

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

МЕТОДЫ И СРЕДСТВА ИНТЕЛЛЕКТУАЛЬНОГО
МОДЕЛИРОВАНИЯ И ПРОЕКТИРОВАНИЯ СИСТЕМ
ПРОМЫШЛЕННОЙ АВТОМАТИКИ

Студент

Д 28.05.24
(подпись, дата)

Дубинин Алексей Викторович
(ФИО полностью)

Руководитель

В.Н. 28.05.24
(подпись, дата)

Дубинин В.Н.
(фамилия, инициалы)

Нормоконтролёр

А.С. 28.05.24
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент

Доцент Пенз. ГТУ 31.05.24
(должность, место работы) (подпись, дата)

Петерский С.В.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

Заведующий кафедрой

М.А.
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

06.06.2024 № 4)

Секретарь ГЭК

Е.И.
(подпись)

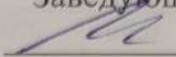
Е.И. Гурин
(фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«05» июня 2023г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Дубинин Алексей Викторович гр. 22ВВИм1, факультета вычислительной техники, направления 09.04.01

2. Тема работы «Методы и средства интеллектуального моделирования и проектирования систем промышленной автоматизации»

Тема утверждена приказом ПГУ № 601а/о от «05» июня 2023г.

3. Руководитель магистерской диссертации д.т.н., профессор Дубинин В.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

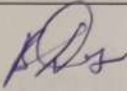
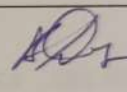
Назначение разработки: иллюстрация научно-методического подхода к онтологическому моделированию и прототипированию промышленных киберфизических систем, а также созданию «цифровых двойников» систем промышленной автоматизации на основе новой технологической платформы семантического Веб на примере производственной системы FESTO.

Разработать методическое, информационное и программное обеспечение для поддержки онтологического подхода к моделированию дискретных систем промышленной автоматизации на примере производственной системы FESTO.

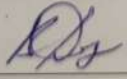
5. Перечень подлежащих разработке вопросов:

- 1) обзор и анализ существующих работ по методам и средствам онтологического моделирования и проектирования систем промышленной автоматизации;
- 2) разработка методики онтологического моделирования дискретных систем промышленной автоматизации;
- 3) разработка онтологии производственной системы FESTO;
- 4) разработка и верификация системы продукционных правил для моделирования динамики производственной системы FESTO;
- 5) реализация продукционных правил на языке SPARQL;
- 6) разработка инструментальной программной системы для поддержки онтологического моделирования дискретных систем промышленной автоматизации.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Обзор и анализ существующих работ по методам и средствам онтологического моделирования и проектирования систем промышленной автоматики	15%	05.06.23– 05.07.23	
2	Разработка методики онтологического моделирования дискретных систем промышленной автоматики	10%	06.07.23– 05.08.23	
3	Разработка онтологии производственной системы FESTO	15%	06.08.23– 05.09.23	
4	Разработка и верификация системы продукционных правил для моделирования динамики производственной системы FESTO	15%	06.09.23– 05.10.23	
5	Реализация продукционных правил на языке SPARQL	20%	06.10.23– 05.12.23	
6	Разработка инструментальной программной системы для поддержки онтологического моделирования дискретных систем промышленной автоматики	15%	06.12.23– 05.02.24	
7	Написание пояснительной записки	10%	19.03.24– 31.05.24	

Дата выдачи задания "05" июня 2023г.

Руководитель магистерской диссертации  05.06.2023 Дубинин В.Н.
(подпись, дата)

Задание к исполнению принял студент  05.06.2023 Дубинин А.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  03.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 154 листа, 56 рисунков, 3 таблицы 62 источника, 2 приложения

FESTO, SPARQL, ОНТОЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ, ОНТОЛОГИЯ, ПРОИЗВОДСТВЕННЫЕ СИСТЕМЫ, ПРОМЫШЛЕННАЯ АВТОМАТИКА, ПРОТОТИПИРОВАНИЕ, ЦИФРОВОЙ ДВОЙНИК, СЕМАНТИЧЕСКИЙ АНАЛИЗ, СЕМАНТИЧЕСКИЙ ВЕБ

Целью работы является иллюстрация научно-методического подхода к онтологическому моделированию и прототипированию промышленных киберфизических систем, а также созданию «цифровых двойников» систем промышленной автоматки на основе новой технологической платформы семантического Веб на примере производственной системы FESTO.

В результате проведенных исследований были получены следующие научные и практические результаты:

- методика онтологического моделирования дискретных систем промышленной автоматки на высоком уровне абстракции,
- общие онтологии станций производственной системы FESTO, а также частные онтологии распределительной, тестирующей и обрабатывающей станций на ее основе,
- семантические ограничения, позволяющих проводить семантический анализ описания станций,
- системы продукционных правил для моделирования динамики функционирования станций и их реализация на языке SPARQL,
- инструментальная программная система для поддержки онтологического (имитационного) моделирования дискретных систем промышленной автоматки.

					ПГУ1.090401.06.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Методы и средства интеллектуального моделирования и проектирования систем промышленной автоматки Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Дубинин А.В.							
Провер.		Дубинин В.Н.					2	154	
Н. Контр		Бычков А.С.				ФВТ, каф. ВТ гр. 22ВВИм1			
									
Утверд.		Митрохин М.А.							

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Дискретные системы промышленной автоматике: особенности, модели и примеры	13
1.1 Дискретные системы промышленной автоматике и их особенности ...	13
1.2 Обзор и анализ существующих работ по методам и средствам моделирования и проектирования систем промышленной автоматике и киберфизических систем.	15
1.3 Производственная система FESTO	24
1.3.1 Неформальное описание	24
1.3.2 Исполнительные устройства	26
1.3.3 Алгоритмы работы	36
2 Онтологическое моделирование производственной системы FESTO	47
2.1 Общая методика онтологического моделирования	47
2.2 Обобщенная онтология станций производственной системы FESTO ..	48
2.2.1 Онтологическая модель	48
2.2.2 Частная онтология тестирующей станции	55
2.2.3 Семантический анализ	61
2.3 Моделирование распределительной станции	66
2.3.1 Онтологическая модель.	66
2.3.2 Продукционные правила и SPARQL–запросы.	69
2.3.3 Верификация системы правил	74
2.4 Моделирование тестирующей станции	76
2.4.1 Онтологическая модель.	76
2.4.2 Продукционные правила и SPARQL–запросы.	78
2.4.3 Верификация системы правил	82

2.5 Моделирование обрабатывающей станции	84
2.5.1 Онтологическая модель	84
2.5.2 Продукционные правила и SPARQL–запросы	86
2.5.3 Верификация системы правил	90
3 Инструментальное средство для поддержки онтологического моделирования на основе языка SPARQL	93
3.1 Назначение, особенности и функции программы	93
3.2 Описание пользовательского интерфейса	94
3.3 UML–диаграмма	102
3.4 Описание модулей (классов)	102
3.5 Руководство пользователя	107
3.6 Имитационное моделирование станций FESTO в инструментальной системе	109
ЗАКЛЮЧЕНИЕ	114
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	118
ПРИЛОЖЕНИЕ А Листинг программы.	126
ПРИЛОЖЕНИЕ А.1 Класс «MainForm»	127
ПРИЛОЖЕНИЕ А.2 Класс «ManualControl»	133
ПРИЛОЖЕНИЕ А.3 Класс «UpdateControl»	135
ПРИЛОЖЕНИЕ А.4 Класс «OptionForm»	140
ПРИЛОЖЕНИЕ А.5 Класс «XMLParser»	142
ПРИЛОЖЕНИЕ В Презентация	145

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения.

ДСПА – дискретные системы промышленной автоматике

КФС – киберфизические системы

ОС – обрабатывающая станция

СПА – системы промышленной автоматике

ТС – тестирующая станция

ЦУЭ – центра управления энергоснабжением

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

MDE (Model-driven engineering) – моделирования на основе моделей

OWL (Web Ontology Language) — язык описания онтологий для семантической паутины

RDF (Resource Description Framework) – структура описания ресурсов

SPARQL (SPARQL Protocol and RDF Query Language) — язык запросов к данным, представленным по модели RDF

SWRL (Semantic Web Rule Language) – язык правил семантической сети

ВВЕДЕНИЕ

Актуальность. Одним из главных атрибутов четвертой промышленной революции (известной также как «Индустрия 4.0») является широкое использование киберфизических систем (КФС) [1]. В связи с принятием концепции «Индустрия 4.0» как определяющей для развития промышленности будущего, возникает актуальная задача разработки методов и средств проектирования систем данного класса, а также их «цифровых двойников». Данное направление является новым и исследования в данной области далеки от завершения.

Важным подклассом КФС являются системы промышленной автоматизации (СПА), непосредственно связанные с промышленным производством. Основными артефактами таких систем являются датчики, исполнительные механизмы и контроллеры. Примерами подобных КФС являются производственные системы, сборочные производства, разумные сети электроснабжения (сети Smart Grid), системы транспортировки и логистические системы и т.д. Каждая из таких систем имеет свои особенности. Производственные системы, как правило, связаны с производством изделий на основе обработки исходных материалов. Одной из главных их особенностей является то, что они в основном относятся к классу дискретных событийных систем.

КФС представляют собой сложные, распределенные в пространстве и времени комплексы, включающие в свой состав как мехатронную, так и кибернетическую части, в которых параллельно протекают процессы различной природы, тесно взаимодействующие между собой, что существенно затрудняет процесс их проектирования.

Важное место в проектировании промышленных КФС, занимает моделирование, которое проводится как с целью верификации, так и с целью оценки производительности. Более того, тонкое и всеобъемлющее моделирование КФС приводит к созданию так называемых «цифровых

двойников» КФС, которые можно использовать в проектировании и обучении КФС практически во всех аспектах.

Несмотря на большое число языков и систем имитационного моделирования, разработка новых подходов к моделированию КФС не прекращается, что связано как с особой ролью КФС в концепции «Индустрия 4.0», так и с появлением новых информационных технологических платформ, среди которых с полным правом можно назвать семантический Web [2].

Перспективным подходом к повышению эффективности процессов моделирования и проектирования КФС является интеллектуализация данных процессов, подразумевающая широкое использование моделей представления знаний, баз знаний и систем логического вывода, а также разработку интеллектуальных средств поддержки. Применение данного подхода позволит сократить сроки и повысить качество разрабатываемых систем. Следует заметить, что несмотря на бурное развитие нейросетевых технологий, классические модели представления знаний прочно удерживают свои позиции в определенных сферах применения, среди которых представление предметных областей. Основными преимуществами моделей такого класса являются высокоуровневый характер представления знаний, отсутствие семантического разрыва между представлениями человека о предметной области и моделями представления знаний, управляемость модели, возможность отслеживания и объяснения результатов логического вывода, легкость интеграции с существующими базами данных и IT-технологиями.

В настоящее время большое значение приобретает онтологическое представление знаний, что связано в двумя факторами. Во-первых, повышением роли концептуальных моделей предметных областей и концептуальных баз знаний, во-вторых, с развитием и распространением концепции семантического Web. Под онтологией понимают формализацию некоторой области знаний с помощью концептуальной схемы [3].

В отличие от обычного Web, который фактически представляет распределенную гипертекстовую базу данных, семантический Web

определяет распределенную базу концептуальных знаний. При этом информация представляется с учетом семантики, имеется возможность семантического поиска. Семантический Web имеет большие интеграционные возможности для включения «разношерстных» ресурсов в непрерывное информационное поле. Нет никаких преград для повторного использования ранее разработанных ресурсов, имеются возможности для совместной работы и обработки данных многими субъектами и программными агентами, а также для организации распределенной обработки данных.

Преимущества использования онтологий в проектировании следующие:

- возможность точно и ясно на высоком уровне выразить семантические свойства,
- наличие логического вывода для доказательства семантических свойств,
- разрешимость задачи классификации и анализа,
- возможность описывать неполные знания,
- легкость перепроектирования системы,
- Web-онтология является разделяемым ресурсом и может быть размещена в сетях Интернет/Интранет для использования людьми или программами [4]. Онтологии находят все большее распространение для описания различных предметных областей [5].

Все большую роль в выявлении ошибок на стадиях проектирования, предшествующих внедрению, приобретает семантический анализ проектов и систем, предполагающий проверку определенных семантических ограничений, отражающих глубинные отношения между артефактами проектирования, не поддающиеся синтаксическому анализу. Семантический анализ не требует интенсивных и ресурсоемких вычислений, какие необходимы при верификации, но при этом способен обнаруживать серьезные ошибки, которые не являются очевидными. В ряде работ (например, [6]) была

показана эффективность использования онтологий и технологий семантического Web в семантическом анализе.

В качестве недостатка (вернее, ограниченности) онтологической модели можно назвать ее статический характер. В последнее время предпринимаются попытки внести некий динамический аспект в онтологическую модель. Как правило, они связаны с использованием внешних средств (например, библиотек языков программирования для обработки онтологий). В качестве примера можно привести библиотеку Thea OWL языка SWI Prolog [7]. Однако, можно выделить подход к трансформации онтологий с помощью стандартных средств семантического Web, а именно: с использованием языка SPARQL – языка запросов в RDF-хранилищах данных [8]. Наличие динамических свойств у онтологической модели важно в двух аспектах:

- 1) возможностью моделировать (имитировать) функционирование рассматриваемой системы,
- 2) возможностью поддерживать процесс проектирования, управляемый моделями (Model-Driven Engineering) [9], который представляется как цепочка моделей, причем каждая последующая модель в цепочке представляется как результат трансформации предыдущей модели.

Нерешенные задачи. На основе анализа существующих источников можно констатировать, что имеется незначительное число работ по онтологическому моделированию систем промышленной автоматизации (русскоязычных работ практически нет). Кроме того, полностью отсутствуют работы, в которых рассматривались бы динамические аспекты онтологического моделирования систем промышленной автоматизации.

На основе вышеизложенного тема магистерской работы, посвященной разработке методов и средств интеллектуального моделирования и проектирования систем промышленной автоматизации, является актуальной.

Объектом исследования являются системы промышленной автоматизации.

Предметом исследования являются методы и средства онтологического моделирования и проектирования систем промышленной автоматизации.

Общей целью данной работы является иллюстрация научно–методического подхода к онтологическому моделированию промышленных КФС [10, 11], особенностью которого является использование онтологических моделей на всех этапах жизненного цикла промышленных КФС. Причем в рамках данного подхода моделируется как кибернетическая, так и физическая составляющие КФС, как статика, так и динамика модели. В качестве иллюстративного примера промышленной КФС рассматривается лабораторная производственная система фирмы FESTO Didactic [12]. Для моделирования динамики функционирования используется язык SPARQL Update [8].

Частными целями работы являются:

- разработка разделяемых информационных ресурсов (а именно: онтологий, то есть, концептуальных баз знаний), описывающих соответствующие предметные области, в сети Интернет. В данном случае разрабатываются онтологии производственных систем,

- создание «цифровых двойников» систем промышленной автоматизации на основе новой технологической платформы, которую представляет семантический Веб. В данной работе скорее должна быть продемонстрирована такая возможность, поскольку моделируемая система довольно простая и, кроме того, система будет рассматриваться на высоком уровне абстракции,

- создание интеллектуальной среды проектирования систем промышленной автоматизации, позволяющей, например, производить семантический анализ описания, а также анализ хода работы системы, производимый во время мониторинга.

Для достижения данных целей в работе решаются следующие задачи:

- проведение обзора и анализа существующих работ по методам и средствам моделирования и проектирования систем промышленной автоматизации (включая методы и средства интеллектуального и онтологического моделирования),

- разработка методики онтологического моделирования дискретных систем промышленной автоматизации,
- разработка обобщенной онтологии станций производственной системы FESTO,
- разработка семантических ограничений, позволяющих проводить семантический анализ,
- разработка частных онтологий конкретных станций производственной системы FESTO,
- разработка и верификация системы продукционных правил для моделирования динамики станций производственной системы FESTO,
- реализация продукционных правил на языке SPARQL,
- разработка инструментальной программной системы для поддержки онтологического (имитационного) моделирования дискретных систем промышленной автоматизации.

Научная новизна. Основным научным результатом магистерской работы состоит в том, что разработана методика онтологического моделирования дискретных систем промышленной автоматизации на высоком уровне абстракции. Особенностью методики является то, что она позволяет моделировать динамику функционирования системы промышленной автоматизации как переход от одной Abox-онтологии к другой Abox-онтологии путем применения трансформационных правил. Разработанная методика проиллюстрирована на примере моделирования производственной системы FESTO.

Практическая ценность работы. Практические результаты, полученные в результате исследований:

- разработаны онтологии станций производственной системы FESTO,
- разработана система продукционных правил для моделирования динамики производственной системы FESTO, проведена верификация данной системы продукционных правил на основе сетей Петри,

- производственные правила для моделирования динамики производственной системы FESTO реализованы на языке SPARQL,
- разработан набор ограничений для проведения семантического анализа описаний станций производственной системы FESTO,
- реализована инструментальная программная система для поддержки онтологического (имитационного) моделирования дискретных систем промышленной автоматизации.

1 Дискретные системы промышленной автоматизации: особенности, модели и примеры

1.1 Дискретные системы промышленной автоматизации и их особенности

Промышленная автоматизация – общее название разнообразных механических, электрических, пневматических, гидравлических и электронных устройств, применяемых для автоматизации технологических процессов, дискретных, непрерывных и гибридных производств.

Примеры применения систем промышленной автоматизации (СПА):

- сборочный конвейер,
- станки с числовым программным управлением,
- промышленные роботы,
- разумные сети электроснабжения (Smart Grid),
- атомные электростанции,
- «умный дом»,
- «умный город»,
- систем логистики,
- транспортные средства,
- системы сигнализации и безопасности.

Элементы СПА представлены ниже:

- датчики – преобразуют физические величины (например, температуру, давление, скорость) в осведомительные сигналы,
- устройства управления – обрабатывают осведомительные сигналы с датчиков в соответствии с заданным алгоритмом управления и выдают управляющие сигналы,
- исполнительные устройства – преобразуют управляющие сигналы в управляющие воздействия на объект (например, включение/выключение двигателя, открытие/закрытие клапана).

Центральную роль в СПА играют устройства управления.

Дискретные системы промышленной автоматики (ДСПА) – это вид СПА, где изменения параметров и состояний происходят скачкообразно, принимая дискретные значения. Эти системы основаны на обработке дискретных сигналов и операций, что отличает их от систем с непрерывным управлением.

Особенности ДСПА перечислены ниже:

- дискретность операций – операции происходят в дискретные моменты времени (а не непрерывно),
- цифровая обработка – ДСПА используют цифровые сигналы и алгоритмы обработки данных для выполнения различных функций управления и контроля процессов,
- логическое управление – в большинстве случаев управление в дискретных системах осуществляется с использованием логических операций,
- программируемость – ДСПА обладают высокой степенью программной настройки и адаптации,
- простота реализации – ДСПА проще проектировать, конструировать и эксплуатировать, чем непрерывные системы,
- надежность – ДСПА менее подвержены влиянию помех и сбоев, чем непрерывные системы,
- низкая стоимость – ДСПА, как правило, дешевле аналоговых систем,
- наглядность – работа ДСПА более понятна и наглядна, чем работа непрерывных систем.

В целом, ДСПА являются ценным инструментом для решения задач управления в различных отраслях промышленности. Их простота, надежность, низкая стоимость и наглядность делают их привлекательным выбором для многих применений.

ДСПА широко используются в дискретном производстве. Дискретное производство характеризуется дискретным во времени изменением

характеристик единиц материального потока. Технологические процессы дискретного производства настроены на обработку либо отдельных единиц производственных продуктов, либо партий продуктов. Собственно, обработка единиц материального потока связана с последовательным выполнением отдельных технологических операций, причем каждая операция имеет фиксированные по времени начало и конец.

Операция, реализуемая в технологическом процессе, это развернутое во времени целенаправленное действие, которое характеризуется своей целью, например, доведение параметров заготовки до заданных значений, перемещение рабочих органов в требуемое положение и т.п., и способом достижения этой цели, например, резка, шлифовка, включение двигателя на перемещение и т.п.

1.2 Обзор и анализ существующих работ по методам и средствам моделирования и проектирования систем промышленной автоматизации и киберфизических систем

Обзор русскоязычной литературы. Ниже представлен краткий обзор русскоязычной литературы (исключая собственные работы) по теме, связанной с онтологическим моделированием и проектированием систем промышленной автоматизации, а также смежным вопросам.

Работа [13] посвящена вопросам построения концептуальной модели предметной области и разработки адекватного и оптимального набора средств ее формального описания. В статье представлены методы и средства решения этих вопросов на основе задачно-ориентированной онтологии, архитектуры Захмана и средств проекта Eclipse Modeling Project.

В статье [14] онтология рассматривается как один из аспектов систематизации, а исследование сосредоточено на анализе сущностей, присутствующих и возникающих в процессе проектирования. На основе онтологии оценивается эффективность проектируемой системы. Данная работа скорее относится к области экономики, вопросы моделирования

промышленного производства и систем промышленной автоматики не затрагиваются.

Монография [15] посвящена современным практикам применения онтологического моделирования при создании корпоративных автоматизированных систем. Рассматривается широкий круг методологических и технологических вопросов моделирования, набор современных программных средств и стандартов представления и обработки онтологических моделей. К сожалению, монография носит общий характер, специфика промышленных предприятий и промышленного производства вообще, а также КФС, в частности, не рассматриваются.

В работе [16] рассматривается применение методов и технологий онтологического моделирования при проектировании и внедрении информационных систем центра управления энергоснабжением (ЦУЭ) Сочинского энергорайона. Причем управление данными рассматривается только на уровне MES– и SCADA–систем.

В работе [17] представлена онтологическая модель насосной станции, однако не указано, для каких целей она создавалась.

В научно–популярной статье [18] отмечено, что нишевая автоматизация не может стать основой интеграции, необходима гиперавтоматизация, получившая реальное подкрепление в виде онтологического подхода к описанию предметной области и графовых баз, как адекватный инструмент реализации масштабных проектов.

Статья [19] посвящена разработке цифрового двойника энергетической системы, который рассматривается в качестве основного инструмента автоматизации управления объектами цифровой децентрализованной декарбонизированной энергетики.

В работе [20] разработана совместимая с Resource Description Framework онтологическая модель управляющего автомата сервиса киберфизической системы, основанная на представлении сервисов, их состояний и выходных символов в виде классов, а входных символов в виде отношений.

В работе [21] предложены схемы организации работ на производственных приборостроительных предприятиях Индустрии 4.0 типа «умная фабрика», оснащенных киберфизическими системами. Рассмотрен метод организации производственных работ по изготовлению изделий приборостроения с применением киберфизических систем и онтологий в условиях предприятий будущего на основе общей теории автоматизации проектирования, однако модели нижних уровней управления не рассматриваются.

В работе [22] рассмотрен объектно–ориентированный подход к проектированию систем управления промышленной автоматикой на основе языка UML и нового международного стандарта IEC 61499. В качестве примера рассматривается производственная система FESTO. Следует отметить, что описание производственной системы FESTO на языке UML можно считать за стартовую точку ее онтологического моделирования.

На основе приведенного выше обзора можно сделать вывод, что нет ни одной русскоязычной работы, непосредственно посвященной онтологическому моделированию систем промышленной автоматикой. Как правило, все работы в той или иной мере касаются моделирования и проектирования информационных систем, связанных с промышленной сферой.

Обзор иностранной литературы. Киберфизические системы (КФС) являются одним из основных атрибутов четвертой промышленной революции, известной также как «Индустрия 4.0» [23]. Примерами промышленных КФС являются разумные сети электроснабжения (Smart Grid) [24], производственные системы, сборочные производства, робототехнические комплексы и т.д.

КФС характеризуются многоаспектностью, структурной и функциональной сложностью. Исследования в данной области лежат на стыке многих дисциплин и находятся пока в начальной стадии разработки. Все это определяет необходимость разработки адекватных методов проектирования

КФС. Наиболее обещающим подходом является подход, основанный на использовании моделей [25]. Обзор методов проектирования, моделирования и интеграции КФС, а также примеры их использования представлены в работе [26].

Разноплановость описаний КФС, состоящих из физической, кибернетической и коммуникационной частей, требуют некоего унифицированного подхода к описанию, который позволял бы простую интеграцию частей в одно целое, повторное использование частей, а также поддерживал бы портабельность и интероперабельность. Данные требования в определенной мере могут быть удовлетворены использованием языков UML, SysML, XML. Однако недостатком этих языков является или нацеленность на представление синтаксической информации, либо ограниченная семантика. Намного большие возможности предлагают технологии семантического Web. Например, описания на основе онтологий позволяют представлять кроме синтаксической информации также и семантическую информацию. Кроме того, имеется возможность проведения онтологического резонинга, что полезно при анализе, верификации и валидации онтологической модели.

Онтологии и онтологические модели начали использоваться в моделировании КФС относительно недавно. В работе [27] предлагается семантическая среда (framework), основанная на использовании моделей, для системного проектирования, отслеживания требований, имитационного моделирования и оценки киберфизических систем. В данной работе для вычислений и принятия решений используются доменные онтологии.

В работе [28] онтологии используются для проверки консистентности различных представлений одной и той же КФС в процессе проектирования.

Работа [29] связана с разработкой структур знаний для поддержки корректного («correct-by-design») проектирования киберфизических систем (КФС). Эта статья представляет новую онтологическую базу знаний и логический вывод для поддержки принятия решений для КФС. Это позволяет разрабатывать детерминированные, доказуемые и выполнимые модели КФС,

поддерживаемые надежной семантикой, что усиливает подход к проектированию КФС на основе управления моделями.

В работе [30] предлагается подход к разработке цифрового представления всей информации, доступной об объекте и из объекта, который может представлять собой аппаратную систему или программную платформу. Цифровое представление основано на формализмах представления семантических знаний, таких как RDF, RDF Schema и OWL. В данной работе также представляется концепция компонента Semantic I4.0, который решает проблемы коммуникации и понимания в сценариях Индустрии 4.0 с использованием семантических технологий.

В работе [31] предлагается контекстная онтология, позволяющая иметь дело с высокодинамичными контекстами киберфизических систем путем явного разграничения системы и ее контекста.

В работе [32] показано, что онтологии в сочетании с ризонером являются полезным средством для семантического анализа управляющих приложений на основе международного стандарта IEC 61499.

Приведенный выше краткий обзор свидетельствует о фактах использования онтологических моделей для моделирования структур КФС, контекстов КФС, верификации проектов КФС, принятия решений в процессе проектирования, представления информации об объекте. Вместе с тем можно констатировать, что на данный момент не существует работ, в которых используются динамические онтологические модели, в которых изменения были бы заложены в их семантику.

Дальнейшим перспективным использованием онтологий является развитие на их основе технологии проектирования на основе управления моделями (Model-Driven Engineering – MDE) [33]. Возникшая новая технология получила название «Проектирование на основе управления онтологиями» (Ontology Driven Engineering – ODE).

Подходы к проектированию, основанные на онтологиях, все чаще используются в области разработки программного обеспечения [34]. В

настоящее время он часто используется в сочетании с проектированием на основе моделей (MDE), но в будущем он способен полностью заменить MDE. В целом, разработка программного обеспечения, основанная на онтологиях, относится к различным способам, с помощью которых онтологии могут способствовать улучшению моделей, методов, приемов и процессов разработки программного обеспечения. Преимущества включают улучшенную отслеживаемость модели/кода, возможность повторного использования артефактов и повышенный уровень совместимости и интеграции систем, а также проверку моделей.

В работе [35] предлагается метод автоматической генерации программного обеспечения систем автоматизации сетей Smart Grid из спецификаций, которые включают схему физической системы и предметно-ориентированные функциональные «рецепты». Сгенерированное программное обеспечение имеет компонентную организацию и реализует децентрализованный подход к управлению сетями Smart Grid, который уменьшает сложность проектирования систем автоматизации и модификации.

Основой подхода на основе управления онтологиями является трансформации онтологий. В работе [36] предлагается язык преобразования для RDF, который называется STTL. Это позволяет преобразовать RDF в RDF, а также любой другой текстовый формат. Оригинальность и мощь STTL в том, что он основан на SPARQL.

Работа [37] определяет язык OntoUML как формальную графовую грамматику и демонстрирует, как модели этого языка могут быть построены комбинированным применением онтологических паттернов, следуя ряду правил преобразования графа. В результате получается версия языка, полностью определенная как формальная грамматика онтологии паттернов.

Обзор собственных работ. В статье [38] рассматриваются вопросы моделирования и верификации производственной системы FESTO на основе высокоуровневых сетей Петри в системе CPNTools.

В статье [11] исследуется использование языка SPARQL в создании моделей мультиагентных систем в семантическом WEB. Авторы предлагают методы, которые помогают повысить эффективность и гибкость разработки таких систем. Они рассматривают применение SPARQL для работы с онтологиями, что позволяет более эффективно управлять данными и ресурсами в семантическом WEB.

В статье [39] предлагается онтологический подход к моделированию сетей Петри с использованием языка SPARQL. Рассматривается онтология сетей Петри, SPARQL-запросы, моделирующие динамику работы, а также вопросы построения клиент-серверных систем для интерпретации сетевых моделей.

В статье [40] рассматриваются два подхода к графическому моделированию производственной системы FESTO, первый из которых основан на Flesh-технологии, а второй – объектно-ориентированный. В рамках второго подхода разработан специализированный язык графического моделирования.

В статье [41] рассматривается замкнутая иерархическая модульная модель манипулятора Pick-and-Place, основанная на формализме сетевых систем «условие-событие».

В статье [42] предлагается абстрактная программная сервис-ориентированная модель системы транспортировки багажа в аэропорту, где каждое устройство системы моделируется отдельным базовым веб-сервисом. На более высоком уровне функционирование системы представлено композитным сервисом, через который осуществляется взаимодействие базовых сервисов. Эта модель может использоваться для имитационного моделирования и прототипирования систем транспортировки багажа, а также в учебном процессе.

В статье [43] предлагается программная сервис-ориентированная модель RnR-манипулятора, включающая как модель управления, так и модель оборудования, взаимодействующие друг с другом. Данная модель может быть

использована для имитационного моделирования и прототипирования реальных систем, а также в учебном процессе.

В статье [44] рассматривается модель производственной системы FESTO на основе продукционных правил, реализованная в инструментальной среде проектирования экспертных систем CLIPS. Для визуализации технологического процесса использована библиотека OpenGL

В статье [45] обсуждается применение концепции "Индустрия 4.0" в развитии промышленного производства и необходимость использования новых подходов к моделированию и проектированию мехатронных систем, которые переходят к статусу киберфизических систем. Предлагается инженерная методика для создания реконфигурируемых программных моделей мехатронных систем на основе сервиса-ориентированной архитектуры. В статье описывается методика построения модели каждого устройства как объекта автоматизации с использованием паттерна проектирования Model/View/Controller. Результатом работы являются программные модели PnP-манипулятора и системы транспортировки багажа в аэропорту, которые могут быть использованы для прототипирования. Для построения моделей используются программные средства Oracle SOA Suite, OpenESB и Netbeans IDE. Гибкость, реконфигурируемость и интероперабельность предложенных моделей позволяют быстро перейти от программных моделей к реальным системам.

