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. Входные дуги направлены от позиции к переходу, а выходные – наоборот. При этом допускается наличие параллельных дуг, т.е. дуг с совпадающими начальными и конечными узлами.

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

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

Функция CL сопоставляет набор цветов каждой позиции из P и каждой переменной из V. 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. Они представляют из себя подмножества непустого множества базовых наборов BS и обозначаются как .

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

Мультимножества используются для представления наборов токенов, аннотаций и режимов переходов, рассматриваемых далее. Стоит отметить, что для мультимножеств не определены операции сложения, вычитания, сравнения, умножения на число и т.д.

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.

Маркировкой называется подмножество непустого множества маркировок позиций MP. Маркировка позиции – это пара , где .

Токенами являются элементы множества, над которым объявлено множество базовых наборов, составляющих мультимножество токенов для некоторой позиции .

Вычислением называется функция EV, сопоставляющая значения выражениям из TRM. Токены начальной маркировки являются результатом вычисления выражения начальной маркировки:

(10)

Начальная маркировка содержит маркировки позиций только с мультимножествами токенов, являющихся результатом вычисления инициализирующего выражения каждой позиции.

Срабатыванием является пара маркировок: до срабатывания и после. Для каждого срабатывания определено мультимножество режимов перехода .

Множество сопоставлений для режима перехода обозначается как . Сопоставление представляет из себя пару , где

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 - .

Сопоставление получается на основе аннотаций входных дуг. При этом множество токенов, выбираемых сопоставлениями из входных позиций, обозначаются как . Множество токенов, переходящих в выходные позиции, обозначается как .

Пусть S – непустое множество токенов позиций в текущей маркировке. Тогда переход t разрешен с режимом tm, если справедливо следующее:

, (11)

, (12)

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

, (13)

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

В масштабе перехода сопоставления должны иметь возможность извлекать только приватизированные токены из позиций, не разделяя их с остальными сопоставлениями. Поэтому, для разрешенного перехода справедливо:

, (14)

Тогда новая маркировка M` получается следующим образом:

, (15)

Сокетами называются пары позиций (или портов) (p1, p2), использующиеся для организации связей между двумя модулями РСП. Сокеты бывают входными (I), выходными (O) и двухсторонними (I/O).

, (16)

Токены, переходящие в выходные (O) и входно-выходные (I/O) позиции, автоматически переходят в связанные через сокеты входные (I) и входно-выходные (I/O) позиции.

, (17)

# Онтологическое моделирование РСП

Онтология РСП включает следующие базовые классы: *CPN* (РСП), *Node* (Узел сети), *Arc* (Дуга), *Place* (Позиция), *Transition* (Переход), *Term* (Выражение), *ColorSet* (Набор цветов), *Variable* (Переменная), *Function* (Функция), *Constant* (Константа), *Multiset* (Мультимножество), *BasisSet* (Базовый набор), *Data* (Значение), *Marking* (Маркировка), *MarkingOfPlace* (Маркировка позиции), *Evaluation* (Вычисление), *Firing* (Срабатывание), *TransitionMode* (Режим перехода), *Binding* (Сопоставление), *Port* (Порт). В онтологии РСП используются следующие объектные свойства: *has\_annotation*, *has\_annotationChunk*, *has\_arc*, *has\_basisSet*, *has\_binding*, *has\_code*, *has\_colorSet*, *has\_data*, *has\_domain*, *has\_firing*, *has\_function*, *has\_guard*, *has\_initialMarking*, *has\_initialTokens*, *has\_marking*, *has\_multisetOfTerms*, *has\_multisetOfTokens*, *has\_multisetOfTransitionModes*, *has\_node*, *has\_place*, *has\_range*, *has\_sourceMarking*, *has\_sourceNode*, *has\_targetMarking*, *has\_targetNode*, *has\_term*, *has\_variable*, *has\_token*, *has\_transition* и *includes\_markingOfPlace*, и свойства по данным: *connected\_to*, *has\_multiplicity*, *has\_name*, *has\_type*, *has\_value* и *reserved\_with*.

