**МОДЕЛИРОВАНИЕ ПРОИЗВОДСТВЕННЫХ СИСТЕМ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА SPARQL**

*An ontological approach to modeling discrete production systems is proposed. To represent the dynamics of the system's functioning, the SPARQL language is used. The approach is illustrated using the example of testing and processing stations of the FESTO production system.*

**Введение**

Важное место в проектировании промышленных киберфизических систем (КФС), к числу которых можно отнести современные производственные системы (ПС), 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* [1]. 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 использование ранее разработанных ресурсов, имеются возможности для совместной работы и обработки данных многими субъектами и программными агентами, а также для организации распределенной обработки данных.

Центральную роль в архитектуре семантического *Web* играют онтологии. Под *онтологией* понимают формализацию некоторой области знаний с помощью концептуальной схемы [2]. Общепризнанным языком онтологий является язык *OWL [3]*. Преимущества использования онтологий в проектировании: 1) возможность точно и ясно на высоком уровне выразить семантические свойства; 2) наличие логического вывода для доказательства семантических свойств; 3) разрешимость задачи классификации и анализа; 4) возможность описывать неполные знания; 5) легкость перепроектирования системы; 6) *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].

Целью данной работы является иллюстрация научно-методического подхода к онтологическому моделированию промышленных КФС [7,8], 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* [9], точнее, две ее станции – тестирующая и обрабатывающая. Для моделирования динамики функционирования используется язык *SPARQL Update* [10].

В данной работе предлагается методика онтологического моделирования промышленных КФС, включающая: 1) разработку онтологии КФС, описывающей как физические компоненты и обрабатываемые детали, так и их состояния; 2) разработку системы продукционных правил, описывающих функционирование системы в терминах изменения состояния компонентов; 3) разработку сетевой модели системы продукционных правил для их верификации; 4) разработку *SPARQL*-запросов, реализующих продукционные правила; 5) разработку клиентского приложения для работы с онтологией КФС; 6) размещение онтологии КФС в семантическом *Web*.

**1. Производственная система фирмы FESTO Didactic**

Производственная система (ПС) *FESTO* предназначена для изготовления некоторых законченных деталей из поступающих на ее вход заготовок [11]. Она состоит из трех станций: распределительной, тестирующей и обрабатывающей. Заготовка представляет собой шайбу из пластмассы или металла. В процессе изготовления детали заготовка последовательно проходит через все три станции. Распределительная станция (РС) служит для «складирования» поступающих от внешнего производителя заготовок с последующей передачей на другие станции, тестирующая станция (ТС) – для контроля и отбраковки нестандартных заготовок, а обрабатывающая станция (ОС)– для сверления в заготовке отверстия. Обработанная заготовка может быть передана внешнему потребителю готовых деталей.

Ниже приводится краткое описание техпроцесса обработки на ТС и ОС. При поступлении детали на ТС (а именно, на тестирующее место) сенсоры-детекторы определяют цвет и материал, из которого изготовлена деталь. При ненормальном цвете или материале заготовка выталкивается (отбраковывается) из ТС. Если цвет и материал заготовки находятся в норме, то она поднимается лифтом к устройству измерения высоты детали (измерителю высоты). Если в результате измерения оказывается, что высота детали нормальная, то заготовка выталкивается в скользящий желоб, по которому она поступает на ОС. В противном случае (ненормальная высота) заготовка опускается на лифте вниз, где она выталкивается из ТС и, таким образом, исключается из дальнейшей обработки. Скользящий желоб имеет заслонку, которая позволяет заблокировать поступление детали на ОС.

На ОС заготовка поступает в лунку поворотного стола. Он провертывается вокруг своей оси на 90 градусов по часовой стрелке, перенося тем самым заготовку в позицию для сверления. Сверление осуществляется с помощью дрели, способной двигаться в вертикальной плоскости. Перед сверлением заготовка прижимается специальным прижимным стержнем (зажимом), входящим в боковую прорезь лунки. После окончания сверления поворотный стол проворачивается еще на 90 градусов по часовой стрелке, и заготовка оказывается в позиции для проверки отверстия. Проверка осуществляется при помощи специального проверочного стержня, опускаемого вниз по центру детали. Если отверстие нормального качества, то стержень проходит сквозь него и касается специального сенсора, что свидетельствует о норме. После окончания проверки отверстия стол поворачивается на одну позицию по часовой стрелке, перенося заготовку в позицию для выгрузки детали из ОС. Выгружаемая деталь передается внешнему потребителю деталей.

**2. Моделирование тестирующей станции FESTO**

На рисунке 1 слева показана ТС ПС *FESTO*, а справа – классы онтологии, которая, по сути, является ее информационной моделью.

