upMotion,
                pDoorCycle := doorCycle,
                downPriority := FALSE),
```

```
                PROCESS downControl: Control (rCallLEDs := callLEDs,
                                rButtonLEDs := buttonLEDs,
                                pReverseControl := upControl,
                                pSameMotion := downMotion,
                                pDoorCycle := doorCycle,
                                downPriority := TRUE)
        );
    END_RESOURCE
END_CONFIGURATION
```

**Описание программы Simulator**

```
PROGRAM Simulator
    VAR
        coord : REAL := 0.0;
    END_VAR

    PROCESS DoorSim
        VAR_INPUT
            open : BOOL;
        END_VAR
        VAR_OUTPUT
            doorclosed : BOOL;
        END_VAR
        VAR CONSTANT
            DOOR_SPEED : REAL := 0.5;
            DOOR_OPEN_COORD : REAL := -50;
        END_VAR
        VAR
            doorCoord : REAL := 0.0;
        END_VAR

        STATE check_open_close LOOPED
            IF open THEN
                doorCoord := doorCoord - DOOR_SPEED;
            ELSE
```

```
                doorCoord := doorCoord + DOOR_SPEED;
            END_IF
            IF doorCoord >= 0.0 THEN
                doorCoord := 0.0;
            END_IF
            IF doorCoord <= DOOR_OPEN_COORD THEN
                doorCoord := DOOR_OPEN_COORD;
            END_IF
            IF doorCoord = 0.0 THEN
                doorclosed := TRUE;
            ELSE
                doorclosed := FALSE;
            END_IF
        END_STATE
END_PROCESS

PROCESS ElevatorSim
    VAR_INPUT
        ELEV_DOWN_COORD : REAL;
    END_VAR
    VAR CONSTANT
        ELEV_ACCEL : REAL := 0.25;
        ELEV_MAX_SPEED : REAL := 0.5;
    END_VAR
    VAR
        v : REAL := 0.0;
    END_VAR
    STATE up_down LOOPED
        IF up THEN
            v := v - ELEV_ACCEL;
        ELSIF down THEN
            v := v + ELEV_ACCEL;
        ELSE
            v := 0.0;
        END_IF
```

```
				IF v > ELEV_MAX_SPEED THEN
					v := ELEV_MAX_SPEED;
				ELSIF v < 0 - ELEV_MAX_SPEED THEN
					v := 0 - ELEV_MAX_SPEED;
				END_IF
				coord := coord + v;
				IF coord < 0.0 THEN
					coord := 0.0;
				ELSIF coord > ELEV_DOWN_COORD THEN
					coord := ELEV_DOWN_COORD;
				END_IF
			END_STATE
		END_PROCESS

		PROCESS FloorSensorSim
			VAR_INPUT
				UPPER_LIMIT : REAL;
				LOWER_LIMIT : REAL;
			END_VAR
			VAR_OUTPUT
				onfloor : BOOL;
			END_VAR
			STATE check_sensor LOOPED
				onfloor := FALSE; (* sensors *)
				IF (coord > LOWER_LIMIT) AND (coord < UPPER_LIMIT) THEN
					onfloor := TRUE;
				END_IF
			END_STATE
		END_PROCESS
END_PROGRAM
```

**Описание программы Controller**

```
PROGRAM Controller
	VAR_INPUT
		numberOfFloors: INT;
```

```
END_VAR

PROCESS Latch
	VAR_INPUT
		call : BOOL;
		open : BOOL;
	END_VAR
	VAR_OUTPUT
		LED : BOOL;
	END_VAR
	VAR
		prev_in : BOOL;
		prev_out : BOOL;
	END_VAR
	STATE init
		prev_in := NOT call;
		prev_out := NOT open;
		SET NEXT;
	END_STATE
	STATE check_ON_OFF LOOPED
		IF call AND NOT prev_in THEN
			LED := TRUE;
		END_IF
		IF open AND NOT prev_out THEN
			LED := FALSE;
		END_IF
		prev_in := call;
		prev_out := open;
	END_STATE
END_PROCESS
PROCESS CheckCurFloor
	VAR_INPUT
		rOnFloors: ARRAY[*] OF BOOL;
		rFloorLEDs: ARRAY[*] OF BOOL;
	END_VAR
```

