УДК 004.413

ПРИНЦИПЫ ПОЛНОЙ И КОРРЕКТНОЙ ТРАНСФОРМАЦИИ СИНХРОНИЗИРУЕМЫХ МОДЕЛЕЙ

В.В. Девятков, **Д.В. Ошкало**

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация
e-mail: deviatkov@bmstu.ru; dmitry.oshkalo@gmail.com

Предложена методика полной и корректной трансформации моделей для решения задачи их синхронизации. В ее основе лежит стратегия построения полного и корректного набора правил трансформации, обеспечивающих выполнение критериев корректности и полноты путем анализа синтаксической и семантической структуры метамоделей с использованием аппарата графовых грамматик. Рассмотрены применяемые в настоящее время подходы для создания правил трансформации моделей, показаны их недостатки, влияющие на результат трансформации, выявлена связь между корректностью метамоделей правилами трансформации, а также полнотой и корректностью процесса трансформации. Предлагаемая методика не только позволяет автоматически создавать указанный набор правил трансформации моделей, но также определять порядок порождения правил. Приведены принципы доказательства этого основополагающего результата.

***Ключевые слова*:** трансформация моделей, синхронизация моделей, графовые грамматики, UML, полнота и корректность трансформации моделей.

THE PRINCIPLES OF COMPLETE AND CORRECT TRANSFORMATION OF THE SYNCHRONIZED MODELS

V.V. Devyatkov, **D.V. Oshkalo**

Bauman Moscow State Technical University, Moscow, Russian Federation
e-mail: deviatkov@bmstu.ru; dmitry.oshkalo@gmail.com

The article describes the method of complete and correct model transformation within a model synchronization scenario. It is based on a strategy of producing a complete and correct set of transformation rules, ensuring the completeness and correctness criteria by analyzing a syntactic and semantic structure of meta-models via graph grammar techniques. Currently used approaches to creating the model transformation rules are considered. Their disadvantages which influence the result of the model transformation process are demonstrated. Correlations between the correctness of the meta-models and transformation rules as well as the completeness and correctness of the transformation process are found. The proposed method allows both creating a correct set of the model transformation rules automatically and determining a sequence of this creation. The evidences of this fundamental result are provided.

***Keywords*:** model transformation, model synchronization, UML, graph grammars, completeness and correctness of model transformation.

Введение. Ключевая особенность процесса разработки программного обеспечения, управляемого моделями, — использование множества формальных структурированных компонентов — моделей, на основе которых путем применения алгоритмов генерации строятся различные артефакты (программный код, конфигурационные файлы,

скрипты, представления данных и т.п.). С одной стороны, применение такого подхода освобождает разработчиков от работы по написанию шаблонов и заготовок кода [1]. С другой стороны, это порождает ряд дополнительных задач, связанных с сопровождением созданных моделей, которые зачастую так или иначе связаны друг с другом, поскольку описывают с разных сторон один и тот же объект (например, схема реляционного хранилища данных и соответствующая ей структура классов, используемая приложением, которое работает с этим хранилищем).

Одна из таких задач — синхронизация моделей, заключающаяся в необходимости внесения адекватных изменений во все взаимосвязанные модели при изменении хотя бы одной из них.

Вследствие большого разнообразия моделей задача синхронизации, как правило, ставится в терминах метамоделей, определяющих структуру моделей, которые в свою очередь являются их экземплярами. Более строго этот факт будет рассмотрен ниже.

На практике синхронизация осуществляется путем последовательных трансформаций, выполняющихся по определенным правилам над частями моделей и переводящих эти модели к адекватному (консистентному) с позиции метамоделей виду [2]. Актуальной, но все еще полностью нерешенной задачей является создание для конкретных классов метамоделей наборов правил трансформации, обладающих полнотой (возможностью внесения всего необходимого перечня изменений) и гарантирующих корректность трансформаций (адекватность внесения изменений с позиции метамоделей).

В настоящее время вследствие того, что модели представляются в виде графов (графовых моделей), широкое применение для их синхронизации получили подходы, основанные на графовых грамматиках [3–5]. Правила вывода в графовых грамматиках в отличие от правил вывода в текстовых грамматиках, в левой и правой частях которых фигурируют символьные представления, используют правила, в левую и правую части которых помещаются фрагменты графовых моделей. В простейшем случае каждое правило графовой грамматики порождает трансформацию графового фрагмента модели, находящегося в левой части правила, в графовый фрагмент. Вид этого фрагмента зависит от того, что находится в правой части правила. Именно поэтому правила графовой грамматики также называются правилами трансформации.