В статье [46] рассматривается представление и трансформация онтологий киберфизических систем с использованием раскрашенных сетей Петри в сфере систем транспортировки багажа и сетей Smart grid.

В статье [47] рассматривается проблема использования формальных моделей при проектировании параллельных, распределенных и мультиагентных систем на основе семантической паутины в контексте концепции семантического веба. Цель исследования заключается в разработке методов и средств реализации высокоуровневых сетей Петри на основе онтологий, что может служить основой для построения систем данного класса.

Материалы и методы исследования основаны на онтологическом подходе к моделированию сложных систем, технологиях семантического веба, языке SPARQL и теории высокоуровневых сетей Петри. Результаты работы включают разработку онтологии раскрашенных сетей Петри, метода интерпретации на основе онтологий и SPARQL, комплекса программ OntoNet для интерпретации раскрашенных сетей Петри в среде семантического веба и сетевой модели потоковой обработки событий. Предложенные методы и средства могут быть использованы для разработки, прототипирования и реализации распределенных, параллельных и многоагентных систем обработки событий, данных и знаний в рамках семантического веба.

В статье [48] рассматривается программная имитационная дискретно–событийная модель тестирующей станции производственной системы FESTO, а также процесс разработки этой модели. Особенностью модели является возможность формирования журнала событий

В статье [49] предлагается подход к синтезу формальных моделей производственных систем на основе метода Process mining. С использованием данного подхода по журналам событий, полученных в ходе имитационных экспериментов, сгенерированы формальные модели процессов, протекающих в тестирующей станции производственной системы FESTO, а также произведено их исследование.

В статье [50] рассматривается функционально–блочная модель тестирующей станции производственной системы FESTO на основе международного стандарта IEC 61499, разработанная с использованием шаблона проектирования MVC.

В статье [51] предлагается методика моделирования статики и динамики тестирующей станции производственной системы FESTO на основе онтологий и их преобразований с использованием языка SPARQL.

В статье [52] предлагается OWL–онтология модульного сетевого формализма, обладающего свойствами временных, самомодифицирующихся, приоритетных и числовых сетей Петри. Для валидации сетевой модели

используются онтологические языки SWRL и SPARQL. Рассматривается пример онтологического представления сетевой модели системы массового обслуживания.

В статье [53] рассматривается имитатор промышленной установки FESTO, представляющий собой распределенную вычислительную систему на основе локальной вычислительной сети. Модель кибернетической части построена на основе международного стандарта IEC 61499, а модель физической части – с использованием языка программирования высокого уровня.

В статье [54] предлагается онтологический подход к моделированию дискретных производственных систем. Для отображения динамики функционирования системы используется язык SPARQL. Подход иллюстрируется на примере тестовой и обрабатывающей станций производственной системы FESTO.

1.3 Производственная система FESTO

1.3.1 Неформальное описание

Производственная система FESTO предназначена для изготовления некоторых законченных деталей из поступающих на ее вход заготовок [12]. Она состоит из трех станций: распределительной (Distribution Station), тестирующей (Testing Station) и обрабатывающей (Processing Station) (рисунки 1.1).

Заготовка представляет собой шайбу из пластмассы или металла. В процессе изготовления детали заготовка последовательно проходит через все три станции. Распределительная станция служит для буферирования поступающих от внешнего производителя заготовок, тестирующая станция – для контроля и отбраковки нестандартных заготовок, а обрабатывающая станция – для сверления в заготовке отверстия. Обработанная деталь может быть передана внешнему потребителю готовых деталей. В дальнейшем для краткости заготовку будем называть деталью (workpiece).



Рисунок 1.1 – Производственная система FESTO

Техпроцесс обработки деталей является одним из определяющих факторов при разработке системы управления. Ниже приводится краткое описание техпроцесса обработки. Внешний производитель помещает детали в магазин распределительной станции, где они могут накапливаться. Буферирование деталей в магазине способствует сглаживанию темпов работы внешнего производителя и производственной системы FESTO. При помощи специального выталкивателя (поршня) деталь выталкивается из дна магазина и сдвигается в выходное место магазина, откуда шарнирный рычаг перемещает ее на тестирующую станцию. В данном случае под выходным местом магазина понимается площадка, на которую из магазина выталкивается деталь. Выходное место магазина не является исполнительным устройством и с ним не связаны никакие датчики. Удерживание детали на шарнирном рычаге осуществляет расположенная на его конце вакуумная присоска.

При поступлении детали на тестирующую станцию (а именно, на тестирующее место) датчики–детекторы определяют цвет и материал, из которого изготовлена деталь. При ненормальном цвете или материале деталь выталкивается (отбраковывается) из тестирующей станции. Если цвет и материал детали находятся в норме, то деталь поднимается лифтом к

устройству измерения высоты детали (измерителю высоты). Если в результате измерения оказывается, что высота детали нормальная, то деталь выталкивается в скользящий желоб, по которому она поступает на обрабатывающую станцию. В противном случае (ненормальная высота) деталь опускается на лифте вниз, где она выталкивается из тестирующей станции и, таким образом, исключается из дальнейшей обработки. Скользящий желоб имеет заслонку, которая позволяет заблокировать поступление детали на обрабатывающую станцию.

На обрабатывающую станцию деталь поступает в лунку рабочего (поворотного) стола. Рабочий стол поворачивается вокруг своей оси на одну позицию по часовой стрелке, перенося тем самым деталь в позицию для сверления. Сверление осуществляется с помощью дрели, способной двигаться по направляющим вверх и вниз. Перед сверлением деталь прижимается специальным прижимным стержнем (зажимом), входящим в боковую прорезь лунки. После окончания сверления рабочий стол поворачивается еще на одну позицию по часовой стрелке и деталь оказывается в позиции для проверки отверстия. Проверка осуществляется с помощью специального проверочного стержня, опускаемого вниз по центру детали. Если отверстие нормального качества, то стержень проходит сквозь него и касается специального датчика, что свидетельствует о норме. После окончания проверки отверстия рабочий стол поворачивается на одну позицию по часовой стрелке, перенося деталь в позицию для выгрузки детали из обрабатывающей станции. Выгружаемая деталь передается внешнему потребителю деталей. В частности, подобным потребителем может быть сортировочная станция, поставляемая на рынок учебной техники той же фирмой FESTO Didactic.

1.3.2 Исполнительные устройства

Устройства распределительной станции. Распределительная станция (Distribution Station) включает в свой состав блок магазина и блок транспортного рычага. В свою очередь блок магазина состоит из вертикальной приемной колбы (цилиндра), играющей роль буфера поступающих заготовок,

и выталкивателя. Блок транспортного рычага состоит из рычага и присоски. Внешний вид распределительной станции показан на рисунке 1.2. Внешний вид блока магазина показан на рисунке 1.3.

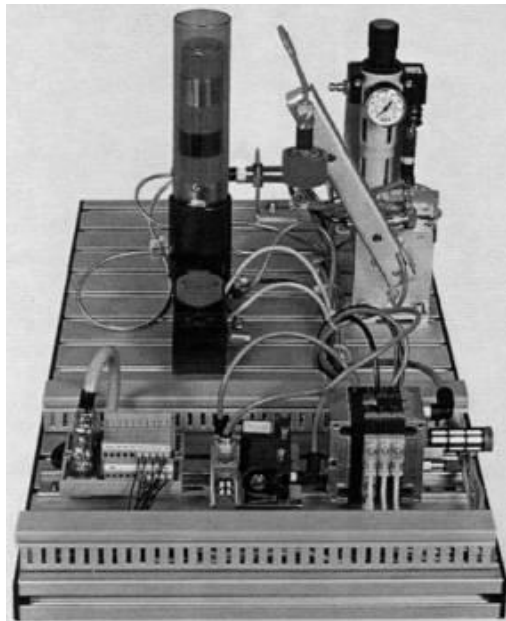


Рисунок 1.2 – Внешний вид распределительной станции



Рисунок 1.3 – Внешний вид блока магазина

Входные сигналы (Input) блока магазина:

- E0.0 Out slidegate valve back,
- E0.1 Out slidegate valve front,
- E0.6 Magazine empty,
- E8.4 Testing station active.

Выходные сигналы (Output) блока магазина:

- A0.0 S M0.0 slidegate_valve_front,
- A0.0 R M0.1 slidegate_valve_back,
- A4.2 Display_H3 («magazine empty»),

– A8.0 Di_active («Distribution active»).

Устройство «Вертикальная приемная колба» (Cylinder) имеет один выходной осведомительный сигнал E0.6 – Магазин пуст. Входных сигналов нет. Примечание: имеется одно входное механическое воздействие – добавление детали в колбу.

Устройство «Выталкиватель» (Slidegate valve) принимает следующие входные сигналы–команды из контроллера:

- A0.0 S M0.0 – выдвинуть поршень вперед,
- A0.0 R M0.1 – втянуть поршень назад.

Выходы (осведомительные сигналы с датчиков) следующие:

- E0.0 – поршень втянут,
- E0.1 – поршень выдвинут.

Внешний вид блока транспортного рычага (Transfer lever) показан на рисунке 1.4.



Рисунок 1.4 – Внешний вид блока транспортного рычага

Входные сигналы (Input) блока транспортного рычага:

- E0.2 Vane Drive at magazine,
- E0.4 Vane Drive at testing station,
- E0.5 Workpieces sucked in.

Выходные сигналы (Output) блока транспортного рычага:

- A0.1 S M0.3 Vana_drive_ma,
- A0.1 R M0.2 Swivel_arm_te,
- A0.2 Di_active («Vakuum_off»),
- A0.3 Di_active («Vakuum_on»),
- A8.0 Di_active («Distribution active»).

Устройство «Рычаг» (Swivel) принимает следующие входные сигналы—команды из контроллера:

- A0.1 – передвинуть рычаг на сторону распределительной станции,
- A0.2 – передвинуть рычаг на сторону тестирующей станции.

Выходы (осведомительные сигналы с датчиков) следующие:

- E0.2 – рычаг находится на стороне распределительной станции,
- E0.4 – рычаг находится на стороне тестирующей станции.

Устройство «Присоска» (Sucker) принимает следующие входные сигналы—команды из контроллера:

- A0.2 – включить вакуум,
- A0.3 – выключить вакуум.

Выход (осведомительный сигнал с датчика) один единственный: E0.5 – деталь «присосана» вакуумом к рычагу (иначе, вакуум включен).

Устройства тестирующей станции. Тестирующая станция (Testing Station) включает в свой состав блок детекторов, блок лифта, блок измерений и блок скользящего желоба. В свою очередь блок лифта состоит из устройства «Лифт» и устройства «Выталкиватель».

Внешний вид тестирующей станции показан на рисунке 1.5. Внешний вид блока детекторов представлен на рисунке 1.6.

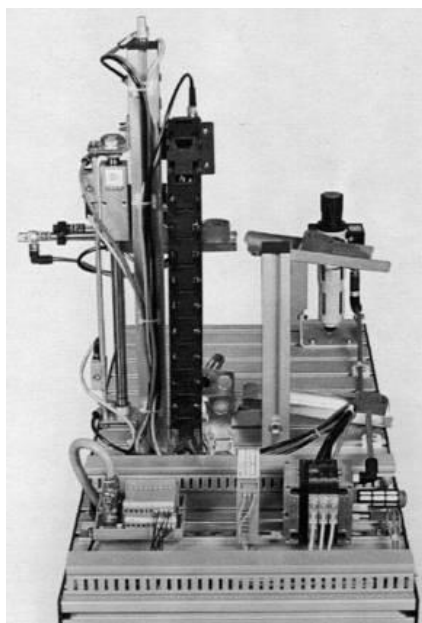


Рисунок 1.5 – Внешний вид тестирующей станции



Рисунок 1.6 – Внешний вид блока детекторов

Входные сигналы (Input) блока детекторов:

- E0.0 Metallic workpiece,
- E0.1 Workpiece in testing station,
- E0.2 Workpiece not black,
- E8.4 Processing station active.

Выходные сигналы (Output) блока детекторов:

- A4.2 Display_H3 («Material testing»),
- A8.4 TS_active («Testing station active»).

Блок детекторов не имеет входов, однако имеет следующие выходы для вывода осведомительных сигналов с датчиков:

- E0.1 – деталь присутствует,
- E0.0 – деталь металлическая,
- E0.2 – деталь нечерная.

Внешний вид блока лифта представлен на рисунке 1.7.



Рисунок 1.7 – Внешний вид блока лифта

Входные сигналы (Input) блока лифта:

- E0.3 Lift down,
- E0.4 Lift up,
- E0.5 Slidegate valve in,

– E8.4 Processing station active.

Выходные сигналы (Output) блока лифта:

- A0.0 Lift_down,
- A0.1 Lift_up,
- A0.2 S M0.0 Slidegate_valve_front,
- A0.2 R M0.1 Slidegate_valve_back,
- A8.4 Di_active («Distribution active»).

Устройство «Лифт (подъемник)» (Lift) принимает следующие входные сигналы–команды из контроллера:

- A0.0 – опустить лифт вниз,
- A0.1 – поднять лифт вверх.

Выходы (осведомительные сигналы с датчиков) следующие:

- E0.3 – лифт находится в верхнем положении,
- E0.4 – лифт находится в нижнем положении.

Устройство «Выталкиватель» (Slidegate valve) принимает следующие входные сигналы–команды из контроллера:

- A0.2 S M0.0 – выдвинуть поршень вперед,
- A0.2 R M0.1 –втянуть поршень назад.

Выход (осведомительный сигнал с датчика) следующий: E0.5 – поршень находится в крайнем положении.

Внешний вид модуля измерений показан на рисунке 1.8.

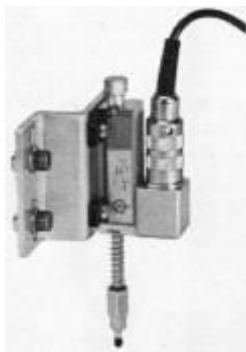


Рисунок 1.8 – Внешний вид модуля измерений

Входные сигналы (Input) модуля измерений:

- E0.6 Measurer out,

- E8.4 Processing station active,
- EW304 Analog_In.

Выходные сигналы (Output) модуля измерений:

- A0.3 S M0.2 Measurer_down,
- A0.3 R M0.3 Measurer_up,
- A8.4 TS_aktiv («Testing station active»).

Модуль измерений имеет следующие входы (сигналы–команды из контроллера):

- A0.3 S M0.2 – опустить измерительный стержень вниз,
- A0.3 R M0.3 – поднять измерительный стержень вверх.

Выходы (осведомительные сигналы с датчиков) следующие:

- E0.6 – измерительный стержень поднят,
- EW304 – измерительный стержень опущен (иначе – аналоговые измерения совершены).

Внешний вид блока скользящего желоба (Slide) показан на рисунке 1.9.

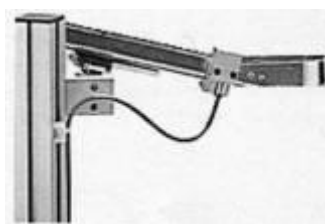


Рисунок 1.9 – Внешний вид блока скользящего желоба

Входной сигнал (Input) блока скользящего желоба: E8.4 Processing station active.

Выходные сигналы (Output) блока скользящего желоба:

- A0.4 S M0.4 Stopper_out,
- A0.4 R M0.5 Stopper_in,
- A8.4 TS_active («Testing station active»).

Блок скользящего желоба имеет следующие входы (сигналы–команды из контроллера):

- A0.4 S M0.4 – опустить заслонку стоппера вниз,
- A0.3 R M0.3 – поднять заслонку стоппера вверх.

Выходы (осведомительные сигналы с датчиков) отсутствуют.

Устройства обрабатывающей станции.

Обрабатывающая станция (Processing Station) включает в свой состав блок поворотного стола, блок дрели и блок проверки качества отверстия в детали. В свою очередь блок дрели состоит из устройства «Дрель» и устройства «Зажим».

Внешний вид тестирующей станции показан на рисунке 1.10. Внешний вид блока детекторов представлен на рисунке 1.11.

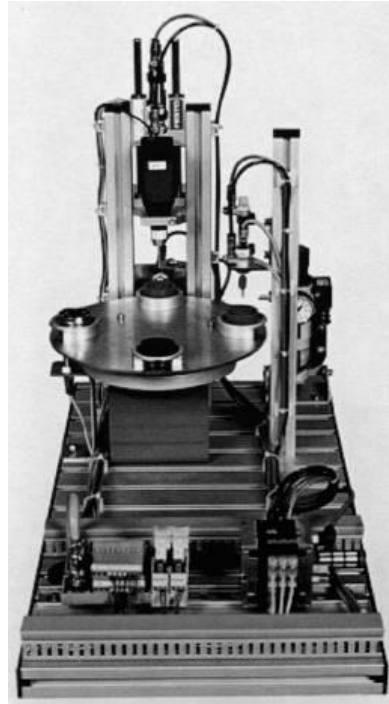


Рисунок 1.10 – Внешний вид обрабатывающей станции



Рисунок 1.11 – Внешний вид блока поворотного стола

Входные сигналы (Input) блока поворотного стола:

- E0.0 Workpiece in surrender,
- E0.1 Table in position,
- E8.4 Testing station active.

Выходные сигналы (Output) блока поворотного стола:

- A0.2 table_rotate,
- A8.4 PS_active («Processing station active»).

Блок поворотного стола принимает следующую сигнал–команду из контроллера: A0.2 – повернуть стол.

Выходы (осведомительные сигналы с датчиков):

- E0.0 – деталь в сдаче,
- E0.1 – поворотный стол находится в позиции (одной из четырех).

Внешний вид блока дрели показан на рисунке 1.12.



Рисунок 1.12 – Внешний вид блока дрели

Входные сигналы (Input) блока дрели:

- E0.2 Bracked in,
- E0.3 Bracked out,
- E0.4 Driller in,
- E0.5 Driller out.

Выходные сигналы (Output) блока дрели:

- A0.1 Drill,
- A0.3 Driller_down,
- A0.4 Driller_up,
- A0.5 S M0.0 Bracked_back,
- A0.5 R M0.1 Bracker_front,
- A8.4 PS_active («Processing station active»).

Устройство «Дрель» (Drill) принимает следующие входные сигналы–команды из контроллера:

- A0.1 – включить вращение дрели,
- A0.3 – опустить дрель вниз,
- A0.4 – поднять дрель вверх.

Выходы устройства «Дрель» (осведомительные сигналы с датчиков) приведены ниже:

- E0.4 – дрель находится в опущенном состоянии,
- E0.5 – дрель находится в поднятом состоянии.

Устройство «Зажим» (Clamp) принимает следующие входные сигналы—команды из контроллера:

- A0.5 S M0.0 – подвести зажимной стержень (зажать деталь),
- A0.5 R M0.1 – отвести зажимной стержень (разжать деталь).

Выходы (осведомительные сигналы с датчиков) устройства «Зажим» приведены ниже:

- E0.2 – зажимной стержень находится в подведенном состоянии (деталь зажата),
- E0.3 – зажимной стержень находится в отведенном состоянии (деталь разжата).

Внешний вид проверки качества отверстия в детали показан на рисунке 1.13.

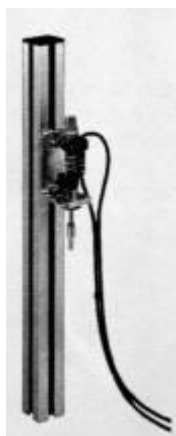


Рисунок 1.13 – Внешний вид блока проверки качества отверстия в детали

Входные сигналы (Input) блока проверки качества отверстия в детали:

- E0.6 Checker in,
- E0.7 Checker out.

Выходные сигналы (Output) блока проверки качества отверстия в детали:

- A0.6 S M0.2 Checker_down,
- A0.6 R M0.3 Checker_up,
- A8.4 PS_active («Processing station active»).

Блок проверки качества отверстия в детали принимает следующие сигналы–команды из контроллера:

- A0.6 S M0.2 – опустить проверочный стержень вниз,
- A0.6 R M0.3 – поднять проверочный стержень вверх.

Выходы (осведомительные сигналы с датчиков) блока проверки качества отверстия приведены ниже:

- E0.6 – проверочный стержень находится в опущенном состоянии,
- E0.6 – проверочный стержень находится в поднятом состоянии.

1.3.3 Алгоритмы работы

На рисунке 1.14 показан алгоритм взаимодействия станций между собой, а также возможность контроля ее оператором и выдачи готовой продукции.

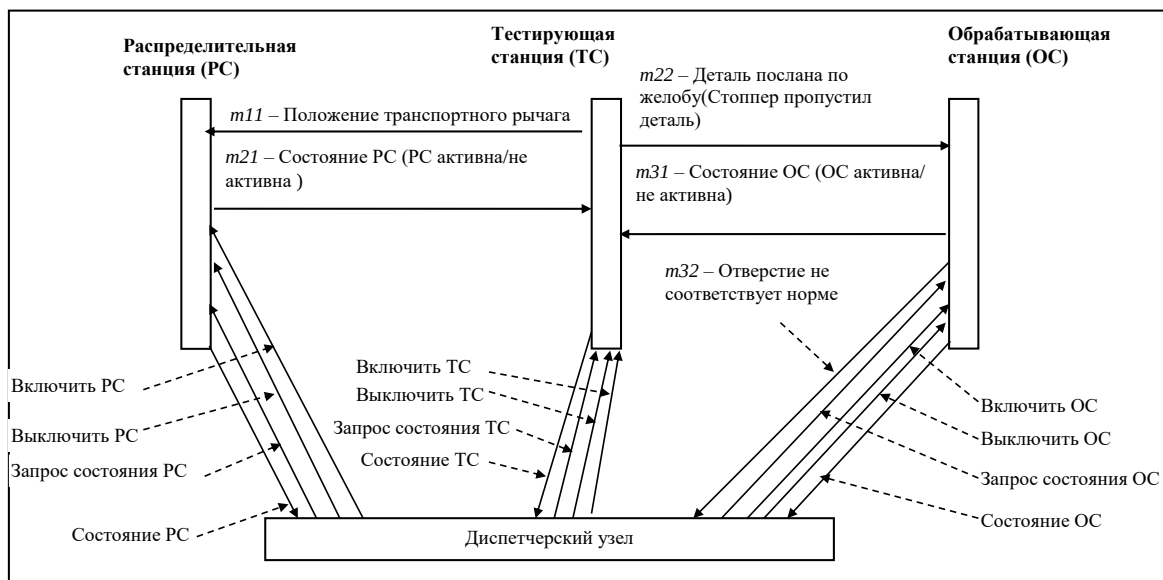


Рисунок 1.14 – Алгоритм взаимодействия станций FESTO

На рисунке 1.14 можно увидеть, что станции взаимодействуют между собой с помощью четырех сигналов: сигналов активности распределительной и обрабатывающей станций, а также сигнала положения транспортного рычага

и сигнала работы стоппера.

Рассмотрим алгоритмов работы функциональных блоков, осуществляющих управление системой. Алгоритмы описаны на языке высокого уровня. При их разработке учитывался принцип взаимодействия станций между собой в соответствии с рисунком 1.14.

При описании системы будем считать, что управление транспортным рычагом полностью осуществляется блоком управления распределительной станции. В представленных ниже алгоритмах не отражены сигналы с диспетчера «Включить РС», «Выключить РС», «Запрос состояния РС».

Условия, используемые в алгоритмах:

- АС – аппаратный сигнал,
- ПП – признак, формируемый программой,
- РС – признак, формируемый на основе сообщения, приходящего из другой станции.

Начальное состояние:

- z11 = «Магазин свободен»,
- z12 = «Выходное место свободно»,
- z13 = «Рычаг на стороне РС»,
- z21' = «ТС свободна».

Ниже представлен последовательный алгоритм функционирования установки FESTO:

НАЧАЛО АЛГОРИТМА

ЦИКЛ

ЕСЛИ Магазин не пуст, ТО

ДЕЛАТЬ

ЕСЛИ транспортный рычаг находится на РС, ТО

Перенести транспортный рычаг на ТС -- чтобы не мешал работе РС

Выдвинуть поршень выталкивателя 1 -- Вытолкнуть деталь в выходное место

Задвинуть поршень выталкивателя 1

Перенести транспортный рычаг на РС -- для захвата детали

Включить вакуум присоски

Перенести транспортный рычаг на ТС -- транспортировка детали

Выключить вакуум присоски

```

Перенести транспортный рычаг на РС      -- чтобы не мешал
работе ТС
ЕСЛИ деталь присутствует на ТС, ТО      -- это сигнал с
датчика
    ЕСЛИ цвет или материал детали не подходящие, ТО      --
    это сигналы с датчика
        ДЕЛАТЬ
            Выдвинуть поршень выталкивателя 2      --
            Вытолкнуть деталь в брак
            Задвинуть поршень выталкивателя 2
            ПЕРЕЙТИ НА Кон_Обр
        КОНЕЦ
    ИНАЧЕ -- на ТС присутствует деталь - нормальная по
    цвету и материалу
        ДЕЛАТЬ
            Поднять лифт
            Опустить измерительный стержень
            Поднять измерительный стержень
            ЕСЛИ высота детали не подходящая, ТО
                ДЕЛАТЬ
                    Опустить лифт
                    Выдвинуть поршень выталкивателя 2      --
                    Вытолкнуть деталь в брак
                    Задвинуть поршень выталкивателя 2
                    ПЕРЕЙТИ НА Кон_Обр
                КОНЕЦ
            Выдвинуть поршень выталкивателя 2      --
            Вытолкнуть деталь в желоб
            Задвинуть поршень выталкивателя 2
            Опустить лифт
            Поднять заслонку желоба      -- для скатывания
            детали на ОС
            Опустить заслонку желоба
            Повернуть стол
            Подвести зажимной стержень
            Включить дрель
            Опустить дрель
            Поднять дрель
            Выключить дрель
            Отвести зажимной стержень
            Повернуть стол
            Подвести проверочный стержень
            Отвести проверочный стержень
            ЕСЛИ отверстие не соответствует норме, ТО
                ПОСЛАТЬ СИГНАЛ N1
            Повернуть стол
            Выгрузить деталь
        КОНЕЦ
    КОНЕЦ
Кон_Обр: КОНЕЦ_ЦИКЛА
КОНЕЦ_АЛГОРИТМА

```

Алгоритм работы распределительной станции.

Условия, используемые в алгоритме:

- z_{11} – состояние выходного места (занято/свободно) [ПП],
- z_{12} – положение транспортного рычага (на стороне РС/на стороне ТС/движется) [АС],
- z_{21}' – Состояние тестирующей станции (ТС) (активна/не активна) [ПС].

Обозначения:

- АС – аппаратный сигнал,
- ПП – признак, формируемый программой,
- ПС – признак, формируемый на основе сообщения, приходящего из другой станции.

Начальное состояние:

- z_{11} = «Выходное место свободно»,
- z_{12} = «Рычаг на стороне РС»,
- z_{21}' = «ТС не активна».

Посылаемое сообщение: m_{11} – положение транспортного рычага.
Посылается на ТС.

Принимаемое сообщение: m_{21} – состояние ТС (готова/не готова к приему детали).

Ниже представлен алгоритм работы распределительной станции:

основная программа

 запустить процесс: прием_сообщений

 запустить процесс: обработка_деталей

конец_программы

процесс: прием сообщений

 цикл (бесконечный)

 принять сообщение (m_{21})

$z_{21}' = m_{21}$ -- установить признак состояния ТС

 конец_цикла

конец_процесса

процесс: обработка_деталей

 цикл (бесконечный)

 выбор

```

1. если z11="выходное место свободно" & z12="рычаг на стороне
тс", то
    делать
        вытолкнуть деталь из магазина в выходное место
        z11= "выходное место занято"
    конец

2. если z11= "выходное место занято" & z12= "рычаг на стороне
рс" & z21'= "тс не активна", то
    делать
        послать сообщение m11 ("рычаг движется на тс") на тс
        перенести деталь с рс на тс
        послать сообщение m11 ("рычаг на стороне тс") на тс
        z12= "выходное место свободно"
    конец

3. если z12= "рычаг на стороне тс" & z21'= "тс активна", то
    делать
        перенести рычаг на рс -- чтобы не мешал тс
        послать сообщение m11 ("рычаг на стороне рс") на тс
    конец

4. если z11="магазин занят" & z12="выходное место свободно" &
z13= "рычаг на стороне рс" & z21'= "тс свободна", то
    делать
        перенести рычаг на тс -- чтобы не мешал рс
        послать сообщение m11 ("рычаг на стороне тс") на тс
    конец

5. если z11= "выходное место занято" & z12= "рычаг на стороне
тс", то
    делать -- как и в правиле 4
        перенести рычаг на тс -- для переноса детали с рс
        послать сообщение m11 ("рычаг на стороне тс") на тс
    конец

конец_выбор
конец_цикла
конец_процесса

```

Состояние ожидания: $z12 = \text{«Рычаг на стороне РС»}$ & $z21' = \text{«ТС занята»}$.

В этом состоянии все действия на РС запрещены: i – выталкивать деталь из магазина нельзя, так как мешает рычаг, ii – переносить деталь на ТС нельзя, так как она занята.

Распределительная станция все время информирует тестирующую станцию о положении транспортного рычага путем послышки соответствующих сообщений. Формирование сообщения производится в момент, когда рычаг прибывает в одно из своих крайних положений или когда изменяется его

положение.

Для Правила 2 считается, что посылаемое сообщение о состоянии рычага приходит на тестирующую станцию быстрее, чем сам рычаг. Это необходимо, чтобы при прибытии рычага на ТС эта ТС не решила вдруг, что рычаг еще на РС. А то она начнет обработку детали, когда на ее стороне лежит рычаг.

Оператор «принять сообщение» является оператором с ожиданием. То есть выполнение оператора заканчивается только тогда, когда принято сообщение.

Конструкция ниже запускает два параллельных процесса:

ЗАПУСТИТЬ ПРОЦЕСС: ПРИЕМ_СООБЩЕНИЙ
ЗАПУСТИТЬ ПРОЦЕСС: ОБРАБОТКА_ДЕТАЛЕЙ

В представленном алгоритме не отражены сигналы с диспетчера «Включить РС», «Выключить РС», «Запрос состояния РС».

Алгоритм работы тестирующей станции.

Признаки, используемые в условиях алгоритма:

- z13' – положение транспортного рычага (на стороне РС/на стороне ТС/движется) [ПС],
- z22 – наличие детали в блоке детекторов (присутствует/отсутствует) [АС],
- z23 – цвет детали (годный/негодный) [АС],
- z24 – материал детали (годный/негодный) [АС],
- z25 – высота детали (годная/негодная) [АС],
- z31' – состояние обрабатывающей станции (ОС) (активна/не активна) [ПС].