```
        VAR_TEMP
            floor : INT;
        END_VAR
        STATE check_floor LOOPED
            FOR floor := 0 TO NUMBER_OF_FLOORS DO
                IF (rOnFloors[floor]) THEN
                    cur := floor;
                    rFloorLEDs[floor] := TRUE;
                ELSE
                    rFloorLEDs[floor] := FALSE;
                END_IF
            END_FOR
        END_STATE
    END_PROCESS

    PROCESS DoorCycle
        VAR_INPUT
            rOpenArray : ARRAY[*] OF BOOL;
            rDoorClosedArray : ARRAY[*] OF BOOL;
        END_VAR
        VAR
            allDoorsClosed: BOOL := TRUE;
        END_VAR
        VAR_TEMP
            floor : INT;
        END_VAR
        STATE choose_door_to_open
            rOpenArray[cur] := TRUE;
            SET NEXT;
        END_STATE
        STATE delay3s
            TIMEOUT T#3s THEN
                FOR floor := 0 TO NUMBER_OF_FLOORS DO
                    rOpenArray[floor] := FALSE;
                END_FOR
```

```
                SET NEXT;
            END_TIMEOUT
        END_STATE
        STATE check_closed
            FOR floor:= 0 TO NUMBER_OF_FLOORS DO
                allDoorsClosed := allDoorsClosed AND rDoorClosedArray[floor];
            END_FOR
            IF (allDoorsClosed) THEN
                STOP;
            END_IF
        END_STATE
END_PROCESS

PROCESS Motion
        VAR_INPUT
            rCallLEDs : ARRAY[*] OF BOOL;
            rButtonLEDs : ARRAY[*] OF BOOL;
        END_VAR
        VAR_OUTPUT
            motion: BOOL;
        END_VAR
        STATE start
            motion:= TRUE;
            IF (rCallLEDs[cur] OR rButtonLEDs[cur]) THEN
                motion:= FALSE;
                STOP;
            END_IF
        END_STATE
END_PROCESS

PROCESS Control
        VAR_PROCESS
            pReverseControl : Control;
            pSameMotion : Motion;
            pDoorCycle : DoorCycle;
```

```
END_VAR
VAR_INPUT
    rCallLEDs : ARRAY[*] OF BOOL;
    rButtonLEDs : ARRAY[*] OF BOOL;
    downPriority : BOOL;
END_VAR
VAR
    aboveCall: BOOL;
    belowCall: BOOL;
    currentCall: BOOL;
END_VAR
VAR_TEMP
    floor : INT;
END_VAR
STATE check_calls
    aboveCall := FALSE;
    belowCall := FALSE;
    currentCall := FALSE;
    IF (rCallLEDs[cur] OR rButtonLEDs[cur]) THEN
        currentCall := TRUE;
    END_IF
    FOR floor := 0 TO (cur - 1) DO
        IF (rCallLEDs[floor] OR rButtonLEDs[floor]) THEN
            belowCall := TRUE;
            EXIT;
        END_IF
    END_FOR
    FOR floor := (cur + 1) TO NUMBER_OF_FLOORS DO
        IF (rCallLEDs[floor] OR rButtonLEDs[floor]) THEN
            aboveCall := TRUE;
        EXIT;
        END_IF
    END_FOR
    IF currentCall THEN
        START PROCESS pDoorCycle;
```

