УДК 519.876.5

СПОСОБ МОДЕЛИРОВАНИЯ РАСПРЕДЕЛЕННЫХ АВТОМАТИЗИРОВАННЫХ ПРОЦЕССОВ И ПРОИЗВОДСТВ

Смирнов Д.П.

Национальный исследовательский университет «МИЭТ», Москва, e-mail: smirnovdp@gmail.com

Активно развивающаяся современная промышленность требует автоматизации производственных процессов. На практике решение этой задачи часто представляет собой сложную распределенную автоматизированную систему, поддержка которой требует наличия средств анализа и оптимизации производительности процессов. Приоритетным направлением в разработке данных средств является имитационное моделирование с использованием графических редакторов для описания исследуемого объекта. Их применение обусловлено простотой и наглядностью процесса моделирования. Однако в случае, когда используются сложные условные выражения или вычисления, имеются зависимости напрямую несвязанных элементов, а также присутствуют схожие, но не одинаковые процессы, всё это либо делает модель нечитаемой, либо невозможной в реализации. В данной статье рассматривается способ моделирования автоматизированных производственных процессов, позволяющий решить описанные проблемы современных средств моделирования с помощью предметно-специфического языка, разработанного для программного применения методики объектно-ориентированного моделирования на основе модифицированных Е-сетей [2]. Применение данного способа даёт возможность описывать сложные распределенные системы, недоступные для моделирования в специализированных графических системах.

Ключевые слова: имитационное моделирование, предметно-ориентированный язык, объектно-ориентированное моделирование, Е-сети

METHOD OF MODELING DISTRIBUTED AUTOMATED PROCESSES AND PRODUCTION

Smirnov D.P.

National Research University of Electronic Technology, Moscow, e-mail: smirnovdp@gmail.com

Modern industry actively develops and requires automation of production processes. In practice, the solution of this problem is often a complex distribution system of automated support tools that require analysis and optimization of process performance. The priority in the development of these tools is a simulation using the graphic editors for the description of the object. Application of these editors is due to the simplicity and visibility of the modeling process. But when using complex conditional description or calculations are based directly unrelated elements, and there are similar but not identical processes, all this makes the model either illegible or impossible to implement. This article describes a method of modeling automated production processes, which allows to solve the problems of modern tools of simulations using domain-specific language developed for the software application techniques object-oriented modeling based on modified E-nets [2]. Application of this method allows to describe complex distributed systems that are impossible for modeling in specialized graphics systems.

Keywords: simulation modeling, domain-specific language, object-oriented modeling, E-nets

Современная промышленность развивается и конкурирует главным образом за счет использования автоматизации в различных производственных процессах. Для этого внедряют автоматизированные средства управления технологическими процессами, модернизируют и роботизируют производство, применяют специализированное программное обеспечение. При этом важным направлением является разработка средств анализа и оптимизации распределённых автоматизированных промышленных процессов. Один из наиболее значимых инструментов для достижения этих целей – имитационное моделирование.

Современные средства имитационного моделирования в основном используют графические редакторы для описания исследуемого объекта. Их применение обусловлено простотой и наглядностью процесса моделирования. Однако в случае, когда используются сложные условные выражения или вычисления, имеются зависимости напрямую несвязанных элементов, а также присутствуют схожие, но не одинаковые процессы, всё это либо сильно увеличивает размеры модели, делая её нечитаемой, либо требует упрощения, в связи с невозможностью реализации.

Данная статья направлена на рассмотрение способа моделирования распределенных автоматизированных процессов в виде предметно-ориентированного языка [5] *OOMDL (Object-Oriented Model Description Language*), разработанного для программного применения методики объектно-ориентированного моделирования (ООМ) на основе модифицированных Е-сетей [2]. Предлагаемый способ даёт возможность представления модифицированных сетей Петри в виде последовательного набора операторов. Синтаксис частично схож с Си

и Си-подобными языками, что было сделано для упрощения его изучения и использования. Применение данного способа даёт возможность описывать сложные распределенные системы, недоступные для моделирования в специализированных графических системах.

Далее для описания конструкций *OOMDL* будет использоваться расширенная форма Бэкуса – Наура (БНФ) [3].

Методика ООМ предполагает использование объекта как базового элемента, представляющего собой описание некоего исследуемого процесса в виде сети Петри [4]. В коде моделирования он будет представлен в следующем виде:

```
объект = 'OBJECT' имя_объекта
[ : имя_базового_объекта ] '{' тело '}'
```

Данная конструкция включает в себя название создаваемого объекта и, если он является наследником, позволяет указать имя базового объекта. В фигурных скобках содержится его описание (тело). В OOMDL границы блоков определяются открывающейся фигурной скобкой в начале и закрывающейся фигурной скобкой в конце. Описание объекта обязательно начинается с определения структурных компонентов, локальных переменных и экземпляров используемых объектов. Всё это необходимо для понимания интерпретатором, какие компоненты применяются в дальнейшем для построения и функционирования сети Петри.