Онтология на уровне классов приведена на рисунке 1.



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.

Ниже рассматривается описание классов и свойств разработанной онтологии. Предложенные аксиомы накладывают семантические ограничения на мощность и диапазон отношений, что позволяет проводить семантический анализ описаний.

Класс CPN описывает модель раскрашенной сети Петри. Он определяется следующей аксиомой: *CPN and (has\_initialMarking only Marking) and (has\_initialMarking max 1 Marking) and (has\_arc only Arc) and (has\_colorSet only ColorSet) and (has\_firing only Firing) and (has\_function only Function) and (has\_marking only Marking) and (has\_node only Node)*

Класс Node описывает узлы РСП и определяется следующей аксиомой: *Node and (has\_name only xsd:string) and (has\_name exactly 1 xsd:string)*

Класс Arc представляет дуги в сети. Его аксиома предложена далее: *Arc and (has\_annotation only Term) and (has\_annotation exactly 1 Term) and (has\_multisetOfTerms only Multiset) and (has\_multisetOfTerms max 1 Multiset) and (has\_sourceNode only Node) and (has\_sourceNode exactly 1 Node) and (has\_targetNode only Node) and (has\_targetNode exactly 1 Node)*

Класс Place описывает позиции РСП. Он является подклассом Node и может использоваться для определения дуг. Аксиома класса Place выглядит следующим образом: *Place and Node and (has\_colorSet only ColorSet) and (has\_colorSet exactly 1 ColorSet) and (has\_initialTokens only Term) and (has\_initialTokens exactly 1 Term)*

Класс Transition описывает переходы РСП. Аналогично, он является подклассом Node и определяется аксиомой: *Transition and Node and (has\_code only Term) and (has\_code max 1 Term) (has\_guard only Term) and (has\_guard exactly 1 Term)*

Класс Term описывает выражения на языке разметки. Аксиома для данного класса: *Term and (has\_value only xsd:string) and (has\_value exactly 1 xsd:string)*

Класс ColorSet описывает набор цветов РСП. Аксиома, определяющая данный класс: *ColorSet and (has\_name only xsd:string) and (has\_name exactly 1 xsd:string)*

Класс Variable описывает переменные в сети и определяется аксиомой: *Variable and Term and (has\_colorSet only ColorSet) and (has\_colorSet exactly 1 ColorSet)*

Класс Function описывает функции модели. Определяющая данный класс аксиома: *Function and (has\_name only xsd:string) and (has\_name exactly 1 xsd:string) and (has\_range only ColorSet) and (has\_range exactly 1 ColorSet) and (has\_domain only ColorSet)*

Класс Constant описывает константы модели. Константы являются функциями без параметров, поэтому считаются подклассом Function. Аксиома класса предложена далее: *Constant and Function and (has\_domain exactly 0 ColorSet)*

Класс Multiset описывает мультимножества РСП. Аксиома для данного класса: *Multiset and (has\_basisSet only BasisSet)*

Класс BasisSet описывает базовые наборы РСП. Аксиома для него приведена далее: *BasisSet and (has\_data only Data) and (has\_data exactly 1 Data) and (has\_multiplicity only xsd:integer) and (has\_multiplicity exactly 1 xsd:integer)*

Класс Data описывает значения, вычисляемые в модели. Он определяется аксиомой: *Data and (has\_value only xsd:string) and (has\_value exactly 1 xsd:string)*

Класс Marking описывает маркировки сети. Его аксиома следующая: *Marking and (includes\_markingOfPlace only MarkingOfPlace)*

Класс MarkingOfPlace описывает маркировки позиций. Определение данного класса в виде аксиомы следующее: *MarkingOfPlace and (has\_multisetOfTokens only Multiset) and (has\_multisetOfTokens exactly 1 Multiset) and (has\_place only Place) and (has\_place exactly 1 Place)*