Популярность таких подходов объясняется, прежде всего, удобством практического применения, которое выражается в наглядности, возможности использования графических редакторов при проектировании правил трансформации, более прозрачных и эффективных методах контроля, верификации и отладки разработанных трансформаций.

Функциональные критерии корректности, характеризующие процессы преобразования моделей, рассмотрены в работах [4, 6–9], где

изучены корректности процессов синхронизации и трансформации моделей. Однако среди критериев, влияющих на корректность этих процессов, можно выделить такие, которые характеризуют связь правил трансформации с обрабатываемыми моделями.

Один из таких критериев — полнота, заключающаяся в том, что любое допустимое изменение в модели должно быть обработано при синхронизации, т.е. набор правил трансформации должен быть составлен так, чтобы в процессе работы с моделями не возникало ситуации, когда внесенное в одну из моделей изменение не может быть перенесено в другие связанные модели.

Другой критерий — корректность структуры обрабатываемых моделей, т.е. их метамоделей, необходимая для гарантии возможности трансформации экземпляров этих метамоделей. Понятие корректности метамоделей будет раскрыто далее.

Выполнение указанных критериев должно быть заложено при разработке набора правил трансформации моделей, однако применяемые подходы [3, 5] в значительной степени базируются на интуитивных и эмпирических приемах обеспечения полноты и корректности набора правил. В связи с этим их использование сопряжено с определенным риском, так как нельзя гарантированно получить набор правил для корректной синхронизации произвольных моделей.

Следовательно, возникает проблема построения набора правил, удовлетворяющего требованию полноты и обеспечивающего корректную трансформацию моделей. Наличие такого набора и алгоритма проверки метамоделей позволит решить еще одну задачу, направленную на обеспечение корректной трансформации — выполнение неявной проверки модели при ее трансформации, если набор правил позволяет отображать все возможные структуры, описываемые метамоделью, а сама метамодель при этом корректна, ошибки при трансформации модели будут указывать на некорректность данной модели, т.е. она не является экземпляром метамодели.

В предложенной работе рассмотрены методики разработки процедур синхронизации моделей, основанные на использовании аппарата графовых грамматик [3], так как в настоящее время программные средства для проектирования трансформаций моделей, использующие этот подход, а также его теоретический базис и возможности его практического применения являются наиболее проработанными. Однако, как будет показано далее, такие подходы обладают недостатками, оказывающими существенное влияние на качество процесса разработки.

Постановка задачи синхронизации графовых моделей на базе одной метамодели. Введем следующие обозначения. Любую графовую модель будем описывать шестеркой $G = (V, E, s, t, l, m)$ [10].

Здесь V — конечное множество вершин ($V = \{\nu_1, \nu_2, \ldots\}$); E — множество ребер ($E = \{\langle \nu_i, \nu_j \rangle | \, \nu_i, \nu_j \in V\}$); $s, t : E \to V$ — целевые функции, которые каждому ребру $e = \langle \nu_i, \nu_j \rangle$ из множества E ставят в соответствие его начальную и конечную вершины $s(e) = \nu_i$, $t(e) = \nu_j$; $l : V \to C_V$ и $m : E \to C_E$ — разметочные функции, которые задают метки вершин и ребер, принадлежащие множествам меток вершин C_V и меток ребер C_E.

Между парой моделей

$$G{=}(V, E, s, t, l, m) \quad \text{и} \quad G_1{=}(V_1, E_1, s_1, t_1, l_1, m_1)$$

возможно бинарное несимметричное отношение метамодель–экземпляр, обозначаемое символом "$\triangleright$". Отношение $G \triangleright G_1$ существует, если

$$\begin{aligned}
&(\forall \nu_1 \in V_1) \quad l_1(\nu_1) \in V \ , \\
&(\forall e_1 \in E_1) \quad m_1(e_1) \ \in E, \\
&l_1(s_1(e_1)) = s(m_1(e_1)), l_1(t_1(e_1)) = t(m_1(e_1)).
\end{aligned}$$

Граф G называется метамоделью модели G_1. Отношение метамодель–экземпляр используется в различных языках моделирования (например, Unified Modeling Language (UML) [11] и Ecore [12]) и служит основой для стандартов разработки программного обеспечения (например, Meta Object Facility (MOF) [13]).

