УДК 004.89

**ONTONET: ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДЛЯ МОДЕЛИРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ СЕМАНТИЧЕСКОГО WEB**

***В. Ю. Каев, В. Н. Дубинин***

**ONTONET: MODELING TOOLS USING SEMANTIC WEB TECHNOLOGIES**

***V. I. Kaev, V. N. Dubinin***

***Аннотация.*** *Предмет.* В данной работе предлагается метод для проектирования и моделирования систем с использованием раскрашенных сетей Петри на основе онтологий и языка SPARQL. Рассматриваются вопросы построения приложений для интерпретации распределенных сетевых моделей. *Методы.* Предложенный подход основан описании области знаний с помощью онтологий, а также на применении *SPARQL*-запросов с целью модификации онтологий в ходе моделирования. *Результаты.* Была составлена онтология раскрашенных сетей Петри, описывающая статическую структуру и динамику работы моделей. Кроме того, был разработан ряд *SPARQL*-запросов, использующихся для произведения первичной инициализации, выявления комбинаций разрешенных переходов и их непосредственного выполнения. *Выводы.* Данный подход развивает идеи построения сервис-ориентированных приложений, актуализируя их в рамках концепции *Web* 3.0.

***Ключевые слова***: моделирование, онтология, раскрашенные сети Петри, семантика, *Web*, *SPARQL*.

***Abstract.*** *Subject.* This paper proposes a method for modelling and designing systems using colored Petri nets based on ontologies and the SPARQL language. The The issues of building applications for the interpretation of distributed network models are considered. *Methods.* The proposed approach is based on the description of the knowledge area using ontologies, as well as on the use of SPARQL queries in order to modify ontologies during modeling. *Results.* An ontology of colored Petri nets was compiled, describing the static structure and dynamics of the models. In addition, a number of SPARQL queries have been developed that are used to perform initial initialization, identify combinations of allowed transitions and execute them directly. *Conclusions.* This approach develops the ideas of building service-oriented applications, updating them within the framework of the concept of Web 3.0.

***Keywords***: modelling, ontology, colored Petri nets, semantics, web, SPARQL.

# Введение

Моделирование является важным этапом разработки систем, в которых понятия параллельности и событийного взаимодействия являются ключевыми [1]. Это относится и к распределенным вычислительным, управляющим и коммуникационным системам промышленного масштаба. Наиболее апробированным средством моделирования считаются раскрашенные сети Петри (РСП). Данный класс сетей Петри отличает наличие свойств, присущих высокоуровневым языкам программирования, при сравнительно компактной форме параметризуемых моделей. Для представления знаний в рамках определенной предметной области, допускающего применение логики, а также позволяющего использовать эти знания повторно, используются онтологии [2]. Они лежат в основе концепции семантического *Web*, разрабатываемой с целью достижения универсальной среды для обмена информацией во всемирной паутине. Онтологическое представление позволяет определить модель РСП в декларативном формате, что упрощает процесс интеграции с другими моделями и приложениями семантической сети. Одним из стандартов семантического web, рекомендуемых консорциумом *W3C*, является язык запросов *SPARQL* [3], использующийся для работы с данными в формате *RDF*. Он предоставляет широкие возможности по извлечению онтологических описаний в форме триплетов и манипуляции ими, что делает его пригодным для моделирования динамических систем.

# Формальное определение используемых раскрашенных сетей Петри

Рассмотрим формальное определение используемого класса сетей Петри. Оно включает в себя две составляющие: представление статической структуры и представление динамики поведения.

Статическая структура может быть представлена кортежем, представленным в (1):

, (1)

Для наглядности, примеры для формального определения будем приводить на простой модели, представленной на рисунке ?...

Основу статической структуры составляют:

* Конечное множество позиций P
* Конечное множество переходов T
* Конечное множество направленных дуг A

При этом множества P и T не пересекаются. Вместе позиции и переходы составляют множество узлов сети N. Дуги представляют собой пары узлов разного типа , задающих направление. При этом допускается наличие параллельных дуг, т.е. дуг с одинаковыми начальными и конечными узлами.

Разметка позиций, переходов и дуг производится с помощью выражений. Множество всех выражений определено как TRM.

Конечное множество наборов цветов CS. Конечное множество переменных V. Для каждой переменной определен цвет из CS.

Функция CP сопоставляет набор цветов каждой позиции из P.

Функция GD сопоставляет сторожевое условие из TRM каждому переходу из T.

Функция AN сопоставляет аннотацию из TRM каждой дуге из A.

Функция IM сопоставляет выражение начальной маркировки из TRM каждой позиции из P.

Таким образом, все определения статической структуры РСП приведены в формулах со (2) по (7)

, (2)

, (3)

, (4)

, (5)

(6)

, (7)

Еще одной немаловажной составляющей РСП являются мультимножества. Оно определяется для непустого множества базовых наборов BS и представляется как функция m от MS, сопоставляющая каждое мультимножество множеству входящих в него базовых наборов.