```
                SET STATE door_cycle;
            ELSIF (downPriority AND belowCall) OR
                (NOT downPriority AND aboveCall) THEN
            START PROCESS pSameMotion;
                SET NEXT;
            ELSIF (downPriority AND aboveCall) OR
                (NOT downPriority AND belowCall) THEN
                START PROCESS pReverseControl;
                STOP;
            END_IF
        END_STATE
        STATE check_stop
            IF (PROCESS pSameMotion IN STATE INACTIVE) THEN
                START PROCESS pDoorCycle;
                SET NEXT;
            END_IF
        END_STATE
        STATE door_cycle
            IF (PROCESS pDoorCycle IN STATE INACTIVE) THEN
                RESTART;
            END_IF
        END_STATE
    END_PROCESS
END_PROGRAM
```

**ПРИЛОЖЕНИЕ В**

Web IDE языка poST

Руководство оператора

Листов 8

Новосибирск, 2022

# СОДЕРЖАНИЕ

*АННОТАЦИЯ* ........ *78*

*1. Назначение программы* ........ *79*

*2. Условия выполнения программы* ........ *80*

**2.1. Минимальный состав аппаратных средств** ........ **80**

**2.2. Минимальный состав программных средств** ........ **80**

**2.3. Требования к оператору** ........ **80**

*3. Выполнение программы* ........ *81*

**3.1. Загрузка и запуск программы** ........ **81**

**3.2. Выполнение программы** ........ **81**

**3.3. Завершение работы программы** ........ **81**

*4. Сообщения оператору* ........ *82*

*5. Лист регистрации изменений* ........ *83*

# АННОТАЦИЯ

В данном программном документе приведено руководство оператора по применению и эксплуатации программной системы, представленной в виде web IDE процесс-ориентированного языка poST.

В разделе «Назначение программы» указаны сведения о назначении программы и перечислены ее функции. В разделе «Условия выполнения программы» перечислены условия, являющиеся необходимыми для выполнения программы. Раздел «Выполнение программы» содержит последовательность действий оператора, которые необходимы для загрузки, запуска, выполнения и завершения программы. В разделе «Сообщения оператору» приведено описание текстовых сообщений и описаны действия оператора при их возникновении.

Оформление программного документа «Руководство оператора» произведено по требованиям ГОСТ 19.505-79 «ЕСПД. Руководство оператора» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

## 1. Назначение программы

Программная система предназначена для использования процесс-ориентированного языка программирования poST с помощью web технологий, представленных в виде web IDE.

Функции:

- Отображение файловой системы;
- Создание / удаление файлов;
- Редактирование файлов;
- Подсветка синтаксиса;
- Выделение ошибок и предупреждений;
- Автодополнение;
- Навигация по коду;
- Поиск по коду;
- Синтаксическая и семантическая проверка;
- Список ошибок и предупреждений, с возможностью перехода в проблемный участок кода;
- Отображение структуры открытого файла;
- Генерация в ST кода;
- Генерация PLCopen XML файла.

## 2. Условия выполнения программы

### 2.1. Минимальный состав аппаратных средств

Для работы программного средства требуется серверная машина со следующими техническими требованиями:

- Процессор, совместимый с архитектурой x86, и тактовой частотой 2 ГГц или выше;
- Оперативную память объемом 4 Гб или выше;
- Жесткий диск объемом 250 Гб или выше;
- Интернет соединение со скоростью 20 мб/с или выше:
- Лицензионная операционная система, unix-подобная или windows;
- Открытые порты для HTTP, PING и SSH соединений.

### 2.2. Минимальный состав программных средств

Для работы программного средства требуется наличие на серверной машине установленной виртуальной машины Java, виртуальную машину NodeJS, менеджер пакетов NPM и менеджер пакетов YARN.

### 2.3. Требования к оператору

Конечный оператор (администратор) программы должен обладать практическими навыками работы с серверными терминалами, уметь настраивать сетевые соединения и иметь опыт работы с пакетными менеджерами NPM и YARN. Для понимания работы системы, администратор должен обладать навыками мониторинга процессов в операционной системе.

## 3. Выполнение программы

### 3.1. Загрузка и запуск программы

Перед запуском программного средства необходимо скачать исходные коды web IDE, загрузить зависимости с помощью пакетного менеджера NPM, используя команду «npm i», и собрать исходные коды web IDE, используя команду «yarn» в начальной директории исходных кодов.

В операционной системе необходимо перенаправить 80 порт на любой свободный порт, создать пользователя с ограниченными правами, открыть созданного пользователя и запустить web IDE в фоновом процессе.

Запуск web IDE производится с помощью пакетного менеджера YARN. Для запуска необходимо зайти в поддиректорию «browser-app» и выполнить команду «yarn run start --hostname <hostname> --port <port>», обеспечив выполнение данной команды в фоновом процессе операционной системы.