Посылаемые сообщения:

- m21 – состояние ТС (готова/не готова к приему детали). Посылается на РС,
- m22 – деталь послана по желобу на ОС. Посылается на ОС.

Принимаемые сообщения:

- m11 – положение транспортного рычага,

– m31 – состояние ОС (готова/не готова к приему детали).

Обозначения:

– АС – аппаратный сигнал,

– ПП – признак, формируемый программой,

– ПС – признак, формируемый на основе сообщения, приходящего из другой станции.

Ниже представлен алгоритм работы тестирующей станции:

основная программа

 запустить процесс: прием_сообщения_1

 запустить процесс: прием_сообщения_2

 запустить процесс: обработка_деталей

конец_программы

процесс: прием сообщения_1

 цикл (бесконечный)

 принять сообщение (m11)

 z13'=m11 – установить признак положения рычага

 конец_цикла

конец_процесса

процесс: прием сообщения_2

 цикл (бесконечный)

 принять сообщение (m31)

 z31'=m31 – установить признак состояние ос

 конец_цикла

конец_процесса

процесс: обработка_деталей

цикл (бесконечный)

 если z22= " деталь в блоке детекторов" & z13'= "рычаг на
стороне рс", то

 делать

 послать сообщение m21 ("тс не готова принять деталь") на
 рс

 если z23="цвет негодный" или z24="материал негодный", то
 делать

 вытолкнуть деталь

 прервать дальнейшую обработку детали

 послать сообщение m21 ("тс готова к приему детали")
 на рс

 конец

 поднять лифт

 измерить высоту детали

 если z25= "высота негодная", то

 делать

 опустить лифт

 вытолкнуть деталь

 прервать дальнейшую обработку детали

```

        послать сообщение m21 ("тс готова к приему детали")
        на рс
    конец
ждать z31' = "ос свободна" -- по сути дела цикл ожидания
сброс признака z31'
послать сообщение m22 ("деталь послана по желобу") на ос
открыть и закрыть заслонку желоба
опустить лифт
послать сообщение m21 ("тс готова к приему детали") на рс
конец
конец_цикл

```

Замечание: под сбросом признака понимается присвоение ему неопределенного значения.

Считается, что сообщение о посылке детали дойдет до ОС быстрее, чем сама деталь скатится по желобу в ОС.

Оператор «ПРИНЯТЬ СООБЩЕНИЕ» является оператором с ожиданием. То есть выполнение оператора заканчивается только тогда, когда принято сообщение.

В представленном алгоритме не отражены сигналы с диспетчера «Включить ТС», «Выключить ТС», «Запрос состояния ТС».

Алгоритм работы обрабатывающей станции

Признаки, используемые в условиях алгоритма:

- z32 – состояние 4-й ячейки (занята деталью/свободна) [АС],
- z33 – соответствие отверстия.

Посылаемые сообщения:

- m31 – состояние ОС (готова/не готова к приему детали). Посылается на ТС,
- m32 – просверленное отверстие не соответствует норме. Посылается на диспетчера.

Принимаемое сообщение: m22 – сообщение от ТС о посылке детали по желобу.

Структуры данных:

D [1..4] – вектор наличия детали в ячейках (лунках). Если D[i]=1, то в i-й ячейке есть деталь.

Ниже представлен алгоритм работы обрабатывающей станции:

```

основная программа
запустить процесс: прием_сообщений
запустить процесс: обработка_деталей
конец_программы
процесс: прием_сообщений
цикл (бесконечный)
    получить сообщение (m22)
    d[1]=1 -- установить признак наличия детали в 1-й ячейке
конец_цикла
конец_процесса
процесс: обработка_деталей
цикл (бесконечный)
    если ( $\exists i \in 1..3 (d[i]=1) \vee z32 = e3.0.3$  "ячейка № 4
        занята") & z31=e3.1 "стол не движется", то
        делать
        послед_обработка_деталей_на_столе
            выдать сообщение m31 ("ос не готова получить деталь")
            на тс
        задержка на время t – время доставки сообщения на тс
        циклический сдвиг вектора d вправо на одну позицию
        повернуть стол на одну позицию
        {
            выдать сигнал a3.2 в блок поворотного стола «повернуть
            стол»
        }
        выдать сообщение m31 ("ос готова получить деталь") на
        тс
конец
конец_цикла
конец_процесса

процедура: послед_обработка_деталей_на_столе
если d[2]= 1, то сверлить
если d[3]= 1, то проверить_отверстие
если z32= e3.0.3"ячейка № 4 занята", то убрать_деталь
конец
процедура: сверлить
подвести зажимной стержень
{
    выдать сигнал a0.5 s m0.0 в блок дрели «подвести зажимной
    стержень (зажать деталь)»
    получить сигнал e3.2 – «деталь зажата»
}
включить дрель
{
    выдать сигнал a3.1.0 в блок дрели «включить вращение дрели»
}
опустить дрель
{
    выдать сигнал a3.3 в блок дрели «опустить дрель вниз»
    получить сигнал e3.4 – «дрель в опущенном состоянии»
}
поднять дрель

```

```

{
выдать сигнал а3.4 – «в блок дрели «поднять дрель вверх»
получить сигнал е3.5 – «дрель в поднятом состоянии»
}
выключить дрель
{
выдать сигнал а3.1.0 в блок дрели «выключить вращение дрели»
}
отвести зажимной стержень
{
    выдать сигнал а0.5 r m0.1 в блок дрели «отвести зажимной
    стержень (разжать деталь)
получить сигнал е3.3 – «деталь разжата»
}
конец_процедуры
процедура: проверить_отверстие
подвести проверочный стержень
{
    выдать сигнал а0.6 s m0.2 в блок проверки качества
    отверстия « опустить проверочный стержень вниз»
получить сигнал е3.6 – «проверочный стержень опущен»
}
отвести проверочный стержень
{
    выдать сигнал а0.6 s m0.3 в блок проверки качества
    отверстия « поднять проверочный стержень вверх»
получить сигнал е3.7 – «проверочный стержень поднят»
}
получить сигнал е3.10.0 v е3.10.1 – «соответствие отверстия»
если z33= е3.10.0 отверстие не соответствует норме, то
    выдать сообщение m32 (“отверстие не соответствует норме”)
    на диспетчер
конец_процедуры
процедура: убрать_деталь
убрать деталь
{
выдать сигнал а3.5 в блок поворотного стола «выгрузить деталь»
получить сигнал е3.9 – «ос свободна»
}
d[4]=0 – сбросить признак наличия детали в 4-й ячейке
конец_процедуры

```

Вектор D представляет разделяемые данные между процессами. Условие $\exists i \in 1..3 (D[i]=1)$ означает, что имеется деталь хотя бы в одной из трех первых лунок.

Возможна ситуация, когда ОС послала сигнал на ТС, что она не готова ПОЛУЧИТЬ деталь, стала поворачивать стол, однако сообщение задержалось и пришло с опозданием на ТС, ТС не получив вовремя сообщения, отрыла заслонку и деталь скатилась по желобу в крутящийся стол. Чтобы исключить

эту ситуацию введена временная задержка после отправки сообщения о неготовности станции. Величина этой временной задержки не меньше, чем время доставки сообщения на ТС.

ОС не готова к приему новой детали лишь при движении (повороте) рабочего стола (поскольку деталь в этом случае не попадет в ячейку).

2 Онтологическое моделирование производственной системы FESTO

2.1 Общая методика онтологического моделирования

Для моделирования дискретных систем промышленной автоматике, функционирующих в среде семантического Web, предлагается использовать подход, основанный на управлении онтологиями (ontology-driven approach) [10]. При этом система представляется как онтология или совокупность взаимосвязанных онтологий, а функционирование системы – как последовательность изменения онтологий. Онтология описывает структуру системы взаимодействующих устройств и процессов, сигналы, которыми они обмениваются между собой, а также наборы данных и знаний, с которыми работает система в целом. Также можно сказать, что онтология в данном случае определяет обобщенное состояние системы. В нашем случае будут использоваться только RDF(S)-онтологии и, таким образом, состояние системы представляет собой экземпляр RDF-графа. В процессе функционирования, как правило, будет изменяться онтология Abox, однако при радикальных изменениях, происходящих в системе, возможно также и изменение онтологии Tbox.

Для реализации трансформации онтологий предлагается использование языка SPARQL Update, являющегося на данный момент самым мощным средством преобразования RDF-графов [8]. Сам по себе язык SPARQL Update является самодостаточным для решения поставленной задачи.

Предложенная методика онтологического моделирования включает следующие шаги:

- разработку обобщенной онтологии станций производственной системы FESTO,
- разработку частных онтологий распределительной, тестирующей и обрабатывающей станций производственной системы FESTO, описывающих как физические компоненты станций и обрабатываемые детали, так и их состояния,

- разработку семантических ограничений, позволяющих проводить семантический анализ,
- разработку системы продукционных правил (ПП), описывающих функционирование станций в терминах изменения состояния компонентов станций,
- верификацию системы продукционных правил на основе сетей Петри,
- разработку SPARQL-запросов, реализующих продукционные правила,
- разработку клиентского приложения для поддержки работы с онтологиями станций,
- информационное имитационное моделирование станций производственной системы FESTO с использованием клиент-серверной системы, включающей разработанное клиентское приложение и SPARQL-сервер,
- размещение онтологий станций производственной системы FESTO в семантическом Web для совместного использования.

2.2 Обобщенная онтология станций производственной системы FESTO

2.2.1 Онтологическая модель

Ниже представлена обобщенная Tbox-онтология станций производственной системы FESTO, с помощью которой можно описать распределительную, тестирующую и обрабатывающую станции данной системы. Упрощенно говоря, Tbox-онтология определяет онтологию на уровне классов. Частные онтологии станций производственной системы FESTO представляются в виде соответствующих Abox-онтологий (то есть, онтологий на уровне экземпляров). Следует заметить, что предложенная Tbox-онтология может использоваться для представления, моделирования и прототипирования других мехатронных систем и систем промышленной автоматизации, а не только станций производственной системы FESTO.

При разработке онтологии были приняты следующие ограничения. Считается, что:

- на каждой станции может быть реализован только один технологический процесс обработки деталей,
- на каждой станции может обрабатываться только одна деталь,
- все операции на каждой из станций выполняются последовательно,
- все изменения состояний устройств производятся скачком в дискретные моменты времени. Непрерывные процессы не рассматриваются, например, не рассматривается движение во времени механических частей устройств (поршня, лифта, поворотного стола и т.д.),
- контекстные состояния станций представляются специальными переменными. Для простоты принято, что они целочисленного типа.

Для одной и той же предметной области можно разработать несколько эквивалентных онтологий, поскольку в принципе имеется несколько различных подходов к описанию даже типовых ситуаций. В качестве примера рассмотрим ситуацию классификации устройств, например, в тестирующей станции FESTO. Для простоты будем предполагать, что все устройства тестирующей станции (класс Device) разбиты на подклассы Pusher (Выталкиватели), Lift (Лифты) и Measurer (Измерители). Первый вариант онтологического описания данной классификации представлен на диаграмме на рисунке 2.1. В данном случае используется системное отношение «класс–подкласс» (subclassOf). Для полноты примера на диаграмме также представлены экземпляры классов (в виде овалов) с использованием отношения instanceOf.

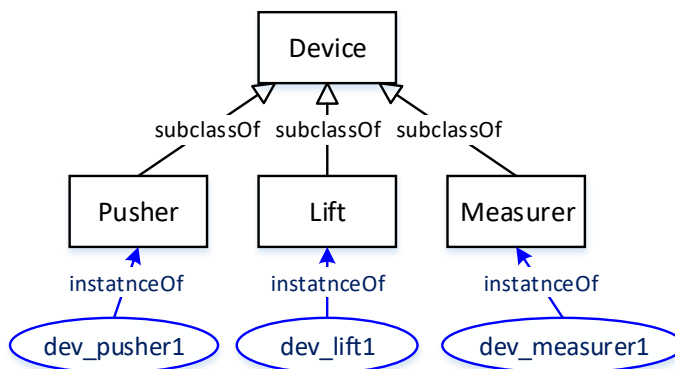


Рисунок 2.1 – Классификация устройств с использованием отношения «класс–подкласс»

Во втором варианте онтологического описания используется пользовательское отношение `has_device_type` «иметь тип устройства» и дополнительный класс типов устройств (класс `Device_Type`) вместе с экземплярами этого класса, представляющими конкретные типы устройств (см. рисунок 2.2).

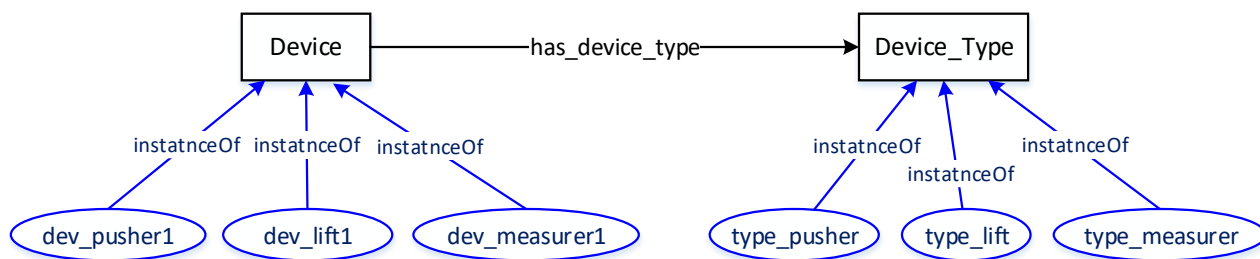


Рисунок 2.2 – Классификация устройств с использованием класса типов устройств и отношения «иметь тип устройства»

Устройства связываются со своими типами фактически с помощью кортежей (триплетов) вида <экземпляр устройства, `has_device_type`, экземпляр типа устройства>. Например, устройство `dev_pusher1` отнесено к типу выталкивателей `type_pusher` при помощи триплета <`dev_pusher1`, `has_device_type`, `type_pusher`>.

Следует отметить, что каждый из этих двух приведенных вариантов описания имеет свои достоинства и недостатки. Первый вариант более простой и естественный, однако соответствующая TBox-онтология не будет являться универсальной, поскольку на каждой станции имеется свой уникальный набор классов устройств. Второй вариант хоть и более громоздкий, тем не менее, он позволяет создать более универсальную TBox-онтологию. Специализация онтологии осуществляется в данном случае путем создания соответствующих экземпляров классов. Поэтому для создания универсальной онтологии станций в данном подразделе магистерской работы был выбран второй вариант. Следует, однако, заметить, что для моделирования динамики функционирования станций с использованием языка SPARQL (подразделы 2.3, 2.4, 2.5), был выбран первый вариант. Это связано с тем, что в этом случае для простоты реализации создания прототипа были разработаны собственные простейшие онтологии станций.

Классы обобщенной Tbox–онтологии в системе Protégé представлены на рисунке 2.3.

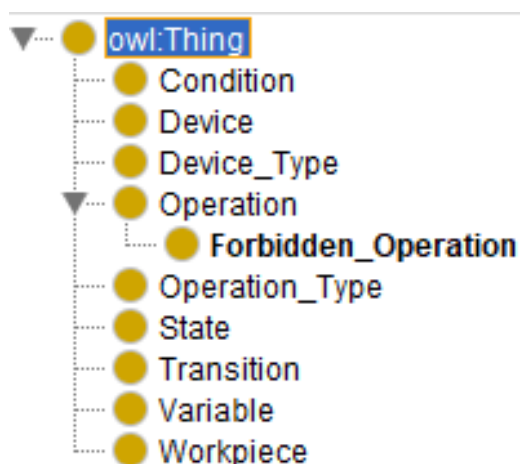


Рисунок 2.3 – Классы обобщенной онтологии в системе Protégé

Смысловая интерпретация классов приведена в таблице 2.1

Таблица 2.1 – Классы обобщенной онтологии

№	Имя класса	Смысловая интерпретация
1	Device	Устройство
2	Device_type	Тип устройства
3	Operation	Технологическая операция
4	Forbidden_Operation	Запрещенная технологическая операция
5	Operation_Type	Тип технологической операции
6	State	Состояние в конечно–автоматной модели техпроцесса
7	Transition	Переход в конечно–автоматной модели техпроцесса
8	Condition	Условие перехода в конечно–автоматной модели техпроцесса
9	Variable	Переменная
10	Workpiece	Обрабатываемая деталь

Класс `Forbidden_Operation` является подклассом класса `Operation`. Вычисления данного подкласса производится в процессе семантического анализа, который описан ниже. Переменные в онтологической модели

используются для хранения контекстной информации о состоянии системы. Например, это может быть флаг, отмечающий, было ли совершено какое-либо действие в прошлом.

Объектные свойства обобщенной онтологии в системе Protégé представлены на рисунке 2.4.

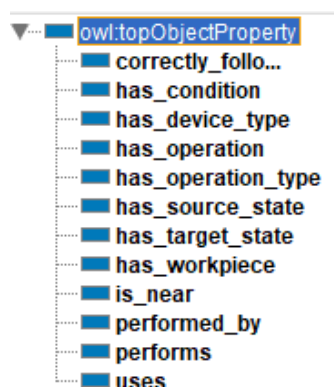


Рисунок 2.4 – Объектные свойства обобщенной онтологии в системе Protégé

Характеристика объектных свойств приведена в таблице 2.2.

Таблица 2.2 – Объектные свойства обобщенной онтологии

Имя объектного свойства	Домен	Диапазон	Смысловая интерпретация
has_device_type	Device	Device_type	Иметь тип устройства
performs	Device_Type	Operation_Type	Может выполнять операцию
performrd_by	Operation	Device	Выполняется устройством
has_operation	State	Operation	Иметь операцию
has_operation_type	Operation	Operation_Type	Иметь тип операции
has_workpiece	Device	Workpiece	Имеет деталь
is_near	Device	Device	Пространственное отношение «быть рядом»
has_source_state	Transition	State	Иметь состояние–источник

Продолжение таблицы 2.2

Имя объектного свойства	Домен	Диапазон	Смысловая интерпретация
has_target_state	Transition	State	Иметь состояние–приемник
has_condition	Transition	Condition	Иметь условие перехода
uses	Condition	Variable, Workpiece, Device	Использовать в вычислении условия
correctly follows	Operation	Operation	Корректно следовать

Следует отметить, что объектное свойство `is_near` является симметричным. Как видно из описания, операция в системе по сути дела определяется парой – типом операции и устройством, на котором она выполняется.

Свойства по данным (datatype properties) обобщенной онтологии в системе Protégé представлены на рисунке 2.5.

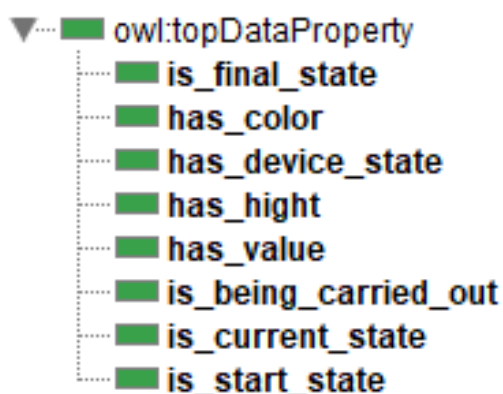


Рисунок 2.5 – Свойства по данным обобщенной онтологии в системе Protégé

Характеристика свойств по данным приведена в таблице 2.3.

Таблица 2.3 – Свойства по данным обобщенной онтологии

Имя свойства по данным	Домен	Тип данных	Смысловая интерпретация
has_state	Device	Integer	Иметь состояние устройства
is_start_state	State	Boolean	Состояние является начальным
is_final_state	State	Boolean	Состояние является конечным
is_current_state	State	Boolean	Состояние является текущим
has_colour	Workpiece	Boolean	Иметь допустимый цвет детали
has_hight	Workpiece	Boolean	Иметь допустимую высоту детали
is_being_carried_out	Operation	Boolean	Операция выполняется в данный текущий момент времени
has_value	Variable	Integer	Иметь значение переменной

Графическое представление обобщенной онтологии в виде диаграммы представлено на рисунке 2.6. На данном рисунке классы онтологии изображаются в виде черных прямоугольников, типы данных – в виде синих овалов, объектные свойства – как черные стрелки, а свойства по данным – как синие стрелки.

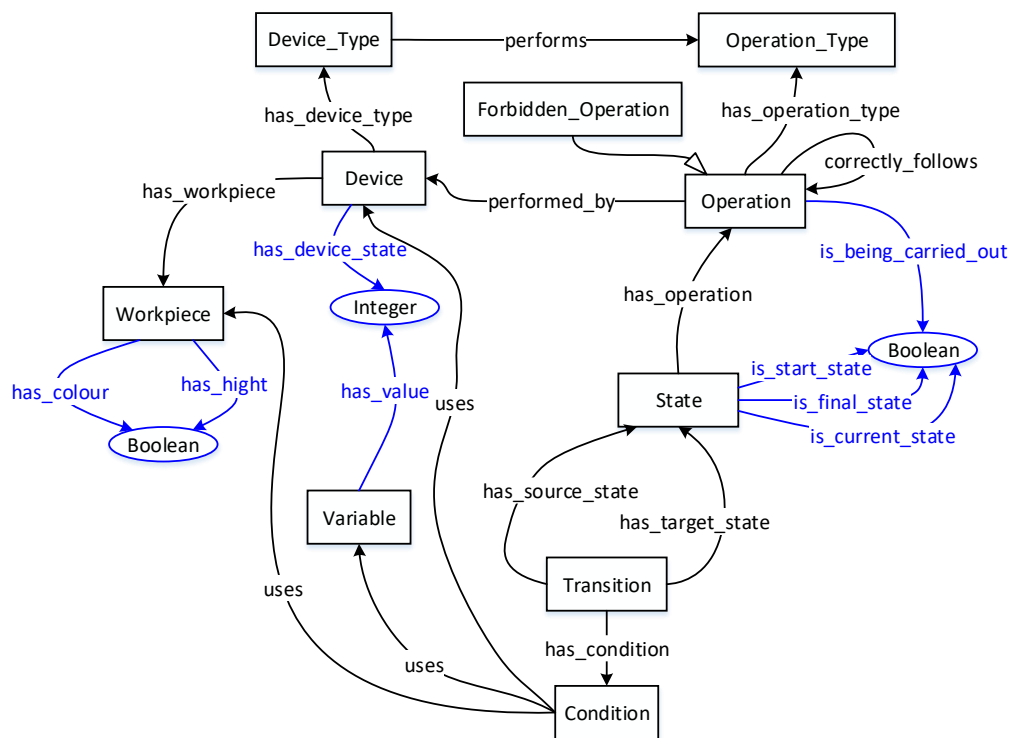


Рисунок 2.6 – Графическое представление обобщенной онтологии

2.2.2 Частная онтология тестирующей станции

Частные онтологии конкретных станций производственной системы FESTO получаются путем дополнения Tbox–онтологии соответствующей Abox–онтологией, иначе, онтологией на уровне экземпляров.

В качестве примера разработаем частную онтологию тестирующей станции системы FESTO. В первую очередь нам надо определиться с составом устройств, входящих в систему, и технологическим процессом обработки детали на этих устройствах.

Каждое устройство в онтологии будет представляться отдельным экземпляром (индивидом) класса Device. Экземпляры (индивиды) класса Device частной онтологии тестирующей станции в системе Protégé представлены на рисунке 2.7.

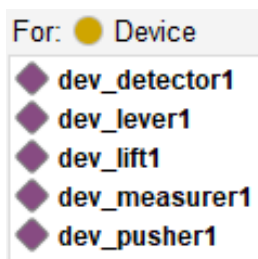


Рисунок 2.7 – Экземпляры класса Device в системе Protégé

Смысловая интерпретация экземпляров класса Device приведена ниже:

- dev_lift1 – лифт № 1,
- dev_measurer1 – измеритель высоты детали № 1,
- dev_pusher1 – выталкиватель № 1,
- dev_lever1 – транспортный рычаг № 1,
- dev_detector1 – блок сенсоров № 1.

Следует заметить, что в состав устройств тестирующей станции включен также транспортный рычаг, который, на самом деле, является общим ресурсом для двух станций – тестирующей и распределительной.

С помощью экземпляров класса Device_Type определяются типы устройств. Экземпляры (индивиды) класса Device_Type частной онтологии тестирующей станции в системе Protégé представлены на рисунке 2.8.

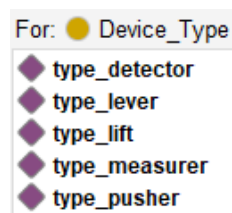


Рисунок 2.8 – Экземпляры класса Device_Type в системе Protégé

Смысловая интерпретация экземпляров класса Device_Type приведена ниже:

- type_pusher – тип устройства «выталкиватель»,
- type_lift_type – тип устройства «лифт»,
- type_measurer – тип устройства «измерительный стержень»,
- type_detector – тип устройства «блок детекторов»,
- type_lever – тип устройства «транспортный рычаг».

Названия операций и типов устройств составлены таким образом, чтобы связь устройства с его типом была очевидной.

С помощью экземпляров класса Operation_Type определяются типы операций. Экземпляры (индивиды) класса Operation_Type частной онтологии тестирующей станции в системе Protégé представлены на рисунке 2.9.

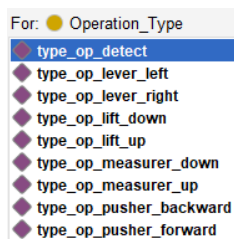


Рисунок 2.9 – Экземпляры класса Operation_Type в системе Protégé

Смысловая интерпретация экземпляров класса Operation_Type приведена ниже:

- type_op_lift_up – тип операции «поднять лифт вверх»,
- type_op_lift_down – тип операции «опустить лифт вниз»,
- type_op_pusher_forward – тип операции «выдвинуть выталкиватель»,
- type_op_pusher_backward – тип операции «втянуть выталкиватель»,
- type_op_measurer_up – тип операции «поднять измеритель вверх»,
- type_op_measurer_down – тип операции «опустить измеритель вниз»,
- type_op_detect – тип операции «съем данных с датчиков»,
- type_op_lever_left – тип операции «передвинуть транспортный рычаг на сторону распределительной станции»,
- type_op_lever_right – тип операции «передвинуть транспортный рычаг на сторону тестирующей станции».

С помощью экземпляров класса Operation определяются операции. Экземпляры (индивиды) класса Operation частной онтологии тестирующей станции в системе Protégé представлены на рисунке 2.10.

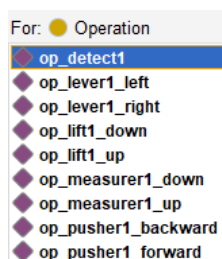


Рисунок 2.10 – Экземпляры класса Operation в системе Protégé

Смысловая интерпретация экземпляров класса Operation приведена ниже:

- op_lift1_up – поднять лифт № 1 вверх,
- op_lift1_down – опустить лифт № 1 вниз,
- op_pusher1_forward – выдвинуть выталкиватель № 1,
- op_pusher1_backward – втянуть выталкиватель № 1,
- op_measure1r_up – поднять измеритель № 1 вверх,
- op_measurer1_down – опустить измеритель № 1 вниз,
- op_detect1 – съем данных с блока датчиков № 1,
- op_lever1_left – передвинуть транспортный рычаг № 1 на сторону распределительной станции,
- op_lever1_right – передвинуть транспортный рычаг № 1 на сторону тестирующей станции.

Названия операций и типов операций составлены таким образом, чтобы связь операции с ее типом была очевидной.

С помощью экземпляров класса Condition определяются условия переходов в конечно–автоматной модели техпроцесса. Экземпляры (индивиды) класса Condition частной онтологии тестирующей станции в системе Protégé представлены на рисунке 2.11.

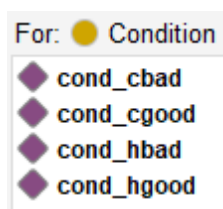


Рисунок 2.11 – Экземпляры класса Condition в системе Protégé

Смысловая интерпретация экземпляров класса Condition приведена ниже:

- cond_cgood – нормальный цвет детали,
- cond_cbad – негодный цвет детали,
- cond_hgood – нормальная высота детали,
- cond_hbad – негодная высота детали.

Далее рассмотрим, как представляется технологический процесс обработки деталей на тестирующей станции. Алгоритм техпроцесса на мнемокоде приведен ниже:

```
Детектирование цвета детали
если цвет детали годный, то
    делать
        Поднять лифт
        Опустить проверочный стержень          --произвести измерение высоты
        детали
        Поднять проверочный стержень
        если высота детали годная, то
            делать
                Выдвинуть выталкиватель      --вытолкнуть деталь в скользящий желоб
                Задвинуть выталкиватель
                Опустить лифт
                конец
            иначе      --высота детали негодная
                делать
                    Опустить лифт
                    Выдвинуть выталкиватель      --вытолкнуть деталь в брак
                    Задвинуть выталкиватель
                    конец
                конец
            иначе      --цвет детали негодный
                делать      --вытолкнуть деталь в брак
                    Выдвинуть выталкиватель
                    Задвинуть выталкиватель
                    конец
```

Данный техпроцесс, представленный в виде конечно–автоматной модели, приведен на рисунке 2.12. В приведенной диаграмме состояния показаны как овалы, а переходы – как стрелки, соединяющие эти овалы. Внутри овалов вписаны названия технологических операций. Надписи около состояний (например, s5) и переходов (например, t9) обозначают их идентификаторы, которые отражены в онтологии как экземпляры классов State и Transition, соответственно. С дугами переходов также могут быть ассоциированы условия переходов (например, cond_cgood), которые представлены экземплярами класса Condition. Начальными состояниями в диаграмме считаются те состояния, которые связаны с вершиной start. В данном случае это состояние s1. Конечными состояниями в диаграмме считаются те состояния, которые связаны с вершиной finish. В данном случае это состояния s6, s10 и s12.

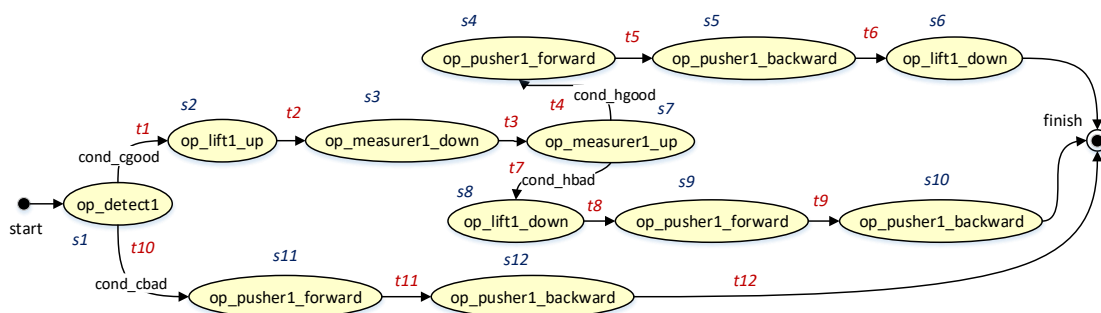


Рисунок 2.12 – Конечно–автоматная модель технологического процесса обработки детали на тестирующей станции FESTO

На рисунке 2.13 представлен пример описания состояния s6 в системе Protégé.