Базовый набор определен как функция BS над некоторым множеством S, сопоставляющая каждый базовый набор с некоторым числом из множества неотрицательных целых чисел N. При этом число называется количеством вхождений (multiplicity) элемента в базовый набор.

Мультимножества используются для представления наборов токенов, аннотаций и режимов (modes) переходов. Стоит отметить, что для мультимножеств не определены операции сложения, вычитания, сравнения, умножения на число и т.д. (в отличие от [Jensen, CPN Tools])

Таким образом, определения для мультимножеств и базовых наборов сформулированы следующим образом.

Пусть определено некоторое непустое множество S. Тогда множеством базовых наборов BS над S является функция , указывающая на число вхождений элемента в базовый набор. При этом справедливо следующее:

, (8)

, (9)

Рассмотрим определения динамики поведения РСП.

Маркировка представляет собой конечное множество MP маркировок позиций . Маркировка позиции – это функция MP, сопоставляющая позиции мультимножество токенов .

Для начальной маркировки справедливо следующее:

, (9)

...

Токенами являются элементы множества, над которым объявлено множество базовых наборов, составляющих мультимножество токенов для некоторой позиции. Токены начальной маркировки связаны с выражением начальной маркировки функцией (парой, множество вычислений) EV.

Срабатыванием (Firing) является пара маркировок: начальной и конечной. Для каждого срабатывания определено мультимножество режимов перехода .

Каждый режим перехода связан с множеством сопоставлений BD (bindings), где . В качестве выражений для сопоставлений используются переменные из множества .

Для срабатывания переходов необходимо вычисление сторожевых условий и аннотаций выходных дуг на основе сопоставлений режимов переходов. Результат вычисления сторожевого условия обозначается как , а результат вычисления аннотации дуги - .

Множество аннотаций входных дуг из p в t обозначается как , множество аннотаций выходных дуг - .

Множество токенов, выбираемых аннотацией входной дуги из позиции p, обозначается как

Переход t разрешен с режимом tm, если справедливо следующее:

, (1)

, (1)

Результирующая маркировка позиции получается путем взятия разницы между текущей маркировкой позиции и множеством извлекаемых токенов, а также присоединением множества новосозданных токенов:

, (1)

При выполнении проверок и вычислении результирующей маркировки достаточно рассматривать лишь позиции, связанные дугами с выбранным переходом.

Кроме того, сопоставления режима перехода должны иметь возможность извлекать приватизированные токены из позиций, не разделяя их с остальными сопоставлениями.

Поэтому, справедливы следующие отношения в масштабе перехода:

, (1)

Тогда новая маркировка M` получается следующим образом:

, (1)

Таким образом, определения для разрешенности и срабатывания шагов:

Шаг разрешен, если справедливо:

Когда шаг разрешен, его срабатывание порождает новую маркировку M`:

Шаг от маркировки к маркировке говорит о том, что доступна напрямую из

Понятия модулей...

Сокеты, порты, композиция модулей

2 Онтологическое моделирование РСП

Ниже рассматривается описание классов и свойств разработанной онтологии. Следует отметить, что для каждого базового класса разрабатывается так называемый «семантически правильный» класс (со знаком @ в конце имени) (иными словами, класс с семантическими ограничениями) для возможности проведения семантического анализа описаний. Суть данного метода представлена в работе [4].

Онтология РСП включает следующие базовые классы: *Place* (Позиция), *Transition* (Переход), *Arc* (Дуга), *Marking* (Маркировка), *Token* (Маркер, Метка), *Pattern* (Шаблон, Кортеж переменных), *Variable* (Переменная кортежа), *Condition* (Сторожевое условие перехода). В онтологии РСП используются следующие объектные свойства: *comes\_from, comes\_to, has\_attribute, has\_condition, has\_marking, has\_pattern, has\_token* и *has\_variable*, и свойства по данным: *has\_data, has\_index* и *has\_name*.

Класс *Place* описывает позиции РСП. Позиции содержат маркировку, причем только одну. Аксиома семантически правильного класса *Place*@ следующая:

Класс *Token* описывает маркеры (метки), содержащиеся в позиции в текущем состоянии сети. Семантически правильный класс *Marking*@ определяется аксиомой: .

Каждый маркер имеет один или более атрибутов. Семантически правильный класс *Token*@ определяется как: .

Поскольку атрибуты упорядочены в виде кортежа, каждый атрибут содержит данные и индекс. Индекс определяет позицию атрибута в кортеже. Аксиома для класса *Attribute*@:

Класс *Transition* описывает переходы РСП. Каждый переход имеет условие срабатывания (класс *Condition*), притом только одно. Аксиома семантически правильного класса: .

Условие задается в виде строки на языке SPARQL. В нем могут использоваться переменные входных дуг, константные значения и логические операторы. Это, в частности, позволяет динамически формировать запросы к онтологии для имитации функционирования РСП.