При перезагрузке сервера необходимо обеспечить автоматический запуск web IDE.

### 3.2. Выполнение программы

Работа программы выполняется в фоновом процессе операционной системы. Операционная система принимает запросы на 80 порт и перенаправляет на выделенный порт. Программа случает HTTP запросы, полученные на выделенный порт, обрабатывает запросы и отправляет ответ на запрос.

Во время выполнения программы, программа обращается к файловой системе операционной системы. Сканирует файлы, детектирует изменения состояния и создает новые файлы и директории.

### 3.3. Завершение работы программы

Для завершения работы программного средства оператору необходимо найти фоновый процесс операционной системы и послать сигнал закрытия.

## 4. Сообщения оператору

В ходе работы программной системы оператор получает информационные сообщения работы web IDE в терминале операционной системы. В сообщения входят информация о подключенных соединениях, о полученных запросах и состоянии работы LS.

В случае работы программной системы в фоновом процессе операционной системы, информационные сообщения сохраняются в файл.

При получения информационного сообщения с ошибкой времени исполнения web IDE, оператор должен сообщить об ошибке разработчикам системы и принять действия для устранения нарушения путем перезапуска системы.

## 5. Лист регистрации изменений

| Лист регистрации изменений | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Номера листов (страниц) | | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа | Подпись | Дата |
| Номер изм. | измененных | замененных | новых | аннулированных | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Таблица 1 – Лист регистрации изменений в программном документе «Руководство оператора»

**ПРИЛОЖЕНИЕ Г**

Web IDE языка poST

Описание программы

Листов 8

Новосибирск, 2020

## СОДЕРЖАНИЕ


*АННОТАЦИЯ* ........ *86*

*1. Общие сведения* ........ *87*

**1.1. Обозначение и наименование программы** ........ **87**

**1.2. Программное обеспечение, необходимое для функционирования программы** ........ **87**

**1.3. Языки программирования** ........ **87**

*2. Функциональное назначение* ........ *88*

**2.1. Назначение программы** ........ **88**

**2.2. Сведения о функциональных ограничениях на применение** ........ **88**

*3. Описание логической структуры* ........ *89*

**3.1. Структура программы** ........ **89**

**3.2. Алгоритм программы** ........ **89**

**3.3. Связи между составными частями программы** ........ **89**

**3.4. Связи программы с другими программами** ........ **89**

*4. Используемые технические средства* ........ *90*

*5. Вызов и загрузка* ........ *91*

*6. Входные данные* ........ *92*

*7. Выходные данные* ........ *93*

*8. Лист регистрации изменений* ........ *94*

# АННОТАЦИЯ

В данном программном документе приведено описание программного средства, представленной в виде web IDE процесс-ориентированного языка poST.

В разделе «Общие сведения» указаны сведения о наименовании программы, необходимом программном обеспечении и перечислены используемые языки программирования. В разделе «Функциональное назначение» перечислены назначение программы и сведения о функциональных ограничениях на применение. Раздел «Описание логической структуры» содержит структуру и алгоритм программы, также связь программы с другими программными элементами. В разделе «Используемые технические средства» указаны типы электронных вычислительных машин и устройств, которые используются при работе программы. В разделе «Вызов и загрузка» указаны способ вызова программы с соответствующего носителя данных и входные точки в программу. В разделе «Входные данные» описаны характер, организация и предварительная подготовка входных данных и формат, описание и способ кодирования входных данных.

Оформление программного документа «Описание программы» произведено по требованиям ГОСТ 19.402-78 «ЕСПД. Описание программы» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

# 1. Общие сведения

## 1.1. Обозначение и наименование программы

Наименование программы — web IDE процесс-ориентированного языка poST.

## 1.2. Программное обеспечение, необходимое для функционирования программы

- Лицензионная операционная система, unix-подобная или windows;
- Виртуальная машина Java;
- Виртуальная машина NodeJS;
- Пакетный менеджер NPM;
- Пакетный менеджер YARN.

## 1.3. Языки программирования

В проекте LS применяется Фреймворк Xtext с языками программирования Java, Xtend и Grammar Language.

В проекте extension применяется язык программирование TypeScript, сценарный язык BASH, язык разметки JSON и расширение Eclipse Theia.

## 2. Функциональное назначение

### 2.1. Назначение программы

Программное средство предназначено для помощи в разработке алгоритмов автоматизации на процесс-ориентированном языке программирования poST с применением web технологий, и трансляции написанного poST-кода в язык программирования ST и формат PLCopen XML Exchange.