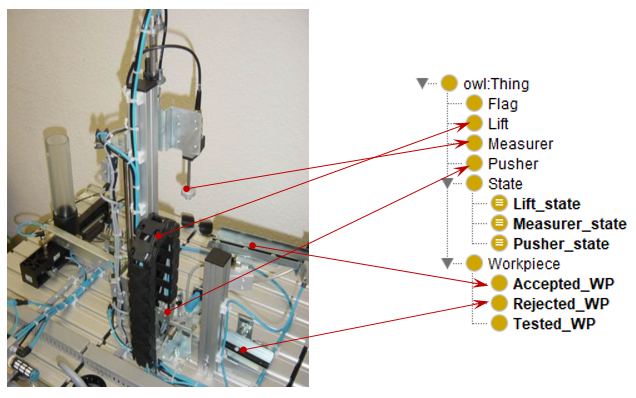
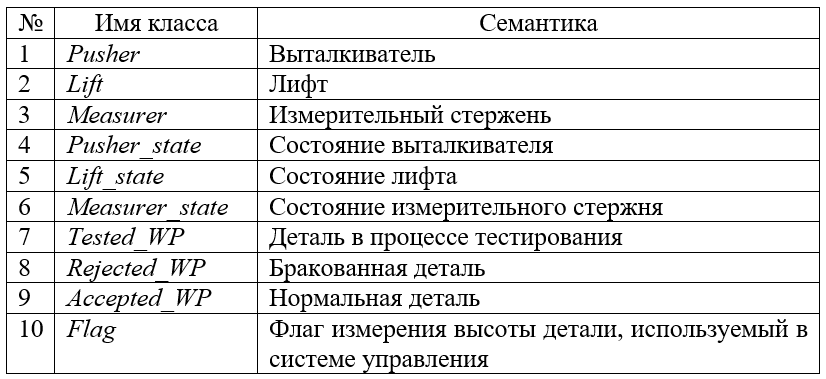


Рисунок 1 – Отображение физических объектов ТС на классы онтологии.

Классы онтологии приведены в таблице 1. Преимущественно, они представляют физические устройства станции.

Таблица 1. Классы онтологии



Объектные свойства онтологии: *has\_pusher\_state, has\_lift\_state* и *has\_measurer\_state* связывают физические устройства тестирующей станции с состояниями соответствующих устройств. Свойства по данным *has\_color* и *has\_hight* 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.

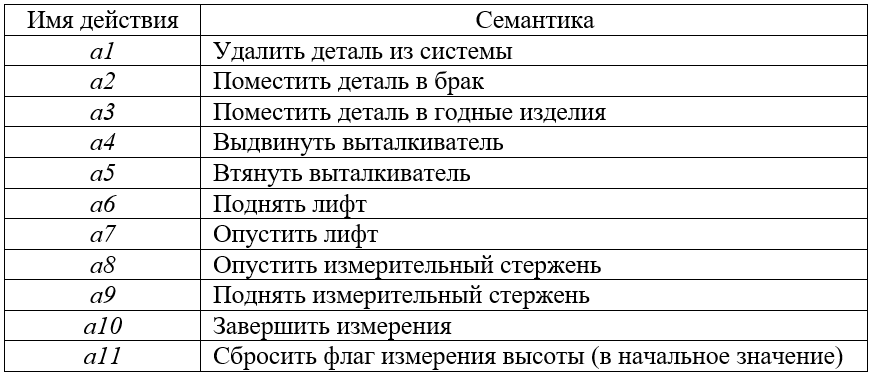
Локальные состояния (условия) в тестирующей станции представлены в таблице 2. Все состояния (условия), по сути, бинарные, то есть могут быть выражены значениями *true* и *false*. Истинное значение условия соответствует выбору первого значения из списка допустимых значений. Например, условие *с4=true* (или просто *с4*) соответствует утверждению «выталкиватель втянут», условие *с5=false* (или просто *¬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 *¬c5* (не *с5*).

В работе ТС могут быть выделены действия, приведенные в таблице 3.

Таблица 2. Локальные состояния (условия) в ТС



Таблица 3. Действия в ТС



Следует отметить, что действия могут быть выражены через новые значения состояний, которые устанавливаются в результате этого действия. Например, действие *a6* может быть выражено как *¬c5* (не *с5*).