Класс *Arc* представляет дуги РСП. Каждая дуга имеет строго один шаблон. Класс *Arc*@ определяется аксиомой: .

Каждый шаблон содержит одну или более переменных. Класс *Pattern*@ определяется как: .

Переменные в шаблоне упорядочены в кортеж. Поэтому переменная содержит имя и индекс. Индекс определяет позицию переменной в кортеже. Таким образом, правильный шаблон определяется как.

Семантически правильная дуга соединяет единственный переход с единственной позицией или единственную позицию с единственным переходом:

Разработанная онтология РСП на уровне классов приведена на рисунке 1. На данном рисунке не представлены семантически правильные классы.

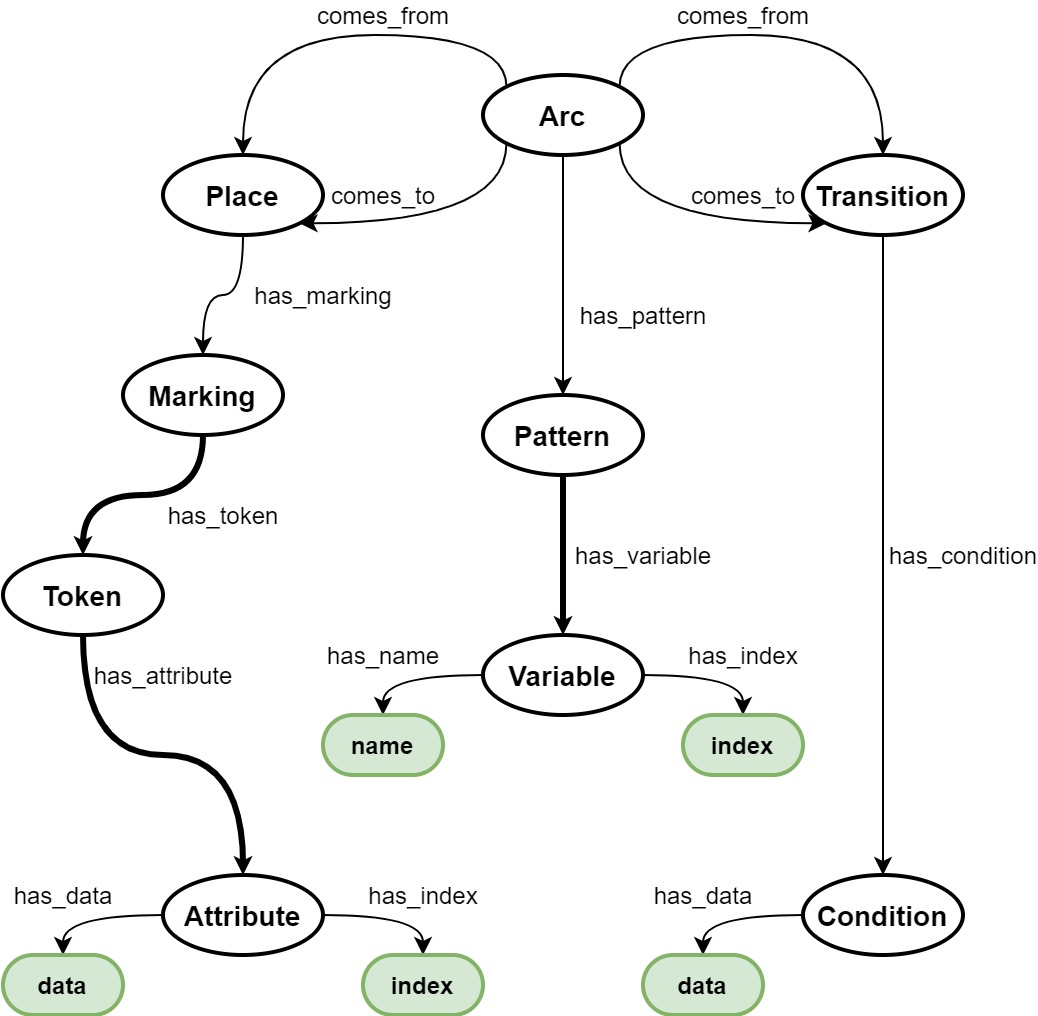


Рисунок 1 - Онтология РСП на уровне TBox

Мощность отношений в графическом представлении онтологии отображена с помощью толщины дуг графа. Классы представлены в виде овалов, а данные классов – в виде прямоугольников зеленого цвета с закругленными углами.

Создание онтологии (в формате OWL) осуществлялось в редакторе онтологий *Protégé* [5]. На рисунке 2 справа в качестве примера представлен фрагмент РСП, а слева - графическое представление онтологии этого сетевого фрагмента на уровне ABox, построенное с использованием плагина *OntoGraf*.

