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 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Козловой Анастасии Викторовны

Тема работы:

РАЗРАБОТКА ПАРСЕРА EDTL-ТРЕБОВАНИЙ В АБСТРАКТНОЕ СИНТАКСИЧЕСКОЕ ДЕРЕВО

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

Руководитель ВКР
к.ф.-м.н.,
доцент КафКТ ФИТ
Гаранина Н. О. /……...
(ФИО) / (подпись)
«31» мая 2022г.

Соруководитель ВКР
к.т.н.,
доцент КафКТ ФИТ
Розов А. С. /……..
(ФИ О) / (подпись)
«31» мая 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

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Зюбин В. Е.
(фамилия, И., О.)
…….
(подпись)
«29» октября 2021г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студентке Козловой Анастасии Викторовне, группы 18201
(фамилия, имя, отчество, номер группы)

Тема «Разработка парсера EDTL-требований в абстрактное синтаксическое дерево»
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе 29.10.21 №0297

Срок сдачи студентом готовой работы 31.05.2022 г.

Исходные данные (или цель работы): разработка ядра Web-IDE языка EDTL, включающего в себя парсер EDTL-требований в абстрактное синтаксическое дерево, синтаксические и семантические проверки.

Структурные части работы: анализ предметной области, разработка синтаксиса и семантических правил языка EDTL, реализация ядра языка EDTL-требований для Web-IDE EDTL, проверка корректности парсера.

Руководитель ВКР
к.ф.-м.н.,
доцент КафКТ ФИТ
Гаранина Н. О. /……..
(ФИ О) / (подпись)
«29» октября 2021г.

Задание приняла к исполнению
Козлова А. В. /……..
(ФИО студента) / (подпись)
«29» октября 2021г.

Соруководитель ВКР
б/с
ст. преподаватель КафКТ ФИТ
Розов А. С. /……..
(ФИ О) / (подпись)
«29» октября 2021г.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

СО РАН - сибирское отделение Российской академии наук.

EDTL - Event-Driven Temporal Logic.

LTL - Linear Temporal Logic.

NL - Natural Language.

poST - process-oriented Structured Text.

БНФ - форма Бэкуса-Наура.

ANTLR – ANother Tool for Language Recognition.

AST - Abstract Syntax Tree, синтаксическое дерево.

DSL - domain-specific language.

IDE - Integrated Development Environment.

LSP - Language Server Protocol.

ПО - Программное обеспечение.

ВВЕДЕНИЕ

На сегодняшний день существует большое количество решений, направленных на улучшение качества программного обеспечения, в основе которых лежат формальные методы. Большинство формальных методов не находят широкого практического применения, так как время выхода на рынок приоритетнее, чем качество разработки. Но в промышленном программировании трудно обойтись без использования формальных методов, поскольку здесь крайне важны безопасность, надежность и поддерживаемость программного продукта.

Часто возникают недопонимания между заказчиками – инженерами-технологами и исполнителями – разработчиками программного обеспечения. Во многих случаях ошибки, такие как противоречия и недостаточность условий, возникают на этапе формулировки требований технического задания. Цена этих ошибок может быть высока и закончиться катастрофой.

Для решения проблем, связанных с этапом формулировки требований к управляющему программному обеспечению киберфизических систем, сотрудниками лаборатории киберфизических систем института автоматики и электрометрии СО РАН (ИАиЭ СО РАН) был предложен шаблон требований на основе управляемой событиями темпоральной логики (Event-Driven Temporal Logic, EDTL). Этот шаблон имеет формальную семантику в виде логики линейного времени LTL и логики предикатов первого порядка [1].

Таким образом, EDTL – это формальная спецификация, которая позволяет пользователям описывать поведение управляющего ПО в терминах событий, а также выполнять логические операции над его входными и выходными данными. Шаблон на основе EDTL помогает организовать эффективное взаимодействие между инженерами-технологами и разработчиками программного обеспечения, а также обеспечивает плавный переход к этапам верификации и проверки согласованности требований.

В отличие от существующих формальных методов представления требований, шаблон EDTL более понятен и интуитивен как для инженеров,

которые думают в терминах событий, так и для программистов, которым не нужно будет углубляться в детали киберфизической системы и позволит им работать с ней как с черным ящиком.

Для более эффективной работы с наборами EDTL-требований возникает потребность в разработке web-IDE для оптимизации ручной работы.

Целью дипломной работы является разработка ядра web-IDE языка EDTL, включающего в себя парсер EDTL-требований в абстрактное синтаксическое дерево, синтаксические и семантические проверки.

Для достижения данной цели поставлены и решены следующие задачи:

- анализ специфики работы с EDTL-требованиями;
- формирование требований к редактору и парсеру;
- разработка формального синтаксиса языка EDTL;
- определение семантических правил языка EDTL;
- реализация синтаксически-ориентированного редактора и парсера;
- проверка корректности работы парсера.

Разработанное ядро web-IDE языка EDTL предоставляет возможность для интеграции языка EDTL в облачные платформы, которые в своей основе используют протокол Language Server [13].

Парсер EDTL-требований предназначен для использования в модулях расширения web-IDE, таких как трансляция EDTL-требований в LTL семантику и выражения естественного языка.

Работа состоит из введения, четырех глав и заключения. В первой главе анализируется предметная область и формируются требования для синтаксически-ориентированного редактора и парсера. Во второй главе описывается синтаксис и семантика разрабатываемого языка. Третья глава посвящена вопросу выбора инструментов разработки, реализации инструментальных средств. Четвертая глава описывает метод и результаты тестирования.

1 Анализ предметной области

1.1 Формальный синтаксис и семантика EDTL-требований

EDTL в первую очередь ориентирован к формированию требований к управляющим системам.

Опишем синтаксис EDTL-требований.

Неформально EDTL-требование представляет собой совокупность системных событий, ограниченных заданными временными взаимосвязями. Эти события являются атрибутами требования EDTL [1].

Определение 1. (EDTL-требования)

EDTL-требование R представляет собой набор следующих EDTL-атрибутов:

R = (trigger, invariant, final, delay, reaction, release).

Графическое представление EDTL-требования показано на рисунке 1.