Рисунок 2.13 – Пример описания состояния s6 в системе Protégé

На рисунке 2.14 представлен пример описания перехода t1 в системе Protégé.

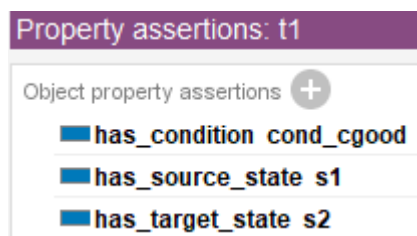


Рисунок 2.14 – Пример описания перехода t1 в системе Protégé

На рисунке 2.15 представлен пример описания операции op_lift1_up в системе Protégé.

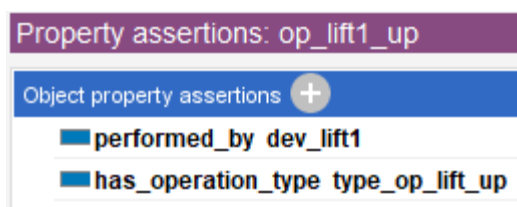


Рисунок 2.15 – Пример описания операции op_lift1_up в системе Protégé

Обрабатываемая деталь является динамическим элементом в системе и может создаваться с использованием средств манипулирования онтологиями, например, с помощью языка SPARQL Update.

2.2.3 Семантический анализ

Все большую роль в выявлении ошибок на стадиях проектирования, предшествующих внедрению, приобретает семантический анализ проектов, предполагающий проверку определенных семантических ограничений, отражающих глубинные отношения между артефактами проектирования, не поддающиеся синтаксическому анализу [55]. Семантический анализ не требует интенсивных и ресурсоемких вычислений, какие необходимы при верификации, но при этом способен обнаруживать серьезные ошибки, которые не являются очевидными.

Семантический анализ основан на определении так называемых «семантически правильных классов» и/или «семантически правильных отношений» и их вычислении в ходе логического вывода [55]. Техника проведения семантического анализа основана на обычной процедуре классификации, осуществляемой штатной системой рассуждений (reasoner).

Следует заметить, что семантический анализ может быть полезен не только на стадии проектирования, но и на стадии эксплуатации системы. При этом работа системы постоянно мониторится, а полученные в результате мониторинга данные анализируются путем проверки специальных семантических ограничений. Если при этом нарушено какое-либо семантическое ограничение, то это свидетельствует (или может свидетельствовать) о нарушении режима правильного функционирования системы. После этого необходимо произвести корректировку работы системы, чтобы избежать возможных инцидентов.

Введем следующую классификацию используемых нами семантических ограничений:

- по степени общности будем различать общие (класс С) и частные (класс Р) семантические ограничения. Первые относятся ко всем системам,

которые описываются данной онтологией. Вторые относятся только к какой-то конкретной системе,

- по степени серьезности будем различать серьезные ограничения (класс S) и предупредительные ограничения (класс W), включающие также синтаксические ограничения,

- по применению на стадиях жизненного цикла системы будем различать ограничения, используемые на стадии проектирования (класс D) и используемые на стадии эксплуатации (класс M),

- по способу представления будем различать ограничения, представленные при помощи аксиом классов и свойств (класс A), и ограничения, представленные в виде SWRL-правил (класс R).

Для представления результирующего типа ограничения будем использовать четырехбуквенную нотацию, причем номер буквы в обозначении определяет номер классификатора, а значение буквы – класс по данному классификатору. Например, запись CSDA означает общее серьезное ограничение, используемое на стадии проектирования, представленное в виде аксиомы.

Общие ограничения. Ниже приведено несколько примеров общих ограничений.

Ограничение R1 (тип CSDR). В диаграмме техпроцесса одна операция корректно следует за другой только в том случае, если соответствующие устройства, используемые в операциях, находятся рядом друг с другом.

В случае нарушения ограничения можно сделать вывод, что порядок следования операций неправильный, и разработчик должен внимательно проверить модель техпроцесса и, соответственно, ее исправить.

Будем считать, что в рамках представленной обобщенной онтологии Ограничение R1 выполняется, если соответствующие операции будут связаны объектным свойством `correctly_follows`. Поэтому данное бинарное отношение должно быть вычислено в процессе логического вывода (ризонинга).

Для более полного визуального восприятия ограничений, представленных в виде правил, а также их визуальной проверки могут быть составлены соответствующие графы запросов [55]. Граф запроса наглядно описывает ситуацию в виде некоторой семантической сети, узлами которой являются переменные. Эти переменные, также как и в правиле SWRL, будем предварять знаком вопроса (?). Дугами в графе запроса являются отношения между соответствующими классами, представляющими интерес.

На рисунке 2.16 показан граф запроса для определения корректного порядка следования операций в техпроцессе. Пунктирная красная дуга с надписью `correctly_follows` представляет дугу корректного следования.

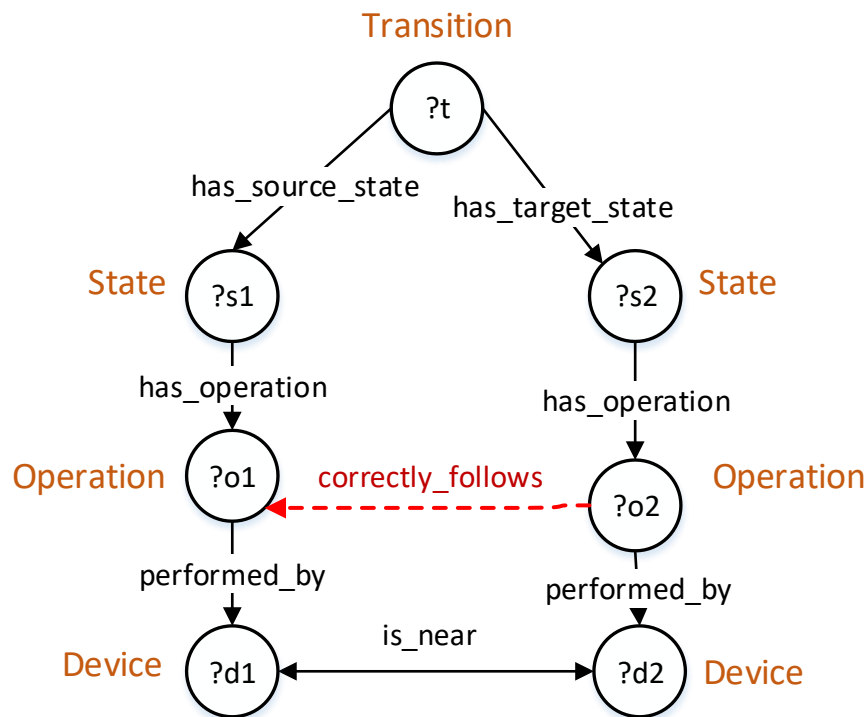


Рисунок 2.16 – Граф запроса для определения корректного порядка следования операций в техпроцессе

Граф запросов можно легко трансформировать в режиме «один к одному» в соответствующее SWRL–правило :

```

Device(?d1) ^ Operation(?o1) ^ performed_by(?o1,?d1) ^ Device(?d2)
^ Operation(?o2) ^ performed_by (?o2,?d2) ^ is_near(?d1,?d2) ^
Transition(?t) ^ State(?s1) ^ has_source_state(?t,?s1) ^
State(?s2) ^ has_target_state(?t,?s2) ^ has_operation(?s1,?o1) ^
has_operation(?s1,?o1)→ correctly_follows(?o2,?o1)

```

Ограничение R2 (тип CWDA). Каждый переход должен иметь одну операцию–источник и одну операцию–приемник.

Особенностью данного ограничения является его направленность на выявление синтаксических ошибок описания. Данные ошибки совершаются разработчиком, как правило, по невнимательности. Для реализации данного ограничения используется механизм аксиом и понятие “правильного” класса [55]. Правильный класс строится на основе соответствующего базового класса, но при этом в его определение включаются дополнительные свойства и ограничения, полученные из анализа предметной области. Техника проведения семантического анализа основана на обычной процедуре классификации, осуществляемой штатной системой рассуждений (reasoner). Собственно семантический анализ сводится к сравнению множеств экземпляров, отнесенных как к базовому классу, так и к его «правильному» аналогу. Равенство данных множеств свидетельствует о семантической корректности описания. Неравенство множеств свидетельствует о наличии ошибки в описании.

Введем класс для описания «правильных» переходов и назовем его Right_Transition. Ниже приведена аксиома для этого класса, выраженная с помощью дескриптивной логики:

```
Right_Transition  $\equiv$   
Transition  $\sqcap$   $\exists$  has_source_state.State  
 $\sqcap$   $\exists$  has_target_state.State
```

Та же самая аксиома на языке редактора выражений системы Protégé, сформированная в пункте Equivalent to для класса Right_Transition, будет выглядеть следующим образом:

```
Transition and has_source_state some State  
and has_target_state some State
```

Частные ограничения. Ниже приведено несколько примеров частных ограничений, касающихся только тестирующей станции.

Ограничение R3 (тип PSMR). Если рычаг находится на стороне тестирующей станции, то нельзя поднимать или опускать лифт.

SWRL–правило, реализующее это ограничение приведено ниже:

```
has_device_state(dev_lever1,1) -  
>Forbidden_Operation(op_lift1_up),  
Forbidden_Operation(op_lift1_down).
```

В соответствии с этим правилом, если состояние транспортного рычага равно 1 (что означает, что рычаг находится на тестирующей станции), то запрещены все операции с лифтом. Это связано с тем, что нахождение транспортного рычага на стороне тестирующей станции делает физически невозможным поднятие или опускание лифта.

Вычисление данного правила осуществляется системой мониторинга всякий раз, когда изменяется состояние системы. Следует отметить, что в этом случае система мониторинга должна быть основана на технологиях семантического Веб, она должна работать с RDF–данными и иметь возможность проведения ризонинга (то есть, логического вывода).

Ограничение R4 (тип PSMR). Если лифт поднимается, то нельзя перемещать транспортный рычаг на сторону тестирующей станции.

SWRL–правило, реализующее это ограничение, приведено ниже:

```
is_being_carried_out(op_lift1_up,true)->  
Forbidden_Operation(op_lever1_right).
```

На первый взгляд может показаться тривиальность данного правила и невозможность его использования на практике, поскольку все операции выполняются последовательно. Однако в случае возможности выполнения параллельных операций (например, когда движением рычага управляет другой (второй) контроллер) данное правило становится актуальным.

Ниже приведено несколько примеров правил типа PWMR без подробных комментариев:

– ограничение R5. Если лифт не поднят или на его платформе нет детали, то нет смысла совершать операции с проверочным стержнем.

– ограничение R6. Если деталь не находится на платформе лифта, то нет смысла совершать операции с выталкивателем.

– ограничение R7. Если на платформе лифта нет детали, то нет смысла поднимать лифт.

Ситуации, которым соответствуют данные правила, не являются какими-то серьезными и опасными, но они являются просто бессмысленными и не приводят к какой-либо цели. Тем не менее, функционирование системы в соответствии с данными правилами, обнаруженное системой мониторинга, может свидетельствовать о проблемах в системе в целом. Если это не приводит к серьезным нарушениям в работе системы в текущий момент, то может привести в последующем.

Пример синтаксического ограничения типа PWDA приведен ниже.

Ограничение R8. Станция имеет по одному устройству каждого типа.

Данное ограничение может быть выражено следующей аксиомой для класса `Device_Type` на языке редактора выражений системы Protégé:

```
has_device_type exactly 1 Device.
```

Знак « $\leftarrow$ » в позиции верхнего индекса используется для представления обратного отношения. Данная аксиома определяет класс типов устройств, которым назначено только по одному устройству.

2.3 Моделирование распределительной станции

2.3.1 Онтологическая модель

На рисунке 2.17 слева показана распределительная станция FESTO, а справа – классы онтологии, полученные в процессе восприятия и абстрагирования.

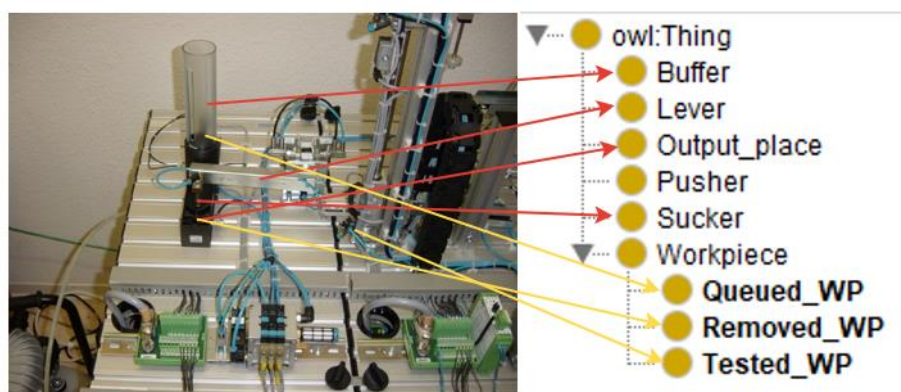


Рисунок 2.17 – Отображение физических объектов распределительной станции (слева) на классы онтологии (справа)

Разработанная онтология включает классы, представляющие физические устройства распределительной станции: буфер деталей (Buffer), транспортный рычаг (Lever), выталкиватель (Pusher), вакуумная присоска (Sucker), выходное место выталкивателя (Output_place), деталь (Workpiece); в зависимости от состояния детали она будет относиться к одному из следующих подклассов: деталь, находящаяся в буфере (Queued_WP), деталь, извлеченная из буфера и находящаяся в распределительной станции (Removed_WP), деталь, перенесенная на тестирующую станцию для проверки (Tested_WP).

Объектные свойства онтологии: указатель на первый элемент очереди (has_IR_pointer), указатель на свободный номер в очереди для записи (has_IW_pointer), втянут – выдвинут (has_pusher_state), вакуум выключен – вакуум включен (has_vacuum_state), на стороне РС – на стороне ТС (has_lever_state), негодная – нормальная (has_color), негодная – нормальная (has_hight), свободно – занято (has_output_state), идентификатор детали в очереди (has_queue_id).

Особенностью этой онтологии является использование свойств по данным для представления состояний устройств. Причем значение всех свойств характеризуется бинарным тип данных.

Начальное состояние распределительной станции следующее: в буфере отсутствуют детали (значения указателей $ir=iw=1$), выталкиватель втянут (false), выходное место свободно от детали (false), транспортный рычаг на стороне распределительной станции (false), вакуум присоски выключен (false).

Информационная модель буфера. Детали в буфере образуют очередь, для которой характерна дисциплина обслуживания FIFO («первый пришел – первый обслужен»). Объекты типа «очередь» широко используются в сфере вычислительной техники. Воспользуемся методом аналогий для моделирования очереди в распределительной станции, приняв за основу модель очереди из данной сферы. При этом надо иметь ввиду, что элементами

очереди в этом случае являются не информационные, а физические объекты (детали).

Для реализации операций записи и чтения из буфера в вычислительной технике часто используется кольцевой буфер (см. рисунок 2.18).

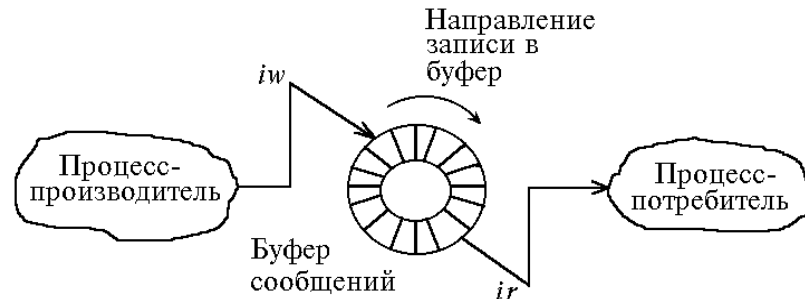


Рисунок 2.18 – Задача «производитель–потребитель»

При этом вводятся два указателя: iw – для записи, ir – для чтения. Указатель iw определяет пустую ячейку в буфере, куда процесс-производитель должен поместить сообщение. Указатель ir определяет ячейку в буфере, откуда процесс-потребитель должен считать сообщение. Условие $iw = ir$ указывает, что буфер пуст. Но это истинно, если данное условие достигается при операции чтения буфера. Если данное условие достигается при операции записи, то это свидетельствует о переполнении буфера.

Для реализации кольцевого буфера можно использовать линейный буфер емкостью N . При этом для сдвига указателей следует использовать сложение с единицей по модулю N (1, 2):

$$iw = (iw + 1) \bmod N \quad (1)$$

$$ir = (ir + 1) \bmod N \quad (2)$$

Условие переполнения выражается таким образом (3):

$$ir = (iw + 1) \bmod N \quad (3)$$

В приведенных ниже правилах (точнее, в правилах B1 и B2) предполагается, что буфер имеет бесконечную емкость, что является

упрощением. Однако данное упрощение в целом не влияет на полезность данной информационной модели. Тем не менее, рассмотрим пути реализации SPARQL–модели, в которой используется буфер ограниченной емкости. Затруднением при реализации является тот факт, что в языке SPARQL нет операции деления по модулю. Для решения проблемы можно использовать функцию «*IF()*» языка SPARQL. Данная функция имеет три аргумента. Если первый аргумент оценивается как «истина», то функция возвращает значение второго аргумента, иначе она возвращает значение третьего аргумента.

Таким образом, увеличение значения указателя *i_w* на единицу, например, по модулю 10 может быть реализовано следующим образом (4):

$$BIND((IF(?i_w+1>10, 1, ?i_w+1)) AS ?i_w2), \quad (4)$$

где *?i_w2* – это переменная с новым значением указателя *i_w*.

Условие переполнения может быть реализовано с помощью двух предложений *BIND* (5, 6):

$$BIND((IF(?i_w+1>10, 1, ?i_w+1)) AS ?i_w2) \quad (5)$$

$$BIND((IF(?i_w2=?i_r), true, false) AS ?overflow) \quad (6)$$

или даже одного предложения *BIND* (7):

$$BIND((IF((IF(?i_w+1>10, 1, ?i_w+1))=?i_r), true, false) AS ?overflow), \quad (7)$$

где переменная «*?overflow*» определяет выполнение (true) или невыполнение (false) условия переполнения буфера.

2.3.2 Продукционные правила и SPARQL–запросы

Правило В1. Поместить в буфер деталь с параметрами «цвет X», «высота Y».

Примечания:

– данное правило создает «анонимную» деталь и связывает с ней необходимые атрибуты (цвет, высоту, порядковый номер в очереди), а также увеличивает на единицу указатель IW очереди,

– в данном правиле не производится проверка на переполнение буфера, считается, что буфер имеет бесконечную емкость.

Запрос на языке SPARQL представлен ниже:

```
DELETE {
?b :has_IW_pointer ?iw}
INSERT {
[] rdf:type :Queued_WP;
:has_queue_id ?iw;
:has_color false;
:has_weight false.
?b :has_IW_pointer ?iw2}
WHERE {
?b rdf:type :Buffer.
?b :has_IW_pointer ?iw.
BIND(?iw2 AS ?iw+1)
}
```

Правило В2. Вытолкнуть деталь из буфера в выходное место.

Ниже приведено продукционное правило В2.

ЕСЛИ буфер не пуст И выталкиватель втянут И выходное место свободно, ТО выдвинуть выталкиватель, переместить деталь из очереди в выходное место, инкрементировать указатель IR очереди.

Примечание: положение транспортного рычага при выталкивании детали не учитывается.

Запрос на языке SPARQL представлен ниже:

```
DELETE {
?p :has_pusher_state false.
?o :has_output_state false.
?b :has_IR_pointer ?ir.
?wp rdf:type :Queued_WP}
INSERT {
?p :has_pusher_state true.      # выдвинуть выталкиватель
?o :has_output_state true.      # занять выходное место
?b :has_IR_pointer ?ir2.        # инкрементировать IR-
указатель очереди
?wp rdf:type :Removed_WP}      # деталь теперь в классе
«вытолкнутых» деталей
WHERE {
?b rdf:type :Buffer.
?b :has_IR_pointer ?ir.
```

```

?b :has_IW_pointer ?iw.
FILTER (?ir != ?iw).           # буфер не пуст
?p rdf:type :Pusher.
?p :has_pusher_state false.    # выталкиватель втянут
?o rdf:type :Output_place.
?o :has_output_state false.    # выходное место свободно
?wp rdf:type :Queued_WP.
?wp :has_queue_id ?ir.        # первая деталь в очереди
BIND(?ir2 AS ?ir+1)}

```

Правило В3. Втянуть выталкиватель.

Ниже приведено продукционное правило В3.

ЕСЛИ выталкиватель выдвинут, ТО втянуть выталкиватель

Запрос на языке SPARQL представлен ниже:

```

DELETE {
?p :has_pusher_state true}
INSERT {
?p :has_pusher_state false}    # втянуть выталкиватель
WHERE {
?p rdf:type :Pusher.
?p :has_pusher_state true}    # выталкиватель выдвинут

```

Правило В4. Перенести транспортный рычаг на сторону распределительной станции.

Продукционное правило В4:

ЕСЛИ выходное место содержит деталь И транспортный рычаг на стороне тестирующей станции (ТС) И вакуум выключен, ТО перенести транспортный рычаг на сторону распределительной станции (РС).

Запрос на языке SPARQL представлен ниже:

```

DELETE {
?t :has_lever_state true}
INSERT {
?t :has_lever_state false}    # перенести транспортный рычаг на
сторону РС
WHERE {
?o rdf:type :Output_place.
?o :has_output_state true.    # выходное место содержит деталь
?t rdf:type :Lever.
?t :has_lever_state true      # транспортный рычаг на стороне ТС
?s rdf:type :Sucker.
?s :has_vacuum_state false}   # вакуум присоски выключен

```

Правило В5. Включить вакуум присоски.

Ниже приведено продукционное правило В5.

ЕСЛИ выходное место содержит деталь И транспортный рычаг на стороне распределительной станции И вакуум присоски выключен, ТО включить вакуум присоски.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?s :has_vacuum_state false }  
INSERT {  
  ?s :has_vacuum_state true}      # включить вакуум присоски  
WHERE {  
  ?o rdf:type :Output_place.  
  ?o :has_output_state true.      # выходное место содержит деталь  
  ?t rdf:type :Lever.  
  ?t :has_lever_state false.      # транспортный рычаг на стороне РС  
  ?s rdf:type :Sucker.  
  ?s :has_vacuum_state false}    # вакуум присоски выключен
```

Правило В6. Перенести транспортный рычаг (с деталью) на тестирующую станцию.

Ниже приведено продукционное правило В6.

ЕСЛИ транспортный рычаг на стороне распределительной станции И выходное место содержит деталь И вакуум присоски включен, ТО перенести транспортный рычаг на сторону тестирующей станции И освободить выходное место.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?t :has_lever_state false.  
  ?o :has_output_state true.}  
INSERT {  
  ?t :has_lever_state true.      # перенести транспортный рычаг на  
  сторону ТС  
  ?o :has_output_state false}    # освободить выходное место  
WHERE {  
  ?t rdf:type :Lever.  
  ?t :has_lever_state false.      # транспортный рычаг на стороне РС  
  ?s rdf:type :Sucker.  
  ?s :has_vacuum_state true.      # вакуум присоски включен  
  ?o rdf:type :Output_place.  
  ?o :has_output_state true.      # выходное место содержит деталь  
}
```

Правило В7. Выключить вакуум присоски.

Ниже приведено продукционное правило В7.

ЕСЛИ транспортный рычаг на стороне тестирующей станции И вакуум присоски включен, ТО Выключить вакуум присоски, включить деталь в состав деталей для проверки на тестирующей станции.

Запрос на языке SPARQL представлен ниже:

```
DELETE {
?s :has_vacuum_state true.
?wp rdf:type :Removed_WP}
INSERT {
?s :has_vacuum_state false.    # выключить вакуум присоски
?wp rdf:type :Tested_WP} # деталь теперь в классе «тестируемых»
деталей
WHERE {
?t rdf:type :Lever.
?t :has_lever_state true.      # транспортный рычаг на стороне ТС
?s rdf:type :Sucker.
?s :has_vacuum_state true.     # вакуум присоски включен
?wp rdf:type :Removed_WP}
```

Запрос на определение состояний устройств системы, а также на наличие обработанных деталей представлен ниже:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://www.semanticweb.org/dubin/ontologies/2024/2/untitled-ontology-4#>
SELECT ?ir ?iw ?sp ?so ?sw ?st ?wp2 ?wp3
WHERE {
?b has_IR_pointer ?ir.
?b has_IW_pointer ?iw.
?p :has_pusher_state ?sp.
?o :has_output_state ?so.
?w :has_vacuum_state ?sw.
?t :has_lever_state ?st
OPTIONAL
?wp2 :Removed_WP.
?wp3 :Tested_WP}
```

Запрос на определение деталей, находящихся в очереди буфера представлен ниже:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://www.semanticweb.org/dubin/ontologies/2024/2/untitled-ontology-4#>
```

```

SELECT ?id ?c ?w
WHERE {
  ?wp rdf:type :Queued_WP.
  ?wp :has_queue_id ?id.
  ?wp :has_color ?c.
  :has_weight ?w}

```

Сценарии обработки детали. Технологический процесс обработки одной детали (при условии, что система находится в начальном состоянии) моделируется выполнением следующей последовательности SPARQL-запросов: B1, B2, B3, B5, B6, B7. В данной последовательности нет правила B4, поскольку нет необходимости перегонять транспортный рычаг с тестирующей станции на распределительную. Это самый простой сценарий обработки. При наличии нескольких деталей в очереди сценарий обработки будет сложнее.

2.3.3 Верификация системы правил

Для верификации системы правил была разработана модель на основе сетей Петри (рисунок 2.19).

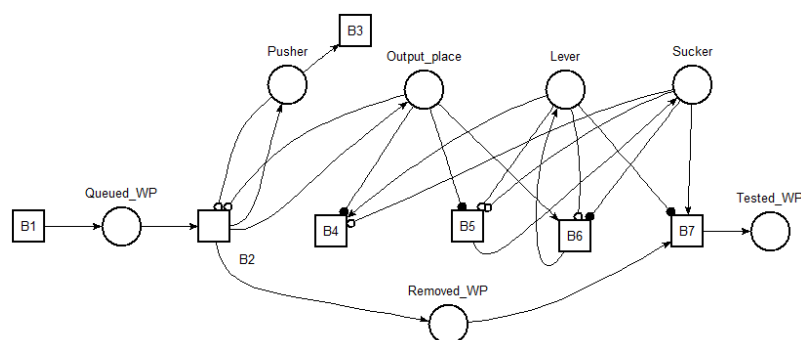


Рисунок 2.19 – Сетевая модель системы продукционных правил, описывающих функционирование тестирующей станции

Каждое правило моделируется отдельным переходом сети Петри. Условия правил представлены позициями сетевой модели, а действия – собственно переходами. Как правило, условия ассоциируются с состояниями вовлеченных в процесс устройств, а также статусом обрабатываемой детали. Переход B1 является генератором деталей, поступающих в систему. Данный переход не имеет входных позиций.

В сетевой используются обычные, ингибиторные дуги и дуги «чтения». В графическом начертании эти дуги отличаются наконечником стрелки.

Причем обычные дуги имеют «стандартный» наконечник в виде «галочки», ингибиторные дуги заканчиваются небольшим не закрашенным кружком, в то время как дуги «чтения» – закрашенным кружком. Чтобы ингибиторная дуга была разрешенной, позиция – источник дуги должна быть не маркирована (то есть, иметь нулевую маркировку).

Особенности (упрощения) сетевой модели представлены ниже:

- деталь представляется как сущность без параметров, в результате чего она моделируется обычной меткой сети Петри,
- очередь деталей, как таковая, в модели не представляется. Поскольку все детали без параметров одинаковы, то вместо очереди используется понятие «куча» (позиция `Queued_WP`).

Начальная маркировка модели соответствует начальному состоянию системы (которое было приведено выше).

Сетевая модель была сформирована и исследована в системе TINA с использованием метода имитационного моделирования (инструмент `Stepper Simulator`). При этом было определено, что:

- нет конфликтных правил,
- процесс обработки детали завершается, и она в итоге перемещается на тестирующую станцию,
- по окончании обработки детали все устройства переходят в свои начальные состояния,
- модель не является полностью детерминированной (см. рисунок 2.20), что, однако, не влияет на правильность функционирования модели.

бракованная деталь (Rejected_WP), нормальная деталь (Accepted_WP). Кроме того, введен класс Flag – флаг измерения высоты детали, используемый в системе управления.

Объектные свойства онтологии: has_pusher_state, has_lift_state и has_mechanic_state связывают физические устройства тестирующей станции с состояниями соответствующих устройств. Свойства по данным has_color и has_height используются для задания цвета и высоты детали, причем для простоты используются логические значения. Значение true означает, что параметр детали находится в норме, а значение false – нет.

При разработке продукционных правил на основе онтологии были выделены следующие локальные состояния, возможные в системе: c1 – деталь в системе (да/нет), c2 – цвет детали (годный /негодный), c3 – высота детали (годная / негодная), c4 – состояние выталкивателя (втянут / выдвинут), c5 – состояние лифта (в нижнем / верхнем положении), c6 – состояние измерительного стержня (поднят / опущен), c7 – флаг измерения высоты (измерения не проводились / измерения проводились), c8 – деталь годная (да /не определено), c9 – деталь бракованная (да /не определено). В скобках указаны возможные значения состояний. Все состояния, по сути, бинарные, то есть могут быть выражены значениями «Истина» и «Ложь». С состояниями связываются условия. Истинное значение условия, связанного с состоянием, соответствует выбору первого значения из списка допустимых значений. Например, условие $c4=true$ (или просто c4) соответствует утверждению «выталкиватель втянут», условие $c5=false$ (или просто $\neg c5$) соответствует утверждению «лифт в верхнем положении». Действия продукционных правил могут быть выражены через новые значения состояний, которые устанавливаются в результате этих действий. Например, действие «Поднять лифт» может быть выражено как $\neg c5$ (не c5).

В работе тестирующей станции могут быть выделены следующие действия: a1– удалить деталь из системы, a2 – поместить деталь в брак, a3 – поместить деталь в годные изделия, a4 – выдвинуть выталкиватель, a5 –

втянуть выталкиватель, а6 – поднять лифт, а7 – опустить лифт, а8 – опустить измерительный стержень, а9 – поднять измерительный стержень, а10 – завершить измерения, а11 – сбросить флаг измерения высоты (в начальное значение). Следует отметить, что действия могут быть выражены через новые значения состояний, которые устанавливаются в результате этого действия. Например, действие а6 может быть выражено как $\neg c5$ (не с5).