Структурные компоненты объекта задаются внутри соответствующих блоков, обозначенных ключевыми словами POSITIONS (позиции), NETS (элементарные сети), где они разбиваются на типы. Для позиций это выбор между SINGLE (простая позиция) и QUEUE (очередь), а для элементарных сетей – разделение на T-сети, X-сети, Y-сети, G-сети и I-сети. Элементарная сеть представляет собой объединение, содержащее один переход, связанные с ним позиции и условия его работы [1]. Каждый структурный компонент может иметь текстовое описание, отображаемое в результатах моделирования, что улучшает восприятие информации при проведении анализа работы системы. Код моделирования структурных компонентов имеет следующий вид:

```
позиции = 'POSITIONS' '{' { ( 'SINGLE' | 'QUEUE' ) имя_позиции [ ':' '/'
комментарий '/' ] { ',' имя_позиции [ ':' '/' комментарий '/' ] } ';' } '}'

сети = 'NETS' '{' { ('T' | 'X' | 'Y' | 'G' | 'I') имя_сети [ ':' комментарий ]
{ ',' имя_сети [ ':' комментарий ] } ';' } '}'
```

Локальные переменные, как и структурные компоненты объекта, объявляются внутри блока, который представлен ключевым словом VARIABLES. В данном случае синтаксис внутри блока схож с языком Си – задаётся тип переменной, её название и начальное значение. Типы переменных: INT (целочисленные), DOUBLE (вещественные), STRING (строковые) и BOOL (булевые). Над строковыми переменными возможно производить только два вида действия: сравнение и присваивание нового значения. Блок локальных переменных описывается по следующему правилу:

```
переменные = 'VARIABLES' '{' { ( 'INT' | 'DOUBLE' | 'STRING' | 'BOOL' )
имя_переменной [ '=' начальное_значение ] { ',' имя_переменной [ '='
начальное_значение ] } ';' } '}'
```

Объекты взаимодействуют друг с другом через экземпляры (элементы) – копии объектов, объявленные в блоке ELEMENTS. Каждый экземпляр внутри блока должен иметь уникальное название, заданное пользователем. Объект может использовать несколько элементов любого другого объекта. При этом нельзя создавать рекурсии, в которых объекты вызывают друг друга или когда объект имеет экземпляры самого себя. Элементы описываются следующим образом:

```
элементы = 'ELEMENTS' '{' { имя_объекта имя_элемента { ',' имя_элемента } ';' }
'}'
```

Порядок следования рассмотренных выше компонентов не важен, необходимо только, чтобы они были заданы в начале объекта. Остальные компоненты могут располагаться в любом порядке, поскольку они не зависят друг от друга.

Порты, через которые происходит взаимодействие с объектом, назначаются из объявленных ранее позиций. Они разделяются на три группы, каждая из которых начинается со своего ключевого слова: IN (входные порты), OUT (выходные порты), INIT (порты начального расположения меток). Любая позиция может входить только в одну из этих групп, что вызвано особенностью строения моделируемой сети: во входных и выходных портах содержатся позиции с отсутствующими одноименными связями, и задание для них меток невозможно, поскольку в дальнейшем они будут объединены с позициями других объектов. Порты объекта задаются в блоке PORTS:

```
порты = 'PORTS' '{' { ( 'IN' | 'OUT'
| 'INIT' ) имя_позиции { ',' имя_позиции }
';' '}'
```

Объявленные ранее элементарные сети должны быть дополнены набором составляющих (тело_сети), включающих в себя указание связей с позициями и используемых атрибутов меток, определение условий перехода, назначение метода преобразования данных и расчет времени перехода метки. Все эти параметры объединены блоком NET:

```
инициализация_сети =
'NET' имя_сети '{' тело_сети '}'
```

Позиции, используемые в работе элементарной сети, задаются в списке связей LINK, где ключевые слова IN и OUT определяют группы входных и выходных позиций. Использование I-сети предполагает возможность определения прерывающих позиций, в связи с чем добавлены отдельные группы ~IN и ~OUT. Очерёдность позиций в элементарной сети определяется порядком их задания. Код моделирования списка связей пишется по следующему правилу:

```
связь_сети = 'LINK' '{' { ( 'IN' | '~IN '
| 'OUT' | '~OUT' ) имя_позиции
{ ',' имя_позиции } ';' } '}'
```

Элементарная сеть имеет возможность взаимодействовать с метками и их атрибутами, для чего используется ключевое слово point, которое включает в себя её описание в момент перехода. В случае, когда входных активных позиций несколько, метка берётся из той, которая была задана раньше остальных. Отдельно для прерывающей позиции применяется ключевое слово ~point. Определение в I-сети, какие позиции активны, возможно благодаря представлению ключевого слова метки в виде булевого значения, представляющего истину при их наличии. Доступ к атрибутам метки осуществляется по следующему правилу:

```
атрибут = 'point''.'название_атрибута
```

