УДК 519.876.5

К.О. Устимов, Н.В. Федоров

АВТОМАТИЗАЦИЯ ПОСТРОЕНИЯ ИМИТАЦИОННОЙ МОДЕЛИ БИЗНЕС-ПРОЦЕССОВ НА ОСНОВЕ МЕТОДОЛОГИИ IDEF0 И РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ

Рассмотрены основные принципы преобразования функциональной модели бизнес-процессов IDEF0 в имитационную модель в виде раскрашенных сетей Петри. Имитационная модель, полученная в результате этого преобразования, может использоваться как базовая модель, которую можно усовершенствовать в соответствии с особенностями реальной системы.

Ключевые слова: функциональное моделирование, методология IDEF0, имитационное моделирование, раскрашенные сети Петри, бизнес-процесс.

Имитационное моделирование применяется для оптимизации бизнес-процессов. Оно позволяет определить, как преобразования повлияют на организацию, ставя эксперименты не на «живой» организации, а на ее модели.

Имитационное моделирование — метод исследования, основанный на том, что изучаемая система заменяется моделью, имитирующей эту систему. Над моделью проводят эксперименты и в результате получают информацию о реальной системе. Имитационное моделирование выполнения бизнес-процессов широко применяется в проектах по реинжинирингу деятельности компаний, когда необходимо заранее спрогнозировать результаты [1].

Рассмотрим подход к автоматизированному построению имитационной модели бизнес-процессов на основе уже имеющейся функциональной модели. Ключевая особенность имитационной модели заключается в наличии определенных принципов проведения имитации. Эти принципы одинаковы для всех имитационных моделей одного типа. Имитационные модели одного и того же типа различаются только структурой, которая может быть схожа со структурой функциональной модели.

Рассмотрим методологию функционального моделирования IDEF0. Данная методология широко применяется для моделирования бизнес-процессов, поэтому к моменту возникновения необходимости проведения имитационного моделирования модель IDEF0 может быть уже построена.

Функциональная модель IDEF0 позволяет описать структуру и функции системы, а также потоки информации и материальных объектов, преобразуемые этими функциями [2]. Далее по тексту для обозначения содержимого потока любого типа будем использовать термин «данные». Мы не можем использовать модель IDEF0 для имитации непосредственно, однако, зная, как должны взаимодействовать между собой функции бизнес-процессов, мы можем на основе имеющейся IDEF0-модели построить имитационную модель.

Для имитационного моделирования будем использовать раскрашенные сети Петри (CPN). Для CPN не существует четкого определения. CPN являются всего лишь некоторым абстрактным термином, обозначающим сети Петри с расширенными возможностями, как минимум, в отношении типизации элементов сетей. В данной статье под термином CPN мы будем понимать раскрашенные сети Петри, разработанные Куртом Йенсеном (университет Орхуса, Дания) [3].

CPN являются совмещением обычных сетей Петри с высокоуровневым языком моделирования CPN ML. Этот язык основан на языке моделирования Standard ML. Перечислим некоторые понятия, которые CPN ML вводит обычную сеть Петри:

- множество цветов — тип данных (простой или сложный);
- маркировка позиции — мультимножество значений определенного типа, связанных с определенной позицией;
- выражение дуги — математическое описание преобразования, выполняемое над маркировкой позиций, связанных с данной дугой.

Перечисленные выше понятия являются довольно абстрактными. Это связано с тем, что CPN ML является языком моделирования общего назначения, что делает CPN универсальным средством имитационного моделирования в определенном классе задач.

Рассмотрим преобразование IDEF0 в CPN. Модель IDEF0 имеет иерархическую структуру, сущность которой выражается термином «декомпозиция». Это означает, что любой блок может быть разложен на основные подблоки посредством создания дочерней диаграммы, В свою очередь, каждый из этих подблоков также может быть разложен на составные части созданием дочерней диаграммы более низкого уровня и т.д. [2]. CPN также могут иметь иерархическую структуру, поэтому достаточно рассмотреть процесс преобразования отдельной IDEF0-диаграммы в CPN.

Преобразование IDEF0-диаграммы в CPN начинается с определения переходов CPN. Каждому функциональному блоку IDEF0-диаграммы взаимнооднозначно сопоставляется переход CPN.

Функциональные блоки в IDEF0-диаграмме связаны между собой непосредственно стрелками. Стрелки определяют каналы, по которым передаются данные. В CPN в роли данных выступают маркировки позиций. Для простой внутренней стрелки создадим следующие элементы CPN:

- позицию CPN и определенный тип данных,
- входящую и выходящую дуги и определенные выражения для них.

На рис. 1 показаны исходная часть диаграммы IDEF0 и полученная из нее часть модели CPN. Для этой сети заданы следующие определения:

```
colset t_Data = string;
val t1_out = 1`"data";
val t1_in = 1`"data";
```

В данном случае будет использоваться строковый тип данных. При срабатывании перехода «Tran 1» позиция «Data» получит одну фишку со значением "data". Переход «Tran 2» может сработать только в том случае, если в позиции «Data» имеется хотя бы одна фишка со значением "data", при этом эта фишка будет извлечена из позиции.