Ниже представлены правила функционирования тестирующей станции FESTO в следующем формате: сначала представлено продукционное правило в текстовом виде, затем краткая форма записи правила в терминах условий, и в конце – реализация продукционного правила в виде запроса на языке SPARQL.

2.4.2 Продукционные правила и SPARQL-запросы

Правило A1: отбраковать деталь с негодным цветом.

Ниже приведено продукционное правило A1.

ЕСЛИ деталь в системе И цвет детали негодный И выталкиватель втянут, ТО выдвинуть выталкиватель, удалить деталь из системы, поместить деталь в брак.

Краткая запись правила: $c1 \& \neg c2 \& c4 \rightarrow \neg c4 \& \neg c1 \& c8$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?p :has_pusher_state :retracted.  
  ?wp rdf:type :Tested_WP.}  
INSERT {  
  ?p :has_pusher_state :extended.  
  ?wp rdf:type :Reject_WP}  
WHERE {  
  ?wp rdf:type :Tested_WP.  
  ?wp :has_color false.  
  ?p rdf:type :Pusher.  
  ?p :has_pusher_state :retracted}
```

Правило A2: поднять деталь для измерения.

Ниже приведено продукционное правило A2.

ЕСЛИ деталь в системе И цвет детали годный И выталкиватель втянут И измерение высоты не произведено И лифт в нижнем положении, ТО поднять лифт.

Краткая запись: $c1 \& c2 \& c4 \& c5 \& c7 \rightarrow \neg c5$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
?l :has_lift_state :down}  
INSERT {  
?l :has_lift_state :up}  
WHERE {  
?wp rdf:type Tested_WP.  
?wp :has_color true.  
?p rdf:type :Pusher.  
?p :has_pusher_state :retracted.  
?f rdf:type :Flag.  
?f :has_value true}
```

Правило A3: измерить высоту детали.

Ниже приведено продукционное правило A3.

ЕСЛИ лифт в верхнем положении И измерительный стержень поднят И измерения высоты не производились, ТО опустить измерительный стержень.

Краткая запись правила: $\neg c5 \& c6 \& c7 \rightarrow \neg c6$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
?m :has_measurer_state :up}  
INSERT {  
?m :has_measurer_state :down}  
WHERE {  
?l rdf:type :Lift.  
?l :has_lift_state :up.  
?m rdf:type :Measurer.  
?m :has_measurer_state :up.  
?f rdf:type :Flag.  
?f :has_value true}
```

Правило A4: негодная высота детали.

Ниже приведено продукционное правило A4.

ЕСЛИ измерительный стержень опущен И высота детали негодная И измерения высоты не произведены, ТО завершить измерения, поднять измерительный стержень.

Краткая запись: $\neg c3 \& \neg c6 \& c7 \rightarrow c6 \& \neg c7$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?m :has_measurer_state :down.  
  ?f :has_value true}  
INSERT {  
  ?m :has_measurer_state :up.  
  ?f :has_value false}  
WHERE {  
  ?m rdf:type :Measurer.  
  ?m :has_measurer_state :down.  
  ?wp :has_hight false.  
  ?f rdf:type :Flag.  
  ?f :has_value true}
```

Правило A5: нормальная высота детали.

Ниже приведено продукционное правило A5.

ЕСЛИ измерительный стержень опущен И высота детали годная И измерения высоты не произведены И выталкиватель втянут И деталь в системе, ТО завершить измерения, поднять измерительный стержень, выдвинуть выталкиватель.

Краткая запись правила: $c1 \& c3 \& \neg c6 \& c7 \& c4 \rightarrow \neg c4 \& c6 \& \neg c7$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?f :has_value true.  
  ?m :has_measurer_state :down.  
  ?p :has_pusher_state :retracted}  
INSERT {  
  ?f :has_value false.  
  ?m :has_measurer_state :up.  
  ?p :has_pusher_state :extended}  
WHERE {  
  ?m rdf:type :Measurer.  
  ?m :has_measurer_state :down.  
  ?wp :has_hight true.  
  ?wp rdf:type :Tested_WP.  
  ?f rdf:type :Flag.  
  ?f :has_value true.  
  ?p rdf:type :Pusher.  
  ?p :has_pusher_state :retracted}
```

Правило A6: опустить лифт после измерения высоты.

Ниже приведено продукционное правило A6.

ЕСЛИ измерительный стержень поднят И измерения высоты произведены И выталкиватель втянут И лифт поднят И измерения высоты произведены, ТО опустить лифт.

Краткая запись: $c4 \& \neg c5 \& c6 \& \neg c7 \rightarrow c5$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?l :has_lift_state :up}  
INSERT {  
  ?l :has_lift_state :down}  
WHERE {  
  ?m rdf:type :Measurer.  
  ?m :has_measurer_state :up.  
  ?p rdf:type :Pusher.  
  ?p :has_pusher_state :retracted.  
  ?f rdf:type :Flag.  
  ?f :has_value false.  
  ?l rdf:type :Lift.  
  ?l :has_lift_state :up}
```

Правило А7: отбраковать деталь с негодной высотой.

Ниже приведено продукционное правило А7.

ЕСЛИ деталь в системе И лифт внизу И измерения высоты произведены И высота детали негодная И выталкиватель втянут, ТО выдвинуть выталкиватель, удалить деталь из системы, поместить деталь в брак, сбросить измерения.

Краткая запись: $c1 \& \neg c3 \& c4 \& \neg c7 \rightarrow \neg c1 \& \neg c4 \& c7 \& c9$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?p :has_pusher_state :retracted.  
  ?wp rdf:type :Tested_WP.  
  ?f :has_value false}  
INSERT {  
  ?p :has_pusher_state :extended.  
  ?wp rdf:type :Reject_WP.  
  ?f :has_value true}  
WHERE {  
  ?wp rdf:type :Tested_WP.  
  ?l rdf:type :Lift.  
  ?l :has_lift_state :down.  
  ?f rdf:type :Flag.  
  ?f :has_value false.  
  ?wp :has_hight false.  
  ?p rdf:type :Pusher.
```

```
?p :has_pusher_state :retracted}
```

Правило А8: втянуть выталкиватель.

Ниже приведено продукционное правило А8.

ЕСЛИ выталкиватель выдвинут, ТО втянуть выталкиватель.

Краткая запись: $\neg c4 \rightarrow c4$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
?p :has_pusher_state :extended}  
INSERT {  
?p :has_pusher_state :retracted}  
WHERE {  
?p rdf:type :Pusher.  
?p :has_pusher_state :extended}
```

Правило А9: принять деталь с годной высотой.

Ниже приведено продукционное правило А9.

ЕСЛИ деталь в системе И лифт внизу И измерения высоты произведены И высота детали годная, ТО удалить деталь из системы, поместить деталь в годные изделия, сбросить измерения.

Краткая запись: $c1 \& c5 \& \neg c7 \& c3 \rightarrow \neg c1 \& c8 \& c7$.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
?wp rdf:type :Tested_WP.  
?f :has_value false}  
INSERT {  
?wp rdf:type :Accepted_WP.  
?f :has_value true}  
WHERE {  
?wp rdf:type :Tested_WP.  
?l rdf:type :Lift.  
?l :has_lift_state :down.  
?f rdf:type :Flag.  
?f :has_value false.  
?wp :has_hight true}
```

2.4.3 Верификация системы правил

В тестирующей станции существуют три сценария обработки деталей:

1) «негодный цвет», 2) «нормальный цвет, негодная высота», 3) «нормальный цвет, нормальная высота». Первому сценарию соответствует последовательность выполнения правил (А1, А8), второму сценарию –

последовательность (A2, A3, A4, A6, A7, A8), третьему сценарию – последовательность (A2, A3, A5, A8, A6, A9).

Для верификации системы правил была разработана модель на основе сетей Петри (рисунок 2.22).

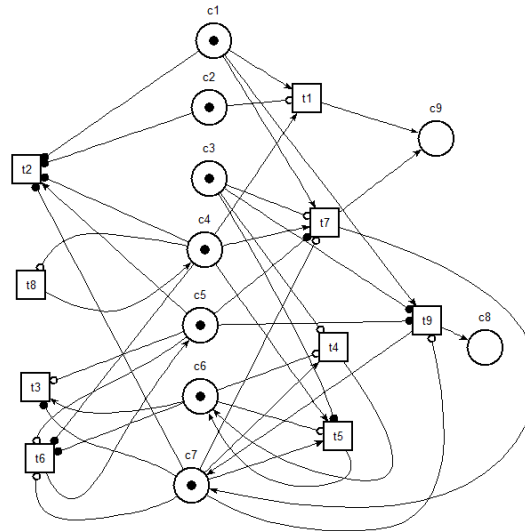


Рисунок 2.22 – Сетевая модель системы продукционных правил, описывающих функционирование тестирующей станции

Каждое правило моделируется отдельным переходом сети Петри. Условия правил представлены позициями сетевой модели, а действия – переходами. В сетевой используются обычные, ингибиторные дуги и дуги «чтения».

Интерпретация начальной маркировки сетевой модели следующая: c1 – деталь в системе (да), c2 – цвет детали (годный), c3 – высота детали (годная), c4 – состояние выталкивателя (втянут), c5 – состояние лифта (в нижнем положении), c6 – состояние измерительного стержня (поднят), c7 – производились ли измерения высоты (нет), c8 – деталь нормальная (не определено), c9 – деталь бракованная (не определено).

Сетевая модель была сформирована и исследована в системе TINA [56] с использованием метода имитационного моделирования. При этом было определено, что при любых параметрах входной детали процесс ее обработки завершается, и она попадает в одну из двух категорий (годная/брак). По окончании тестирования детали все устройства тестирующей станции переходят в свои начальные состояния. Важным результатом оказалось

обнаружение детерминированности сетевой модели, поскольку из этого вытекают важные свойства системы продукционных правил:

- правила являются неконфликтными,
- на каждом шаге активным является только одно правило.

Последнее свойство значительно упрощает систему интерпретации правил.

2.5 Моделирование обрабатывающей станции

2.5.1 Онтологическая модель

На рисунке 2.23 слева показана обрабатывающая станция FESTO, а справа – классы онтологии, представляющие основные объекты этой станции.

Смысловая интерпретация классов онтологии представлена ниже: Table_hole – лунки поворотного стола, Drill – дрель, Clamp – зажим, Checker – проверочный стержень, Flag – флаг, определяющий запрос на поворот стола. Будем считать, что данный флаг контролируется системой управления.

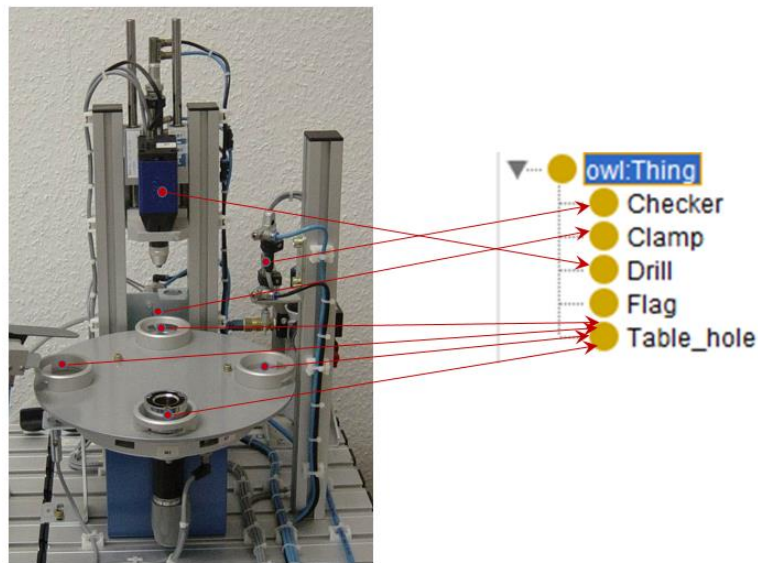


Рисунок 2.23 – Отображение физических объектов тестирующей станции (слева) на классы онтологии (справа)

Особенностью онтологии является использование свойств по данным для представления состояний устройств. Смысловая интерпретация свойств по данным следующая: has_drill_state – состояние дрели, has_clamp_state – состояние зажима, has_checker_state – состояние проверочного стержня,

has_hole_state – состояние лунки поворотного стола, has_flag_value – значение флага запроса на поворот стола.

Значение всех свойств характеризуется бинарным типом данных. Таким образом, каждое устройство имеет два состояния. Например, значение true относительно дрели соответствует состоянию «Находится в верхнем положении и выключена», а значение false – состоянию «Находится в нижнем положении и включена». Как правило, значение true соответствует состоянию устройства в начальном состоянии (или наличию детали в лунке). Изменение состояния устройств происходит «скачком» при выполнении соответствующего производственного правила.

Каждый класс устройств представлен одним единственным индивидом (например, drill_1, clamp_1, checker_1, flag_1). Однако класс для лунок представлен четырьмя индивидами L1, L2, L3 и L4, поскольку стол имеет четыре лунки. В нашей модели лунки – это не физические сущности, а скорее, позиции №№ 1,2,3,4 для деталей на поворотном столе. Стол может поворачиваться на 90 градусов по часовой стрелке. При подобном повороте можно считать, что происходит перемещение деталей по позициям поворотного стола. Если предположить, что L1, L2, L3 и L4 – это переменные, определяющие состояния лунок, то поворот моделируется последовательным выполнением операторов присваивания: $L4=L3$, $L3=L2$, $L2=L1$, $L1=0$. Обнуление первой лунки определяется тем, что считается, что новая деталь на станцию не поступает до тех пор, пока не будет полностью обработана текущая деталь на станции.

Начальное состояние обрабатывающей станции следующее:

- лунка L1 содержит деталь (true),
- дрель находится в верхнем положении (true),
- зажим отведен (true),
- проверочный стержень находится в верхнем положении (true),
- установлен флаг запроса поворота стола (true).

Ниже представлены правила функционирования обрабатывающей станции FESTO.

2.5.2 Продукционные правила и SPARQL-запросы

Правило Б1: поворот стола после прихода новой детали.

Ниже приведено продукционное правило Б1.

ЕСЛИ Лунка L1 содержит деталь И дрель в верхнем положении И зажим отведен И проверочный стержень в верхнем положении И установлен флаг запроса поворота стола, ТО повернуть стол (действие Д1), сбросить флаг запроса поворота стола.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
?f :has_flag_value true}  
INSERT {  
?f :has_flag_value false}  
WHERE {  
:L1 rdf:type :Table_hole.  
:L1 :has_hole_state true.  
?d rdf:type :Drill.  
?d :has_drill_state true.  
?c rdf:type :Clamp.  
?c :has_clamp_state true.  
?s rdf:type :Checker.  
?s :has_checker_state true.  
?f rdf:type :Flag.  
?f :has_flag_value true}
```

Действие Д1: поворот стола.

Поворот стола является комплексным действием, состоящим в модели из ряда шагов, поэтому для его реализации разработан отдельный SPARQL-запрос, приведенный ниже:

```
DELETE {  
:L1 :has_hole_state ?x1.  
:L2 :has_hole_state ?x2.  
:L3 :has_hole_state ?x3.  
:L4 :has_hole_state ?x4}  
INSERT {  
:L1 :has_hole_state false.  
:L2 :has_hole_state ?x1.  
:L3 :has_hole_state ?x2.  
:L4 :has_hole_state ?x3}  
WHERE {  
:L1 rdf:type :Table_hole.  
:L2 rdf:type :Table_hole.
```

```

:L3 rdf:type :Table_hole.
:L4 rdf:type :Table_hole.
:L1 :has_hole_state ?x1.
:L2 :has_hole_state ?x2.
:L3 :has_hole_state ?x3.
:L4 :has_hole_state ?x4}

```

Правило Б2: зажать деталь.

Ниже приведено продукционное правило Б2.

ЕСЛИ лунка L2 содержит деталь И дрель в верхнем положении И зажим отведен И сброшен флаг запроса поворота стола, ТО зажать деталь.

Запрос на языке SPARQL представлен ниже:

```

DELETE {
?c :has_clamp_state true}
INSERT {
?c :has_clamp_state false}
WHERE {
:L2 rdf:type :Table_hole.
:L2 :has_hole_state true.
?d rdf:type :Drill.
?d :has_drill_state true.
?c rdf:type :Clamp.
?c :has_clamp_state true.
?f rdf:type :Flag.
?f :has_flag_value false}

```

Правило Б3: просверлить отверстие в детали.

Ниже приведено продукционное правило Б3.

ЕСЛИ деталь зажата, ТО включить и опустить дрель.

Запрос на языке SPARQL представлен ниже:

```

DELETE {
?d :has_drill_state true}
INSERT {
?d :has_drill_state false}
WHERE {
?c rdf:type :Clamp.
?c :has_clamp_state false.
?d rdf:type :Drill}

```

Правило Б4: поднять дрель, отвести зажим.

Ниже приведено продукционное правило Б4.

ЕСЛИ дрель опущена, ТО поднять и выключить дрель, отвести зажим, установить флаг запроса поворота стола.

Запрос на языке SPARQL представлен ниже:

```

DELETE {
?d :has_drill_state false.
?c :has_clamp_state false.
?f :has_flag_value false}
INSERT {
?d :has_drill_state true.
?c :has_clamp_state true.
?f :has_flag_value true}
WHERE {
?d rdf:type :Drill.
?d :has_drill_state false.
?c rdf:type :Clamp.
?f rdf:type :Flag}

```

Правило Б5: повернуть стол после сверления.

Ниже приведено продукционное правило Б5.

ЕСЛИ лунка L2 содержит деталь И дрель в верхнем положении И зажим отведен И установлен флаг запроса поворота стола, ТО повернуть стол, сбросить флаг запроса поворота стола.

Запрос на языке SPARQL представлен ниже:

```

DELETE {
?f :has_flag_value true}
INSERT {
?f :has_flag_value false}
WHERE {
:L2 rdf:type :Table_hole.
:L2 :has_hole_state true.
?d rdf:type :Drill.
?d :has_drill_state true.
?c rdf:type :Clamp.
?c :has_clamp_state true.
?f rdf:type :Flag.
?f :has_flag_value true}

```

Сразу после выполнения данного SPARQL-запроса должен быть выполнен SPARQL-запрос, реализующий действие Д1 (см. выше).

Правило Б6: опустить проверочный стержень.

Ниже приведено продукционное правило Б6.

ЕСЛИ лунка L3 содержит деталь И проверочный стержень находится в верхнем положении И сброшен флаг запроса поворота стола, ТО опустить проверочный стержень.

Запрос на языке SPARQL представлен ниже:

```

DELETE {

```

```

?s :has_checker_state true}
INSERT {
?s :has_checker_state false}
WHERE {
:L3 rdf:type :Table_hole.
:L3 :has_hole_state true.
?s rdf:type :Checker.
?s :has_checker_state true.
?f rdf:type :Flag.
?f :has_flag_value false}

```

Правило Б7: поднять проверочный стержень.

Ниже приведено продукционное правило Б7.

ЕСЛИ проверочный стержень опущен, ТО поднять проверочный стержень, установить флаг запроса поворота стола.

Запрос на языке SPARQL представлен ниже:

```

DELETE {?s:has_checker_state false.
?f :has_flag_value false}
INSERT {
?s :has_checker_state true.
?f :has_flag_value true}
WHERE {
?s rdf:type :Checker.
?s :has_checker_state false.
?f rdf:type :Flag}

```

Правило Б8: повернуть стол после проверки отверстия.

Ниже приведено продукционное правило Б8.

ЕСЛИ лунка L3 содержит деталь И проверочный стержень находится в верхнем положении И установлен флаг запроса поворота стола, ТО повернуть стол, сбросить флаг запроса поворота стола.

Запрос на языке SPARQL представлен ниже:

```

DELETE {
?f :has_flag_value true}
INSERT {
?f :has_flag_value false}
WHERE {
:L3 rdf:type :Table_hole.
:L3 :has_hole_state true.
?s rdf:type :Checker.
?s :has_checker_state true.
?f rdf:type :Flag.
?f :has_flag_value true}

```

Сразу после выполнения данного SPARQL-запроса должен быть выполнен SPARQL-запрос, реализующий действие Д1 (см. выше).

Правило Б9: уход детали из системы.

Ниже приведено продукционное правило Б9.

ЕСЛИ лунка L4 содержит деталь И сброшен флаг запроса поворота стола, ТО убрать деталь из лунки L4, установить флаг запроса поворота стола.

Запрос на языке SPARQL представлен ниже:

```
DELETE {  
  ?f :has_flag_value false.  
  :L4 :has_hole_state true}  
INSERT {  
  ?f :has_flag_value true.  
  :L4 :has_hole_state false}  
WHERE {  
  :L4 rdf:type :Table_hole.  
  :L4 :has_hole_state true.  
  ?f rdf:type :Flag.  
  ?f :has_flag_value false}
```

2.5.3 Верификация системы правил

Технологический процесс обработки детали моделируется выполнением следующей последовательности SPARQL-запросов: Б1, Д1, Б2, Б3, Б4, Б5, Д1, Б6, Б7, Б8, Д1, Б9.

Для верификации системы правил серии Б была разработана модель на основе сетей Петри (рисунок 2.24). В отличие от сетевой модели на рисунке 2.22, данная сетевая модель не является прямым отображением 1:1 соответствующих продукционных правил. Это является следствием того, что невозможно одним переходом промоделировать комбинированное действие по повороту стола, где требуется последовательно менять состояния четырех лунок ($L4=L3$, $L3=L2$, $L2=L1$, $L1=0$). В сетевой модели на рисунке 2.24 поворот стола моделируется верхним фрагментом сети, включающим переходы e1, e2, e3, t1, t2, t3 и позиции L1, L2, L3, L4, d1, d2, startR, freeR. Наличие метки в позиции freeR свидетельствует о том, процесс поворота стола закончен. Переходы R1..R9 непосредственно моделируют соответствующие продукционные правила Б1..Б9.

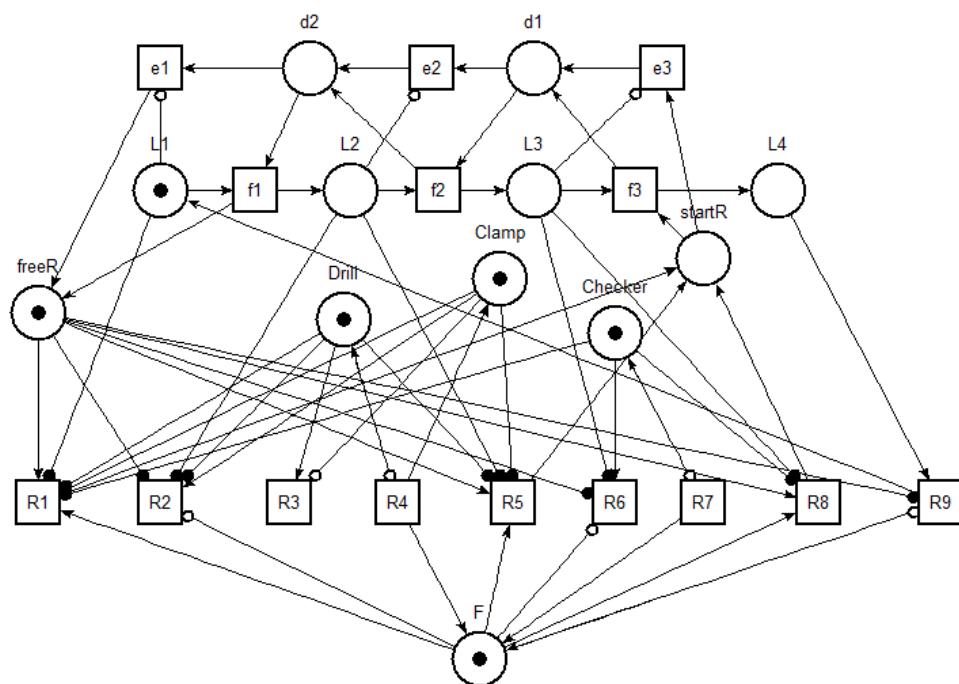


Рисунок 2.24 – Сетевая модель функционирования обрабатывающей станции FESTO

Сеть Петри, представленная на рисунке 2.24, была промоделирована в системе TINA [56]. Анализ модели продемонстрировал правильность функционирования системы продукционных правил. Также было показано, что в один момент времени разрешенным может быть только одно правило, что упрощает создание клиентского приложения для моделирования обрабатывающей станции.

В заключение следует отметить, что разработка онтологических моделей тестирующей и обрабатывающей станций системы FESTO производилась с использованием системы Protégé [57]. Для вычисления SPARQL-запросов использовался сервер Apache Jena Fuseki [58]. Клиентская часть для моделирования тестирующей станции разработана на языке JavaScript с использованием библиотеки React, предназначенной для создания пользовательских интерфейсов, а для моделирования обрабатывающей станции – на языке C#.

В человеко–машинном интерфейсе клиентских приложений представлены признаки, показывающие степень активности продукционных правил. Для определения того, является некоторое правило разрешенным или

неразрешенным, используются ASK–запросы языка SPARQL, возвращающие логические значения true или false. Пример ASK–запроса для правила A1 «Отбраковать деталь с негодным цветом» приведен ниже:

```
ASK {  
  ?wp rdf:type :Tested_WP.  
  ?wp :has_color false.  
  ?p rdf:type :Pusher.  
  ?p :has_pusher_state :retracted}
```

3 Инструментальное средство для поддержки онтологического моделирования на основе языка SPARQL

3.1 Назначение, особенности и функции программы

SPARQL/C-M – это программная система для ручного управления выполнением SPARQL UPDATE – запросов, хранящихся в формате RDF (Resource Description Framework).

Программа имеет 8 функций:

1) ввод установочной информации о SPARQL-сервере и рабочем наборе данных (включая IP-адрес сервера, номер порта и имя набора данных), а также установка префиксов (если это необходимо);

2) управление текстовыми файлами со SPARQL Update-запросами:

- загрузка текстового файла со SPARQL Update-запросами;
- просмотр текстовых файлов со SPARQL Update-запросами;
- редактирование текстовых файлов со SPARQL Update-запросами;

3) управление RDF-наборам данных:

- создание нового набора данных;
- загрузка в него данных из файла;
- удаление набора данных;
- просмотр списка наборов данных, существующих на SPARQL-сервере, и выбор из него активного (текущего) набора данных;
- просмотр выбранного набора данных;
- установка / добавление / изменение префиксов и их сокращений, как стандартных типа rdf, rdfs, owl, xsd, так и пользовательских;

4) возможность ввода с экрана произвольного SPARQL– запроса на выборку, его выполнение на сервере и вывод на экран результатов его выполнения;

5) справочная система, показывающая назначение программы, основные шаги и приемы по работе с программой;

6) отображение статуса SPARQL Update-запросов, в частности, разрешен запрос или нет, ошибочен ли он или нет, сколько решений имеется для данного запроса;

7) посылка SPARQL Update-запроса и его выполнение на сервере при нажатии соответствующей кнопки;

8) отображение информации об успешности или не успешности выполнения SPARQL Update-запроса.

3.2 Описание пользовательского интерфейса

Основное меню приложения SPARQL/C-M (рисунок 3.1) обеспечивает доступ к основным функциям и операциям, необходимым для работы с данными в формате RDF с использованием языка запросов SPARQL.

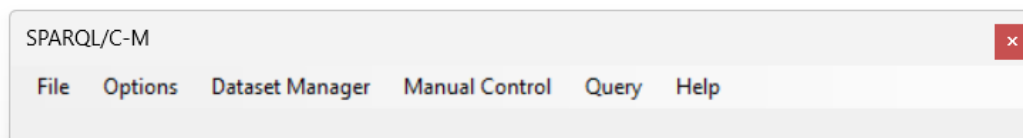


Рисунок 3.1 – Основное меню приложения

Меню «File» (рисунок 3.2) предоставляет функции для работы с файлами и завершения работы с приложением.

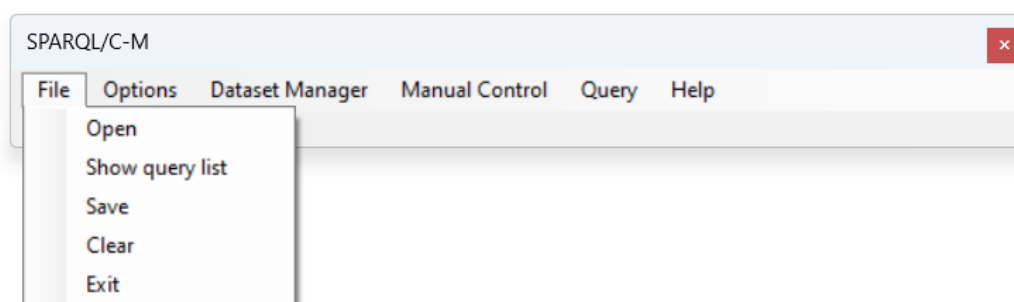


Рисунок 3.2 – Меню «File»

Пункты меню «File»:

- Open – загрузить текстовый файл со SPARQL Update – запросами,
- Show query list – показать список SPARQL Update – запросов,
- Edit query – открыть SPARQL Update – запрос для редактирования,
- Save – сохранить текстовый файл со SPARQL Update – запросами после их редактирования,

- Clear – очистить систему (сброс в исходное состояние),
- Exit – выйти из системы.

Пункт меню (и, соответственно, окно «Options») (рисунок 3.3) предоставляет пользователю возможность настройки параметров подключения к серверу RDF.

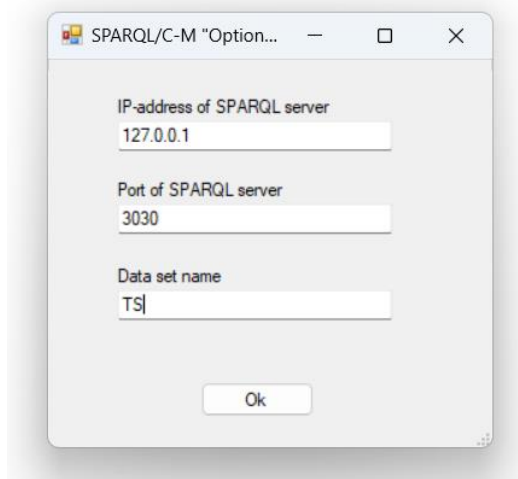


Рисунок 3.3 – Окно «Options»

Действия доступные в этой форме:

- IP-address of SPARQL server – позволяет пользователю ввести IP-адрес SPARQL-сервера (иначе, точки доступа SPARQL). Если поле останется пустым или будет введен неверный формат IP-адреса, отображается соответствующее сообщение об ошибке,
- Port of SPARQL server – позволяет пользователю указать порт для подключения к SPARQL-серверу. Если поле останется пустым или будет введено нечисловое значение, отображается соответствующее сообщение об ошибке,
- Data set name – позволяет пользователю ввести название набора данных на SPARQL-сервере. Если поле останется пустым, отображается соответствующее сообщение об ошибке.