Имеются ограничения на используемые атрибуты в каждой конкретной элементарной сети. Блок ATTRIBUTES содержит их название и типы (INT, DOUBLE и т.п.). В проектируемой модели важно, чтобы поступающие на вход элементарной сети метки содержали объявленные в этом блоке атрибуты, но не обязательно только их. Данные ограничения необходимы для возможности использования объявленных атрибутов в последующих компонентах. Отсюда следует, что при стандартной работе, когда содержание метки не влияет на переход, данный блок не является обязательным. Правило его описания выглядит следующим образом:

```
атрибуты_сети = 'ATTRIBUTES' '{' {
( ' INT' | 'DOUBLE' | 'STRING' | 'BOOL' )
имя_атрибута { ',' имя_атрибута } ';' } '}'
```

Связь с позициями и набор используемых атрибутов пишутся в начале описания элементарной сети, поскольку последующие компоненты напрямую зависят от них и их используют.

Метод преобразования данных изменяет значения атрибутов меток для выходных позиций, используя для этого входные данные и значения доступных ему переменных. Данный метод представляет собой блок ACTION. Основным его инструментом является условный оператор IF...ELSE. Помимо этого имеется возможность производить математические операции и присваивать значения переменным и атрибутам. Синтаксис операторов и операций такой же, как в Си-подобных языках. I-сеть, в отличие от других элементарных сетей, имеет дополнительный метод преобразования ~ACTION, который выполняется при возникновении события прерывания.

Условия перехода задаются отдельно для входных и выходных позиций в блоках ACTIVE_IN и ACTIVE_OUT. Связано это с тем, что для таких типов элементарных сетей, как X-сеть и Y-сеть, возможно задавать условия только для одного вида позиций. Зарезервированные в *OOMDL* переменные _in и _out определяют собой активные входные и выходные позиции. Данные переменные можно использовать только в соответствующих их названию блоках через операцию присваивания. В теле блока разрешено использование условных операторов, применение математических операций и имеется возможность определять наличие метки в позиции. При использовании

позиции в условных выражениях она выдает ответ булевого типа, где значение true предполагает наличие в ней метки. В случае позиции-очереди для выходных позиций всегда будет выдаваться значение false.

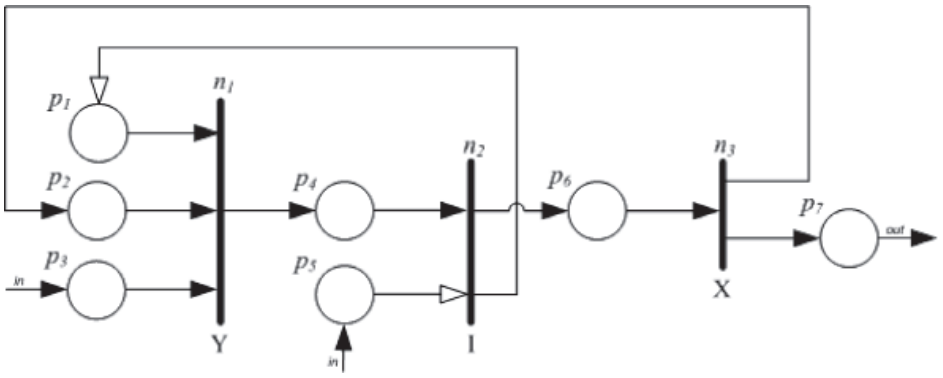
Время выполнения перехода по умолчанию определяется равным нулю, то есть действие происходит мгновенно. Однако имеется возможность задать это значение некоторой постоянной или набором условий. Для этого используется блок `DELAY` с зарезервированной переменной `_delay`, которая и определяет время задержки перехода. Значение `_delay` помимо данного блока может использоваться в методе преобразования данных, без возможности его изменения. При этом прошедшее время ожидания перехода хранится в переменной `_time`, с которой нельзя использовать операцию присваивания.

Рассмотренные компоненты представляют полный набор для описания объекта. Помимо объектов, код моделирования может ещё содержать глобальные переменные. Они объявляются в начале моделирования и содержатся в блоке `GLOBAL_VARIABLES`. Содержание блока идентично `VARIABLES`. При этом обращение к данным переменным внутри объектов начинается с префикса `glob` и точки, тем самым отделяя локальные переменные от глобальных со схожим именем:

```
глобальная_переменная =

'glob''.'название_переменной
```

Применение языка моделирования можно рассмотреть на примере обработки запросов с прерываниями. Такая особенность рабочего процесса довольно часто присутствует в различных сферах производства. Сетевая схема процесса изображена на рисунке и ниже представлен код её описания.