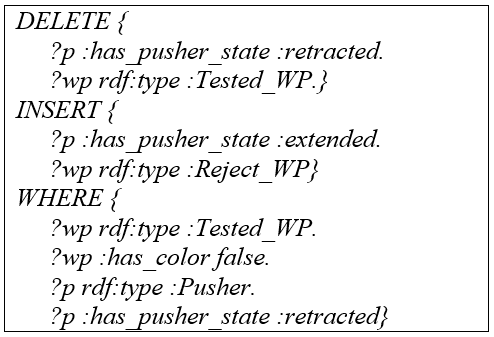
Ниже представлены правила функционирования ТС в следующем формате: сначала представлено продукционное правило в текстовом виде, затем краткая форма записи правила в терминах условий, и в конце – реализация продукционного правила в виде запроса на языке *SPARQL*.

**Правило А1. *Отбраковать деталь с негодным цветом***

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&¬c2&c4 →¬c4&¬c1&с8*.

Запрос на языке *SPARQL*:

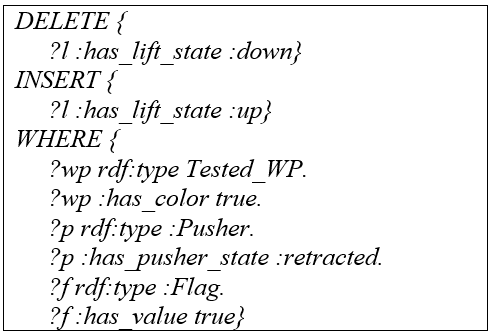


**Правило А2. *Поднять деталь для измерения***

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 → ¬c5*.

Запрос на языке *SPARQL*:



**Правило А3. *Измерить высоту детали***

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

Краткая запись правила: *¬c5&c6&c7 → ¬c6*.

Запрос на языке *SPARQL*:

|  |
| --- |
| *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 А4. *Негодная высота детали***

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

Краткая запись:*¬c3&¬c6&c7 → c6&¬c7*.

Запрос на языке *SPARQL*:

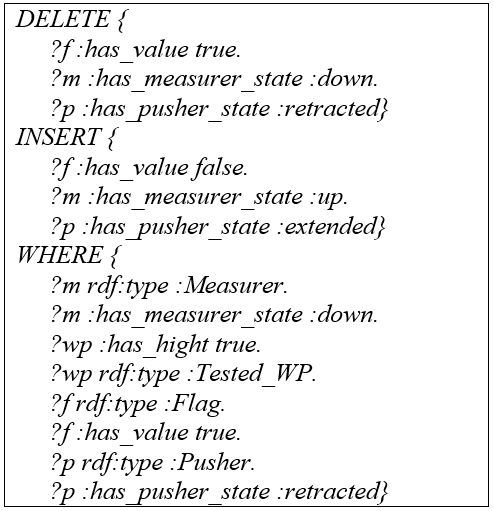
|  |
| --- |
| *DELETE {*  *?m :has\_measurer\_state :down.*  *?f :has\_value true}*  *INSERT {*  *?m :has\_measurer\_state :up.*  *?f :has\_value false}*  *WHERE {*  *?m rdf:type :Measurer.*  *?m :has\_measurer\_state :down.*  *?wp :has\_hight false.*  *?f rdf:type :Flag.*  *?f :has\_value true}* |

**Правило А5. *Нормальная высота детали***

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: *с1&c3&¬c6&c7&c4 → ¬c4&c6&¬c7*.

Запрос на языке *SPARQL*:



**Правило А6. *Опустить лифт после измерения высоты***

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

Краткая запись:*c4&¬c5&c6&¬c7 → c5*.

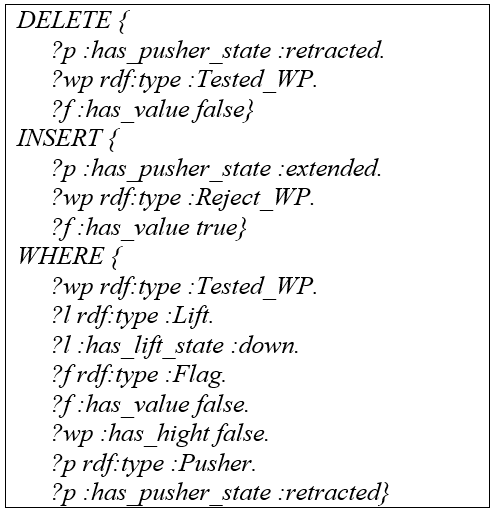
Запрос на языке *SPARQL*:

|  |
| --- |
| *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***

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.