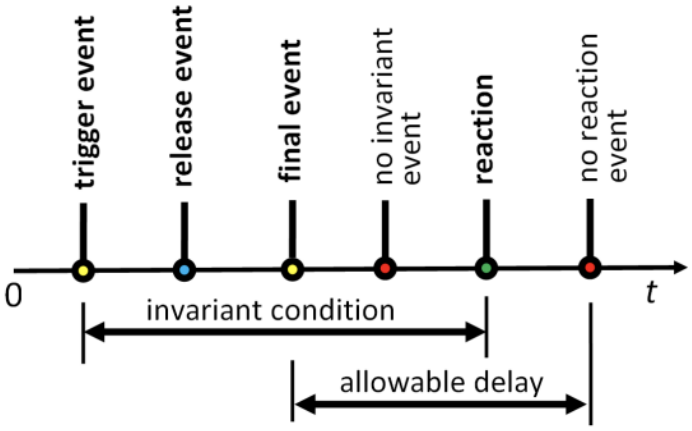

Рисунок 1 - EDTL-требования в графическом виде

Приводится следующее неформальное описание атрибутов:

- **trigger** (триггер) – событие, после которого invariant должен быть истинным, пока не произойдет событие release или reaction. Это событие также является отправной точкой для тайм-аутов моя создания final/release событий (если таковые имеются);

- **invariant** (инвариант) – утверждение, которое должно быть истинным с момента возникновения события trigger до момента release или reaction;
- **final** (финал) – событие, после которого должна произойти реакция с допустимой задержкой. Это событие всегда следует за событием trigger;
- **delay** (задержка) – лимит времени после события final, в течение которого должно произойти событие reaction;
- **reaction** (реакция) – утверждение, которое должно стать истинным в пределах delay от final;
- **release** (высвобождение) – при этом событии, требование считается выполненным.

Этому неформальному описанию соответствует следующее описание семантики требований EDTL на естественном языке:

После каждого trigger-события invariant должен оставаться истинным либо до release, либо до final. Invariant также должен оставаться истинным после final до release или reaction, и, кроме того, событие reaction должно иметь место в пределах указанного допустимого delay от события final.

Значения EDTL-атрибутов представляют собой EDTL-формулы. EDTL-формулы делятся на два класса: формулы состояния и формулы событий. Неформально формулы состояния утверждают о значениях системных переменных в данный момент времени, а формулы событий утверждают о событиях, которые просто происходят или не происходят, т.е. об изменении/сохранении значений переменных с предыдущего момента времени.

EDTL-формулы строятся из утверждений следующим образом:

Определение 2. (EDTL-формулы)

Пусть p – утверждение о событии или о времени, φ и ψ - EDTL-формулы. Тогда:

- формулы состояния:
    - true, false и p — атомарные EDTL-формулы,

- φ∧ψ есть конъюнкция φ и ψ,
- φ∨ψ — дизъюнкция φ и ψ,
- ¬φ — отрицание φ;

- формулы событий:
  - \p — падающий фронт: значение p меняется с false на true,
  - /p — нарастающий фронт: значение p меняется с true на false,
  - p – низкое установившееся состояние: значение p остается равным false,
  - ~ p – высокое установившееся состояние: значение p остается равным true.

В первую очередь планируется использование Web-IDE для формулирования EDTL-требований к управляющему программному обеспечению, написанному на языке poST, который представлен в следующем разделе.

1.2 Язык poST

Современные управляющие системы проектируются на программируемом логическом контроллере (ПЛК) с использованием языков стандарта IEC 61131-3 [9]. Среди языков данного стандарта наиболее подходящими языками для табличных спецификаций – текстовый язык ST или его процесс-ориентированное расширение poST [8]. Таким образом, чтобы обеспечить синтаксическую однородность процедуры разработки систем управления, в качестве синтаксической основы языка EDTL был выбран язык poST.

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 [8].

Для разработки синтаксиса редактора EDTL-требований используются синтаксические конструкции и ключевые слова языка poST из таблицы 1.

Синтаксис	Описание

VAR_INPUT <Объявление переменных> **END_VAR**	Объявление входных переменных.
VAR_OUTPUT <Объявление переменных> **END_VAR**	Объявление выходных переменных.
AT	Оператор привязки программной переменной к аппаратной.
#T<Время>	Ключевое слово для объявления константы времени.

Таблица 1 - Синтаксические конструкции языка poST

Тип объявленной переменной может быть BOOL или INTEGER. Запись константы времени соответствует шаблону #T[d][m][s][ms].

1.3 Требования к синтаксически-ориентированному редактору и парсеру

Исходно EDTL-требования представлялись в виде таблицы, однако эмпирическое исследование способов записи EDTL-требований показало необходимость разработки текстового представления требований. Кроме того, поскольку EDTL-требования предназначены для задания свойств программ на языке poST, решено использовать в текстовом представлении требований синтаксические конструкции этого языка. Для более удобной разработки EDTL-требований инженерами и программистами предполагается внедрить в EDTL редактор общепринятые сервисы, такие как подсветка синтаксиса, автодополнение, подсветка ошибок и т. д.

На основе вышеприведенного анализа сформулированы следующие требования к синтаксически-ориентированному редактору и парсеру:

- ориентация на язык poST;
- переиспользование парсера для модулей расширения;

– синтаксический анализ кода языка EDTL;
– подсветка синтаксиса;
– автодополнение кода;
– семантический анализ;
– возможность интеграции в web-IDE.

Требования к входным данным: входными данными будет являться корректный текстовый код на языке EDTL с учетом формального синтаксиса и семантических правил.

Требования к выходным данным: выходными данными будет являться сгенерированное абстрактное синтаксическое дерево.

2 Разработка синтаксиса и семантических правил языка EDTL

2.1 Синтаксис текстового представления языка EDTL

Для реализации синтаксически-ориентированного редактора языка EDTL необходимо продумать синтаксис конструкций, не встречающихся в языке poST, таких как объявление требований EDTL, макросов и аббревиатур.

В таблице 2 представлены синтаксис этих новых конструкций и их неформальное объяснение. Эта таблица использует определение 1 формального синтаксиса EDTL-требований из раздела 1.1.