Окно «Dataset Manager» (рисунок 3.4) предоставляет функции для управления наборами данных на SPARQL-сервере.

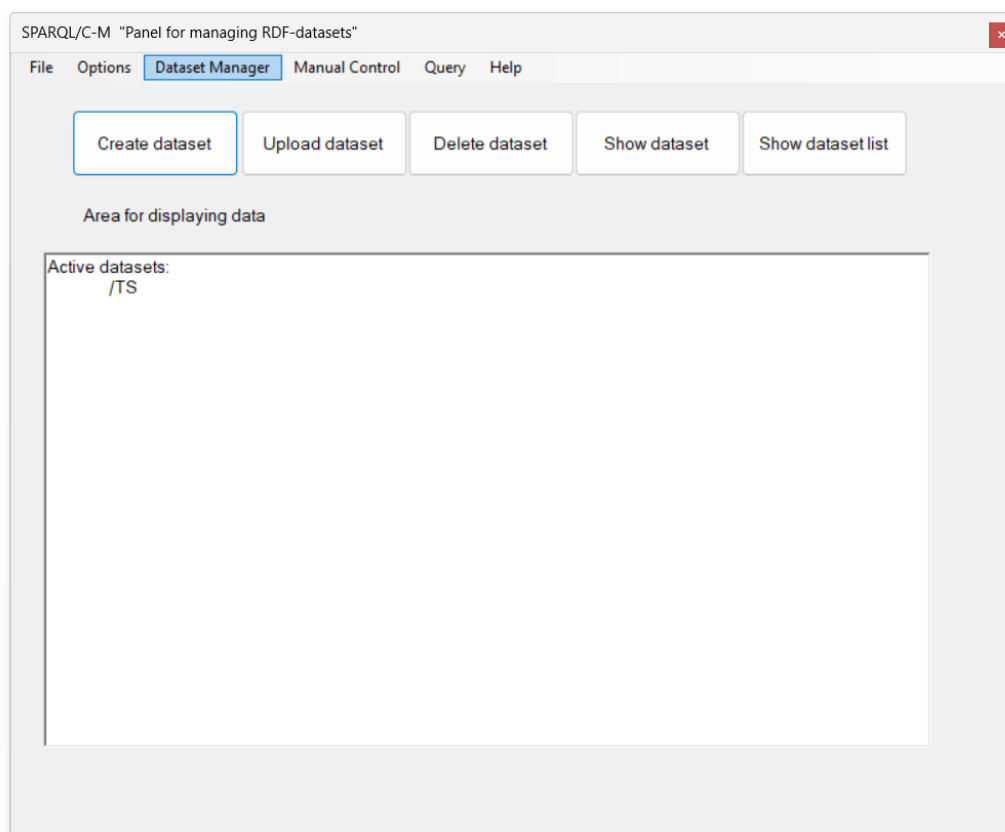


Рисунок 3.4 – Окно «Dataset Manager»

Окно «Dataset Manager» включает в себя следующие пункты вложенного меню:

- Create dataset – создание нового набора данных на SPARQL-сервере,
- Upload dataset – загрузка в набор данных на SPARQL-сервере данных из файла,
- Delete dataset – удаление набора данных на SPARQL-сервере,
- Show dataset list – просмотр списка наборов данных, существующих на SPARQL-сервере,
- Show dataset – просмотр выбранного набора данных.

Окно «Query» (рисунок 3.5) предоставляет пользователю возможность вводить и выполнять SPARQL-запросы, а также просматривать и анализировать результаты этих запросов.

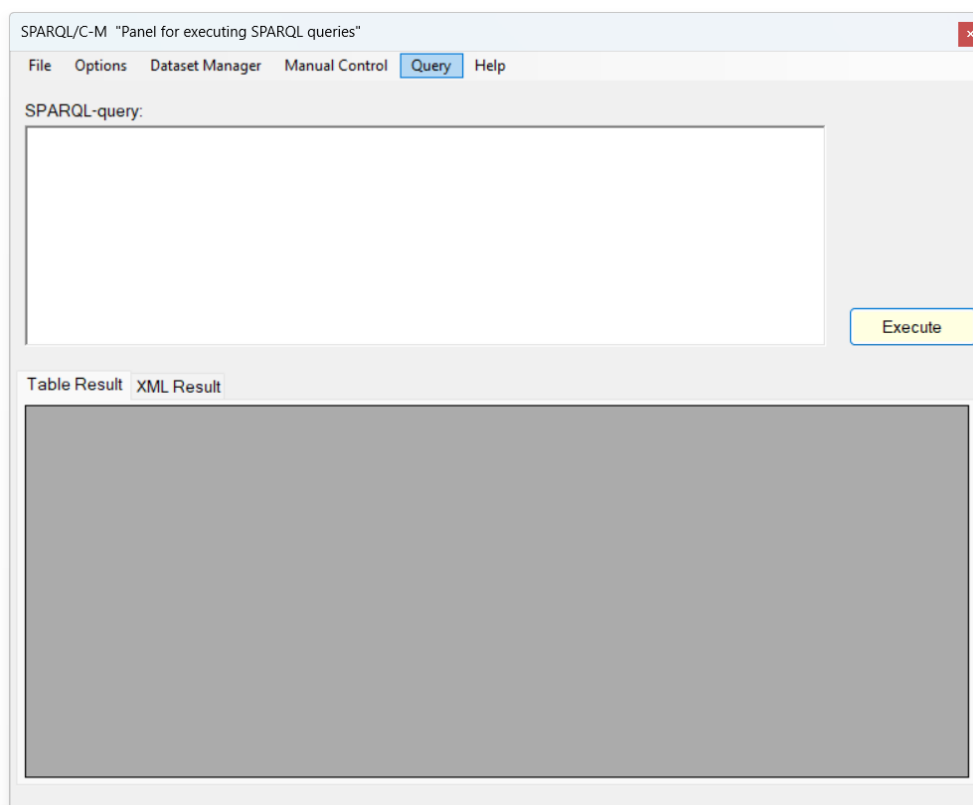


Рисунок 3.5 – Окно «Query»

Окно «Query» содержит следующие элементы:

- текстовое поле SPARQL-query – это место, куда пользователь может вводить или вставлять SPARQL-запросы для выполнения. Здесь пользователь может создавать или редактировать запросы перед их отправкой на SPARQL-сервер,
- текстовое поле для результатов выполнения запроса (под вкладками Table Result/XML Result) – в это текстовое поле выводятся результаты выполнения запроса или сообщения об ошибках при выполнении запроса. Пользователь может видеть текстовое представление результатов запроса или получать уведомления о проблемах, возникающих при выполнении запроса,
- кнопка «Execute» – при нажатии на эту кнопку приложение отправляет введенный пользователем SPARQL-запрос на сервер RDF для выполнения. Результаты запроса отображаются в текстовом поле для вывода результатов,
- вкладка Table Result – позволяет отображать результаты выполнения запроса в соответствующем поле в виде таблицы. Если запрос возвращает наборы данных, они будут отображаться здесь в удобном табличном формате,

– вкладка XML Result – позволяет отображать результаты выполнения запроса в виде XML–документа.

Окно «Panel for executing SPARQL queries» (рисунок 3.6) обеспечивает пользователя возможностью управлять списком SPARQL–запросов, редактировать их и выполнять на SPARQL–сервере, обеспечивая удобный и эффективный способ работы с данными.

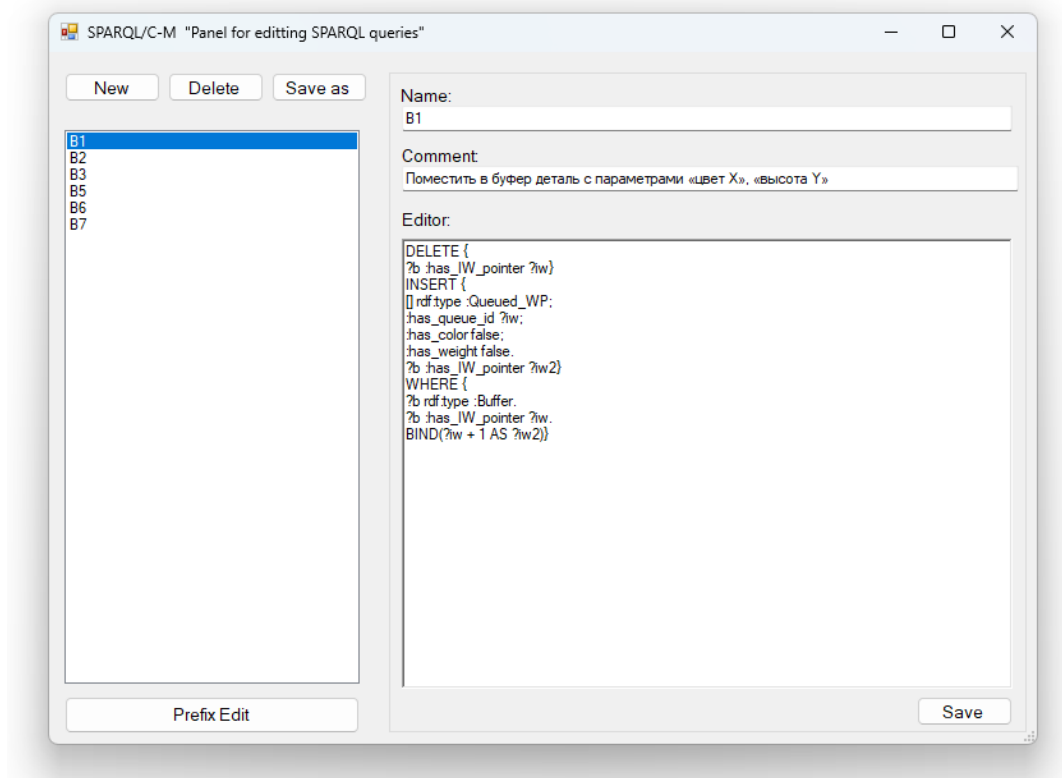


Рисунок 3.6 – Окно «Panel for executing SPARQL queries»

Окно «Panel for executing SPARQL queries» включает в себя следующие элементы:

- список запросов (queryListBox) – в этом списке отображаются доступные SPARQL–запросы. Пользователь может выбрать запрос из списка для его редактирования или выполнения,

- текстовые поля для редактирования имени запроса и комментария (Name, Comment) – в этих полях пользователь может просмотреть или изменить имя и комментарий выбранного запроса,

– текстовое поле Editor – в этом поле пользователь может просматривать или редактировать текст выбранного запроса. Здесь можно вносить изменения в сам запрос,

– кнопка «Prefix Edit» – при нажатии открывается форма для редактирования списка префиксов (рисунок 3.7), которые используются в запросах,

– кнопка «New» – создает новый запрос и добавляет его в список запросов,

– кнопка «Delete» – удаляет выбранный запрос из списка,

– кнопка «Save as» – позволяет сохранить текущий список запросов в файле,

– кнопка «Save» – сохраняет внесенные изменения в выбранный запрос.

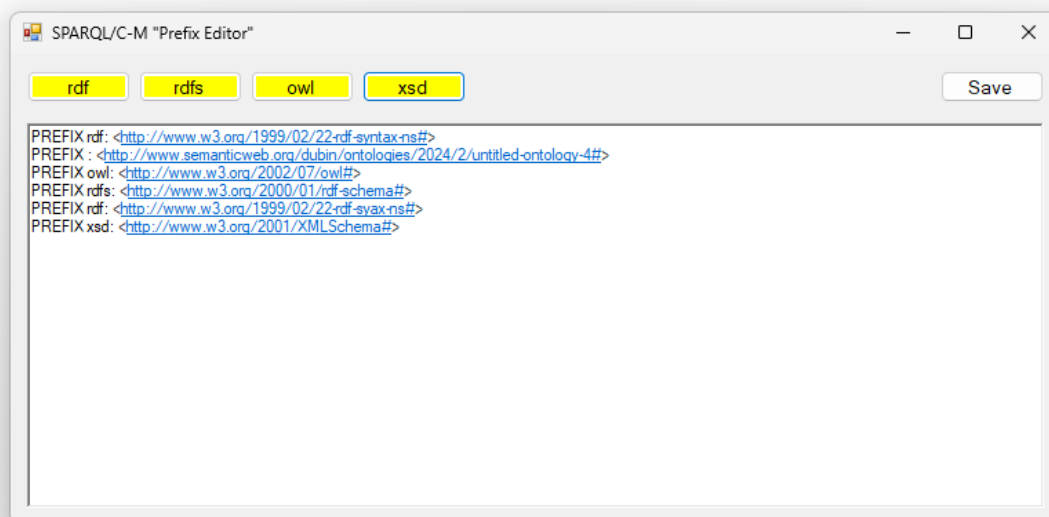


Рисунок 3.7 – Окно «Prefix Editor»

Окно «Manual Control» (рисунок 3.8) обеспечивает пользователю возможность взаимодействия с элементами управления для выполнения SPARQL-запросов, обновления данных на сервере и контроля над процессом выполнения запросов.

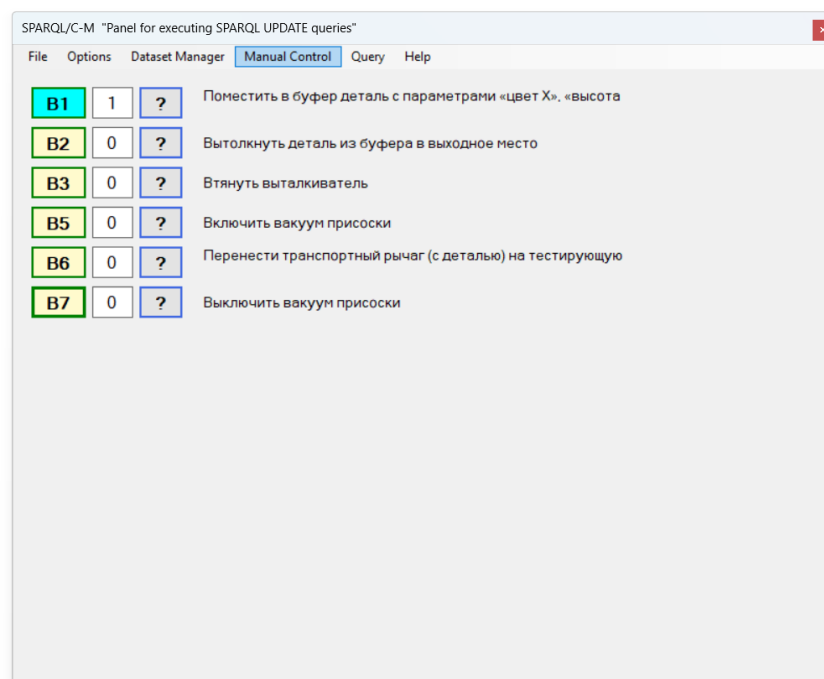


Рисунок 3.8 – Окно «Manual Control»

Окно «Manual Control» включает в себя следующие элементы:

- панель для отображения элементов управления – на этой панели располагаются элементы управления для выполнения SPARQL-запросов. Эти элементы управления создаются динамически в зависимости от списка доступных запросов,
- элементы управления – каждый элемент управления представляет собой отдельный SPARQL-запрос, который может быть выполнен на сервере. Элементы управления обновляются и отображаются на панели в соответствии с переданным списком запросов.

Например, на рисунке 3.9 представлено отображение правила B1 – Поместить в буфер деталь с параметрами «цвет X», «высота Y».

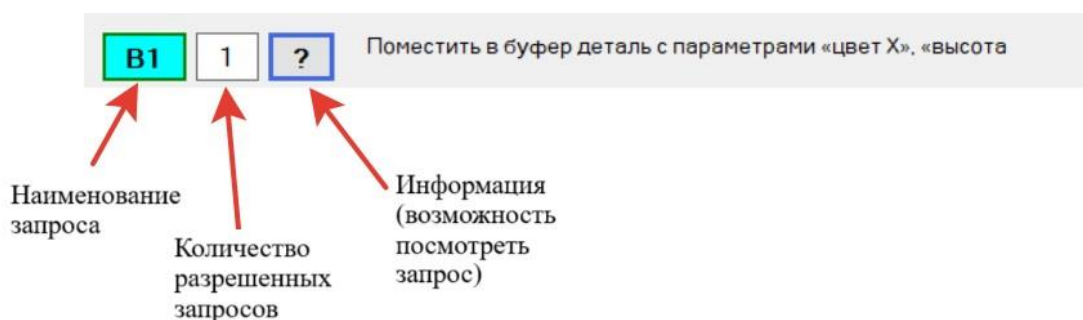


Рисунок 3.9 – Правило B1 на панели для отображения элементов управления

Цвета кнопок изменяются в зависимости от результата выполнения SPARQL-запросов или операций обновления данных на сервере. Вот как это происходит:

- циановая кнопка – запуск выполнения ASK-запроса или операции обновления данных (иначе, SPARQL Update запроса) на сервере,
- желтая кнопка – ошибка парсинга запроса или запрос завершился с ошибкой,
- бледно-желтая кнопка – результат выполнения запроса ASK вернул false,
- синяя кнопка с красной рамкой – ошибка выполнения операции обновления данных на сервере.

Окно «Help» (рисунок 3.10) позволяет пользователям быстро получить информацию о различных аспектах использования программы. Каждый раздел содержит полезные сведения о том, как использовать функции программы, что они означают и как они работают.

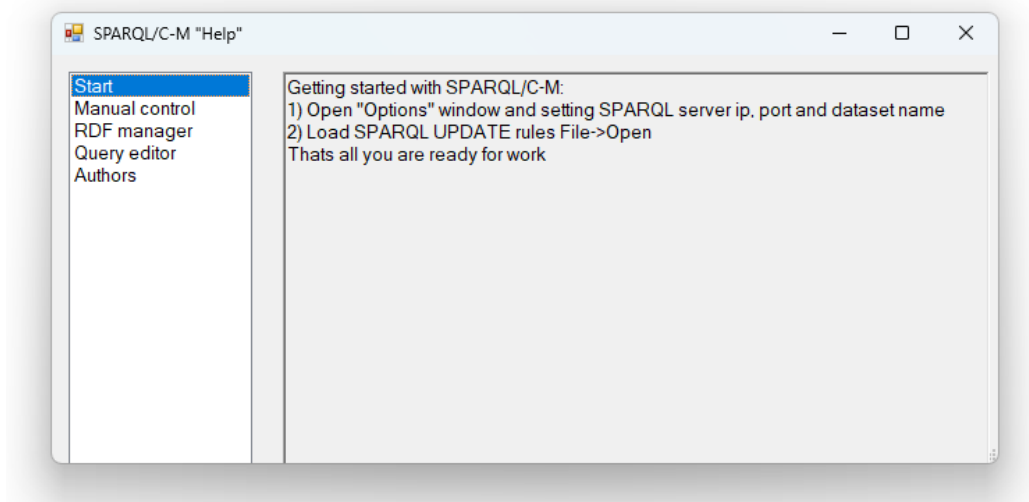


Рисунок 3.10 – Окно «Help»

3.3 UML–диаграмма

Ниже на рисунке 3.11 представлена диаграмма классов программы.

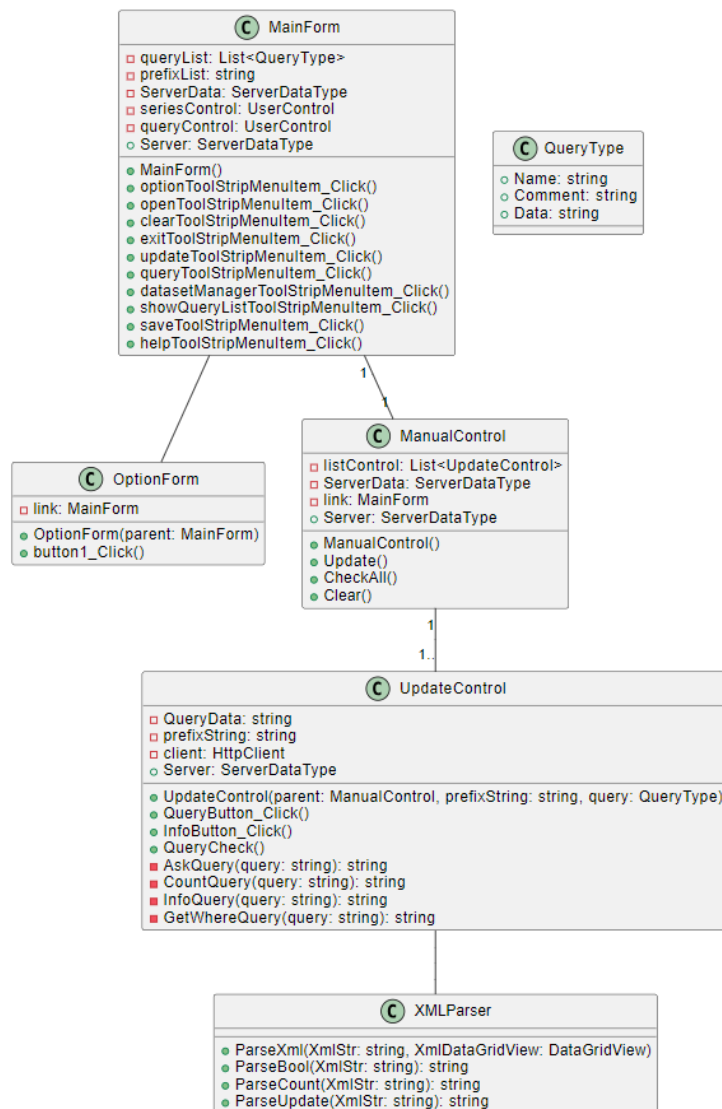


Рисунок 3.11 – UML–диаграмма

3.4 Описание модулей (классов)

Класс MainForm является основным пользовательским интерфейсом в приложении SparqlClient. Он предоставляет доступ к основным функциям приложения, таким как выполнение SPARQL–запросов, настройка сервера и управление данными набора RDF. Кроме того, он отвечает за отображение различных элементов управления в зависимости от выбора пользователя из меню.

Класс содержит такие поля как:

- queryList – список объектов типа QueryType, представляющих информацию о SPARQL-запросах,
- prefixList – строка, содержащая информацию о префиксах для SPARQL-запросов,
- ServerData – данные о сервере типа ServerDataType, включая IP-адрес, порт и имя набора данных,
- seriesControl – элемент управления для выполнения серии запросов,
- queryControl – элемент управления для выполнения отдельных SPARQL-запросов.

Класс содержит такие методы как:

- MainForm() – конструктор класса, инициализирует поля класса и настраивает интерфейс формы,
- Server – свойство для доступа к данным о сервере,
- optionToolStripMenuItem_Click() – метод, вызываемый при нажатии на элемент меню "Option", открывает окно настроек,
- openToolStripMenuItem_Click() – метод, вызываемый при нажатии на элемент меню "Open", открывает файл с SPARQL-запросами,
- clearToolStripMenuItem_Click() – метод, вызываемый при нажатии на элемент меню "Clear", переводит программу в стартовое состояние,
- exitToolStripMenuItem_Click() – метод, вызываемый при нажатии на элемент меню "Exit", закрывает приложение,
- updateToolStripMenuItem_Click() – метод, вызываемый при нажатии на элемент меню «Manual Control», отображает элемент управления для выполнения серии запросов,
- queryToolStripMenuItem_Click() – метод, вызываемый при нажатии на элемент меню «Query», отображает элемент управления для выполнения отдельных запросов,
- datasetManagerToolStripMenuItem_Click() – метод, вызываемый при нажатии на элемент меню «Dataset Manager», отображает элемент управления для управления набором данных RDF,

- `showQueryListToolStripMenuItem_Click()` – метод, вызываемый при нажатии на элемент меню «Show Query List», отображает окно для просмотра и редактирования списка запросов,

- `saveToolStripMenuItem_Click()` – метод, вызываемый при нажатии на элемент меню «Save», сохраняет текущие данные о запросах и префиксах в файл,

- `helpToolStripMenuItem_Click()` – метод, вызываемый при нажатии на элемент меню «Help», отображает окно справки.

Класс `ManualControl` – предоставляет интерфейс для ручного управления выполнением набором `Sparql` запросов. Он содержит список элементов управления `UpdateControl`, которые отвечают за выполнение отдельных SPARQL-запросов для обновления данных на сервере RDF.

Класс содержит такие поля как:

- `listControl` – список элементов управления `UpdateControl`,

- `ServerData` – данные о сервере типа `ServerDataType`.

Класс содержит такие методы как:

- `ManualControl()`: Конструктор класса, инициализирует список элементов управления и данные о сервере,

- `Server` – свойство для установки данных о сервере,

- `Update()` – метод для обновления списка элементов управления на основе предоставленных префиксов и списка SPARQL-запросов,

- `CheckAll()` – метод для выполнения проверки всех элементов управления,

- `Clear()` – метод для очистки списка элементов управления.

Класс `UpdateControl` – предоставляет возможность пользователям отправлять SPARQL-запросы на сервер RDF и отслеживать результаты выполнения. Он содержит методы для формирования различных типов запросов на основе введенного текста запроса и префиксов. Кроме того, он обрабатывает события нажатия кнопок на элементе управления и отправляет соответствующие запросы на сервер. Результаты выполнения запросов

отображаются пользователю, позволяя ему взаимодействовать с данными на сервере RDF.

Класс содержит такие поля как:

- QueryData – строка, содержащая текст SPARQL-запроса,
- prefixString – строка, содержащая информацию о префиксах для SPARQL-запроса,
- client – объект класса HttpClient, используемый для отправки HTTP-запросов на сервер RDF,
- ServerData – данные о сервере типа ServerDataType,
- Link – ссылка на родительский элемент управления ManualControl.

Класс содержит такие методы как:

- UpdateControl (parent: ManualControl, prefixString: string, query: QueryType) – конструктор класса, инициализирует поля класса и устанавливает информацию о префиксах и тексте запроса,
- Server – свойство для доступа к данным о сервере,
- QueryButton_Click() – метод, вызываемый при нажатии на кнопку выполнения запроса, отправляет SPARQL-запрос на сервер и отслеживает результат выполнения,
- InfoButton_Click() – метод, вызываемый при нажатии на кнопку для отправки информационного SPARQL-запроса на сервер,
- QueryCheck() – метод, проверяющий данные сервера и вызывающий метод AskQuery,
- AskQuery(query: string): string – метод для формирования SPARQL-запроса типа ASK на основе переданной строки запроса,
- CountQuery(query: string): string – метод для формирования SPARQL-запроса типа SELECT с функцией COUNT на основе переданной строки запроса,
- InfoQuery(query: string): string – метод для формирования информационного SPARQL-запроса на основе переданной строки запроса,

– `GetWhereQuery(query: string): string` – метод для извлечения фрагмента запроса, содержащего блок `WHERE`.

Класс `OptionForm` – предоставляет пользователю интерфейс для настройки параметров сервера. При открытии формы, текущие данные о сервере загружаются и отображаются в соответствующих полях. Пользователь может внести изменения в поля IP-адреса, порта и имени набора данных. После нажатия кнопки «ОК», введенные данные проверяются на корректность. Если все данные заполнены корректно, они обновляют данные о сервере в главной форме, и окно настроек закрывается. Если введены некорректные данные, пользователю выводятся соответствующие сообщения об ошибке.

Класс содержит такие поля как:

– `link`: Ссылка на главную форму `MainForm`.

Класс содержит такие методы как:

– `OptionForm(parent: MainForm)`: Конструктор класса, инициализирует поля класса и устанавливает данные о сервере из главной формы,

– `button1_Click()`: Метод, вызываемый при нажатии на кнопку "ОК", проверяет введенные данные, обновляет данные о сервере и закрывает окно настроек.

Класс `XMLParser` – предоставляет набор методов для разбора XML-данных, полученных в ответ на различные типы запросов к серверу RDF. Он позволяет приложению эффективно обрабатывать XML-ответы, извлекать необходимую информацию и представлять её пользователю в удобном виде. Кроме того, класс обеспечивает обработку возможных ошибок при разборе XML-данных, что способствует повышению надёжности приложения.

Класс содержит такие методы как:

– `ParseXml(XmlStr: string, XmlDataGridView: DataGridView)`: Метод для разбора XML-строки и отображения данных в элементе управления `DataGridView`. Он извлекает информацию из XML, формирует

соответствующий объект DataTable и связывает его с элементом управления DataGridView, чтобы пользователь мог просматривать данные,

- ParseBool(XmlStr: string): string: Метод для разбора XML-строки и извлечения логического значения (true или false), представленного в формате SPARQL-запроса,

- ParseCount(XmlStr: string): string: Метод для разбора XML-строки и извлечения количества найденных подграфов,

- ParseUpdate(XmlStr: string): string: Метод для разбора XML-строки и извлечения информации об успешности выполнения SPARQL-запроса на обновление данных на сервере RDF.

3.5 Руководство пользователя

Разработанная клиентская программа SparqlClient предназначена для поддержки удаленного выполнения пользовательских SPARQL Update-запросов на SPARQL-сервере. В совокупности клиентская программа SparqlClient и SPARQL-сервер образуют проблемно-ориентированную клиент-серверную систему. Для ее нормального функционирования необходимо выполнение следующих системных требований:

- IBM PC-совместимый ПК, ЦП не ниже Pentium Core 2 Duo,
- операционная система Windows 7/10,
- SPARQL-сервер Apache Jena Fuseki версии 3.16 и выше,
- минимальный объем оперативной памяти компьютера – 2 Гб.

Для начала работы с программой выполните следующие 7 шагов:

- 1) запустите SPARQL-сервер Apache Jena Fuseki. Для этого перейдите в каталог сервера и дважды щелкните мышкой на файле fuseki-server.bat. Появится консольное окно черного цвета, в которое будут выводиться сообщения. Последним сообщением должна быть строчка:

```
INFO Server      :: Started 2024/05/10 21:16:00 MSK on port 3030
```

Это означает, что SPARQL-сервер запущен на порту 3030.

- 2) запустите клиентскую программу SparqlClient. Для этого перейдите в каталог Sparql C-M и щелкните два раза мышкой на файле

Sparql C–M.exe. Сразу после этого на экране монитора появится начальное окно программы (см. рисунок 3.1),

3) чтобы загрузить набор данных на сервер откройте меню «Dataset manager» (рисунок 3.4), далее необходимо нажать кнопку «Create dataset» и загрузить набор данных на сервер нажатием кнопки «Upload dataset»,

4) откройте меню «Options» (рисунок 3.3) и установите параметры сервера SPARQL (IP–адрес, порт, название набора данных),

5) загрузите файл с правилами SPARQL UPDATE через меню «File → Open» (рисунок 3.2). Этот файл содержит правила, которые будут использоваться для обновления данных на сервере,

6) отредактировать загруженные правила (если нужно) можно через меню «File → Show query list»,

7) для начала выполнения запросов необходимо открыть меню «Manual Control». «Manual Control» (рисунок 3.8) обеспечивает пользователю возможность взаимодействия с элементами управления для выполнения SPARQL–запросов, обновления данных на сервере и контроля над процессом выполнения запросов. Каждое правило отображается в виде отдельного управляющего элемента (рисунок 3.9), позволяющего пользователю взаимодействовать с соответствующим правилом.