Класс Evaluation описывает вычисление выражения. Он определяется аксиомой следующего вида: *Evaluation and (has\_data only Data) and (has\_data exactly 1 Data) and (has\_term only Term) and (has\_term exactly 1 Term)*

Класс Firing описывает срабатывание перехода в сети. Представление данного класса в виде аксиомы приведено далее: *Firing and (has\_multisetOfTransitionModes only Multiset) and (has\_multisetOfTransitionModes exactly 1 Multiset) and (has\_sourceMarking only Marking) and (has\_sourceMarking exactly 1 Marking) and (has\_targetMarking only Marking) and (has\_targetMarking exactly 1 Marking)*

Класс TransitionMode описывает режим перехода. Он является подклассом Data, поскольку также является значением для базового набора и используется при формировании мультимножеств режимов перехода. Определяющая аксиома для данного класса: *TransitionMode and Data and (has\_transition only Transition) and (has\_transition exactly 1 Transition) and (has\_binding only Binding)*

Класс Binding описывает сопоставление переменных и их значений. Он является частным случаем и, как следствие, подклассом Evaluation. Его отличие заключается в наличии дополнительных отношений выборки базовых наборов. Данный класс определяется следующей аксиомой: *Binding and Evaluation and (has\_annotationChunk only BasisSet) and (has\_annotationChunk exactly 1 BasisSet) and (has\_token only BasisSet) and (has\_token exactly 1 BasisSet) and (has\_variable only Variable) and (has\_variable exactly 1 Variable)*

Класс Port описывает порт для межсетевого взаимодействия. Он определяется аксиомой: *Port and (has\_place only Place) and (has\_place exactly 1 Place) and (has\_type only {"I/O" , "In" , "Out"}) and (has\_type exactly 1 xsd:string) and (reserved\_with only xsd:string) and (reserved\_with max 1 xsd:string) and (connected\_to only xsd:string)*

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

# Моделирование динамики функционирования РСП с использованием SPARQL

Сам процесс моделирования представляет собой симуляцию срабатываний переходов посредством выполнения цепочек SPARQL-запросов (транзакций) к онтологии. Фрагменты и описание этих запросов приводятся ниже. Стоит отметить, что повторяющиеся и незначительные части запросов заменены поясняющими комментариями в угоду краткости.

Алгоритм выполнения запросов в виде блок-схемы представлен на рисунке ?...

Первый запрос используется для инициализации системы. Он позволяет получить все статичные составляющие сети, а именно: идентификаторы переходов, их сторожевые условия и участки кода, идентификаторы дуг, их тип и выражение аннотации, идентификаторы позиций и выражения их начальных маркировок. Фрагмент данного запроса представлен далее:

|  |
| --- |
| PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  PREFIX t: <http://www.onto.net/core/>  PREFIX a: <http://www.onto.net/abox/heads-and-tails/>  SELECT ?type ?id ?guard ?code ?arc ?arc\_type ?place ?term ?term\_value ?value  FROM <http://localhost:3030/ontonet/data/tbox>  FROM <http://localhost:3030/ontonet/data/abox>  WHERE {  ?cpn rdf:type t:CPN.  {  ?cpn t:has\_node ?transition.  ?transition rdf:type t:Transition.  BIND (?transition as ?id)  BIND ("transitions" as ?type)  {  # выборка сторожевого условия и блока кода  }  UNION  {  # выборка входных и выходных дуг  }  }  UNION  {  {  # выборка выражений аннотаций дуг  }  UNION  {  # выборка выражений начальных маркировок позиций  }  BIND (?arc\_or\_place as ?id)  OPTIONAL {  # выборка вычисленных выражений  }  }  }  ORDER BY ?type ?arc |

В дальнейших фрагментах запросов префиксы и графы аналогичные, поэтому будут опущены.