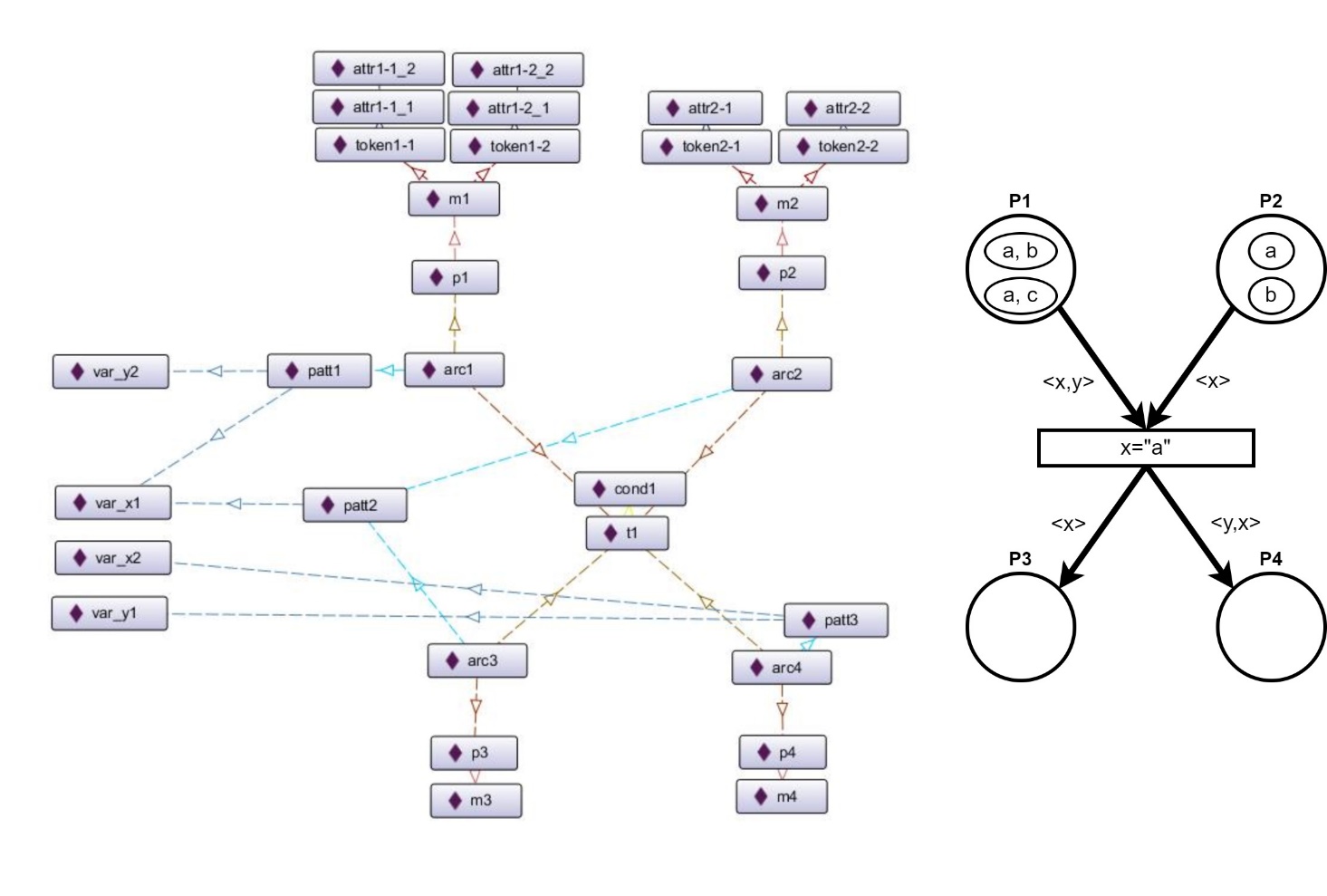


Рисунок 2 – Пример онтологии РСП на уровне ABox

3 Определение/Моделирование динамики функционирования РСП с использованием SPARQL

Сам процесс моделирования представляет собой симуляцию срабатываний переходов посредством выполнения цепочек SPARQL-запросов (транзакций) к онтологии. Фрагменты этих запросов приводятся ниже.

Первый запрос позволяет получить данные по переходам РСП.

|  |
| --- |
| PREFIX : <http://www.semanticweb.org/baker/ontologies/2020/9/OntoNet-CPN-onlotogy#>  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  SELECT ?transition ?condition\_data ?variables  WHERE {  ?transition rdf:type :Transition.  ?transition :has\_condition ?condition.  ?condition :has\_data ?condition\_data  {  SELECT ?transition (GROUP\_CONCAT(?var\_i\_name; SEPARATOR=",") as ?variables)  WHERE {{  SELECT DISTINCT ?transition ?var\_i\_name  WHERE {  ?arc\_i :comes\_to ?transition;  :has\_pattern ?patt\_i.  ?patt\_i :has\_variable ?var\_i.  ?var\_i :has\_name ?var\_i\_name.  }  }}  GROUP BY ?transition  }  } |

В результате выполнения данного запроса получаем URI переходов, их сторожевые условия на языке SPARQL и список переменных, использующихся в шаблонах входных дуг каждого перехода.