Синтаксис	Описание
REQ <Имя требования> <Тело требования> **END_REQ**	Требования. Тело требования состоит из набора нижеописанных EDTL-атрибутов.
TRIGGER <Тело атрибута>	Атрибут trigger EDTL-требования. Тело атрибута.
INVARIANT <Тело атрибута>	Атрибут invariant EDTL-требования. Тело атрибута.
FINAL <Тело атрибута>	Атрибут final EDTL-требования. Тело атрибута.
DELAY <Тело атрибута>	Атрибут delay EDTL-требования. Тело атрибута.
REACTION <Тело атрибута>	Атрибут reaction EDTL-требования. Тело атрибута.
RELEASE <Тело атрибута>	Атрибут release EDTL-требования. Тело атрибута.
NL <Тело комментария>	Атрибут EDTL-требования на естественном языке.

Таблица 2 - Описание новых конструкций

Тело атрибута может состоять из:

- формул EDTL;
- вызов объявленных в коде макросов;
- вызов объявленных в коде сокращений;
- комментарий атрибута на естественном языке.

NL комментарии необходимы для перевода EDTL-требований в естественный язык.

В соответствии со спецификой EDTL-формул необходимо ввести новые операторы, представленные в таблице 3. Эта таблица использует определение 2 формального синтаксиса EDTL-формул из раздела 1.1.

Синтаксис	Описание
FE	Оператор Falling edge.
RE	Оператор Rising edge.
LOW	Оператор Low steady-state.
HIGH	Оператор High steady-state.
TAU <Тело тау>	Оператор времени. Телом тау является временная константа.

Таблица 3 - Описание новых операторов

В таблице 4 представлен синтаксис конструкций макросов и сокращений.

Синтаксис	Описание
MACROS <Имя макроса> <Аргументы макроса> <Тело макроса> **END_MACROS**	Макросы. Аргументами объявления макроса являются переменные. Тело макроса состоит из формул EDTL.
ABBR <Имя сокращения> <Тело сокращения>	Сокращения (аббревиатуры). Тело сокращения состоит из формул EDTL.

END_ABBR	

Таблица 4 - Описание макросов и сокращений

Макросы и атрибуты позволяют разработчикам выносить громоздкие часто повторяемые выражения в отдельные конструкции, что делает код более компактным.

Конструкции и операторы, описанные выше, удовлетворяют требованиям из главы 1.3.

Помимо нововведенных конструкций и операторов используются необходимые конструкции языка poST, описанные в таблице 1 главы 1.2.

2.2 Грамматика текстового представления языка EDTL

Для реализации синтаксически-ориентированного редактора и парсера языка EDTL с новыми конструкциями необходимо описать грамматику.

В приложении А описывается грамматика языка EDTL в расширенной БНФ с описанием выражений для построения правил.

В частности, для описания требований EDTL введены следующие правила:

requirement_name: identifier

requirement_declaration:

'REQ' requirement_name
('TRIGGER' ‘:=’ expression ';' ('NL:' "" nl_attribute "" ';')?)?
('INVARIANT' ‘:=’ expression ';' ('NL:' "" nl_attribute "" ';')?)?
('FINAL' ‘:=’ expression ';' ("NL:' "" nl_attribute "" ';')?)?
('DELAY' ‘:=’ expression ';' ('NL:' "" nl_attribute "" ';')?)?
('REACTION' ‘:=’ expression ';' ('NL:' "" nl_attribute "" ';')?)?
('RELEASE' ‘:=’ expression ';' ('NL:' "" nl_attribute "" ';')?)?
'END_REQ'

nl_attribute: string

Идентификатор имен identifier соответствует стандарту IEC 61131-3. Объявление атрибутов было решено сделать аналогично объявлению переменных в языке Pascal. Конструкция expression описана ниже. Комментарий

на естественном языке для атрибута nl_attribute представляет собой строку, заключенную в парные двойные кавычки.

Для описания выражений EDTL-атрибутов введены следующие правила, описывающие как стандартные конструкции, так и нововведенные:

expression: xor_expression (or_operator xor_expression)*

or_operator: '||’ | 'OR'

xor_expression: and_expression ('XOR' and_expression)*

and_expression: comp_expression (and_operator comp_expression)*

and_operator: '&&' | 'AND'

comp_expression: equ_expression (comp_operator equ_expression)*

comp_operator: '<' | '>' | '<=' | '>='

equ_expression: un_expression (equ_operator un_expression)*

equ_operator: '==' | '<>'

un_expression: primary_expression | un_operator primary_expression

un_operator: not_operator | 'FE' | 'RE' | 'HIGH' | 'LOW'

not_operator: '!' | 'NOT'

primary_expression: constant | abbr_variable | '(' expression ')' | tau_expression | macros_name '(' param_assignment (',' param_assignment)* ')'

tau_expression: 'TAU' '(' time_literal ')'

abbr_variable: variable | abbr_name

Правила для унарных операторов содержат в себе синтаксис операторов из таблицы 3. Для наглядности описания временных констант введена конструкция TAU. Также добавлены символьные варианты операторов AND, OR, NOT, что позволяет писать код в двух стилях: буквенном и символьном. Это дает возможность писать код в более привычном для разработчика стиле.

Для описания макросов и аббревиатур введены следующие правила:

macros_name: identifier

macros_declaration: 'MACROS' macros_name '(' macros_var_list_init ')'

expression

'END_MACROS'

abbr_name: identifier

abbr_declaration: 'ABBR' abbr_name
expression
'END_ABBR'

Правило macros_var_list_init задает аргументы для макроса. Аббревиатура представляет собой макрос без аргументов.

2.3 Семантический анализ

Код языка EDTL, помимо соответствия определенных грамматикой синтаксическим правилам, должен подвергаться семантическому контролю.

Были определены следующие семантические правила с указанием в круглых скобках типа диагностического сообщения:

- Проверка неиспользованных переменных - если переменная была определена, но нигде не использовалась (Warning).
- Проверка конфлика имен: имена переменных, требований, аббревиатур и макросов не должны дублироваться (Error).
- Проверка присваивания значения переменной: если тип переменной не соответствует присваемому значению (Error).
- Проверка использования: в объявлении конструкции MACROS не должно быть выражений MACROS (Error), а при объявлении конструкции ABBR не должно быть выражений ABBR и MACROS (Error).
- Стиль операторов: код рекомендуется писать с использованием только одного стиля: символьного или буквенного (Warning).

Данные семантические правила помогают корректному составлению требований, написанию кода в едином стиле для читабельности, а также уменьшают вероятность появления ошибок.

