

УДК 004.89; ГРНТИ 50.07.03

МОДЕЛИРОВАНИЕ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА SPARQL

В.Ю. Каев, В.Н. Дубинин

Пензенский государственный университет,
Российская Федерация, Пенза, veld4n@gmail.com

Аннотация. В данной работе предлагается метод моделирования раскрашенных сетей Петри на основе онтологий и языка SPARQL. Рассматривается онтология раскрашенных сетей Петри, SPARQL-запросы, моделирующие динамику работы, а также вопросы построения приложений для интерпретации сетевых моделей.

Ключевые слова: моделирование, онтология, раскрашенные сети Петри, динамика, SPARQL.

MODELLING OF COLORED PETRI NETS USING SPARQL LANGUAGE

V.I. Kaev, V.N. Dubinin

Penza State University,
Russia, Penza, veld4n@gmail.com

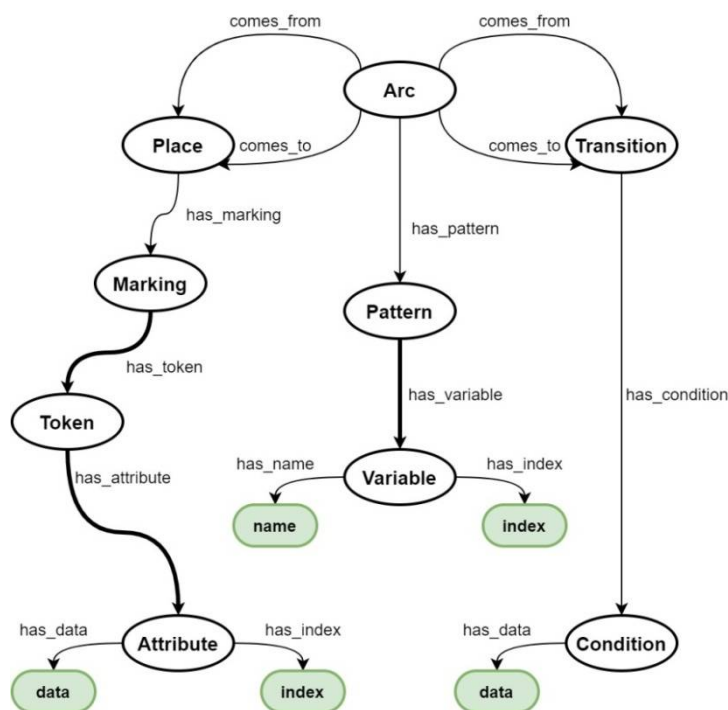
Abstract. This paper proposes a method for colored Petri nets modelling based on ontologies and the SPARQL language. The ontology of colored of Petri nets, SPARQL queries for modelling the dynamics of functioning, as well as issues of building applications to simulate network models are considered.

Keywords: modelling, ontology, coloured Petri nets, dynamics, SPARQL.

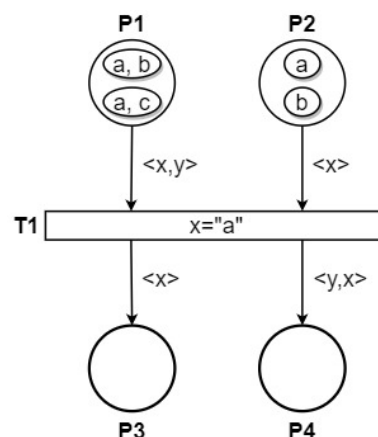
Раскрашенные сети Петри (РСП) широко используются как средство моделирования систем, в которых понятия параллельности и событийного взаимодействия являются ключевыми [1]. Данный класс сетей Петри отличает наличие свойств, присущих высокоуровневым языкам программирования, при сравнительно компактной форме параметризуемых моделей. Это, в частности, делает их подходящими для использования при моделировании, проектировании и реализации распределенных вычислительных, управляющих и коммуникационных систем в промышленном масштабе. Для представления знаний в рамках определенной предметной области, допускающего применение логики, а также позволяющего использовать эти знания повторно, используются онтологии [2]. Они являются основой развивающейся концепции семантического web, которая расширяет возможности машинной интерпретации данных в сети. 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.

В данной работе рассматривается моделирование РСП с использованием онтологий и SPARQL-запросов. Используемая онтология РСП на уровне концепций приведена на рисунке 1,а. Простой пример РСП представлен на рисунке 1,б. Онтология РСП включает в себя: классы Place (Позиция), Arc (Дуга), Transition (Переход), Marking (Маркировка позиции), Token (Метка), Attribute (Атрибут), Pattern (Шаблон дуги), Variable (Переменная шаблона) и Condition (Сторожевое условие перехода); объектные свойства: comes_from и comes_to (для входного и выходного узла дуги), has_marking (для маркировки позиции), has_token (для меток), has_attribute (для атрибутов меток), has_pattern (для шаблона), has_variable (для переменных шаблона), has_condition (для условия перехода); свойства по данным: has_data (для данных маркера и условия перехода), has_index (для определения позиции атрибута/переменной в кортеже) и has_name (для задания имени переменной). Атрибуты меток и

переменные шаблонов упорядочены в кортеж. Данная онтология была сформирована в редакторе онтологий Protégé [4].



a)



б)

Рис. 1. Представления раскрашенных сетей Петри: а – онтология на уровне TBox; б – графическое представление (пример)

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 PCП.

```

PREFIX : <http://www.semanticweb.org/baker/ontologies/2020/9/OntoNet-CPN-ontology#>
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-onlology#>
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 пределах 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 шаблонам 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.

На практике данный подход был применен при разработке клиент-серверной системы интерпретации РСП под названием *OntoNet*. Она позволяет пользователю загружать на сервер онтологию РСП и моделировать ее выполнение в ручном режиме. В качестве сервера используется *Apache Jena Fuseki Server* [5], взаимодействие организовано по протоколу HTTP. Клиент был разработан на языках *JavaScript/TypeScript*. Он спроектирован с использованием модульной архитектуры, поэтому отдельные его модули могут использоваться в других проектах. Каждый модуль имеет свой интерфейс взаимодействия, предоставляющий широкий диапазон методов для получения состояния сети, определения разрешенных переходов, пошагового моделирования и т.д. Состояние РСП выводится в виде таблицы, отражающей маркировку позиций, однако представление отделено от логики системы и может быть заменено на более продвинутое или специализированное.

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.

Библиографический список

1. Jensen K., Kristensen L.M. Coloured Petri Nets. Modelling and Validation of Concurrent Systems // Springer-Verlag Berlin Heidelberg, 2009. – 384 с.
2. Jacksi K. Introduction to Semantic Web. // University of Zakho Department of Computer Science March, 2014.
3. SPARQL 1.1 Update. W3C Recommendation 21 March 2013. [Электронный ресурс]. – Режим доступа: <https://www.w3.org/TR/2013/REC-sparql11-update-20130321/>. – Дата доступа: 05.12.2020.
4. Protégé [Электронный ресурс]. Режим доступа: <http://protege.stanford.edu/>. – Дата доступа: 17.10.2020.
5. Apache Jena Fuseki [Электронный ресурс]. – Режим доступа: <https://jena.apache.org/documentation/fuseki2/>. – Дата доступа: 17.12.2020.