С информацией по переходам становится возможным определение факта разрешенности перехода при определенных комбинациях меток во входных позициях. Общий вид SPARQL-запроса, отвечающего за это, приведен ниже.

|  |
| --- |
| PREFIX : <http://www.semanticweb.org/baker/ontologies/2020/9/OntoNet-CPN-onlotogy#>  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  SELECT ?place\_i ?token\_i ?values <значения переменных...>  WHERE {  <выборка значений переменных...>  BIND((<сторожевое условие перехода>) as ?res)  ?arc\_i :comes\_to <URI перехода>;  :comes\_from ?place\_i;  :has\_pattern ?patt\_i.  ?place\_i :has\_marking ?marking\_i.  ?marking\_i :has\_token ?token\_i.  <выделение значений переменных, сгруппированных по меткам>  <проверка корректного сопоставления значения переменной для всех входных дуг...>  BIND((<конкатенация значений переменных в строку через запятую>) as ?values)  }  GROUP BY ?place\_i ?token\_i ?res ?values <значения переменных...>  HAVING (?res = true)  ORDER BY ?values |

Данный запрос определяет комбинации меток, при которых разрешен переход, а также осуществляет согласование переменных шаблонов входных дуг и значений атрибутов меток из входных позиций. Здесь и далее в угловые скобки заключены части запроса, формируемые динамически на основе данных из предыдущего запроса: идентификатора перехода, сторожевого условия перехода и списка переменных.

Рассмотрим составляющие этого запроса в отдельности на примере простой РСП, приведенной выше. Поиск значений осуществляем для перехода *t1*. Выборка значения и индекса переменной *x* для нее будет выглядеть следующим образом.

|  |
| --- |
| {SELECT DISTINCT ?x ?x\_index  WHERE {  ?arc\_i :comes\_to :t1;  :comes\_from ?place\_i;  :has\_pattern ?patt\_i.  ?place\_i :has\_marking ?marking\_i.  ?marking\_i :has\_token ?token\_i.  ?token\_i :has\_attribute ?attr\_i.  ?attr\_i :has\_data ?x;  :has\_index ?x\_index.  ?patt\_i :has\_variable ?var\_i.  ?var\_i :has\_name ?var\_i\_name;  :has\_index ?var\_i\_index.  FILTER(?var\_i\_name = "x" && ?x\_index = ?var\_i\_index)  }} |

В результате получаем все возможные значения переменной *x*, а также их индексы в кортежах атрибутов в метках входных позиций. Выделение известных значений и индексов переменных *x* и *y*, сгруппированных по меткам, представлено ниже.

|  |
| --- |
| {  SELECT ?token\_i ?patt\_i  (COUNT(?attr\_i) as ?attr\_i\_count) (COUNT(?var\_i) as ?var\_i\_count)  ?x ?x\_index ?y ?y\_index  WHERE {  ?patt\_i :has\_variable ?var\_i.  ?token\_i :has\_attribute ?attr\_i.  OPTIONAL {  SELECT ?token\_i ?x ?x\_index  WHERE {  ?token\_i :has\_attribute ?attr\_i.  ?attr\_i :has\_data ?x;  :has\_index ?x\_index.  }  }  FILTER(NOT EXISTS {  ?token\_i :has\_attribute ?attrx\_i.  ?attrx\_i :has\_data ?\_var;  :has\_index ?\_var\_index.  FILTER ((?\_var != ?x || ?\_var\_index != ?x\_index) && (?\_var != ?y || ?\_var\_index != ?y\_index))  })  }  GROUP BY ?token\_i ?patt\_i ?x ?x\_index ?y ?y\_index  HAVING (?attr\_i\_count = ?var\_i\_count)  } |

Данный фрагмент запроса ограничивает комбинации значений переменных по меткам, используя ранее извлеченные данные об их индексах в пределак кортежей атрибутов. Проверка корректного сопоставления значения переменной *x* для всех входных дуг приведена далее.

|  |
| --- |
| FILTER (!BOUND(?x) || NOT EXISTS {  SELECT ?place2\_i ?x (COUNT(?attr2\_i\_data) as ?x2\_count)  WHERE {  ?arc2\_i :comes\_to :t1;  :comes\_from ?place2\_i;  :has\_pattern ?patt2\_i.  ?place2\_i :has\_marking ?marking2\_i.  ?marking2\_i :has\_token ?token2\_i.  ?token2\_i :has\_attribute ?attr2\_i.  ?attr2\_i :has\_index ?attr2\_i\_index.  ?patt2\_i :has\_variable ?var2\_i.  ?var2\_i :has\_name ?var2\_i\_name;  :has\_index ?var2\_i\_index.  FILTER(?var2\_i\_name = "x" && ?attr2\_i\_index = ?var2\_i\_index)  OPTIONAL {  ?attr2\_i :has\_data ?attr2\_i\_data;  FILTER(?attr2\_i\_data = ?x)  }  }  GROUP BY ?place2\_i ?x  HAVING (?x2\_count = 0)  }) |

В данном случае проверяется факт наличия аналогичного значения переменной *x* в других входных позициях, где шаблон дуги содержит эту переменную.