3 Выбор инструментальных средств реализации парсера

3.1 Выбор средства разработки парсера

Исходя из требований, выдвинутых в главе 1.4, был выбран фреймворк Xtext[2].

Xtext позволяет реализовать собственный DSL. Этот фреймворк предоставляет возможность написать грамматику и получить на ее основе парсер, который генерирует модель классов для абстрактного синтаксического дерева, что позволяет удобно работать с абстрактным синтаксическим деревом, делать по нему обход для семантических проверок, а также предоставляет полнофункциональную настраиваемую IDE на основе Eclipse с навигацией по коду, подсветкой синтаксиса и автодополнением.

Xtext использует ANTLR (ANother Tool for Language Recognition) - генератор нисходящих парсеров для формальных языков, который реализует алгоритм LL(*) [3]. ANTLR преобразует контекстно-свободную грамматику, определяющую язык, из вида расширенной БНФ в исходный код для распознавания языка.

Реализованный DSL можно собрать в архив LS-jar для последующей интеграции языка в Web-IDE.

3.2 Реализация парсера

В фреймворке Xtext для описания грамматики языка реализован DSL под названием Grammar Language, который по синтаксису близок к расширенной БНФ. В нём описываются терминальные и нетерминальные правила, и по их описанию создается лексический анализатор вместе с парсером [2, 3].

Все отдельно взятые нетерминальные правила языка Grammar Language после генерации парсера транслируются в классы Java. Экземпляры этих классов являются частями семантического дерева [4].

Grammar Language поддерживает перекрестные ссылки. В абстрактном синтаксическом дереве при их использовании будет храниться ссылка на ранее описанный объект. Для работы с EDTL-требованиями этот механизм реализован,

в частности, с аббревиатурами, когда при описании выражения атрибута встречается имя аббревиатуры, в абстрактном синтаксическом дереве будет храниться ссылка на ранее объявленную аббревиатуру, а не ее имя.

В приложении Б описана вся грамматика языка EDTL, написанного в формате Grammar Language.

3.3 Реализация семантического анализа

В соответствии с разработанными семантическими проверками из главы 2.3 на языке Xtend[5] реализован модуль семантического анализа построенного абстрактного синтаксического дерева для выдачи предупреждений и выявлений ошибок семантики.

В языке Xtend перед объявлением метода проверки сущностей абстрактного синтаксического дерева необходимо указать аннотацию @Check. При каждом изменении кода EDTL автоматически перестраивается AST, а также вызываются методы его проверки.

Ошибки типов warning и error будут подчеркивать код желтым и красным цветами соответственно [6]. На рисунках 2 и 3 показано, как выглядит код EDTL с реализованным семантическим анализом в Eclipse IDE.

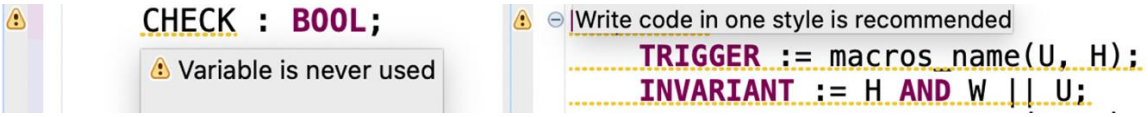

Рисунок 2 - Семантический анализ кода EDTL с ошибками типа warning в Eclipse IDE

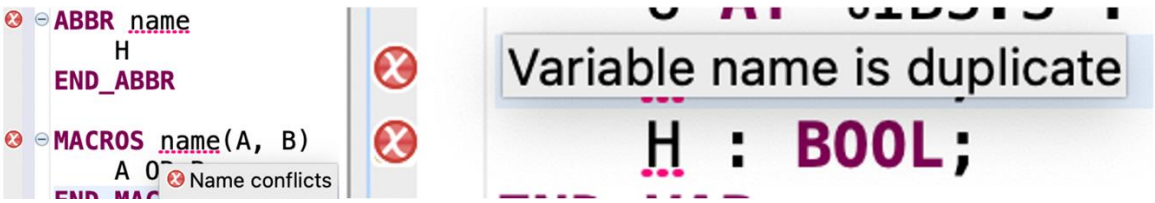

Рисунок 3 - Семантический анализ кода EDTL с ошибками типа error в Eclipse IDE

3.4 Реализация генератора подстановки макросов, сокращений и вызова генераторов

Для того чтобы воспользоваться преимуществами аббревиатур и макросов, необходимо дополнительное перестроение абстрактного синтаксического дерева. Также как и в случае с модулем семантического анализа необходимо обходить AST с помощью Xtend. Для перестроения AST нужно переопределить

построенное автоматически AST и обходить его вглубь для нахождения ссылок на макросы и аббревиатуры. В случае с аббревиатурами достаточно заменить найденную ссылку на копию выражения аббревиатуры. Для макросов необходимо производить подстановку аргументов. Пусть объявлен макрос larger(A, B) с выражением A > B. Тогда при вызове макроса larger(H, D) берется ссылка на объявленный макрос, создается HashMap, ставящий в соответствие аргументы вызова и аргументы объявления, и во время обхода AST производится подстановка. После подстановки аналогично аббревиатуре копируется выражение полученного макроса с подставленными аргументами.

Для возможности использования парсера модулями расширения необходимо объявить зависимость, имеющую название “точка расширения” (extension point) в файле описания плагина plugin.xml. Она объявляет контракт, который должны реализовать модули расширения для подключения к парсеру [11]. Это нужно для того, чтобы генератор ядра смог увидеть модули расширения и вызвал их в переопределенном от абстрактного класса генератора методе doGenerate [4].

3.5 Реализация Language Server

Были проанализированы подходы к реализации Language Server Protocol[13]. Среди них самой популярной оказалась реализация от Eclipse фреймворка LSP4J [12], позволяющая реализовывать LS на Java-подобных языках программирования. Ядро языка EDTL реализуется средствами фреймворка Xtext, а также на языке программирования Xtend, который транслируется в язык Java, и выходные артефакты проектов языка EDTL имеют Java формат, что позволяет переиспользовать фрейворк LSP4J для ядра языка EDTL.