Запрос для получения последней проициализированной маркировки используется для определения необходимости произведения инициализации. Он позволяет получить информацию по мультимножествам токенов последней маркировки. Его фрагмент приведен ниже:

|  |
| --- |
| SELECT DISTINCT ?id ?basis\_set ?value ?multiplicity  WHERE {  ?cpn rdf:type t:CPN;  t:has\_marking ?marking.  #last marking  FILTER NOT EXISTS {  ?firing rdf:type t:Firing;  t:has\_sourceMarking ?marking.  }  ?marking t:includes\_markingOfPlace ?mop.  ?mop t:has\_place ?place;  t:has\_multisetOfTokens ?multiset.  OPTIONAL {  # получение проинициализированных данных  }  FILTER (BOUND(?basis\_set))  BIND(?place as ?id)  } |

Запрос для произведения первичной инициализации маркировки применяется движком после вычисления требуемых выражений и получения мультимножеств токенов и частей аннотаций.

|  |
| --- |
| WITH <http://localhost:3030/ontonet/data/abox>  INSERT {  # добавление начальной маркировки, маркировок позиций и мультимножеств вычисленных токенов  }  WHERE {  GRAPH <http://localhost:3030/ontonet/data/tbox> {  ?cpn rdf:type t:CPN.  }  BIND(IRI(CONCAT(STR(a:), "marking\_", STRUUID())) as ?initial\_marking)  {  # привязка UUID позиции  # формирование UUID маркировок позиций и мультимножеств токенов  {  # формирование базового набора токенов, количества вхождений и значений токенов  UNION  {  # аналогично для другого базового набора токенов...  }  BIND(IRI(CONCAT(STR(a:), "eval\_", STRUUID())) as ?eval)  BIND(a:P1\_initial-tokens-term as ?initial\_tokens\_term)  BIND(IRI(CONCAT(STR(a:), "data\_", STRUUID())) as ?eval\_data)  BIND("{\"prop\": \"JSON-string\"}" as ?eval\_data\_value)  }  UNION  {  # аналогично для другой позиции...  }  };  WITH <http://localhost:3030/ontonet/data/abox>  INSERT {  # добавление мультимножеств аннотаций дуг  }  WHERE {  {  # привязка UUID дуги  BIND(IRI(CONCAT(STR(a:), "mul\_", STRUUID())) as ?multiset)  {  # формирование базового набора выражений аннотации, кол-ва вхождений и значений шаблонов  }  UNION  {  # аналогично для другого базового набора выражений аннотации...  }  # формирование UUID вычислений для каждого выражения  }  UNION  {  # аналогично для другой дуги...  }  } |

Запрос для добавления в сеть всех возможных сопоставлений переменных и значений используется с целью последующей фильтрации этих значений. Это увеличивает гибкость системы, поскольку основной упор остается на формировании немногословных SPARQL-запросах.

|  |
| --- |
| INSERT {  # добавление сопоставления переменной и значения  }  WHERE {  {  # формирование UUID и значений для сопоставления, переменной и значения  }  UNION  {  # аналогично для другого сопоставления...  }  } |

Данные, полученные в ходе запроса свободных сопоставлений, применяются для формирования следующего запроса.

|  |
| --- |
| SELECT ?transition ?annotation\_bs ?token\_bs (?binding as ?id)  WHERE {  # получение сопоставления по связанной аннотации дуги  FILTER NOT EXISTS {  # проверка наличия режимов перехода  }  } |

Запрос для формирования потенциальных режимов перехода используется для агрегации сопоставлений с целью последующей фильтрации.