Переход не разрешен, если для него не существует ни одного варианта сопоставления значений переменным шаблонов входных дуг. Общий вид SPARQL-запроса, выполняющего разрешенный переход, представлен ниже.

|  |
| --- |
| DELETE {  ?marking\_i :has\_token ?token\_i.  ?token\_i rdf:type :Token;  :has\_attribute ?token\_i\_attr.  ?token\_i\_attr rdf:type :Attribute;  :has\_data ?token\_i\_attr\_data;  :has\_index ?token\_i\_attr\_index.  }  INSERT {  ?marking\_o :has\_token ?token\_new.  <формирование меток в выходных позициях>  }  WHERE {  {  VALUES ?token\_i { <URI меток, проходящих через переход...> } # <-- :token1-1 :token2-1  ?arc\_i :comes\_to <URI перехода>;  :comes\_from ?place\_i;  :has\_pattern ?patt\_i.  ?place\_i :has\_marking ?marking\_i.  ?marking\_i :has\_token ?token\_i.  ?token\_i :has\_attribute ?token\_i\_attr.  ?token\_i\_attr :has\_data ?token\_i\_attr\_data;  :has\_index ?token\_i\_attr\_index.  } UNION {  ?arc\_o :comes\_from <URI перехода>;  :comes\_to ?place\_o;  :has\_pattern ?patt\_o.  ?place\_o :has\_marking ?marking\_o.  BIND(UUID() as ?token\_new)  <выборка атрибутов по шаблонам выходных дуг...>  }  } |

Данный запрос удаляет метки, определенные в ходе выполнения предыдущего запроса, из выходных позиций, и формирует метки в выходных позициях на основе шаблонов выходных дуг. Идентификаторы новых меток формируются с помощью функции UUID.

Аналогично, рассмотрим составляющие этого запроса. Фрагмент запроса для формирования данных по новым меткам выходных позиций при наличии сопоставленных переменным *x* и *y* значений (“a” для *x* и “b” для *y*) приведен ниже:

|  |
| --- |
| ?token\_new rdf:type :Token;  :has\_attribute ?x;  :has\_attribute ?y.  ?x rdf:type :Attribute;  :has\_data "a";  :has\_index ?x\_index.  ?y rdf:type :Attribute;  :has\_data "b";  :has\_index ?y\_index. |

Здесь в онтологию вставляются (insert) данные о новых метках и их атрибутах. Выборка атрибутов для переменной *x* производится следующим образом.

|  |
| --- |
| OPTIONAL {  SELECT ?patt\_o ?x ?x\_index  WHERE {  ?patt\_o :has\_variable ?var\_o.  ?var\_o :has\_name ?var\_o\_name;  :has\_index ?x\_index.  FILTER(?var\_o\_name = "x")  BIND(UUID() as ?x)  }  } |

В данном фрагменте, исходя из шаблонов выходных дуг, выбираются сформированные идентификаторы атрибутов, а также индексы для будущих кортежей в составе меток.

Для получения маркировки РСП на текущем шаге моделирования используется следующий SPARQL-запрос.

|  |
| --- |
| SELECT ?place ?token (GROUP\_CONCAT(?data;SEPARATOR=",") AS ?token\_data)  WHERE {  ?place rdf:type :Place.  ?place :has\_marking ?marking.  ?marking :has\_token ?token.  OPTIONAL {  ?token :has\_attribute ?attr.  {SELECT ?attr ?data  WHERE {  ?attr :has\_data ?data.  ?attr :has\_index ?index.  } ORDER BY ?index  }}  }  GROUP BY ?place ?token  ORDER BY ?place |

Он позволяет получить позиции РСП и метки, содержащиеся в них.

4 Распределенная интерпретация РСП: проблемы и решения

Что подразумевает распределенная структура системы моделирования? Есть 3 варианта:

1) распределенное хранение данных (ресурсов);

2) распределенная обработка федеративных SPARQL-запросов (SERVICE);

3) распределение узлов иерархических РСП

Какой вариант распределенной структуры выбрать и почему? Наиболее интересен 3 вариант, реализующий иерархические РСП в сети. Потенциальная польза от него больше:

+ позволяет реализовывать микросервисы из иерархических моделей

+ дает возможность вести разработку узлов параллельно

Что насчет федеративных запросов? В приоритете такая структура, узлы которой имеют URI лишь связанных с ними других узлов. В таком случае, сбор информации возможен с помощью федеративных запросов или иных протоколов взаимодействия между узлами

Что из себя представляет узел иерархической РСП? Узел является автономной ABox онтологией, имеющей входные и выходные узлы (позиции).

Узел используется вместо перехода.