Language Server ядра web-IDE для языка EDTL должен быть собран в виде jar файла, собранного по технологии Fat Jar. Данная технология гарантирует запуск LS на компьютерах с установленной JVM, а также собирает все необходимые зависимости и обеспечивает базовые функции LSP, такие как

автодополнение кода, выявление и подчеркивание ошибок с предупреждениями и рефакторинг кода.

Для сборки ядра web-IDE для языка EDTL в LS-jar был применен сторонний скрипт Gradle shadowJar [10], чтобы обеспечить его интеграцию в Web интерфейс.

Ранее ядро web-IDE для языка EDTL было реализовано в проекте со сборкой Maven. Чтобы воспользоваться данным скриптом, необходимо создание второго проекта со сборкой Gradle. Это нужно для того, чтобы в одном проекте не находилось две версии исходных кодов, что способствует внесению путаницы для Gradle.

В новом проекте со сборкой Gradle необходимо вызвать скрипт shadowJar, автоматически подгружающий все необходимые зависимости и собирающий исходные коды, из чего формирует LS-jar. Собранный Java архив позволяет интегрировать синтаксически-ориентированный редактор языка EDTL в web-IDE.

4 Тестирование парсера

Тестирование работоспособности парсера EDTL-требований в абстрактное синтаксическое дерево проводилось с помощью технологии UniTESK SynTESK [14]. UniTESK позволяет автоматически сгенерировать позитивные и негативные тесты на основе заданной грамматики.

На вход UniTESK подается EDTL-грамматика в виде БНФ.

На выходе UniTESK получается архив, содержащий позитивные и негативные тесты для парсера. Тест представляет собой код на языке EDTL.

Для прохождения сгенерированных тестов был написан модуль на языке Java. На вход jar-файлу с парсером EDTL по циклу передаются тесты.

Негативные тесты считаются пройденными, если парсер считает код теста ошибочным, но не падает с исключением.

По итогам тестирования пройдено 3622/3622 негативных тестов. Это показывает, что парсер защищен от неверных входов – потенциальных уязвимостей.

ЗАКЛЮЧЕНИЕ

Проводился анализ формального синтаксиса EDTL-требований и языка управляющих систем poST. Разрабатывался синтаксис текстового представления языка EDTL, разрабатывалось ядро web-IDE для языка EDTL.

Были получены следующие результаты:

- предложен синтаксис текстового представления языка EDTL;
- реализовано ядро web-IDE для языка EDTL, включающее в себя:
  - синтаксически-ориентированный редактор EDTL-требований,
  - парсер EDTL-требований в абстрактное синтаксическое дерево,
  - синтаксический анализ,
  - семантический контроль,
  - реализация макросов и аббревиатур;
- собран LS-jar для использования ядра в web интерфейсе;
- проверена корректность работы парсера с помощью технологии UniTESK SynTESK на негативных тестах.

Ядро web-IDE языка EDTL будет использоваться в следующих модулях расширения:

- модуль трансляции EDTL-требований в формальную LTL семантику;
- модуль трансляции в выражения на естественном языке;
- модуль динамической верификации.

Также планируется переиспользование ядра для возможности расширения web-IDE другими модулями, такими как контроль непротиворечивости, полноты, избыточности и целостности требований.

Работа представлена на 60-ой Международной научной студенческой конференции 2022 в секции «Информационные технологии» в подсекции «Инструментальные и прикладные программные системы» [7], по итогам выступления получен диплом третьей степени.

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 ознакомлена 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*

« ____ »_____20 __г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Vladimir Zyubin, Igor Anureev, Natalia Garanina, Sergey Staroletov, Andrei Rozov, Tatiana Liakh. Event-Driven Temporal Logic Pattern for Control Software Requirements Specification. // International Conference on Fundamentals of Software Engineering. — FSEN 2021: Fundamentals of Software Engineering. — P. 92-107

2. Xtext Documentation [Электронный ресурс] // Xtext [сайт]. URL: https://www.eclipse.org/Xtext/documentation/ (дата обращения: 30.04.2022)

3. Xtext Grammar Language [Электронный ресурс] // Xtext [сайт]. URL: https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html (дата обращения: 30.04.2022)

4. Xtext Language Implementation [Электронный ресурс] // Xtext [сайт]. URL: https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html (дата обращения: 30.04.2022)

5. Xtend Classes and Members [Электронный ресурс] // Xtend [сайт]. URL:https://www.eclipse.org/xtend/documentation/202_xtend_classes_members.html (дата обращения: 30.04.2022)

6. Xtend Expressions [Электронный ресурс] // Xtend [сайт]. URL: https://www.eclipse.org/xtend/documentation/203_xtend_expressions.html (дата обращения: 30.04.2022)

7. Козлова А. В., “Интегрированная среда разработки для EDTL-требований”, МНСК-2022. – В печати.

8. 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.

9. IEC 61131-3. Programmable controllers. Part 3: Programming languages // International Electrotechnic Commission. 2013.

10. Gradle Shadow [Электронный ресурс] // ShadowJar User Guide [сайт]. URL: https://imperceptiblethoughts.com/shadow/introduction/ (дата обращения: 30.04.2022)

11. Eclipse Wiki [Электронный ресурс] // Eclipse Foundation [сайт]. URL: https://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points%3F (дата обращения: 30.04.2022)

12. Eclipse LSP4J [Электронный ресурс] // Github Repository [сайт]. URL:]https://github.com/eclipse/lsp4j (дата обращения: 30.04.2022)

13. 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)

14. UniTESK [Электронный ресурс] // SynTESK UniTESK [сайт]. URL: http://www.unitesk.ru/content/category/7/20/53/ (дата обращения 30.04.2022)

ПРИЛОЖЕНИЕ А

Грамматика языка EDTL в нотации расширенной БНФ.

Терминальные текстовые символы состоят из соответствующий строки символов, заключенной в парные одиночные кавычки.

Нетерминальные текстовые символы представлены строками букв нижнего регистра, числами и символом подчеркивания «_».

Правила вывода имеют следующию форму:

non_terminal_symbol: extended_structure

Данное правило можно прочитать как: «non_terminal_symbol» включает «extended_structure». Расширенные структуры можно конструировать в соответствии со следующими правилами:

Любой терминальный символ - расширенная структура.

Любой нетерминальный символ - расширенная структура.

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

(S) - означает собственно S.

(S)* - означает ноль или большее число сцеплений S.