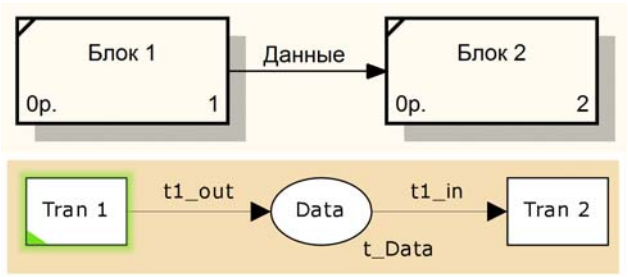

Рис. 1

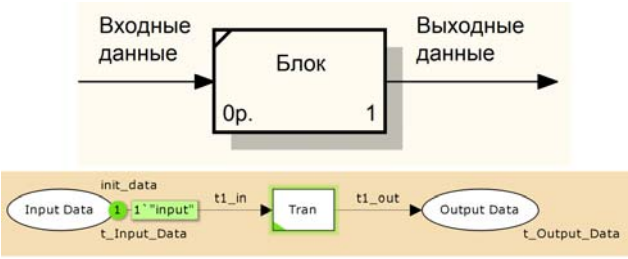

Рис. 2

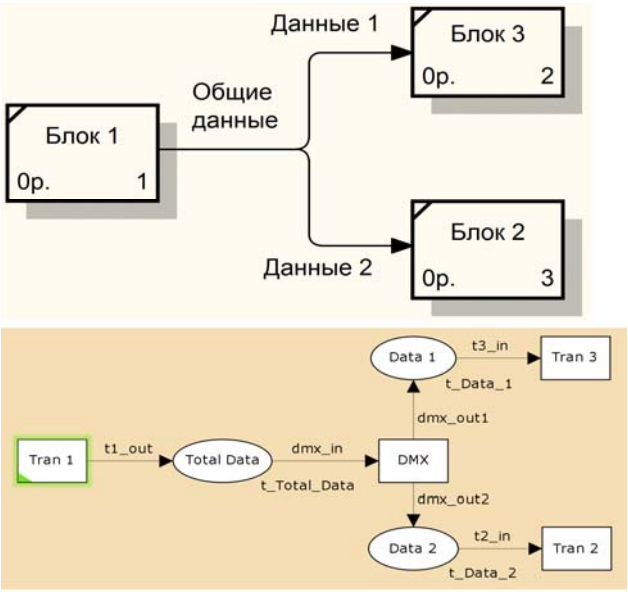

Рис. 3

val dmx_in = 1`"data1, data2";
val dmx_out1 = 1`"data1";
val dmx_out2 = 1`"data2";
val t3_in = 1`"data1";
val t2_in = 1`"data2";

В случае с внешней стрелкой преобразование будет таким же, за исключением одного блока и одной стрелки (см. рис. 2). Для этой сети заданы следующие определения:

colset t_Input_Data = string;
colset t_Output_Data = string;
val init_data = 1`"input";
val t1_in = 1`"input";
val t1_out = 1`"output";

В отличие от случая с внутренней стрелкой, для внешней входной стрелки была создана позиция «Input Data» с начальной маркировкой в виде одной фишки со значением "input". Эта фишка будет извлечена из позиции при срабатывании перехода «Tran», а в позицию «Output Data» будет помещена одна фишка со значением "output".

Рассмотрим теперь преобразование, связанное с ветвлением стрелки. В этом случае в CPN будет добавлен дополнительный переход, который будет разделять поток данных на составляющие потоки. На рис. 3 показано такое преобразование. Для этой сети заданы следующие определения:

colset t_Total_Data = string;
colset t_Data_1 = string;
colset t_Data_2 = string;
val t1_out = 1`"data1, data2";