|  |
| --- |
| INSERT {  # добавление режимов перехода и их сопоставлений  }  WHERE {  # transition level  {  # привязка UUID перехода    # на уровне базового набора аннотации  {  VALUES (<?переменные...>) { (<id переменных...>) (t:NULL...) } # в рамках одного токена  # аналогично для остальных токенов...  # для комбинации выбирается хотя бы один токен  FILTER (?b1\_x != t:NULL || ?b2\_x != t:NULL)  }  {  # аналогично для других базовых наборов аннотации...  }  OPTIONAL {  BIND(IRI(CONCAT(STR(a:), "tm\_", STRUUID())) as ?transition\_mode)  }  # далее построчное разбиение сопоставлений  {  VALUES <?переменная> { <id переменной> }  BIND (<?переменная> as ?binding)  } UNION {  # аналогично для остальных переменных комбинации...  }  }  UNION  {  # аналогично для другого перехода...  }  } |

Запрос для исключения режимов перехода, для которых переход не разрешен, осуществляет фильтрацию режимов перехода по двум признакам. Во-первых, значения переменных должны быть непротиворечивы в рамках одного режима перехода. Во-вторых, аннотации входных дуг должны быть способны извлечь зарезервированные токены из позиций, не допуская конфликтов.

|  |
| --- |
| DELETE {  # удаление режима перехода  }  WHERE {  {  SELECT DISTINCT ?transition\_mode  WHERE {  {  ?transition\_mode rdf:type t:TransitionMode.  FILTER EXISTS {  ?transition\_mode rdf:type t:TransitionMode;  t:has\_binding ?binding1;  t:has\_binding ?binding2.  FILTER (?binding1 != ?binding2)  # проверка несовпадения значений одной переменной в режиме перехода  FILTER (?value1 != ?value2)  }  }  UNION  {  # недостаточное количество токенов в позициях  {  SELECT ?transition\_mode ((?n + SUM(?diff)) as ?res)  WHERE {  {  SELECT DISTINCT ?transition\_mode ?anno\_chunk\_bs ?token\_bs  WHERE {  # выборка...  }  }  ?anno\_chunk\_bs t:has\_multiplicity ?anno\_chunk\_multiplicity.  ?token\_bs t:has\_multiplicity ?token\_multiplitity.  BIND (?anno\_chunk\_multiplicity as ?k)  BIND (?token\_multiplitity as ?n)  {  SELECT DISTINCT ?anno\_chunk\_bs (SUM(?n) as ?sum\_n)  WHERE {  {  SELECT DISTINCT ?anno\_chunk\_bs ?token\_bs  WHERE {  # выборка...  }  }  ?token\_bs t:has\_multiplicity ?token\_multiplitity.  BIND (?token\_multiplitity as ?n)  }  GROUP BY ?anno\_chunk\_bs  }  BIND ((?sum\_n - ?n - ?k) as ?diff) # степень выборки токена базовыми наборами аннотации  }  GROUP BY ?transition\_mode ?token\_bs ?n  }  FILTER (?res < 0) # разница между кол-вом доступных и выбираемых токенов должна быть >= 0  }  }  }  { ?transition\_mode t:has\_binding ?binding. }  UNION  { ?transition\_mode t:has\_transition ?transition. }  } |

Запрос для получения свободных режимов перехода для использования в текущем срабатывании предназначен для клиентской стороны и позволяет определить возможные режимы перехода при текущей маркировке.

|  |
| --- |
| SELECT ?id  WHERE {  ?transition\_mode rdf:type t:TransitionMode.  FILTER NOT EXISTS {  # проверка наличия срабатывания с данным режимом  }  BIND (?transition\_mode as ?id)  } |

Запрос для добавления срабатывания с определенным режимом перехода аналогично производится по инициативе клиента и сигнализирует о начале выполнения перехода.

|  |
| --- |
| INSERT {  # добавление срабатывания, связанных маркировок и мультимножества режимов перехода  }  WHERE {  GRAPH <http://localhost:3030/ontonet/data/tbox> {  ?cpn rdf:type t:CPN.  }  ?cpn t:has\_marking ?marking.  FILTER NOT EXISTS {  # проверка наличия другого срабатывания с данной маркировкой  }  # привязка UUID режима перехода    # формирование UUID для срабатывания, мультимножества и базовых наборов  } |