Перейдем к постановке задачи синхронизации моделей. Пусть даны модели G_1, G_2, G_1', G_2', причем $G_1' \triangleright G_1$, $G_2' \triangleright G_2$, задано отношение консистентности [2] между моделями, обозначаемое символом "$\Leftrightarrow$". Отношение консистентности $G_1 \Leftrightarrow G_2$ между моделями G_1, G_2 означает, что информация, представленная в моделях G_1, G_2, семантически адекватна с позиции метамоделей G_1', G_2'.

В этом случае синхронизацией модели G_2 с моделью G_1 будем называть такое преобразование P метамодели G_2' с помощью множества правил трансформации T, в результате которого получается модель G_2, совместимая с моделью G_1. Такое преобразование обозначим как $P(G_2', T) = G_2$ и $G_1 \Leftrightarrow G_2$.

Для того чтобы решить главную задачу формирования полного и корректного множества T, в настоящей статье предложен путь, состоящий в решении следующих подзадач.

1. Найти условия, необходимые для формирования полного множества правил трансформации T, которое позволяет для любой пары метамодель–модель осуществить преобразование P.

2. Построить формальную стратегию получения полного и корректного множества T правил трансформации.

3. Доказать полноту стратегии, т.е. возможность получения всех моделей G_2.

Рассмотрим решение перечисленных подзадач в указанном выше порядке.

Условия, необходимые для формирования полного множества правил трансформации. Основная идея формирования полного множества правил трансформации T для преобразования P основана на том, что модели G'_1, G'_2 должны быть совместимыми экземплярами единственной метамодели G, являющейся описанием языка моделирования. В настоящее время известно много программных средств создания правил трансформации моделей, основанных на графовых грамматиках [3, 5]. В основном эти средства представляют собой редакторы и отладчики "ручного" создания правил трансформации без каких-либо гарантий их полноты и непротиворечивости. Главным образом, используются два подхода к разработке множества правил трансформации:

1) задание вручную в редакторе метамоделей синхронизируемых моделей и условий их совместимости. Отладка результатов синхронизации на тестовых примерах;

2) задание примеров синхронизированных моделей, на основе которых автоматически генерируются правила трансформации.

Проверка полноты и непротиворечивости не осуществляется и оставляется на откуп создателя правил трансформации. Процедуры построения правил трансформации по метамодели интуитивны и не формализованы. Ответ на вопрос, действительно ли полученная система правил трансформации достаточна для выполнения синхронизации моделей, остается открытым. Вносимые в модель изменения могут быть ошибочными с позиции описания, заложенного в метамодели, поэтому необходим механизм, который позволит проверять корректность преобразований модели. Созданные вручную правила трансформации могут приводить к некорректной синхронизации при непроработанности стратегии применения правил синхронизации с учетом особенностей метамодели.

Не снижая уровня общности, для решения поставленных подзадач описания моделей и метамоделей будем использовать диаграммы классов языка UML, как наиболее популярного в настоящее время. Диаграмма классов — орграф, каждой вершине которого сопоставляется некоторый класс. Вершины (классы), соединяются дугами, каждая из которых соответствует одному из следующих трех отношений: ассоциация; наследование; композиция. Дугам на диаграмме классов присваивают имя соответствующего им отношения.

Метамодель — граф, задающий различные типы вершин и дуг. Создание моделей по метамодели предполагает анализ ее элементов, который позволяет сформировать полное множество правил трансформации T, допускаемых метамоделью G. При наличии некоторой метамодели G, прежде всего, необходимо выявить, позволяет ли эта метамодель в принципе создать множество T. Анализ метамодели для

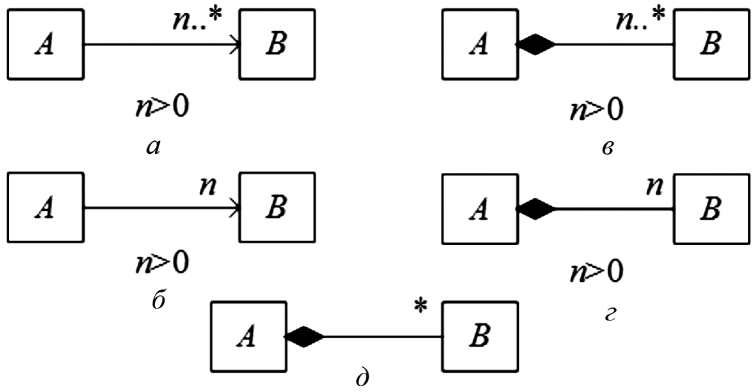

Рис. 1. Примеры зависимостей элементов в диаграммах классов UML

подобного выявления предполагает наличие некоторой стратегии обхода ее элементов.