Модель обработки запросов с прерываниями

```
OBJECT interruption {
  POSITIONS{ SINGLE p1, p2, p3, p4, p5, p6, p7; }
  NETS { Y n1; I n2; X n3; }
  PORTS { IN p3, p5; OUT p7; }
  NET n1 {
    LINK { IN p1, p2, p3; OUT p4; }
  }
  NET n2 {
    LINK { IN p4; ~IN p5; OUT p6; ~OUT p1; }
    ATTRIBUTES { INT work; }
    DELAY { _delay = point.work; }
    ACTION { point.work = 0; }
    ~ACTION {
      IF (point) { point.work = _delay - _time;}
    }
  }
  NET n3 {
    LINK { IN p6; OUT p7, p2;}
    ATTRIBUTES { INT work; }
    ACTIVE_OUT {
      IF ( point.work != 0) { _out = p7; }
      ELSE { _out = p2; }
    }
  }
}
```

Схема построена с помощью трех элементарных сетей. В качестве запросов в модели используются метки, содержащие атрибут `work` со временем, которое необходимо для их обработки. *Y*-сеть n_1 отвечает за их объединение и выстраивание в одну общую очередь. Её входные позиции представляют собой различные источники запросов. Позиция p_3 является входным портом и основным каналом поступления запросов. При работе сети без прерываний метки поступают только по данному каналу и выполняются последовательно друг за другом. Позиция-очередь p_1 хранит в себе прерывающие сигналы, которые требуют немедленной обработки. Метки в эту позицию попадают уже после остановки выполнения предыдущего запроса, который отправляется в позицию-очередь p_2 для ожидания продолжения своей обработки. Приоритетность позиций определяется сверху вниз, где p_1 имеет наивысший приоритет, а p_3 – самый низкий. Выбранная элементарной сетью метка переходит в позицию p_4.

Непосредственная обработка запросов происходит в *I*-сети n_2. Время перехода определяется атрибутом `work` метки из позиции p_4. Так работает элементарная сеть, пока не поступит прерывающий сигнал во входной порт p_5. Сигнал представляет собой запрос в виде метки, который необходимо обработать вне очереди. Без прерывания действие, производимое сетью после перехода, устанавливает значение атрибута `work` равным нулю, в противном случае выполняется блок `~ACTION`, где при наличии метки в процессе перехода вычисляется оставшееся время обработки. Запросы из позиции p_4 переходят в позицию p_6, а из позиции p_5 в позицию p_1.

После обработки запрос из позиции p_6 идет в элементарную сеть n_3 типа *X*. Она определяет, что дальше делать с запросом. Имеются два маршрута, по которым метка может следовать. Они выбираются в зависимости от условий перехода `ACTIVE_OUT`. Если у запроса атрибут `work` не равен нулю, то он отправляется в позицию p_2 для завершения обработки, иначе идет в выходной порт.

Заключение

Представленный способ моделирования в виде предметно-ориентированного языка OOMDL позволяет описывать производственные процессы по методике ООМ компактно и наглядно. Благодаря этому в дальнейшем появляются возможности автоматизации процесса разработки программного обеспечения, контроля исходных кодов модели и создания расширяемых библиотек элементов.

Список литературы

1. Илюшечкина Л.В., Разработка средств моделирования для исследования систем распределённой обработки информации: дис. канд. технических наук. – М., 2002.

2. Костин А.Е., Модели и алгоритмы организации распределенной обработки данных в информационных системах: докт. дисс. – М., 2002.

3. ISO/IEC 14977: 1996(E) [Электронный ресурс] // Сайт университета Кембриджа. – URL: https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf (дата обращения 11.11.2015).

4. NUTT G.J. Evaluation Nets for Computer System Performance Analysis, AFIPS FJCC, vol. 41, 1972, Pt. 1.

5. Voelter M. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages, 2013.

References

1. Ilyushechkina L.V. *Razrabotka sredstv modelirovaniya dlya issledovaniya sistem raspredelennoy obrabotki informacii*. Kand. tekhnicheskikh nauk, Diss. [Development of modeling tools for the research of systems of distributed information processing. PhD. Diss.]. Moscow, 2002.

2. Kostin A.E. *Modeli i algoritmy organizatsii raspredelennoy obrabotki dannykh v informatsionnykh sistemakh*. Dokt, Diss. [Models and algorithms for the organization of distributed data processing in information systems. Doct. Diss.]. Moscow, 2002.

3. ISO/IEC 14977: 1996(E) Available at https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf (accessed 11 November 2015)

4. NUTT G.J. Evaluation Nets for Computer System Performance Analysis, AFIPS FJCC, vol. 41, 1972, Pt. 1.

5. Voelter M. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages, 2013.