(S)+ - означает одно или большее число сцеплений S.

(S)? - означает ноль или одно появление S.

Если S1 и S2 – это расширенные структуры, тогда следующие выражения являются расширенными структурами:

S1 | S2 - выбор, означающее выбор S1 или S2.

S1 S2 - сцепление, означающее S1, за которым следует S2.

Сцепление предшествует выбору, то есть S1 | S2 S3 - эквивалентно S1 | (S2 S3), S1 S2 | S3 - эквивалентно (S1 S2) | S3.

Идентификаторы

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

digit: '0' ... '9'

octal_digit: '0' ... '7'

identifier: letter (letter | digit)*

boolean_literal: 'true' | 'false'

Числовые литералы

integer: digit+

Литерал времени

time_literal: '#T' (integer 'd')? (integer 'h')? (integer 'm')? (integer 's')? (integer 'ms')?

Элементарные типы данных

variable_type: 'BOOL' | 'INT’

Выражения EDTL

expression: xor_expression (or_operator xor_expression)*

or_operator: '||’ | 'OR'

xor_expression: and_expression ('XOR' and_expression)*

and_expression: comp_expression (and_operator comp_expression)*

and_operator: '&&' | 'AND'

comp_expression: equ_expression (comp_operator equ_expression)*

comp_operator: '<' | '>' | '<=' | '>='

equ_expression: un_expression (equ_operator un_expression)*

equ_operator: '==' | '<>'

un_expression: primary_expression | un_operator primary_expression

un_operator: not_operator | 'FE' | 'RE' | 'HIGH' | 'LOW'

not_operator: '!' | 'NOT'

primary_expression: constant | abbr_variable | '(' expression ')' | tau_expression | macros_name '(' param_assignment (',' param_assignment)* ')'

constant: integer | boolean_literal

tau_expression: 'TAU' '(' time_literal ')'

abbr_variable: variable | abbr_name

Объявление переменных

variable: variable_name

variable_name: identifier

direct_variable: '%'location_prefix (size_prefix)? integer ('.'integer)*

location_prefix: 'I' | 'Q' | 'M'

size_prefix: 'X' | 'B' | 'W' | 'D' | 'L'

var_declaration: variable ('AT' direct_variable)? ':' variable_type

decl_var_input:

'VAR_INPUT'

('INPUT_PORTS_COUNTER' decl_symb integer)? (var_declaration ';')* 'END_VAR'

decl_var_output:

'VAR_OUTPUT'

('OUTPUT_PORTS_COUNTER' decl_symb integer)? (var_declaration ';')*

'END_VAR'

macros_var_list_init: variable (',' variable)*

decl_symb: ':='

Присваивание значений переменным

var_init: 'VAR_INIT'

(variable decl_symb value=(integer | boolean_literal) ';')*

'END_VAR'

Объявление требований

requirement_name: identifier

requirement_declaration:

'REQ' requirement_name

('TRIGGER' decl_symb expression ';' ('NL:' "" nl_attribute "" ';')?)?

('INVARIANT' decl_symb expression ';' ('NL:' "" nl_attribute "" ';')?)?

('FINAL' decl_symb expression ';' ("NL:' "" nl_attribute "" ';')?)?

('DELAY' decl_symb expression ';' ('NL:' "" nl_attribute "" ';')?)?

('REACTION' decl_symb expression ';' ('NL:' "" nl_attribute "" ';')?)?

('RELEASE' decl_symb expression ';' ('NL:' "" nl_attribute "" ';')?)?

'END_REQ'

nl_attribute: identifier

Объявление макросов

macros_name: identifier

macros_declaration: 'MACROS' macros_name '(' macros_var_list_init ')'
expression
'END_MACROS'

Объявление аббревиатур

abbr_name: identifier

abbr_declaration: 'ABBR' abbr_name
expression
'END_ABBR'