Примеры зависимости элементов в диаграммах классов UML приведены на рис. 1 [11]. На диаграммах, представленных на рис. 1, *а*, *б*, класс A зависит от класса B, поскольку для его создания необходимо иметь n элементов класса B, а на диаграммах, показанных на рис.1, *в–д*, зависимым является класс B, так как он является частью класса A.

Для анализа влияния зависимостей между элементами метамодели на возможность создания множества T преобразуем граф метамодели $G = (V, E, s, t, l, m)$ в граф зависимостей между элементами $G_{dep} = (V_{dep}, E_{dep}, s_{dep}, t_{dep})$, где $V_{dep} = V$, E_{dep} — множество дуг, зависящее от меток дуг графа метамодели; дуга $\langle \nu_i, \nu_j \rangle$, направленная из вершины ν_i в вершину ν_j, указывает, что вершина ν_j зависит от вершины ν_i (класс ν_j создается после класса ν_i); $s_{dep}, t_{dep} : E_{dep} \to V_{dep}$ — функции, которые каждой дуге графа модели ставят в соответствие инцидентные вершины. В случае UML множество дуг графа зависимостей можно определить как

$$E_{dep} = \begin{cases} e_{dep} : s_{dep}(e_{dep}) = s(e) \wedge t_{dep}(e_{dep}) = t(e), \text{ если } e - \\ \text{композиция} \\ e_{dep} = \varnothing, \text{ если } e - \text{ассоциация с ограничениями} \\ 0..n, 0..^*,^*, n > 0; \\ e_{dep} : s_{dep}(e_{dep}) = t(e) \wedge t_{dep}(e_{dep}) = s(e), \text{ если } e - \\ \text{ассоциация с ограничениями } n, n..*, n > 0. \end{cases}$$

Для того чтобы можно было создать экземпляр метамодели, в полученном графе зависимостей должны отсутствовать контуры. Множество вершин E_{dep} ориентированного графа без контуров можно разбить на непересекающиеся подмножества $N_1, \ldots, N_r$ следующим образом [14]:

$$N_1 = \left\{ v \in V_{dep} | V_{dep}^{-1}(v) = \varnothing \right\},$$
$$N_2 = \left\{ v \in V_{dep} \backslash N_1 | V_{dep}^{-1}(v) \subseteq N_1 \right\},$$
$$N_3 = \left\{ v \in V_{dep} \backslash (N_1 \bigcup N_2) | V_{dep}^{-1}(v) \subseteq N_1 \bigcup N_2 \right\},$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$N_r = \left\{ v \in V_{dep} \backslash \bigcup_{i=1}^{r-1} N_i | V_{dep}^{-1}(v) \subseteq \bigcup_{i=1}^{r-1} N_i \right\},$$

где r — наименьшее число, такое, что $V_{dep} \backslash \bigcup_{i=1}^{r} N_i = \varnothing$.

Полученные подмножества упорядочены так, что, если вершина входит в подмножество с номером i, то следующая за ней вершина в графе входит в подмножество с номером, большим i. Полученные непересекающиеся подмножества называются уровнями. Чтобы выполнить задачи разбиения множества вершин графа зависимостей на уровни может быть использован алгоритм, приведенный ниже:

```
ModelElements = V_dep; LevelElements = ∅; Level = 0;
while ModelElements ≠ ∅ {
    Level = Level + 1;
    LevelElements[Level] = getLevelElements(ModelElements);
    if(LevelElements[Level] = ∅){
      exit_with_error ;
    }
    remove_all(ModelElements, LevelElements[Level]);
}

        getLevelElements(ModelElements){
            result = ∅;
             foreach  e : ModelElements  {
              if(getAscendants(e)  ⊆ V_dep\ModelElements){
                push(result, e);
              }
            }
             return  result;
        }
```

В качестве входных данных алгоритм использует множество вершин графа зависимостей. В цикле выполняется поиск элементов множества V_{dep}, соответствующих текущему уровню (функция $getLevelElements$). Поиск осуществляется среди всех элементов, уровень которых еще не был установлен, согласно приведенному выше определению разбиения множества вершин ориентированного графа без контуров (функция $getAscendants$ возвращает набор вершин, из которых выходит дуга, входящая в текущую вершину e). После того,

как элементы текущего уровня найдены, они удаляются из множества $ModelElements$, затем происходит поиск вершин следующего уровня. Алгоритм работает до тех пор, пока множество $ModelElements$ не станет пустым.