Данные позиции инициализируются при загрузке онтологии на сервер адресами URL.

Передача данных между узлами осуществляется по протоколу UDP, т.к. подтверждение получения в нем не требуется

Что насчет TBox онтологии (core)? Онтологию можно вынести на отдельный сервер по типу https://www.w3.org/2002/07/owl

Если допустить возможность хранения собственных экземпляров TBox, то функциональные возможности благодаря обратной совместимости разных версий онтологии будут ограничены лишь используемой версией

Как представляются оконечные узлы (позиции) в узле иерархии? В узле объявляются экземпляры Port, связанные с определенными позициями. Для них определяется тип (input/output) и URI(URL) связанного узла

Где хранится информация о срабатываниях firings? Экземпляры firing хранятся на каждом узле. Они создаются при срабатывании переходов в узле или при внешних входных воздействиях

Где хранится информация об ощих term’ах, вроде констант? Эта информация могла бы храниться в корневой (core) онтологии на отдельном сервере

Как можно реализовать временные РСП в рамках распределенной структуры? Требуется синхронизация глобального времени для узлов распределенной РСП.

Возможно, это реализуется федеративными запросами по узлам с целью определения ближайшего времени перехода

Как синхронизировать глобальное время между узлами иерархической РСП? Есть вариант с рассылкой запросов от узла к соседним узлам. В запрос включается хэш, а также с ростом глубины запроса пополняется список URL узлов, пробрасывающих эти запросы по сети. Окончанием синхронизации считается получение ответов от всех соседних узлов, которым этот запрос был адресован.

Также, возможен федеративный запрос, извлекающий в ходе выполнения новые узлы сети.

Какие есть виды взаимодействий узлов иерархической сети? Каждый узел отвечает за срабатывание только своих переходов.

При формировании меток в позиции, являющейся еще и выходным портом, узел пересылает связанному соседнему узлу SPARQL-запрос на формирование, который аналогичен выполненному внутри самого узла.

Все функционирование сводится к передаче меток через порты. Решение о выполнении перехода осуществляется агентами

Как исполняется иерархическая распределенная СП? 2 варианта:

- комплексное выполнение переходов через глобальные запросы

- сконцентрированное выполнение переходов в рамках узлов

Второй вариант является приоритетным

Какое применение у реализуемой иерархической РСП? Выполнять моделирование с использованием РСП в рамках сети

Как выполняется моделирование в рамках узлов? Каждый узел содержит фрагмент сети и отвечает за логический вывод (ризонинг).

Сервер предоставляет агентам доступ (GET) к состоянию фрагмента сети, а также интерфейс (REST? SPARQL) для выполнения разрешенных переходов в рамках фрагмента сети.

Также он взаимодействует со связанными узлами для передачи токенов

Какие существуют подходы к организации подключения модулей/узлов? - попарное подключение 1 к 1

- разделение общего ресурса

В чем плюсы и минусы попарного подключения? + можно реализовать подписку на события

+ в связке всех со всеми реализуется общий ресурс

+ упрощенная синхронизация и процесс подключения

+ разделение портов на In, Out и I/O

В чем плюсы и минусы организации контактной позиции как общего ресурса? - все порты являются двухсторонними I/O

- подключение всех со всеми

В чем заключается отличие реализуемой системы от классических иерархических СП? Каждый узел сети является модулем. Допускается связь 1 – много для модулей.

Как решается проблема синхронизации при выполнении переходов в узлах, имеющих общие порты? На узлах генерируется уникальная последовательность (хеш), с помощью которой резервируются локальные порты, используемые при переходе.

Выполняется попытка занятия используемых портов на удаленных узлах. Хеш при этом используется для определения приоритета узла.

Если заняты удаленные порты, занимаются и локальные порты, после чего выполняется переход и порты освобождаются.

В чем плюсы и минусы такой синхронизации? + хорошо работает для иерархических сетей (1 к 1: дочерний модуль к родительскому), позволяя определить и выполнить приоритетный переход

- не решает проблему коллизий при множественных связях между модулями (1 порт используется 2+ другими модулями), что приводит к отменам выполнения переходов на всех взаимодействующих модулях

5 Архитектура OntoNet

Реализуется серверное приложение с интерфейсом командной строки, структура которого показана на рисунке 1. Оно состоит из следующих модулей: интерфейса командной строки (CLI), веб-сервера, движка (Engine), механизма логического вывода (Reasoner) и связующего ядра (Core).