Каждый управляющий элемент содержит:

- кнопку выполнения правила, которая изменяет цвет в зависимости от статуса выполнения,
- поле с количеством разрешенных запросов,
- кнопку информации для просмотра запроса,
- комментарий к правилу.

Если у вас возникли вопросы по использованию программы, вы можете обратиться к разделу «Help». Там содержится подробная информация о различных аспектах работы с программой.

3.6 Имитационное моделирование станций FESTO в инструментальной системе

Ниже на рисунке 3.12 представлена визуализация динамики функционирования модели распределительной станции (моделируется выполнением следующей последовательности SPARQL-запросов: B1, B2, B3, B5, B6, B7).

Смысловая интерпретация последовательности этих запросов следующая:

- B1 – поместить в буфер деталь с параметрами «цвет X», «высота Y»,
- B2 – вытолкнуть деталь из буфера в выходное место,
- B3 – втянуть выталкиватель,
- B5 – включить вакуум присоски,
- B6 – перенести транспортный рычаг (с деталью) на тестирующую станцию,
- B7 – выключить вакуум присоски.

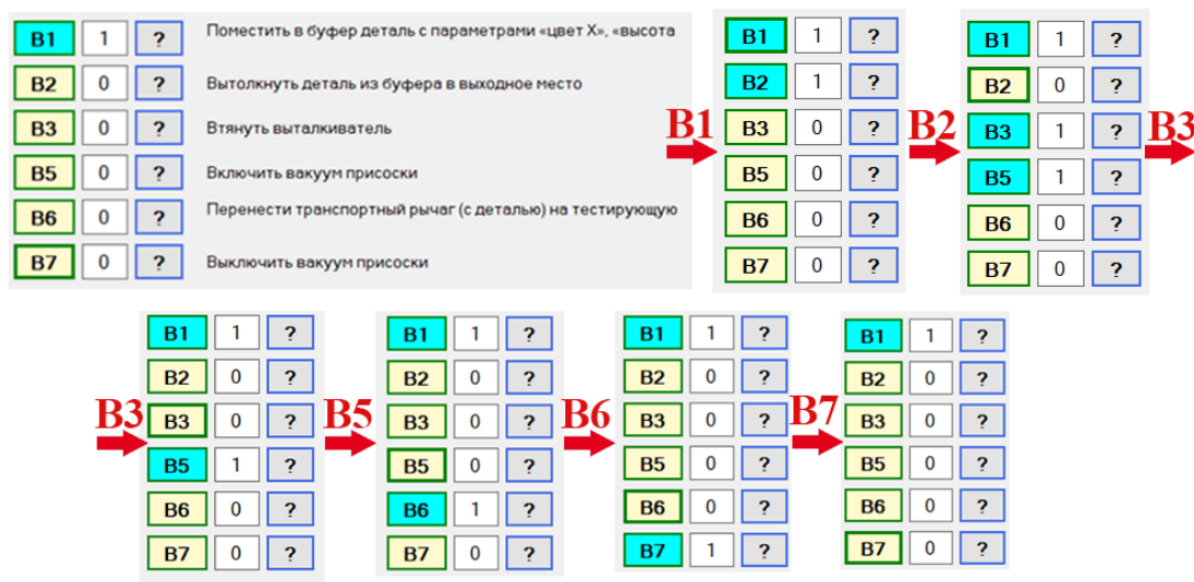


Рисунок 3.12 – Визуализация динамики функционирования модели распределительной станции

В тестирующей станции существуют три сценария обработки деталей: 1) «негодный цвет», 2) «нормальный цвет, негодная высота», 3) «нормальный цвет, нормальная высота».

Ниже на рисунках 3.13–3.15 представлена визуализация динамики функционирования модели тестирующей станции для каждого из трех сценариев:

1) Сценарий «негодный цвет» (моделируется выполнением следующей последовательности SPARQL-запросов: A1, A8),

Смысловая интерпретация последовательности этих запросов следующая:

- A1 – отбраковать деталь с негодным цветом,
- A8 – втянуть выталкиватель.

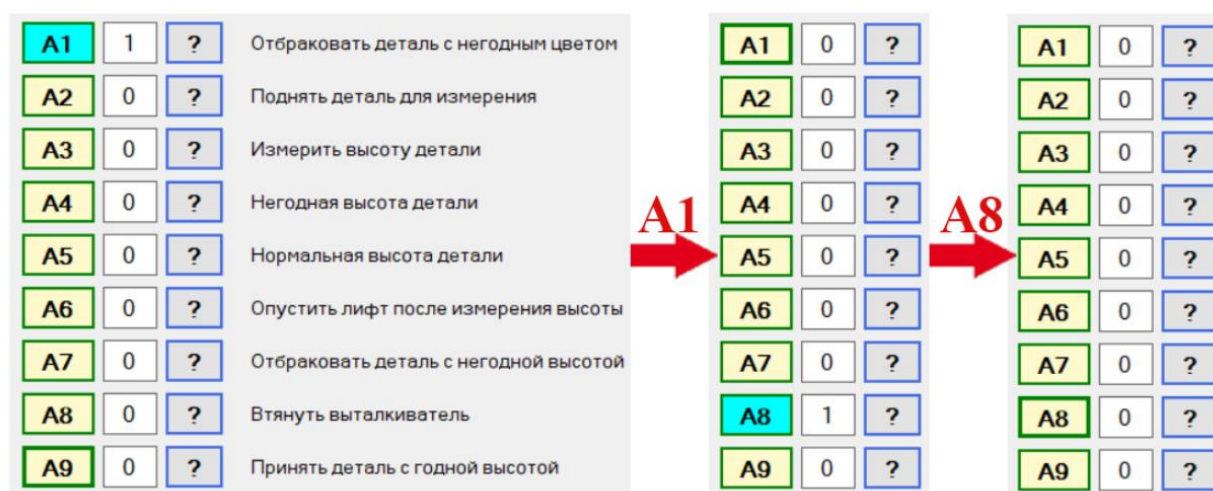


Рисунок 3.13 – Визуализация динамики функционирования модели тестирующей станции (сценарий «негодный цвет»)

2) сценарий «нормальный цвет, негодная высота» (моделируется выполнением следующей последовательности SPARQL-запросов: A2, A3, A4, A6, A7, A8),

Смысловая интерпретация последовательности этих запросов следующая:

- A2 – поднять деталь для измерения,
- A3 – измерить высоту детали,
- A4 – негодная высота детали,
- A6 – опустить лифт после измерения высоты,
- A7 – отбраковать деталь с негодной высотой,
- A8 – втянуть выталкиватель.



Рисунок 3.14 – Визуализация динамики функционирования модели тестирующей станции (сценарий «нормальный цвет, негодная высота»)

3) сценарий «нормальный цвет, нормальная высота» (моделируется выполнением следующей последовательности SPARQL-запросов: A2, A3, A5, A8, A6, A9).

Смысловая интерпретация последовательности этих запросов следующая:

- A2 – поднять деталь для измерения,
- A3 – измерить высоту детали,
- A5 – нормальная высота детали,
- A8 – втянуть выталкиватель,
- A6 – опустить лифт после измерения высоты,
- A9 – принять деталь с годной высотой.



Рисунок 3.15 – Визуализация динамики функционирования модели тестирующей станции (сценарий «нормальный цвет, нормальная высота»)

Ниже на рисунке 3.16 представлена визуализация динамики функционирования модели обрабатывающей станции (моделируется выполнением следующей последовательности SPARQL-запросов: Q1, D1, Q2, Q3, Q4, Q5, D1, Q6, Q7, Q8, D1, Q9).

Смысловая интерпретация последовательности этих запросов следующая:

- Q1 – поворот стола после прихода новой детали,
- D1 – поворот стола,
- Q2 – зажать детали,
- Q3 – просверлить отверстие в детали,
- Q4 – поднять дрель, отвести зажим,
- Q5 – повернуть стол после сверления,
- D1 – поворот стола,

- Q6 – опустить проверочный стержень,
- Q7 – поднять проверочный стержень,
- Q8 – повернуть стол после проверки отверстия,
- D1 – поворот стола,
- Q9 – уход детали из системы.

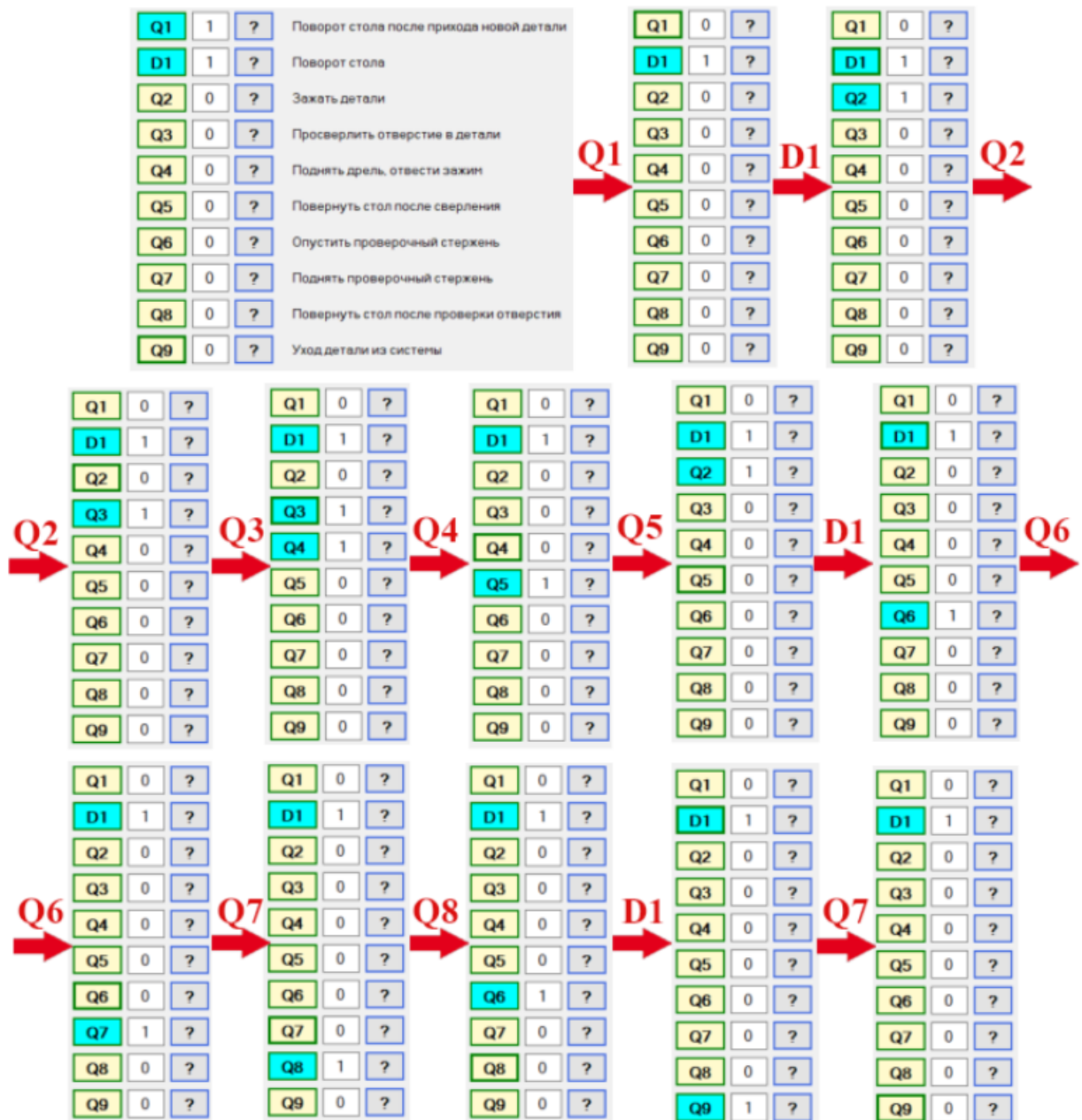


Рисунок 3.16 – Визуализация динамики функционирования модели обрабатывающей станции

ЗАКЛЮЧЕНИЕ

Основным результатом представленной магистерской работы является то, что была продемонстрирована возможность прототипирования систем промышленной автоматики и киберфизических систем (КФС) и показан путь создания их “цифровых двойников” (Digital Twins) на основе семантического Веб, который представляет собой, по сути, новую технологическую платформу в сети Интернет. Преимущества онтологий и семантического Веб хорошо известны из литературы.

Основной проблемой онтологического моделирования систем промышленной автоматики и КФС (а также других технических систем) является описание и имитация динамики их функционирования. В рамках представленной магистерской работы эта проблема была решена путем использования языка SPARQL и его расширения SPARQL Update, позволяющего динамически изменять онтологию, чем и достигается моделирование динамики функционирования системы.

Задача онтологического моделирования систем промышленной автоматики и КФС является «масштабируемой» в том плане, что можно моделировать системы практически любой сложности, используя одну и ту же методику. Увеличение степени сложности системы будет приводить лишь к увеличению сложности основополагающей онтологии и SPARQL-запросов. Для ускорения процесса проектирования онтологии и увеличения степени ее «читабельности», онтология может быть построена по модульному принципу. Кроме того, наличие инструментальных программных средств поддержки может значительно упростить задачи разработчика и пользователя.

В ходе выполнения данной работы магистерской были получены следующие научные и практические результаты:

- проведен обзор и анализ существующих работ по методам и средствам моделирования и проектирования систем промышленной автоматики. На основе анализа существующих источников можно констатировать, что имеется незначительное число работ по онтологическому моделированию

систем промышленной автоматике (русскоязычных работ практически нет). Кроме того, полностью отсутствуют работы, в которых рассматривались бы динамические аспекты онтологического моделирования систем промышленной автоматике и КФС,

- разработана методика онтологического моделирования дискретных систем промышленной автоматике на высоком уровне абстракции. Это означает, что в качестве основных артефактов проектирования рассматривались устройства и технологические операции, а не сигналы,

- разработана общая Tbox-онтология станций производственной системы FESTO. Данная онтология может использоваться для представления, моделирования и прототипирования других мехатронных систем и систем промышленной автоматике (а не только станций производственной системы FESTO),

- разработаны частные TBox+Abox-онтологии распределительной, тестирующей и обрабатывающей станций производственной системы FESTO на основе общей онтологии. Причем были разработаны две частные онтологии тестирующей станции, специализация одной из которых достигалась путем использования соответствующих экземпляров классов, а другая – путем использования подклассов,

- разработаны семантические ограничения, позволяющие проводить семантический анализ описания станций, приведены примеры общих и частных ограничений,

- разработаны системы продукционных правил для моделирования динамики функционирования станций. Продукционная модель представления знаний является наиболее популярной, четкой и понятной для разработчиков и пользователей,

- разработаны SPARQL Update – запросы типа DELETE-INSERT-WHERE, реализующие продукционные правила и трансформирующие Abox-онтологию. На данный момент язык SPARQL Update является самым мощным

средством преобразования онтологий, что является ключевым моментом в моделировании динамики функционирования,

- разработана инструментальная программная система SPARQL/C–M для поддержки онтологического (имитационного) моделирования дискретных систем промышленной автоматики. Одной из основных функций системы является ручное управление выполнением SPARQL Update – запросов в интерактивном режиме.

Полученные результаты способствуют развитию теории и практики интеллектуального моделирования и проектирования сложных систем промышленной автоматики, что является важным шагом на пути к реализации концепции Индустрии 4.0.

Использование полученных в магистерской работе результатов по онтологическому моделированию систем промышленной автоматики позволит:

- формализовать предметную область в сфере промышленной автоматики,
- уменьшить число ошибок проектирования,
- увеличить степень повторного использования разработанных артефактов проектирования и предложенных решений,
- увеличить надежность проектируемых систем,
- автоматизировать процесс проектирования,
- повысить качество проектных решений,
- ускорить процесс проектирования,
- уменьшить затраты на разработку, модификацию и сопровождение.

Направлениями дальнейших исследований являются:

- онтологическое моделирование других станций производственной системы FESTO (например, сортирующей станции),
- интеграция моделей станций в одну общую онтологическую модель системы FESTO, учитывая механизмы взаимодействия между ними,

- размещение разработанных онтологий в семантическом Веб в качестве разделяемых ресурсов,

- моделирование структурных изменений при функционировании системы.

В данной работе динамика функционирования была представлена через изменение, по сути, скалярных величин, характеризующих (глобальное) состояние системы. Однако предложенный подход к моделированию на основе изменения онтологий допускает более глубокие и кардинальные изменения системных состояний – на уровне графов. Это позволяет моделировать структурные изменения системы как на физическом, так и на логическом уровне. Данная возможность полезна при моделировании реконфигурируемых и адаптивных КФС. Таким образом, требуется разработка новых примеров для демонстрации такого подхода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Jazdi N. Cyber physical systems in the context of Industry 4.0 // IEEE International Conference on Automation, Quality and Testing, Robotics, 2014, P.1-4.
2. Szeredi P., Lukácsy G., Benkő T. The Semantic Web Explained: The Technology and Mathematics behind Web 3.0. Cambridge University Press, 2014. 478 p.
3. Gruber T.R. A Translation Approach to Portable Ontology Specifications // Knowledge Acquisition, 1993. N 5(2). P. 199–220.
4. Keet C. M. An Introduction to Ontology Engineering. College Publications, 2018. 344 p.
5. Горшков С. Введение в онтологическое моделирование. – ТриниДата, 2016. 166 с.
6. Dai W., Dubinin V., Vyatkin V. Automatically Generated Layered Ontological Models for Semantic Analysis of Component–Based Control Systems // IEEE Transactions on Industrial Informatics. . – 2013. Vol. 9, Issue 4. – P. 2124–2136.
7. Thea OWL library [Электронный ресурс] – URL: <http://vangelisv.github.io/thea> Дата обращения: 13.03.2024.
8. DuCharme B. Learning SPARQL: Querying and Updating with SPARQL 1.1. O'Reilly, 2013. 386 p.
9. Sendall S., Kozaczynski W. Model transformation: The heart and soul of model–driven software development // IEEE Software. Special Issue on Model–Driven Software Development. – 2003.– 20(5).–P.42–45
10. Yang C.–W., Dubinin V., Vyatkin V. Ontology Driven Approach to Generate Distributed Automation Control from Substation Automation Design // IEEE Transactions on Industrial Informatics, 2017, Volume 13, Issue 2, P. 668–679.
11. Дубинин В.Н., Дубинин А.В., Янг Ч.–В., Вяткин В.В. Использование языка SPARQL в онтологическом моделировании мультиагентных систем в семантическом Web // Известия ВУЗов. Поволжский регион. Технические науки. – 2020. – № 1. – С. 4–18. (БАК)

12. FESTO Didactic [Электронный ресурс] – URL: <https://www.festo-didactic.com>. Дата обращения: 17.03.2024.

13. Самонов А.В. Методика разработки интеллектуальных средств проектирования автоматизированных систем управления технологическими процессами // Интеллектуальные технологии на транспорте. – 2020. – № 4. – С. 10–22.

14. Ахмедьянова Г.Ф., Пищухин А.М. Онтологический подход к проектированию научно–производственных систем // Онтология проектирования. –2022. – том 12, № 1. – С. 57–67.

15. Онтологическое моделирование предприятий: методы и технологии: монография; [отв. ред. С. В. Горшков]; предисл. С. В. Горшкова. – Екатеринбург: Изд–во Урал. ун–та, 2019. – 236 с.

16. Мельник Е.Н., Бадалов А.Ю, Шведин Б.Я. и др. Онтологические модели для систем управления электроснабжением олимпийских объектов в Сочи // Онтология проектирования. – 2014. № 1. – С. 6–23.

17. Отт П. К., Зобнин Б. Б. Онтологическая модель насосной станции по очистке КРВ // Система управления экологической безопасностью: сборник трудов XIV международной научно–практической конференции (Екатеринбург, 20–21 мая 2020 г.). — Екатеринбург: УрФУ, 2020. — С. 185–190.

18. Гайдукова Е. Ключ к гиперавтоматизации: онтологические модели // Открытые системы. СУБД. – 2021. – № 3. – С.21–23.

19. Андриюшкевич С.К., Ковалев С.П., Нефедов Е.И. Разработка цифрового двойника энергетической системы на основе онтологической модели // Автоматизация в промышленности. – 2020. – №1.

20. Шилов Н.Г., Щекотов М.С. Онтологическое моделирование управляющих автоматов сервисов киберфизических систем // Моделирование систем и процессов. – 2017. – № 5. – С. 80–87.

21. Организация цифровых производств Индустрии 4. 0 на основе киберфизических систем и онтологий // Научно–технический вестник

информационных технологий, механики и оптики. – 2018. – том 18, № 2. – С. 268–277.

22. Дубинин В.Н., Вяткин В. Проектирование распределенных систем управления промышленными процессами с использованием UML–FB // Известия высших учебных заведений. Поволжский регион. Сер. Технические науки. Пенза: Изд-во Пенз. гос. ун-та, 2004. N 2 (11). С.136–146.

23. Lee, E.A. "The Past, Present, and Future of Cyber–Physical Systems: A Focus on Models." *Sensors*, vol. 15, pp. 4837–4869, February 2015.

24. Gupta, B.B., Akhtar, T. "A survey on smart power grid: frameworks, tools, security issues, and solutions." *Annals of Telecommunications*, vol. 72, no. 9–10, pp. 517–549, 2017.

25. Austin, M. "Model–based systems engineering for cyber–physical systems." Keynote Presentation at International Conference on Systems (ICONS 2014), Nice, France, February 26, 2014.

26. Hehenberger, P., Vogel–Heuser, B., Bradley, D., Eynard, B., Tomiyama, T., Achiche, S. "Design, Modelling, Simulation and Integration of Cyber Physical Systems: Methods and Applications." *Computers in Industry*, vol. 82, pp. 273–289, 2016.

27. Delgoshaei, P., Austin, M.A., Pertzborn, A.J. "A Semantic Framework for Modeling and Simulation of Cyber–Physical Systems." *International Journal on Advances in Systems and Measurements*, vol. 7, no. 3 & 4, 2014.

28. Vanherpen, K., et al. "Ontological reasoning for consistency in the design of cyber–physical systems." In: 1st International Workshop on Cyber–Physical Production Systems, CPPS@CPSWeek 2016, Vienna, Austria, 2016.

29. Petnga, L., Austin, M. "An ontological framework for knowledge modeling and decision support in cyber–physical systems." *Advanced Engineering Informatics*, vol. 30, no. 1, pp. 77, 2016.

30. Grangel–González, et al. "Towards a Semantic Administrative Shell for Industry 4.0 Components." In: IEEE Tenth International Conference on Semantic Computing (ICSC), 2016.

31. Brings, J., Daun, M., Hildebrandt, C., Törsleff, S. "An Ontological Context Modeling Framework for Coping with the Dynamic Contexts of Cyber–physical Systems." In: Proceedings of the 6th International Conference on Model–Driven Engineering and Software Development (MODELSWARD 2018), pp. 396–403, 2018.
32. Dai, W., Dubinin, V., Vyatkin, V. "Automatically generated layered ontological models for semantic analysis of component–based control systems." IEEE Trans. Ind. Informat., vol. 9, no. 4, pp. 2124–2136, 2013.
33. Sendall, S., Kozaczynski, W. "Model transformation: The heart and soul of model–driven software development." IEEE Softw., vol. 20, no. 5, pp. 42–45, Sep./Oct. 2003.
34. Pan, J.Z., Staab, S., Aßmann, U., Ebert, J., Zhao, Y. *Ontology–Driven Software Development*. Berlin, Germany: Springer, 2012.
35. Yang, C.–W., Dubinin, V., Vyatkin, V. "Ontology Driven Approach to Generate Distributed Automation Control from Substation Automation Design." IEEE Transactions on Industrial Informatics, vol. 13, no. 2, pp. 668–679, 2017.
36. Corby, O., Zucker, C.F. "A Transformation Language for RDF based on SPARQL." In: Web Information Systems and Technologies (WEBIST 2015), pp. 318–340.
37. Zambon, E., Guizzardi, G. "Formal Definition of a General Ontology Pattern Language using a Graph Grammar." In: Federated Conference on Computer Science and Information Systems (FedCSIS), 2017.
38. Дубинин А.В., Ручкин М.А. Моделирование и верификация производственных систем в CPN Tools // Новые информационные технологии и системы (НИТиС 2019): сб. науч. статей XVI Междунар. науч.–техн. конф., 27–29 ноября 2019 г., Пенза – Пенза: Изд–во ПГУ, 2019. – С. 62–65.
39. Дубинин В.Н., Дубинин А.В., Климкина Л.П. Онтологическое моделирование сетей Петри с использованием языка SPARQL // Материалы III Международного научно–технического форума «Современные технологии в науке и образовании» (СТНО–2020), том 4, Рязань, 2020. – С. 119–124.

40. Дубинин А.В., Ручкин М.А. Графическое моделирование техпроцессов в производственной системе FESTO // Сб. статей VII Ежегодной всероссийской межвузовской науч.–практ. конф. «Информационные технологии в науке и образовании. Проблемы и перспективы» (ВМНПК – 2020), Пенза, 2020. – С. 393–396.

41. Дубинин А.В., Дубинин В.Н., Ручкин М.А. NCES–модель PnP–манипулятора // Сб. статей VII Ежегодной всероссийской межвузовской науч.–практ. конф. «Информационные технологии в науке и образовании. Проблемы и перспективы» (ВМНПК – 2020), Пенза, 2020. – С. 391–393.

42. Ручкин М.А., Дубинин А.В., Дубинин В.Н. Сервис–ориентированная модель системы транспортировки багажа аэропорта // Сборник научных статей Международной научно–технической конференции "Современные информационные технологии", Пенза, 2020, вып. 31. – С. 6–13.

43. Ручкин М.А., Дубинин А.В., Дубинин В.Н., Бычков А.С. Программная модель PnP–манипулятора на основе сервис–ориентированной архитектуры // Новые информационные технологии и системы (НИТиС–2020) : сб. науч. статей XVII Междунар. науч.–техн. конф., 18–19 ноября 2020 г., Пенза – Пенза: Изд–во ПГУ, 2020. – С. 193–197.

44. Дубинин А.В., Ручкин М.А., Дубинин В.Н., Бычков А.С. Моделирование производственной системы FESTO на основе продукционных правил в среде CLIPS // Новые информационные технологии и системы (НИТиС–2020): сб. науч. статей XVII Междунар. науч.–техн. конф., 18–19 ноября 2020 г., Пенза – Пенза: Изд–во ПГУ, 2020. – С. 189–192.

45. Дубинин В.Н., Дубинин А.В., Ручкин М.А. Программные модели мехатронных систем на основе сервис–ориентированной архитектуры // Модели, системы, сети в экономике, технике, природе и обществе. – 2020. – № 4 (36). – С. 97–108. (БАК)

46. Дубинин В.Н., Дубинин А.В., Климкина Л.П. Представление и преобразование онтологических описаний киберфизических систем с использованием раскрашенных сетей Петри // Материалы XVI

Международной научно–технической конференции «Оптико–электронные приборы и устройства в системах распознавания образов и обработки изображений», Курск, 2021. С. 109–111.

47. Каев В.Ю., Дубинин В.Н., Дубинин А.В., Климкина Л.П. Моделирование раскрашенных сетей Петри с использованием технологий семантического Web // Известия ВУЗов. Поволжский регион. Технические науки. – 2022. – № 1. – С. 62–77. (БАК)

48. Дубинин А.В., Дубинин В.Н., Бычков А.С. Программная имитационная модель тестирующей станции производственной системы FESTO // Новые информационные технологии и системы (НИТиС–2022) : сб. науч. статей XIX Междунар. науч.–техн. конф., 17–18 ноября 2022 г., Пенза – Пенза: Изд–во ПГУ, 2022. – С. 67–71.

49. Дубинин А.В., Дубинин В.Н. Генерация формальных моделей производственной системы FESTO на основе Process Mining // Новые информационные технологии и системы (НИТиС–2022) : сб. науч. статей XIX Междунар. науч.–техн. конф., 17–18 ноября 2022 г., Пенза – Пенза: Изд–во ПГУ, 2022. – С. 80–84.

50. Дубинин А.В., Ручкин М.А. Разработка распределенной функционально–блочной модели тестирующей станции системы FESTO // Новые информационные технологии и системы (НИТиС–2022) : сб. науч. статей XIX Междунар. науч.–техн. конф., 17–18 ноября 2022 г., Пенза – Пенза: Изд–во ПГУ, 2022. – С. 77–80.

51. Дубинин А.В., Дубинин В.Н. Онтологическое моделирование тестирующей станции производственной системы FESTO // Сб. ст. X Ежегодн. всеросс. межвуз. науч.–практ. конф. «Информационные технологии в науке и образовании. Проблемы и перспективы (ВМНПК – 2023)», Пенза, 2023. – С. 388–391.

52. Дубинин В.Н., Дубинин А.В., Климкина Л.П. Онтология сетевого формализма // Материалы VI Международного научно–технического форума

«Современные технологии в науке и образовании» (СТНО–2023), Том 4, Рязань, 2023. – С. 150–155.

53. Дубинин А.В., Ручкин М.А. Распределенная система для имитационного моделирования лабораторной модульной производственной установки // Новые информационные технологии и системы (НИТиС–2023) : сб. науч. статей XX Междунар. науч.–техн. конф., 16–17 ноября 2023 г., Пенза – Пенза: Изд–во ПГУ, 2023. – С. 312–316.

54. Дубинин А.В., Ручкин М.А., Дубинин В.Н. Моделирование производственных систем с использованием языка SPARQL // Сборник научных статей Международной научно–технической конференции "Современные информационные технологии", Пенза, 2023, вып. 38.

55. Дубинин В.Н., Вяткин В.В. Модели функциональных блоков IEC 61499, их проверка и трансформации в проектировании распределенных систем управления: монография. – Пенза: Изд–во ПГУ, 2012. – 348 с.

56. The TINA toolbox Home Page – Time Petri Net Analyzer [Электронный ресурс]. Режим доступа: <http://projects.laas.fr/tina/>

57. Protégé. A free, open–source ontology editor and framework for building intelligent systems [Электронный ресурс] – URL: <https://protege.stanford.edu/>. Дата обращения: 10.03.2024.

58. Apache Jena Fuseki [Электронный ресурс] – URL: <https://jena.apache.org/documentation/fuseki2/>. Дата обращения: 27.02.2024.