- Программа решает следующие задачи:
    - Помощь в написании кода:
    - Подсветка синтаксиса;
    - Навигация по коду;
    - Поиск по коду;
    - Автодополнение;
    - Отображение структуры открытого файла;
- Синтаксическая и семантическая проверка:
    - Выделение ошибок и предупреждений;
    - Список ошибок и предупреждений, с возможностью перехода в проблемный участок кода;
- Генерация в ST кода;
- Генерация PLCopen XML файла.

### 2.2. Сведения о функциональных ограничениях на применение

Работа программного обеспечения производится средствами интернет соединения по HTTP протоколу.

Для доступа к программному обеспечению необходим браузер с интернет соединением.

## 3. Описание логической структуры

### 3.1. Структура программы

Программное средство разделено на 2 части с применением разных технологических решений:

- LS языка poST;
- Theia расширения для языка poST.

LS разрабатывается с применением Фреймворка Xtext и языков программирования Java, Xtend и Grammar Language. Входными данными является грамматика языка poST, по которой генерируется основа ядра. Некоторые части основы модифицированы, такие как линковщик, вариатор, генератор. Для функциональности LSP используется готовая библиотека без изменений. По исходным кодам генерируется JAR файл, содержащий LS языка poST.

Расширение реализуется с применением расширение Eclipse Theia и языка программирования TypeScript. Входными данными является код на языке TypeScript. Реальзуеются несколько модулей. Модуль poST LS реализуется отдельно, в котором производится запуск JAR с LS и открытие канала связи. Так же реализуются модули web IDE и локальной версии IDE. По исходным кодам собираются запускаемые артефакты JavaScript.

### 3.2. Алгоритм программы

Алгоритм работы разделен на 2 части: клиентская и серверная часть. На клиентской части пользователь создает запросы через пользовательский интерфейс программы и отправляет запрос на сервер. Запрос формируется по протоку LSP, после чего оборачивается HTTP протоколом. При получении запроса на сервере, HTTP обертка уберётся и внутренний LSP запрос передается LS. Коммуникация между LS и серверной частью Theia происходит по открытому каналу связи, реализуемый через сокеты операционной системы. При получении LSP запроса, LS выполняет соответствующую логику и отправляет LSP ответ. Ответ получается на серверной части Theia, оборачивается HTTP протоколом и отправляется на клиентскую часть.

### 3.3. Связи между составными частями программы

Связи между клиентской и серверной частью осуществляется средствами интернет соединение по HTTP протоколу. Связь между серверной частью Theia и LS осуществляется через сокеты операционной системы и передаются в текстовом формате. Запросы используют LSP протокол.

### 3.4. Связи программы с другими программами

В системе используется платформа Eclipse Theia, для которой реализуется расширение.

## 4. Используемые технические средства

В программном средстве используются следующие технические средства:

- Фреймворк Xtext;
- Язык программирования Java;
- Язык программирования Xtend;
- Язык программирования Grammar Language;
- Язык программирования TypeSript;
- Виртуальная машина Java;
- Виртуальная машина NodeJS;
- Пакетный мекнеджер NPM;
- Пакетный менеджер YARN.

## 5. Вызов и загрузка

Перед запуском программного средства необходимо скачать исходные коды web IDE, загрузить зависимости с помощью пакетного менеджера NPM, используя команду «npm i», и собрать исходные коды web IDE, используя команду «yarn» в начальной директории исходных кодов.

В операционной системе необходимо перенаправить 80 порт на любой свободный порт, создать пользователя с ограниченными правами, открыть созданного пользователя и запустить web IDE в фоновом процессе.

Запуск web IDE производится с помощью пакетного менеджера YARN. Для запуска необходимо зайти в поддиректорию «browser-app» и выполнить команду «yarn run start --hostname <hostname> --port <port>», обеспечив выполнение данной команды в фоновом процессе операционной системы.

При перезагрузке сервера необходимо обеспечить автоматический запуск web IDE.

## 6. Входные данные

Входными данными клиентской части являются действия пользователя.

Входными данными серверной части является HTTP запросы.

Входными данными LS являются LSP запросы.

## 7. Выходные данные

Выходными данными клиентской части являются визуальные изменение графического пользовательского интерфейса.

Выходными данными серверной части является HTTP ответы.

Выходными данными LS являются LSP ответы.

## 8. Лист регистрации изменений

| Лист регистрации изменений | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Номера листов (страниц) | | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа | Подпись | Дата |
| Номер изм. | измененных | замененных | новых | аннулированных | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Таблица 2 – Лист регистрации изменений в программном документе «Описание программы»