Запрос для получения информации о возможных действиях по логическому выводу со стороны движка позволяет определить необходимость произведения срабатывания.

|  |
| --- |
| SELECT ?type ?id  WHERE {  ?firing rdf:type t:Firing.  FILTER NOT EXISTS {  # исключение обработанных срабатываний  }  BIND ("firing" as ?type)  BIND (?firing as ?id)  } |

Дальнейшие действия движка начинаются с отправки запроса для получения данных о срабатывании.

|  |
| --- |
| SELECT ?type ?id ?variable\_name ?value ?anno\_chunk\_bs ?token\_bs  WHERE {  # привязка UUID срабатывания  ?firing t:has\_multisetOfTransitionModes ?multiset.  ?multiset t:has\_basisSet ?basis\_set.  ?basis\_set t:has\_data ?transition\_mode.    {  BIND ("multiplicity" as ?type)  # выборка количества вхождений токенов и шаблонов  }  UNION  {  BIND ("relation" as ?type)  # получение выборок токенов частями аннотации  }  UNION  {  BIND ("transition" as ?type)  # получение UUID перехода  }  UNION  {  BIND ("binding" as ?type)  # получение переменных и значений сопоставлений для режима перехода  }  }  ORDER BY ?type |

Запрос для получения комбинаций выборки токенов возвращает все возможные варианты извлечения токенов из входных позиций в данном режиме перехода.

|  |
| --- |
| SELECT <?переменные токенов...>  WHERE {  BIND (<кол-во вхождений для частей аннотации> as <?переменная части аннотации>)  # аналогично для других частей аннотации...    {  SELECT <?переменные токенов...> <?переменные частей аннотации...>  WHERE {  {  VALUES <?переменная части аннотации для токена> { <значения от 0 до кол-ва вхожд. аннотации>}  BIND ((<?переменные части аннотации для токена+...>) as <?переменная токена>)  FILTER (<?переменная токена> <= <кол-во вхождений для токена>)  }  {  # аналогично для других токенов...  }  BIND ((<?переменные части аннотации для токенов+...>) as <?переменная части аннотации>)  # аналогичто для других частей аннотации...  }  }  }  LIMIT 1 |

Запрос для выполнения перехода состоит из трех частей. Две из них не столь значительны и осуществляют очистку онтологии от излишних триплетов. Третья часть определяет стабильные и новые маркировки позиций и базовые наборы токенов, и осуществляет добавление новой маркировки, завершая таким образом срабатывание перехода. Запрос позволяет сохранять в новой маркировке сети отношения с нетронутыми маркировками позиций, а в новых маркировках позиций – с нетронутыми базовыми наборами токенов. Это позволяет упростить сравнение маркировок при обходе цепочки срабатываний.