ПРИЛОЖЕНИЕ Б

Грамматика языка EDTL в нотации Grammar Language (Xtext).

```
grammar su.nsk.iae.edtl.Edtl hidden(WS, ML_COMMENT, SL_COMMENT)
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
generate edtl "http://www.nsk.su/iae/edtl/Edtl"
Model:
(
	declVarInput+=DeclVarInput |
	declVarOutput+=DeclVarOutput |
	varInits+=VarInit |
	abbrs+=Abbr |
	macroses+=Macros |
	reqs+=Requirement
)*;
VariableType returns ecore::EString:
	'BOOL' | 'INT';
DeclVarInput:
	{DeclVarInput}
	'VAR_INPUT'
	('INPUT_PORTS_COUNTER' DECL_SYMB inputCounter=INTEGER ';')?
	(varDecls+=VarDeclaration ';')*
	 'END_VAR';
DeclVarOutput:
	{DeclVarOutput}
	'VAR_OUTPUT'
	('OUTPUT_PORTS_COUNTER'  DECL_SYMB  outputCounter=INTEGER
';')?
	(varDecls+=VarDeclaration ';')*
	'END_VAR';
VarDeclaration:
```

```
        v=Variable ('AT' location=DIRECT_VARIABLE)? ':' type=VariableType;
VarInit:
        {VarInit}
        'VAR_INIT'
        (varAssign+=VarAssign ';')*
        'END_VAR';
VarAssign:
        variable=[Variable]  DECL_SYMB  value=(INTEGER  |
BOOLEAN_LITERAL);
Abbr:
        'ABBR' name=ID
                expr=Expression
        'END_ABBR';
Macros:
        'MACROS' name=ID '(' (args=VarList)? ')'
                expr=Expression
        'END_MACROS';
VarList:
        vars+=Variable (',' vars+=Variable)*;
Requirement:
        'REQ' name=ID
                (
                        'TRIGGER'  DECL_SYMB  trigExpr=Expression ';'
                        ('NL:' nlTrig=STRING ';')?
                )?
                (
                        'INVARIANT'  DECL_SYMB  invExpr=Expression ';'
                        ('NL:' nlInv=STRING ';')?
                )?
                (
```

```
                'FINAL' DECL_SYMB finalExpr=Expression ';'
                ('NL:' nlFinal=STRING ';')?
            )?
            (
                'DELAY' DECL_SYMB delayExpr=Expression ';'
                ('NL:' nlDelay=STRING ';')?
            )?
            (
                'REACTION' DECL_SYMB reacExpr=Expression ';'
                ('NL:' nlReac=STRING ';')?
            )?
            (
                'RELEASE' DECL_SYMB relExpr=Expression ';'
                ('NL:' nlRel=STRING ';')?
            )?
        'END_REQ';
terminal DECL_SYMB:
        ':=';
Variable:
        name=ID;
Expression:
        XorExpression ({Expression.left=current} orOp=OR_OPERATOR
right=XorExpression)*;
terminal OR_OPERATOR:
        '||' | 'OR';
XorExpression:
        AndExpression ({XorExpression.left=current} XOR_OPERATOR
right=AndExpression)*;
XOR_OPERATOR:
        'XOR';
```

```
AndExpression:
    CompExpression ({AndExpression.left=current} andOp=AND_OPERATOR
right=CompExpression)*;
AND_OPERATOR:
    '&&' | 'AND';
CompExpression:
    EquExpression ({CompExpression.left=current} compOp=CompOperator
right=AndExpression)*;
EquExpression:
    UnExpression ({EquExpression.left=current} equOp=EquOperator
right=UnExpression)*;
enum EquOperator:
    EQUAL='==' | NOT_EQUAL='<>';
enum CompOperator:
    LESS='<' | GREATER='>' | LESS_EQU='<=' | GREATER_EQU='>=';
UnOperator returns ecore::EString:
    NotOperator | 'FE' | 'RE' | 'HIGH' | 'LOW' ;
NotOperator:
    'NOT' | '!';
UnExpression:
    PrimaryExpression | unOp=UnOperator right=PrimaryExpression;
TauExpression:
    'TAU' '(' (time=TimeLiteral) ')';
TimeLiteral:
     TIME_PREF_LITERAL interval=INTERVAL;
TIME_PREF_LITERAL returns ecore::EString:
    '#T';
terminal INTERVAL returns ecore::EString:
    (INTEGER 'd')? (INTEGER 'h')? (INTEGER 'm')? (INTEGER 's')? (INTEGER
'ms')?;
```

```
PrimaryExpression:
	constant=Constant | tau=TauExpression | v=[CrossVarAbbr] | macros=[Macros]'(' (args=ParamAssignmentElements)? ')' | '(' nestExpr=Expression ')';
Constant:
	INTEGER | BOOLEAN_LITERAL;
ParamAssignmentElements:
	elements+=[CrossVarAbbr] (',' elements+=[CrossVarAbbr])*;
CrossVarAbbr:
	Variable | Abbr;
terminal DIRECT_VARIABLE returns ecore::EString:
	'%' DIRECT_TYPE_PREFIX DIRECT_SIZE_PREFIX OCTAL_DIGIT ('.' OCTAL_DIGIT)*;
terminal fragment DIRECT_TYPE_PREFIX:
	'I' | 'Q' | 'M';
terminal fragment DIRECT_SIZE_PREFIX:
	'X' | 'B' | 'W' | 'D' | 'L';
terminal INTEGER returns ecore::EString:
	DIGIT+;
terminal fragment LETTER:
	'A'..'Z' | 'a'..'z' | '_';
terminal fragment BIT:
	'0' | '1';
terminal fragment OCTAL_DIGIT:
	'0'..'7';
terminal fragment DIGIT:
	'0'..'9';
terminal fragment HEX_DIGIT:
	DIGIT | 'A'..'F';
terminal BOOLEAN_LITERAL:
```

```
        'TRUE' | 'FALSE';
terminal ID returns ecore::EString:
        LETTER (LETTER | DIGIT)*;
terminal ML_COMMENT:
   '/*' -> '*/' | '(*' -> '*)';
terminal SL_COMMENT:
   '//' !('\n'|'\r')* ('\r'? '\n')?;
terminal WS:
   (' '|'\t'|'\r'|'\n')+;
terminal STRING:
    "" ( '\\'('b'|'t'|'n'|'f'|'r'|'u'|""|"'"|'\\') | !('\\'|"") )* "" | "'" ( '\\'('b'|'t'|'n'|'f'|'r'|'u'|""|"'"|'\\') | !('\\'|"'") )* "'";
```

ПРИЛОЖЕНИЕ В

Плагин языка программирования EDTL для Eclipse IDE

Руководство оператора

Листов 8

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном программном документе приведено руководство оператора по применению и эксплуатации программной системы, представленной в виде плагина языка EDTL для Eclipse IDE.

В данном документе, 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 языка EDTL с помощью инструментальных средств Eclipse IDE.

Функции программы:

- подсветка синтаксиса;
- автодополнение;
- синтаксическая и семантическая проверка;
- выделение ошибок и предупреждений;
- список ошибок и предупреждений;
- отображение структуры открытого файла.

2 Условия выполнения программы

2.1 Минимальный состав аппаратных средств

Для работы программного средства требуется персональный компьютер, включающий в себя:

- процессор с архитектурой x86 с тактовой частотой 2 ГГц или выше;
- оперативную память объемом 2 Гб или выше;
- жесткий диск объемом 256 Гб или выше;
- монитор с разрешением экрана не менее 1024 на 768 пикселей;
- устройство ввода: клавиатура, мышь;
- операционная система с графическим интерфейсом.

2.2 Минимальный состав программных средств

Для работы программного средства требуется наличие на персональном компьютере установленной программой Eclipse IDE, а также наличие Java Virtual Machine.

2.3 Требования к оператору

Конечный оператор (пользователь) программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы. Для понимания работы системы, пользователь должен обладать знаниями написания EDTL-требований и знать структуру программ, написанных на языке EDTL.

3 Выполнение программы

3.1 Загрузка и запуск программы

Перед запуском программного средства необходимо установить плагин Eclipse IDE путём скачивания файлов данного плагина из репозитория ядра EDTL.

После запуска Eclipse IDE необходимо создать или открыть существующий EDTL проект.

Для создания проекта необходимо в открытом workspace создать Java проект (File → New → Project... → Java Project). Далее необходимо создать файл с расширением «.edtl», после чего появится диагностическое сообщение с предложением превратить данный проект в Xtext проект. При согласии данный проект становится проектом на языке программирования EDTL.

3.2 Выполнение программы

Работа программы заключается в написании кода на языке программирования EDTL в открытом текстовом редакторе для файлов с расширением «.edtl».