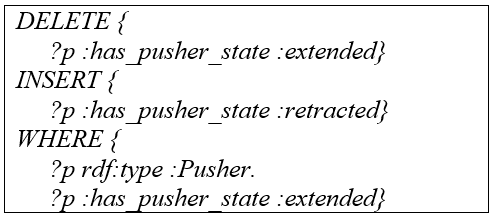
Краткая запись:*c1&¬c3&c4 &¬c7 → ¬c1&¬c4&c7&c9*. Запрос на языке *SPARQL*:



**Правило А8. *Втянуть выталкиватель***

ЕСЛИ выталкиватель выдвинут, ТО втянуть выталкиватель.

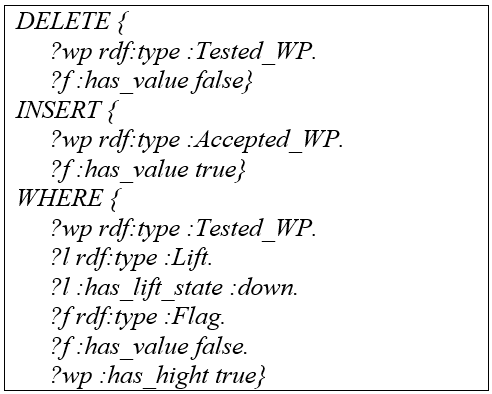
Краткая запись:*¬c4 → c4*. Запрос на языке *SPARQL*:



**Правило А9. Принять деталь с годной высотой**

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&с5&¬c7&c3→ ¬c1&c8&c7*. Запрос на языке *SPARQL*:



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 (А2, А3, А4, А6, А7, А8), третьему сценарию – последовательность (А2, А3, А5, А8, А6, А9).

Для верификации системы правил была разработана модель на основе сетей Петри (рисунок 2).

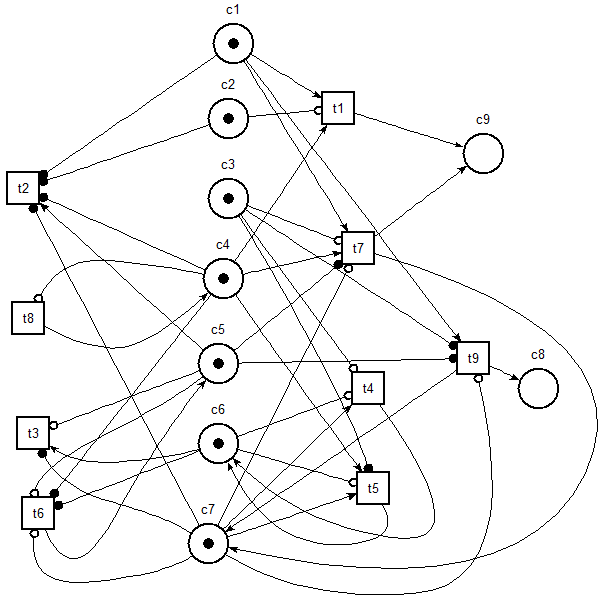


Рисунок 2 – Сетевая модель системы продукционных правил, описывающих функционирование тестирующей станции.

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 позиций сетевой модели и начальной маркировки представлена в таблице 4.

Таблица 4. Интерпретация позиций сетевой модели и начальной маркировки



Сеть Петри из рисунка 2 была промоделирована в системе *TINA* [12] на основе симуляции. Анализ модели показал правильность ее функционирования. Также было обнаружено, что в системе разрешенным может быть только один переход (иными словами, только одно правило). Данное свойство детерминированности сетевой модели значительно упрощает реализацию программной системы интерпретации правил.

**2. Моделирование обрабатывающей станции FESTO**

На рисунке 3 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.

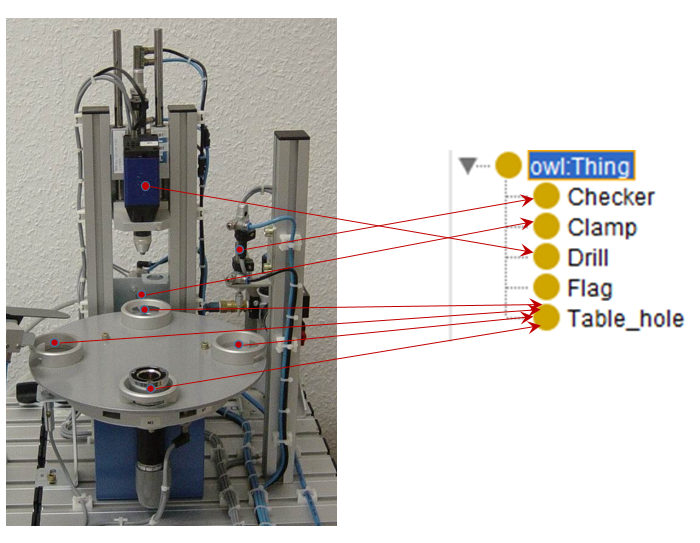


Рисунок 3 – Отображение физических объектов обрабатывающей станции (слева) на классы онтологии (справа)

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 продукционного 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: 1) лунка *L1* содержит деталь (*true*); 2) дрель находится в верхнем положении (*true*); 3) зажим отведен (*true*); 4) проверочный стержень находится в верхнем положении (*true*); 5) установлен флаг запроса поворота стола (*true*).