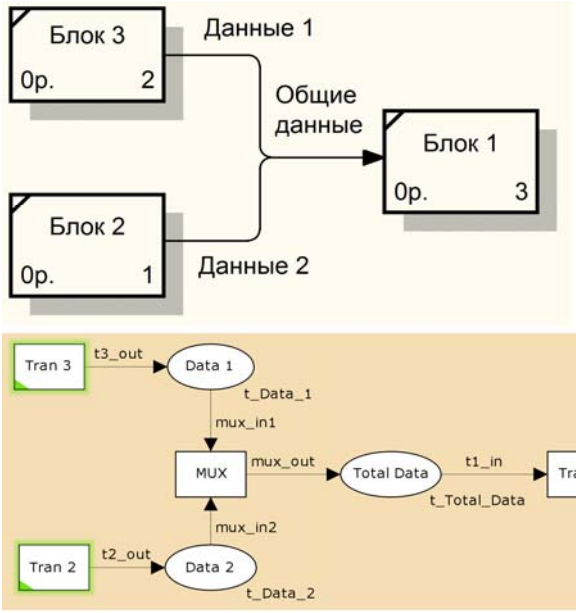

Рис. 4

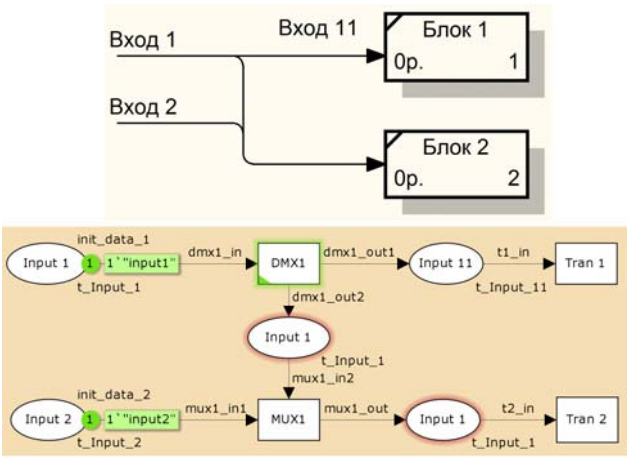

Рис. 5

Переход «DMX» может сработать только, когда в позиции «Total Data» имеется хотя бы одна фишка со значением "data1, data2". При срабатывании этого перехода в позициях «Data 1» и «Data 2» появятся по одной фишке со значениями, соответственно, "data1" и "data2".

На рис. 4 показано аналогичное преобразование слияния стрелок. Для этой сети заданы следующие определения:

```
colset t_Total_Data = string;
colset t_Data_1 = string;
colset t_Data_2 = string;
val t1_in = 1`"data1, data2";
val mux_out = 1`"data1, data2";
val mux_in1 = 1`"data1";
val mux_in2 = 1`"data2";
val t3_out = 1`"data1";
val t2_out = 1`"data2";
```

Переход «MUX» может сработать только, когда в позициях «Data 1» и «Data 2» имеется хотя бы по одной фишке со значениями, соответственно, "data1" и "data2". При срабатывании этого перехода в позиции «Total Data» появится одна фишка со значением "data1, data2".

Рассмотрим более сложный пример ветвления и слияния, показанный на рис. 5. Для этой сети заданы следующие определения:

```
colset t_Input_1 = string;
colset t_Input_2 = string;
colset t_Input_11 = string;
val init_data_1 = 1`"input1";
val init_data_2 = 1`"input2";
val dmx1_in = 1`"input1";
val dmx1_out1 = 1`"input11";
val t1_in = 1`"input11";
val dmx1_out2 = 1`"input1";
```

```
val mux1_in1 = 1`"input2";
val mux1_in2 = 1`"input1";
val mux1_out = 1`"input1";
val t2_in = 1`"input1";
```

В этом примере поток «Вход 1» входит в «Блок 2». От этого потока сначала отделяется составная часть «Вход 11» и входит в «Блок 1». Затем к нему присоединяется дополнительная часть «Вход 2». Таким образом, мы имеем пять сегментов стрелок, которые соединены с помощью двух узлов связывания. В соответствии с этим в CPN были созданы пять позиций и два дополнительных перехода. Обратим внимание на то, что три из этих позиций имеют одинаковые имена «Input 1» и одинаковые типы данных t_Input_1. Используя этот прием, мы добиваемся однозначности преобразования.

Руководствуясь вышеописанными принципами преобразования, можно преобразовать в CPN любую IDEF0-диаграмму стандартного вида, не содержащую туннелей и стрелок вызовов. Полученная модель будет являться базовой имитационной моделью бизнес-процессов. Данные принципы полностью формализованы, поэтому процесс построения базовой модели может быть выполнен автоматически. Следующим этапом следует уточнить типы данных, выражения дуг, начальную маркировку и другие описания в соответствии с особенностями реальной системы. Также можно внести изменения в структуру базовой модели и добавить новые описания и логику моделирования, предоставляемые языком CPN ML.

СПИСОК ЛИТЕРАТУРЫ

1. http://www.businessstudio.ru/ procedures/ business/immodel/

2. Р 50.1.028 — 2001. Рекомендации по стандартизации. Методология функционального моделирования.

3. *Kurt Jensen, Lars M. Kristensen.* Coloured Petri Nets. Modelling and Validation of Concurrent Systems — Springer, 2009. ГИАБ

КОРОТКО ОБ АВТОРАХ

Устимов Кирилл Олегович – аспирант, avader@yandex.ru

Федоров Николай Владимирович – кандидат технических наук, доцент, зам. зав. кафедрой, fnv1@mail.msiu.ru

кафедра информационной безопасности, Московский государственный индустриальный университет (ФГБОУ ВПО «МГИУ»).