|  |
| --- |
| INSERT {  # добавление новой маркировки со стабильными и новыми маркировками позиций  # добавление новых маркировок позиций и мультимножеств токенов  }  WHERE {  # формирование UUID новой маркировки  # привязка UUID текущей маркировки  {  GRAPH <http://localhost:3030/ontonet/data/tbox> {  ?cpn rdf:type t:CPN.  }  }  UNION  {  ?firing t:has\_sourceMarking ?source\_marking.  {  ?source\_marking t:includes\_markingOfPlace ?marking\_of\_place.  MINUS { # маркировки позиций с извлекаемыми из них токенами }  MINUS { # маркировки позиций с добавляемыми в них токенами }  BIND(?marking\_of\_place as ?stable\_marking\_of\_place)  }  UNION  {  ?source\_marking t:includes\_markingOfPlace ?marking\_of\_place.  FILTER(  EXISTS {  # удаляемые токены  }  || EXISTS {  # добавляемые токены  }  )  # формирование UUID новой маркировки позиции и нового мультимножества токенов  ?marking\_of\_place t:has\_place ?place;  t:has\_multisetOfTokens ?multiset.  {  ?multiset t:has\_basisSet ?token\_bs.  MINUS { # удаляемые токены }  MINUS { # добавляемые токены }  BIND(?token\_bs as ?stable\_token\_bs)  }  UNION  {  ?multiset t:has\_basisSet ?token\_bs.  FILTER(  EXISTS {  # удаляемые токены  }  || EXISTS {  # добавляемые токены  }  )  # формирование UUID и значений для нового базового набора токенов  OPTIONAL { # удаляемые токены }  OPTIONAL { # добавляемые токены }  BIND ((?token\_multiplicity - ?del\_multiplicity + ?add\_multiplicity) as ?multiplicity)  FILTER (?multiplicity > 0)  }  UNION  {  # для новых уникальных токенов  # формирование UUID и значений для нового базового набора токенов  MINUS { # проверка наличия аналогичных токенов }  }  }  }  }  DELETE {  # удаление неиспользованных режимов перехода  }  WHERE {  # выборка свободных режимов перехода  };  DELETE {  # удаление неиспользованных сопоставлений  }  WHERE {  # выборка свободных сопоставлений  } |

# Распределенная интерпретация РСП: проблемы и решения

Распределенная структура РСП может быть достигнута несколькими способами:

1) распределенное хранение данных (ресурсов);

2) распределенная обработка федеративных SPARQL-запросов;

3) распределение узлов иерархических РСП.

Третий вариант выглядит наиболее подходящим, так как:

а) позволяет реализовывать микросервисы из иерархических моделей;

б) дает возможность вести параллельную разработку узлов.

Дадим определение узла иерарзической РСП. Узел – это автономная ABox онтология, которая имеет входные и выходные порты для организации межузлового взаимодействия. Порты инициализируются URL связанных узлов и типом (input/output).

Tbox онтология при этом может выноситься на отдельный сервер. Однако допускается и хранение собственных экземпляров TBox-онтологии. В таком случае функциональных возможности благодаря обратной совместимости будут ограничены лишь используемой версией.

Срабатывания в распределенных РСП хранятся на каждом узле в отдельности и отражают действия в рамках их внутренних процессов, а также внешних воздействий связанных узлов. При этом общие выражения и константы могут выноситься в TBox-онтологию в виде исключения, если это предпочтительно.

Узел на SPARQL-сервере в совокупности с управляющей программой (движком) образует модуль.

Каждый модуль отвечает за срабатывание только своих переходов. При формировании меток в позиции, являющейся еще и выходным портом, модуль пересылает связанному связанному узлу SPARQL-запрос на формирование, который аналогичен выполненному внутри самого модуля.

Все функционирование сводится к передаче меток через порты. Решение о выполнении перехода принимается клиентской программой.

Есть два варианта исполнения иерархических распределенных РСП:

1) комплексное выполнение переходов через глобальные запросы

2) сконцентрированное выполнение переходов в рамках узлов

Второй вариант является предпочтительным по причине того, что это позволяет инкапсулировать схожий функционал в узлах и следовать принципу единственной ответственности.

Существует два подхода к организации подключения модулей:

1) попарное подключение 1 к 1;

2) разделение общего ресурса.

Попарное подключение обладает следующими достоинствами:

а) можно реализовать подписку на события;

б) в связке “всех со всеми” реализуется общий ресурс;

в) упрощенная синхронизация и процесс подключения;

г) разделение портов на In, Out и I/O.

В свою очередь, организация контактной позиции как общего ресурса усложняет синхронизацию, а также не допускает разделения портов на входные и выходные (все порты являются двухсторонними, I/O).

В отличие от классических иерархических СП каждый узел сети является модулем. Допускается связь “один – много” для модулей.

При срабатывании переходов и изменении позиций, являющихся выходными портами, отрабатывает следующий алгоритм синхронизации. На узле 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 в случае одновременного выполнения переходов, затрагивающих контактные позиции.