При любом изменении написанного кода, происходит статический анализ, и в случае выявления нарушения синтаксических или семантических особенностей языка EDTL, выделяется место нарушения.

Генерация абстрактного синтаксического дерева происходит автоматически после каждого изменения кода программы.

3.3 Завершение работы программы

Для завершения работы программного средства оператору необходимо закрыть окно с Eclipse IDE с помощью кнопки «X» на рамке окна.

4 Сообщения оператору

В ходе работы программной системы оператор может получить уведомление о нарушении синтаксических или семантических особенностей языка EDTL в написанном коде. Место нарушения в случае ошибки подсвечивается красным, или желтым в случае предупреждения. Данное предупреждение вместе с номером строки, в котором оно совершено, дублируется в специализированном окне со списком ошибок. Диагностическое сообщение, с описанием нарушения, можно увидеть наведя мышкой на проблемный участок кода, или в окне со списком ошибок.

При наличии ошибок кода построение абстрактного синтаксического дерева не происходит. При их наличии оператор должен принять действия для устранения нарушения путем изменения кода на языке EDTL.

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 5 - Лист регистрации изменений в программном документе «Руководство оператора»

ПРИЛОЖЕНИЕ Г

Плагин языка программирования EDTL для Eclipse IDE

Описание программы

Листов 12

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном программном документе приведено описание программного средства, представленной в виде плагина языка программирования EDTL для Eclipse IDE.

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 – плагин языка программирования EDTL для Eclipse IDE.

1.2 Программное обеспечение, необходимое для функционирования программы

- Операционная система с графическим интерфейсом;
- Интегрированная среда разработки: Eclipse IDE;

1.3 Языки программирования

Программное средство основано на применении фреймворка Xtext от Eclipse.

Исходные языки программирования: Grammar Language, Xtend, Java. Средство разработки: Eclipse IDE.

2 Функциональное назначение

2.1 Назначение программы

Программное средство предназначено для помощи в разработке EDTL-требований на языке программирования EDTL, а также предназначено для трансляции написанного кода в формулы LTL, естественный язык.

Программа решает следующие задачи:

- Помощь в написании кода:
    - Подсветка синтаксиса;
    - Автодополнение;
    - Отображение структуры открытого файла;
- Синтаксическая и семантическая проверка:
    - Выделение ошибок и предупреждений;
    - Список ошибок и предупреждений;
- Генерация абстрактного синтаксического дерева для модулей расширения.

2.2 Сведения о функциональных ограничениях на применение

Написанный код на языке программирования EDTL должен находиться в одном файле с расширением «.edtl».

3 Описание логической структуры

3.1 Структура программы

При разработке программного средства использовался фреймворк Xtext. Входными данными для него стала грамматика языка EDTL в нотации Grammar Language. По написанной грамматике Xtext генерирует артефакты для плагина, парсер и представление абстрактного синтаксического дерева (AST).

Реализованы следующие части:

- Семантическая проверка;
- Генерация абстрактного синтаксического дерева;

3.2 Алгоритм программы

По заданной грамматике Xtext генерирует все необходимые файлы для работы плагина. После генерации, в плагине сразу же доступна подсветка синтаксиса, автодополнение и синтаксический анализ.

Семантический анализ реализован путем написания кастомных правил статического анализа. Перед объявлением метода проверки сущностей абстрактного синтаксического дерева указывается аннотация @Check. При каждом изменении кода EDTL автоматически перестраивается AST, а также вызываются методы его проверки.

Реализованы следующие методы проверки:

- checkVariable_VarDeclarationRepetition(Variable) – проверка дублирования объявления переменной;
- checkRequirementRepetition_RequirementRepetition(Requirement) – проверка дублирования объявления требования;
- checkAbbr_AbbrRepetition(Abbr) – проверка дублирования объявления аббревиатуры;
- checkMacros_MacrosRepetition(Macros) – проверка дублирования объявления макроса;
- checkNameConflicts(Variable) – проверка конфликтов имен переменных;

- checkNameConflicts(Requirement) – проверка конфликтов имен требований;
- checkNameConflicts(Abbr) – проверка конфликтов имен аббревиатур;
- checkVariable_Assign(VarAssign) – проверка соответствия типа переменной и присваемого значения;
- checkAbbr_Expressions(PrimaryExpression) – проверка использования аббревиатур и макросов в аббревиатурах;
- checkMacros_Expressions(PrimaryExpression) – проверка использования макросов в макросах;
- checkVariable_NeverUse(Variable) – проверка неиспользуемых переменных;
- checkOperatorStyle(Model) – проверка единого стиля написания программы.

В модуле генератора происходит объявление генераторов модулей расширения с помощью метода initGenerators().

В методе beforeGenerate(Recourse, IFileSystemAccess2, IGeneratorContext) реализована подстановка макросов и аббревиатур.

В методе doGenerate(Recourse, IFileSystemAccess2, IGeneratorContext) вызывается метод initGenerators().

3.3 Связи между составными частями программы

В модуле генератора происходит объявление генераторов модулей расширения с помощью метода initGenerators().

3.4 Связи программы с другими программами

Программное средства работает через Eclipse IDE.

4 Используемые технические средства

Программное средство эксплуатируется на персональном компьютере, оснащенном операционной системой с графическим интерфейсом, Eclipse IDE и Java Virtual Machine. Режим работы – в формате графического приложения, с взаимодействием с оператором. Входные и выходные данные хранятся на жестком диске.

5 Вызов и загрузка

Перед запуском программного средства необходимо установить плагин Eclipse IDE путём скачивания файлов данного плагина из репозитория EDTL.

После запуска Eclipse IDE необходимо создать или открыть существующий EDTL проект.

Для создания проекта необходимо в открытом workspace создать Java проект (File → New → Project... → Java Project). Далее необходимо создать файл с расширением «.edtl», после чего появится диагностическое сообщение с предложением превратить данный проект в Xtext проект. При согласии данный проект становится проектом на языке программирования EDTL.

6 Входные данные

Входными данными будет являться корректный текстовый код на языке EDTL с учетом формального синтаксиса и семантических правил.

7 Выходные данные

Выходными данными будет являться сгенерированное абстрактное синтаксическое дерево.

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 6 - Лист регистрации изменений в программном документе «Описание программы»