Ниже представлены правила функционирования обрабатывающей станции *FESTO*.

**Правило Б1. *Поворот стола после прихода новой детали***

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*:

|  |
| --- |
| *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***

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*:

|  |
| --- |
| *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***

ЕСЛИ деталь зажата, ТО включить и опустить дрель.

Ниже представлены два варианта реализующего *SPARQL*-запроса. В первом варианте используется переменная *?d*, конкретизируемая единственным экземпляром класса *Drill*, а во втором варианте используется явное указание *drill\_1 -* единственного и конкретного экземпляра класса *Drill*. Оба запроса являются эквивалентными. Небольшим преимуществом запросов первого типа является то, что нет необходимости знать имена конкретных индивидов классов.

Запрос на языке *SPARQL* (вариант 1):

|  |
| --- |
| *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}* |

Запрос на языке *SPARQL* (вариант 2):

|  |
| --- |
| *DELETE {*  *:drill\_1 :has\_drill\_state true}*  *INSERT {*  *:drill\_1 :has\_drill\_state false}*  *WHERE {*  *?c rdf:type :Clamp.*  *?c :has\_clamp\_state false}* |

**Правило Б4. *Поднять дрель, отвести зажим***

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

Запрос на языке *SPARQL*:

|  |
| --- |
| *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***

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*:

|  |
| --- |
| *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***

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*:

|  |
| --- |
| *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***

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

Запрос на языке *SPARQL*:

|  |
| --- |
| *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***

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*:

|  |
| --- |
| *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***

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*:

|  |
| --- |
| *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}* |

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 4). В отличие от сетевой модели на рисунке 2, 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 4 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.

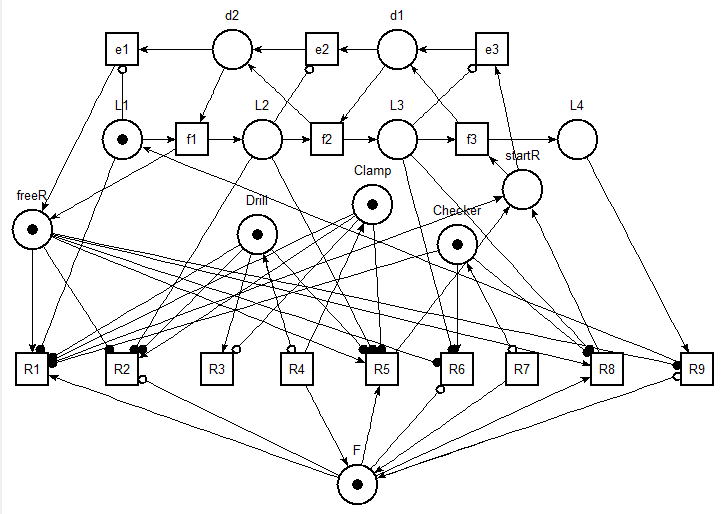


Рисунок 4 – Сетевая модель функционирования обрабатывающей станции *FESTO*

Сеть Петри, представленная на рисунке 4, была промоделирована в системе *TINA* [12]. 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*. Для вычисления *SPARQL*-запросов использовался сервер *Apache Jena Fuseki*. 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 С#.

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 А1 «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}* |

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.

1. Szeredi P., Lukácsy G., Benkő T. The Semantic Web Explained: The Technology and Mathematics behind Web 3.0. Cambridge University Press, 2014. 478 p.

2. Gruber T.R. A Translation Approach to Portable Ontology Specifications // Knowledge Acquisition, 1993. N 5(2). P. 199-220.

3. W3C Консорциум. Язык Web-онтологий OWL [Электронный ресурс]. - http://www.w3.org/2004/OWL.

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. 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.

7. 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.

8. 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. C. 4-18.

9. FESTO Didactic [Электронный ресурс]. Режим доступа: https://www.festo-didactic.com.

10. DuCharme B. Learning SPARQL: Querying and Updating with SPARQL 1.1. O'Reilly, 2013. 386 p.

11. 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.

12. The TINA toolbox Home Page - Time Petri Net Analyzer [Электронный ресурс]. Режим доступа: http://projects.laas.fr/tina/