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

Такой алгоритм особенно подходит для иерархических сетей (1 к 1, дочерний модуль к родительскому), позволяя определить и выполнить приоритетный переход.

# Архитектура OntoNet

Модули системы представлена на рисунке 1. Оно состоит из следующих модулей: интерфейса командной строки (CLI), веб-сервера, движка (Engine), механизма логического вывода (Reasoner) и связующего ядра (Core).

Система написана на TypeScript и предлагает возможность обработки разметки сети с помощью выражений на JavaScript.

Взаимодействие осуществляется через CLI, который позволяет проинициализировать систему путем подгрузки файлов formulas.js и descriptor.js, установить соединение со SPARQL-сервером по URL, загрузить ABox онтологию и наблюдать за процессом работы модуля.

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 вычисление выражений на JavaScript.



Рисунок 2 - Организация модулей OntoNet

Данная архитектура позволяет в дальнейшем с легкостью развивать систему, например, заменить CLI на полноценное приложение с графическим интерфейсом или встроить базу данных.

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

В качестве примера реализации распределенной системы была выбрана задача синхронизации “Обедающие философы”. Раскрашенная сеть Петри представлена на рисунке 3. Она состоит из трех модулей со связанными позициями (портами) P2-P6, P3-P8, P5-P9. Объявлены наборы цветов для отражения состояния вилок (класс F) и философов (класс S).



Рисунок 3 - Модель системы

Данная модель отражает наиболее важные качества системы: синхронизацию портов при распределенном исполнении и выразительность языка, использующегося для разметки сети.

Визуализация Abox онтологии модели в OntoGraf приведена на рисунке ?...

Состояние фрагментов сети помодульно в ходе моделирования отражено в таблице 1.

Таблица 1 - Ход моделирования

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Module1 | | | Module2 | | | Module3 | | |
| P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 |
| Idle | Fork | Fork | Idle | Fork | Fork | Idle | Fork | Fork |
| Firing T1<x: “Fork”, y: “Fork”, s: “Idle”> | | | | | | | | |
| **Dining** | undef | undef | Idle | Fork | undef | Idle | undef | Fork |
| Firing T1<x: undefined, y: undefined, s: “Dining”> | | | | | | | | |
| **Idle** | Fork | Fork | Idle | Fork | Fork | Idle | Fork | Fork |
| Firing T2<x: “Fork”, y: “Fork”, s: “Idle”> | | | | | | | | |
| Idle | undef | Fork | **Dining** | undef | undef | Idle | Fork | undef |
| Firing T2<x: undefined, y: undefined, s: “Dining”> | | | | | | | | |
| Idle | Fork | Fork | **Idle** | Fork | Fork | Idle | Fork | Fork |
| Firing T3<x: “Fork”, y: “Fork”, s: “Idle”> | | | | | | | | |
| Idle | Fork | undef | Idle | undef | Fork | **Dining** | undef | undef |
| Firing T3<x: undefined, y: undefined, s: “Dining”> | | | | | | | | |
| Idle | Fork | Fork | Idle | Fork | Fork | **Idle** | Fork | Fork |

Изменяющиеся состояния выделены жирным шрифтом.

Модули могут оперироваться в том числе и одним мультизадачным клиентским приложением.

# Выводы

В представленной статье показан метод моделирования раскрашенных сетей Петри с использованием языка SPARQL. Используемый подход обладает повышенной гибкостью и расширяемостью. Полученная система может использоваться для проектирования, прототипирования и моделирования распределенных систем в сети Интернет.

Направлением дальнейших исследований является адаптация метода под моделирование других классов раскрашенных сетей Петри.

…

***Библиографический список***

1. Jensen K., Kristensen L.M. Coloured Petri Nets. Modelling and Validation of Concurrent System // Springer-Verlag Berlin Heidelberg, 2009, 384 с.

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***

...