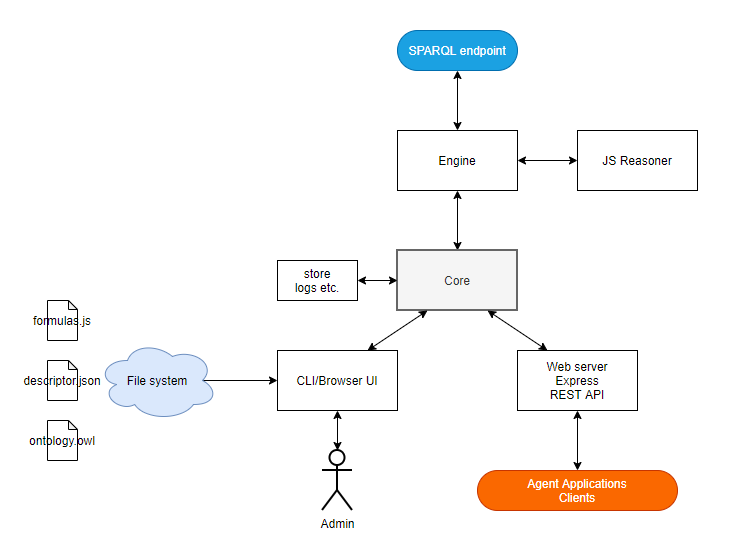


Рисунок 1 - Структура разрабатываемого инструмента

Администратор взаимодействует с приложением через терминал, который позволяет конфигурировать систему, задавая URL адрес SPARQL-сервера, подгружая на сервер файлы онтологии, а также файл с JS-реализацией алгебры РСП. Веб-сервер принимает запросы от агентов и перенаправляет их на SPARQL-сервер, после чего дает команду движку. Движок взаимодействует непосредственно с оконечной точкой SPARQL, выполняя при необходимости с помощью ризонера логический вывод новых концепций.

6 Использование OntoNet для проектирования, прототипирования, моделирования и реализации распределенных систем обработки и управления

Выводы/Заключение

В результате выполнения данного курсового проекта было разработано веб приложение, позволяющее получить начальные маркировки позиций и мультимножества аннотаций дуг на основе онтологического представления РСП. Полученный цифровой продукт может использоваться в целях обучения или исследования.

Направлением дальнейшей работы является расширение возможностей логического вывода и распределенная интерпретация РСП в сети.

Направлением дальнейших исследований является введение модульности и иерархии в «онтологические» РСП, развитие инструментальной системы OntoNet для возможности реализации на ее основе распределенных вычислительных и управляющих систем в контексте семантического web. Перспективным является использование мультимодельного подхода к построению и реализации сложных систем (например, информационных моделей киберфизических систем), когда одна часть системы представляется с использованием одной модели (например, тех же РСП), другая часть – с использованием другой модели (например, функциональных блоков стандарта IEC 61499), и эти модели интегрируются на базе онтологий.

Список литературы/использованных источников/Sources, без ГОСТов, в порядке [упоминания]

***Библиографический список***

1. Jensen K., Kristensen L.M. Coloured Petri Nets. Modelling and Validation of Concurrent Systems // Springer-Verlag Berlin Heidelberg, 2009.

2. Jacksi K. Introduction to Semantic Web. // University of Zakho Department of Computer Science March, 2014.

3. Semantic Web. W3C Consortium. [Электронный ресурс]. – Режим доступа: https://www.w3.org/standards/semanticweb/. – Дата доступа: 11.01.2021.

4. Gašević, D. Interoperable Petri Net Models via Ontology / D. Gašević, V. Devedžić // International Journal of Web Engineering and Technology. – Vol. 3, N 4, 2007. – P. 374-396.

5. Vidal J.C., Lama M., Bugarín A. OPENET: Ontology-based engine for high-level Petri nets // Expert Systems with Applications. – 2010. – N 37. – P. 6493–6509.

6. Protege 5 Documentation [Электронный ресурс]. – Режим доступа: https://protegeproject.github.io/protege/. – Дата доступа: 06.01.2021.

7. Apache Jena Fuseki [Электронный ресурс]. – Режим доступа: https://jena.apache.org/documentation/fuseki2/. – Дата доступа: 24.12.2020.

***References***

1. Jensen K., Kristensen L.M. Coloured Petri Nets. Modelling and Validation of Concurrent Systems // Springer-Verlag Berlin Heidelberg, 2009.

2. Jacksi K. Introduction to Semantic Web. // University of Zakho Department of Computer Science March, 2014.

3. Semantic Web. W3C Consortium. [Электронный ресурс]. – Режим доступа: https://www.w3.org/standards/semanticweb/. – Дата доступа: 11.01.2021.

4. Gašević, D. Interoperable Petri Net Models via Ontology / D. Gašević, V. Devedžić // International Journal of Web Engineering and Technology. – Vol. 3, N 4, 2007. – P. 374-396.

5. Vidal J.C., Lama M., Bugarín A. OPENET: Ontology-based engine for high-level Petri nets // Expert Systems with Applications. – 2010. – N 37. – P. 6493–6509.

6. Protege 5 Documentation [Электронный ресурс]. – Режим доступа: https://protegeproject.github.io/protege/. – Дата доступа: 06.01.2021.

7. Apache Jena Fuseki [Электронный ресурс]. – Режим доступа: https://jena.apache.org/documentation/fuseki2/. – Дата доступа: 24.12.2020.