59. W3C Консорциум. Язык Web–онтологий OWL [Электронный ресурс] – URL: <http://www.w3.org/2004/OWL>. Дата обращения: 28.02.2024.

60. López, J., et al. "Process Mining for trust monitoring." In: 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, pp. 605–610, IEEE, 2015.

61. Yoo, Y.S., et al. "A Study on Continuous Monitoring Reinforcement for Sales Audit Using Process Mining Under Big Data Environment." Journal of Internet Computing and Services, vol. 17, no. 6, pp. 123–131, 2016.

62. Kingsley, O., et al. "Using semantic-based approach to manage perspectives of process mining: Application on improving learning process domain data." In: 2016 IEEE International Conference on Big Data (Big Data), pp. 3529–3538, IEEE, 2016.

ПРИЛОЖЕНИЕ А

(Обязательное)

Методы и средства интеллектуального моделирования и проектирования
систем промышленной автоматике

Листинг программы

ПРИЛОЖЕНИЕ А.1

Класс «MainForm»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace SparqlClient
{
    public struct ServerDataType
    {
        public string IPAddress;
        public string Port;
        public string DatasetName;
    }

    public partial class MainForm : Form
    {
        public List<QueryType> queryList;
        public string prefixList;
        public ServerDataType ServerData;

        public MainForm()
        {
            InitializeComponent();
            queryList = new List<QueryType>();
            prefixList = string.Empty;
            ServerData.IPAddress = "127.0.0.1";
            ServerData.Port = "3030";
            ServerData.DatasetName = string.Empty;
            this.seriesControl.Server = ServerData;
            this.queryControl.Server = ServerData;
        }
        public ServerDataType Server
        {
            get { return ServerData; }
            set { ServerData = value; }
        }

        //Обработок нажатия кнопок панели инструментов главного
        окна
    }
}
```

```

        private void optionToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            var option = new OptionForm(this);
            option.ShowDialog();
            this.seriesControl.Server = ServerData;
            this.queryControl.Server = ServerData;
        }

        private void openToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            var fileContent = string.Empty;
            var filePath = string.Empty;

            using (OpenFileDialog openFileDialog = new
OpenFileDialog())
            {
                openFileDialog.InitialDirectory = "c:\\";
                openFileDialog.Filter = "txt files
(*.txt)|*.txt|All files (*.*)|*.*";
                openFileDialog.FilterIndex = 2;
                openFileDialog.RestoreDirectory = true;

                if (openFileDialog.ShowDialog() ==
DialogResult.OK)
                {
                    //Get the path of specified file
                    filePath = openFileDialog.FileName;

                    //Read the contents of the file into a
stream
                    var fileStream = openFileDialog.OpenFile();

                    using (StreamReader reader = new
StreamReader(fileStream))
                    {
                        fileContent = reader.ReadToEnd();
                        string[] fileContentLine =
fileContent.Split(new string[] { "\r\n", "\n", "\r" },
StringSplitOptions.RemoveEmptyEntries);
                        //DataList.Clear();
                        bool readPrefix = false;
                        bool readQuery = false;
                        QueryType element = new QueryType();
                        foreach (string line in fileContentLine)
                        {
                            //начало считывание префиксов
                            if (line.StartsWith("#PREFIX"))
                            {
                                readQuery = false;
                                readPrefix = true;
                                continue;

```

```

    }
    //начало считывание запроса
    if (line.StartsWith("#QUERY"))
    {
        if(readQuery)
        {
            queryList.Add(element);
        }
        readPrefix = false;
        readQuery = true;
        string[] stringValues =
line.Split(new char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);
        element = new QueryType();
        element.Name = stringValues[1];
        foreach (QueryType query in
queryList)
            if (query.Name ==
element.Name)
            {
                DialogResult result =
MessageBox.Show(
                    "Query " +
query.Name + " already exist. Rewrite?",
                    "Warning",
                    MessageBoxButtons.YesNo,
                    MessageBoxIcon.Information,
                    MessageBoxDefaultButton.Button1,
                    MessageBoxOptions.DefaultDesktopOnly);
                if (result ==
DialogResult.Yes)
                {
                    queryList.Remove(query);
                }
                else
                {
                    readQuery = false;
                    break;
                }
            }
        for (int i = 2; i <
stringValues.Count(); i++)
        {
            element.Comment +=
stringValues[i] + " ";
        }
        continue;
    }
    //обнаружен конец файла с правилами
    if (line.StartsWith("#END"))

```

```

        {
            if (readQuery)
            {
                queryList.Add(element);
            }

this.seriesControl.Update(prefixList, queryList);
            break;
        }
        if (readPrefix)
        {
            prefixList +=
SearchPrefix(line);
        }
        if (readQuery)
        {
            element.Data += line + "\r\n";
        }
    }
}

private string SearchPrefix(string nPrefix)
{
    string result = string.Empty;
    string[] lines = nPrefix.Split(new char[] { '\r',
'\n' }, StringSplitOptions.RemoveEmptyEntries);
    foreach (string str in lines)
        if (!prefixList.Contains(str))
            result += str + "\r\n";
    return result;
}

private void clearToolStripMenuItem_Click(object sender,
EventArgs e)
{
    queryList.Clear();
    prefixList = string.Empty;
    this.seriesControl.Clear();
    this.queryControl.Clear();
    ServerData.IpAddress = "127.0.0.1";
    ServerData.Port = "3030";
    ServerData.DatasetName = string.Empty;
    this.seriesControl.Server = ServerData;
    this.queryControl.Server = ServerData;
    this.Text = "SPARQL/C-M";
    this.Controls.Remove(this.rdfManager);
    this.Controls.Remove(this.queryControl);
    this.Controls.Remove(this.seriesControl);
}

```

```

        this.Size = new System.Drawing.Size(590, 83);
    }

    private void exitToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        this.Close();
    }

    private void updateToolStripMenuItem_Click(object
sender, EventArgs e)
    {
        this.seriesControl.Update(prefixList, queryList);
        this.Controls.Remove(this.queryControl);
        this.Controls.Remove(this.rdfManager);
        this.Controls.Add(this.seriesControl);
        this.Text = "SPARQL/C-M \\"Panel for executing
SPARQL UPDATE queries\\"";
        this.Size = new System.Drawing.Size(800, 650);
        this.seriesControl.Server = ServerData;

        seriesControl.CheckAll();
    }

    private void queryToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        this.Controls.Remove(this.seriesControl);
        this.Controls.Remove(this.rdfManager);
        this.Controls.Add(this.queryControl);
        this.Text = "SPARQL/C-M \\"Panel for executing
SPARQL queries\\"";
        this.Size = new System.Drawing.Size(800, 650);
        this.queryControl.Server = ServerData;
    }

    private void
datasetManagerToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        this.Controls.Remove(this.seriesControl);
        this.Controls.Remove(this.queryControl);
        this.Controls.Add(this.rdfManager);
        this.Text = "SPARQL/C-M \\"Panel for managing RDF-
datasets\\"";
        this.Size = new System.Drawing.Size(800, 650);
    }

    private void showQueryListToolStripMenuItem_Click(object
sender, EventArgs e)
    {
        QueryEditorForm dialog = new
QueryEditorForm(prefixList, queryList);

```

```

        dialog.ShowDialog();
        prefixList = dialog.PrefixList;
        queryList = dialog.QueryData;
        this.seriesControl.Update(prefixList, queryList);
    }

    private void saveToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        using (SaveFileDialog saveFileDialog = new
SaveFileDialog())
        {
            saveFileDialog.Filter = "Text files(*.txt) |
*.txt |All files(*.*)|*.*";
            if (saveFileDialog.ShowDialog() ==
DialogResult.Cancel)
                return;
            string filename = saveFileDialog.FileName;
            string saveFile = "#PREFIX\n" + prefixList;
            foreach (QueryType query in queryList)
                saveFile += "#QUERY " + query.Name + " " +
query.Comment + "\n" + query.Data + "\n";
            saveFile += "#END\n";
            System.IO.File.WriteAllText(filename, saveFile);
        }
    }

    private void helpToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        HelpForm dialog = new HelpForm();
        dialog.ShowDialog();
    }
}

```

ПРИЛОЖЕНИЕ А.2

Класс «ManualControl»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SparqlClient
{
    public partial class ManualControl : UserControl
    {
        List<UpdateControl> listControl;
        ServerDataType ServerData;
        public ManualControl()
        {
            InitializeComponent();
            listControl = new List<UpdateControl>();
            ServerData.IpAddress = String.Empty;
            ServerData.Port = String.Empty;
            ServerData.DatasetName = String.Empty;
        }
        public ServerDataType Server
        {
            set
            {
                ServerData = value;
                foreach (UpdateControl control in listControl)
                    control.Server = ServerData;
            }
        }

        public void Update(string prefixList, List<QueryType>
queryData)
        {
            Clear();
            List<QueryType> localData = new List<QueryType>();
            foreach (QueryType query in queryData)
            {
                QueryType clone = new QueryType(query);
                localData.Add(clone);
            }
            localData.Reverse();
            foreach (QueryType query in localData)
            {
                UpdateControl control = new UpdateControl(this,
prefixList, query);
```

```

        listControl.Add(control);
        panell.Controls.Add(control);
        control.Dock = DockStyle.Top;
        control.Server = ServerData;
        control.QueryCheck();
    }
    CheckAll();
}

public void CheckAll()
{
    foreach (UpdateControl control in listControl)
    {
        control.QueryCheck();
    }
}

public void Clear()
{
    foreach (UpdateControl control in listControl)
        panell.Controls.Remove(control);
    listControl.Clear();
}
}
}

```

ПРИЛОЖЕНИЕ А.3

Класс «UpdateControl»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net.Http;

namespace SparqlClient
{
    public partial class UpdateControl : UserControl
    {
        public String QueryData;
        public String prefixString;
        HttpClient client = new HttpClient();
        static ServerDataType ServerData;
        static ManualControl link;
        public UpdateControl(ManualControl parent, string
prefixString, QueryType query)
        {
            InitializeComponent();
            QueryData = query.Data;
            this.prefixString = prefixString;
            QueryButton.Text = query.Name;
            CommentLabel.Text = query.Comment;
            CountLabel.Text = String.Empty;

            ServerData.IpAddress = String.Empty;
            ServerData.Port = String.Empty;
            ServerData.DatasetName = String.Empty;

            link = parent;
        }

        public ServerDataType Server
        {
            set { ServerData = value; }
        }

        private async void QueryButton_Click(object sender,
EventArgs e)
        {
            string query = prefixString + " " + QueryData;
            await SendUpdateRequest(client, query, QueryButton,
CountLabel);
        }
    }
}
```

```

    }

    private async void InfoButton_Click(object sender,
EventArgs e)
    {
        await SendInfoRequest(client, InfoQuery(QueryData),
QueryButton.Text);
    }

    public async void QueryCheck()
    {
        if(!string.IsNullOrEmpty(ServerData.IpAddress) &&
!string.IsNullOrEmpty(ServerData.Port) &&
!string.IsNullOrEmpty(ServerData.DatasetName))
            await SendAskRequest(client,
AskQuery(QueryData),CountQuery(QueryData), QueryButton,
CountLabel);
    }

    private string AskQuery(string query)
    {
        string result = prefixString+" ASK ";
        string whereStr = GetWhereQuery(query);
        return result + whereStr;
    }

    private string CountQuery(string query)
    {
        string result = prefixString+" SELECT (COUNT(*) AS
?n) ";
        string whereStr = GetWhereQuery(query);
        return result + whereStr;
    }

    private string InfoQuery(string query)
    {
        string result = prefixString+" SELECT * ";
        string whereStr = GetWhereQuery(query);
        return result + whereStr;
    }

    private string GetWhereQuery(string query)
    {
        string result = string.Empty;
        string[] words = query.Split(new string[] { "\r\n",
"\n","\r" }, StringSplitOptions.RemoveEmptyEntries);
        bool flagRead = false;
        bool flagCheack = true;
        foreach (string word in words)
        {
            if(flagCheack)
                if (word.StartsWith("WHERE") ||
word.StartsWith("where"))
            {

```

```

        flagRead = true;
        flagCheack = false;
    }
    if(flagRead)
    {
        result += word + "\r\n";
    }
}
return result;
}

/* static string CharToHexString(char myChar)
{
    // Convert char to string of hex using "X2" format
specifier
    return ((int)myChar).ToString("X2");
}*/

static async Task SendUpdateRequest(HttpClient handler,
string query, Button btn, Label lbl)
{
    StringContent cont = new StringContent("update=" +
query.Replace("+", "%2B"), Encoding.UTF8, "application/x-www-
form-urlencoded");
    HttpResponseMessage response = await
handler.PostAsync("http://" + ServerData.IpAddress + ":" +
ServerData.Port + "/" + ServerData.DatasetName+"/update", cont);
    var jsonResponse = await
response.Content.ReadAsStringAsync();
    string result = jsonResponse.ToString();
    XMLParser worker = new XMLParser();
    string parse = worker.ParseUpdate(result);
    if (parse.StartsWith("Success"))
    {
        link.CheckAll();
    }
    else
    {
        btn.BackColor = Color.DodgerBlue;
        btn.FlatAppearance.BorderColor = Color.Red;
        lbl.Text = string.Empty;
        MessageBox.Show( result, "Error execute query ",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

static async Task SendAskRequest(HttpClient handler,
string ask, string count, Button btn, Label lbl)
{
    StringContent cont = new StringContent(ask,
Encoding.UTF8, "application/sparql-query");

```

```

        HttpResponseMessage response = await
handler.PostAsync("http://" + ServerData.IpAddress + ":" +
ServerData.Port + "/" + ServerData.DatasetName + "/sparql",
cont);

        var jsonResponse = await
response.Content.ReadAsStringAsync();
        string result = jsonResponse.ToString();
        XMLParser worker = new XMLParser();
        if (result.StartsWith("Parse error"))
        {
            btn.BackColor = Color.Yellow;
            btn.FlatAppearance.BorderColor = Color.Red;
            lbl.Text = string.Empty;
        }
        else
        {
            string boolRes = worker.ParseBool(result);
            if (boolRes.StartsWith("true"))
            {
                btn.BackColor = Color.Cyan;
                btn.FlatAppearance.BorderColor =
Color.Green;

                await SendCountRequest(handler, count, lbl);
            }
            else if (boolRes.StartsWith("false"))
            {
                btn.BackColor = Color.LemonChiffon;
                btn.FlatAppearance.BorderColor =
Color.Green;

                lbl.Text = "0";
            }
        }
    }

    static async Task SendCountRequest(HttpClient handler,
string count, Label lbl)
    {
        StringContent cont = new StringContent(count,
Encoding.UTF8, "application/sparql-query");
        HttpResponseMessage response = await
handler.PostAsync("http://" + ServerData.IpAddress + ":" +
ServerData.Port + "/" + ServerData.DatasetName + "/sparql",
cont);

        var jsonResponse = await
response.Content.ReadAsStringAsync();
        string result = jsonResponse.ToString();
        XMLParser worker = new XMLParser();
        lbl.Text = worker.ParseCount(result);
    }

    static async Task SendInfoRequest(HttpClient handler,
string query, string Handle)
    {

```

```

        StringContent cont = new StringContent(query,
Encoding.UTF8, "application/sparql-query");
        HttpResponseMessage response = await
handler.PostAsync("http://" + ServerData.IpAddress + ":" +
ServerData.Port + "/" + ServerData.DatasetName + "/sparql",
cont);
        var jsonResponse = await
response.Content.ReadAsStringAsync();
        string result = jsonResponse.ToString();
        InfoDialog dialog = new InfoDialog(result, Handle);
        dialog.ShowDialog();
    }
}
}

```

ПРИЛОЖЕНИЕ А.4

Класс «OptionForm»

```
using System;
using System.Windows.Forms;
using System.Net;
namespace SparqlClient
{
    public partial class OptionForm : Form
    {
        MainForm link;
        public OptionForm(MainForm parent)
        {
            InitializeComponent();
            link = parent;
            ErrorLabel.Text = string.Empty;
            IpError.Text = string.Empty;
            PortError.Text = string.Empty;
            DatasetError.Text = string.Empty;

            ServerDataType ServerData = link.Server;
            IpBox.Text = ServerData.IpAddress;
            PortBox.Text = ServerData.Port;
            DatasetBox.Text = ServerData.DatasetName;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            ErrorLabel.Text = string.Empty;
            IpError.Text = string.Empty;
            PortError.Text = string.Empty;
            DatasetError.Text = string.Empty;

            if (string.IsNullOrEmpty(IpBox.Text))
            {
                IpError.Text = "*";
                ErrorLabel.Text = "Empty IP address";
            }
            else if (string.IsNullOrEmpty(PortBox.Text))
            {
                PortError.Text = "*";
                ErrorLabel.Text = "Empty Port";
            }
            else if (string.IsNullOrEmpty(DatasetBox.Text))
            {
                DatasetError.Text = "*";
                ErrorLabel.Text = "Empty Port";
            }
            else if (IpBox.Text.Split(new char[] { '.' },
StringSplitOptions.RemoveEmptyEntries).Length == 4)
            {
                IPAddress ipAddr;
```

```

        if (IPAddress.TryParse(IpBox.Text, out ipAddr))
        {
            int n;
            // IP is valid
            if (int.TryParse(PortBox.Text, out n))
            {
                ServerDataType ServerData;
                ServerData.IpAddress = IpBox.Text;
                ServerData.Port = PortBox.Text;
                ServerData.DatasetName =
DatasetBox.Text;

                link.Server = ServerData;
                this.Close();
            }
            else
            {
                PortError.Text = "*";
                ErrorLabel.Text = "Bad Port";
            }
        }
        else
        {
            // invalid IP
            IpError.Text = "*";
            ErrorLabel.Text = "Bad IP address";
        }
    }
    else
    {
        // invalid IP
        IpError.Text = "*";
        ErrorLabel.Text = "Bad IP address";
    }
}
}
}

```

ПРИЛОЖЕНИЕ А.5

Класс «XMLParser»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.Drawing;
using System.Windows.Forms;
using System.Data;

namespace SparqlClient
{
    class XMLParser
    {
        public void ParseXml(string XmlStr, DataGridView
XmlDataGridView)
        {
            try
            {
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.LoadXml(XmlStr);

                DataTable dt = new DataTable();

                dt.Columns.Add("#", typeof(string));

                // Чтение заголовков из head
                XmlNodeList headNodes =
xmlDoc.GetElementsByTagName("head");
                foreach (XmlNode headNode in headNodes)
                {
                    foreach (XmlNode variableNode in
headNode.ChildNodes)
                    {
                        string columnName =
variableNode.Attributes["name"].Value;
                        dt.Columns.Add(columnName,
typeof(string));
                    }
                }
                // Чтение данных из XML и добавление их в
DataTable
                XmlNodeList results =
xmlDoc.GetElementsByTagName("result");
                int index = 1;
                foreach (XmlNode result in results)
                {
                    DataRow row = dt.NewRow();
                    row["#"] = index++;
                }
            }
        }
    }
}
```

```

        foreach (XmlNode binding in
result.ChildNodes)
        {
            string name =
binding.Attributes["name"].Value;
            string value = binding.ChildNodes.Count
> 0 ? ExtractFragment(binding.ChildNodes[0].InnerText) :
string.Empty;

            row[name] = value;
        }

        dt.Rows.Add(row);
    }

    // Привязка DataTable к GridView
    XmlDataGridView.DataSource = dt;
    XmlDataGridView.AutoGenerateColumns = true;
}
catch (Exception ex)
{
    MessageBox.Show("Error loading XML file:
{ex.Message}", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
}

static string ExtractFragment(string uri)
{
    int index = uri.LastIndexOf('#');
    return (index != -1 && index < uri.Length - 1) ?
uri.Substring(index + 1) : uri;
}

public string ParseBool(string XmlStr)
{
    try
    {
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.LoadXml(XmlStr);
        XmlNodeList node =
xmlDoc.GetElementsByTagName("boolean");
        return node[0].InnerText;
    }
    catch (Exception ex)
    {
    }
    return "null";
}

public string ParseCount(string XmlStr)
{
    try

```

```

        {
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.LoadXml(XmlStr);
            XmlNodeList results =
xmlDoc.GetElementsByTagName("result");
            XmlNode binding = results[0].ChildNodes[0];
            string value = binding.ChildNodes.Count > 0 ?
ExtractFragment(binding.ChildNodes[0].InnerText) : string.Empty;
            return value;
        }
        catch (Exception ex)
        {
        }
        return "null";
    }

    public string ParseUpdate(string XmlStr)
    {
        try
        {
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.LoadXml(XmlStr);
            XmlNodeList results =
xmlDoc.GetElementsByTagName("body");
            XmlNode binding = results[0].ChildNodes[0];
            string value = binding.ChildNodes.Count > 0 ?
ExtractFragment(binding.ChildNodes[0].InnerText) : string.Empty;
            return value;
        }
        catch (Exception ex)
        {
            return "error";
        }
    }
}

```

ПРИЛОЖЕНИЕ В

(Обязательное)

Методы и средства интеллектуального моделирования и проектирования
систем промышленной автоматике

Презентация

МЕТОДЫ И СРЕДСТВА ИНТЕЛЛЕКТУАЛЬНОГО МОДЕЛИРОВАНИЯ И ПРОЕКТИРОВАНИЯ СИСТЕМ ПРОМЫШЛЕННОЙ АВТОМАТИКИ



Выполнил:
Студент группы 22ВВИм1
Дубинин А.В.
Руководитель:
д.т.н., профессор каф. ВТ
Дубинин В.Н.

2024

Актуальность

2

- **1. Промышленная революция (Индустрия 4.0):**
 - Четвертая промышленная революция характеризуется внедрением киберфизических систем (КФС).
 - КФС интегрируют вычислительные и физические процессы, что позволяет создавать интеллектуальные системы.
- **2. Важность разработки КФС:**
 - Необходимость разработки методов и средств для проектирования и эксплуатации КФС.
 - Создание цифровых двойников (digital twins) как неотъемлемая часть КФС для моделирования и оптимизации процессов.
- **3. Онтологическое моделирование:**
 - Онтологическое моделирование играет ключевую роль в обеспечении эффективности и качества проектирования КФС.
 - Обеспечение формализации знаний и данных, что позволяет лучше понимать и управлять сложными системами.
- **4. Преимущества онтологического подхода:**
 - Повышение совместимости и интероперабельности различных систем и компонентов.
 - Улучшение процесса обмена информацией между системами за счет использования общих понятий и терминов.

Научная новизна работы

3

- Разработка методики онтологического моделирования дискретных систем промышленности
- ленной автоматике
- Использование онтологий для создания цифровых двойников промышленных систем:
- Интеграция онтологического моделирования с языком SPARQL
- Разработка продукционных правил для моделирования динамики промышленных систем на языке SPARQL
- Создание инструментальной программной системы для поддержки онтологического моделирования:
- Формализация и стандартизация знаний о промышленных системах:
- Практическая апробация разработанных методик и инструментов на примере производственной системы FESTO

Практические результаты

4

- Создание цифровых двойников систем промышленной автоматике
- Разработка продукционных правил на языке SPARQL:
- Создание инструментальной программной системы для поддержки онтологического моделирования
- Повышение совместимости и интероперабельности промышленных систем
- Практическая апробация на примере производственной системы FESTO
- Обеспечение поддержки принятия решений на основе анализа данных и знаний
- Снижение затрат и повышение качества производства
- Интеграция онтологического моделирования с существующими системами

Цель работы

5

Целью работы является иллюстрация научно-методического подхода к онтологическому моделированию и прототипированию промышленных киберфизических систем, а также созданию «цифровых двойников» систем промышленной автоматизации на основе новой технологической платформы семантического Веб на примере производственной системы FESTO.

Производственная система FESTO 6

Производственная система *FESTO* предназначена для изготовления некоторых законченных деталей из поступающих на ее вход заготовок. Она состоит из трех станций: распределительной, тестирующей и обрабатывающей.



Онтологическое моделирование системы FESTO

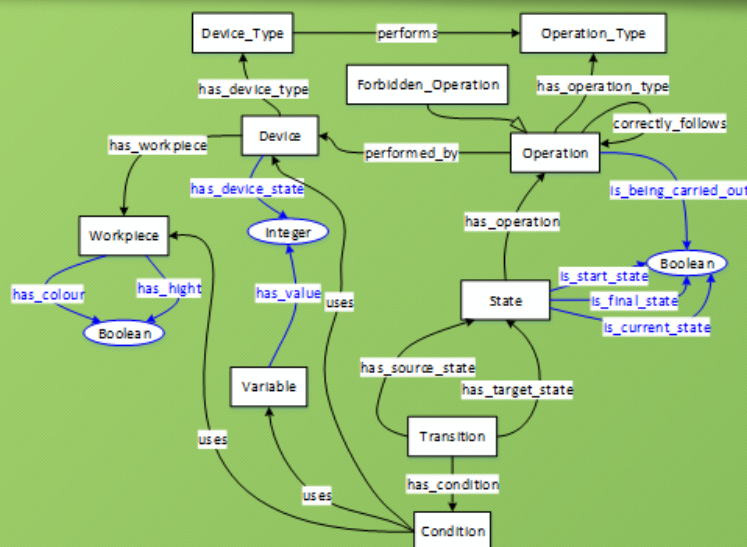
7

Была разработана общая онтология станций производственной системы FESTO, а также частные онтологии для распределительной, тестирующей и обрабатывающей станций. Это позволило формализовать знания о работе этих станций и разработать продукционные правила для моделирования их динамики.



Графическое представление обобщенной онтологии

8



Семантический анализ и продукционные правила

9

В ходе работы были разработаны семантические ограничения для анализа описания станций и продукционные правила для моделирования их динамики на языке SPARQL. Верификация системы правил подтвердила их корректность и эффективность.



Продукционные правила и SPARQL-запросы

10

Правило В4. Перенести транспортный рычаг на сторону распределительной станции

Продукционное правило В4:

ЕСЛИ выходное место содержит деталь И транспортный рычаг на стороне тестирующей станции (ТС) И вакуум выключен, ТО перенести транспортный рычаг на сторону распределительной станции (РС).



Внешний вид блока транспортного рычага

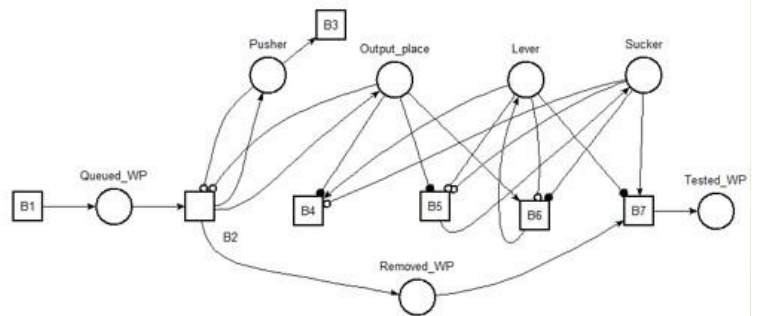
Запрос на языке SPARQL:

```
DELETE {  
  ?t :has_lever_state true}  
INSERT {  
  ?t :has_lever_state false} # перенести транспортный рычаг на  
  сторону РС  
WHERE {  
  ?o rdf:type :Output_place.  
  ?o :has_output_state true. # выходное место содержит деталь  
  ?t rdf:type :Lever.  
  ?t :has_lever_state true # транспортный рычаг на стороне ТС  
  ?s rdf:type :Sucker.  
  ?s :has_vacuum_state false} # вакуум присоски выключен
```

Для верификации системы для каждой станции была разработана модель на основе сетей Петри

Каждое правило моделируется отдельным переходом сети Петри. Условия правил представлены позициями сетевой модели, а действия - собственно переходами.

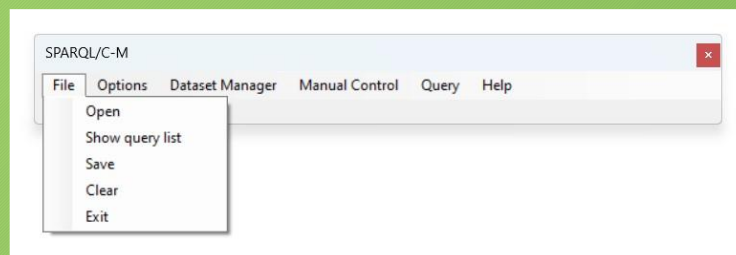
Сетевая модель системы продукционных правил, описывающих функционирование тестирующей станции



Инструментальная программная система

12

Была разработана инструментальная система SPARQL/C-M для ручного управления выполнением SPARQL UPDATE - запросов, хранящихся в формате RDF . Она включает средства автоматизации создания и проверки онтологий, интеграции с существующими системами проектирования и управления, а также поддержки принятия решений на основе анализа данных и знаний.



Назначение, особенности и функции программы

13

- 1) ввод установочной информации о SPARQL-сервере и рабочем наборе данных
- 2) управление текстовыми файлами со SPARQL Update-запросами
- 3) управление RDF-наборам данных
- 4) возможность ввода с экрана произвольного SPARQL-запроса на выборку, его выполнение на сервере и вывод на экран результатов его выполнения;
- 6) отображение статуса SPARQL Update-запросов, в частности, разрешен запрос или нет, ошибочен ли он или нет, сколько решений имеется для данного запроса;
- 7) посылка SPARQL Update-запроса и его выполнение на сервере при нажатии соответствующей кнопки;
- 8) отображение информации об успешности или не успешности выполнения SPARQL Update-запроса.

Окно «Manual Control»

14

Обеспечивает пользователю возможность взаимодействия с элементами управления для выполнения SPARQL-запросов, обновления данных на сервере и контроля над процессом выполнения запросов.

Наименование запроса	Количество разрешенных запросов	Информация (возможность посмотреть запрос)
B1	1	?
B2	0	?
B3	0	?
B5	0	?
B6	0	?
B7	0	?

Поместить в буфер деталь с параметрами «цвет X», «высота»

Цвета кнопок изменяются в зависимости от результата выполнения SPARQL-запросов или операций обновления данных на сервере.

Имитационное моделирование станций FESTO в инструментальной системе 15

Смысловая интерпретация
последовательности этих запросов
следующая:

- B1 - поместить в буфер деталь с параметрами «цвет X», «высота Y»,
- B2 - вытолкнуть деталь из буфера в выходное место,
- B3 - втянуть выталкиватель,
- B5 - включить вакуум присоски,
- B6 - перенести транспортный рычаг (с деталью) на тестирующую станцию,
- B7 - выключить вакуум присоски.

Визуализация динамики функционирования
модели распределительной станции



Заключение

16

Основным результатом представленной магистерской работы является то, что была продемонстрирована возможность прототипирования систем промышленной автоматизации и киберфизических систем (КФС) и показан путь создания их «цифровых двойников» (Digital Twins) на основе семантического Веб