Поскольку заранее не известно, содержит ли граф зависимостей контуры (что свидетельствует о некорректности исходной метамодели), на каждом шаге алгоритма выполняется проверка. Эта проверка основана на том, что, если граф зависимостей содержит контуры, существует некоторое непустое подмножество его вершин, для которых условие разбиения на уровни не выполняется, так как не существует дуг, ведущих из тех вершин предыдущего уровня, в одну из вершин, рассматриваемых на текущем шаге. В этой ситуации функция $getLevelElements$ ничего не вернет.

Аналогично можно построить алгоритм разбиения графа зависимостей на уровни при использовании другого представления графа, например, в виде матрицы инциденций или матрицы смежности [14].

Проиллюстрируем рассмотренный алгоритм на примере. Исходная UML-диаграмма приведена на рис. 2, *а*, полученный на ее основе граф зависимостей — на рис. 2, *б*. Используемый алгоритм представлен ниже.

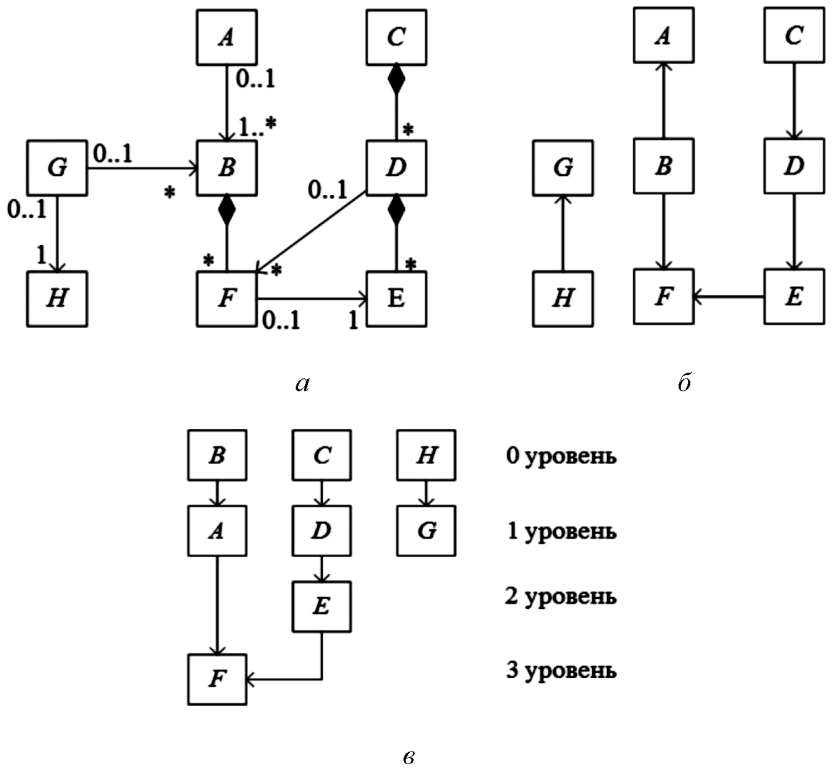

Рис. 2. Пример исходной UML-диаграммы (*а*), полученный на ее основе граф зависимостей (*б*) и результат работы алгоритма (*в*)

Шаг 0. Начало работы алгоритма. $ModelElements = \{A, B, C, D, E, F, H, G\}$.

Шаг 1. Поиск вершин, в которые не входит ни одна дуга (1 уровень): $LevelElements[1]=\{B, C, H\} ModelElements=\{A, D, E, F, G\}$.

Шаг 2. Поиск вершин, в которые ведут дуги из вершин: $LevelElements[1] : LevelElements[2] = \{A, D, G\} ModelElements = \{E, F\}$.

Шаг 3. Поиск вершин, в которые ведут дуги из вершин двух предыдущих уровней: $LevelElements[3] = \{E\} ModelElements = \{F\}$.

Шаг 4. Поиск вершин, в которые ведут дуги из вершин трех предыдущих уровней: $LevelElements[4] = \{F\} ModelElements = \{\}$.

Шаг 5. Множество $ModelElements$ пустое, прекращение работы.

Формальная стратегия получения полного и корректного множества правил трансформации. Эта стратегия состоит из двух этапов: 1) построение по метамодели графа зависимостей, разбиение множества вершин графа зависимостей на уровни; 2) построение множества правил трансформации, соответствующих этому разбиению. Рассмотрим данную стратегию на примере реляционной метамодели, представленной в виде диаграммы классов на языке UML (рис. 3). Граф зависимостей строить нет необходимости, поскольку в таком случае структура его очевидна, и порядок обхода вершин метамодели следующий: $Schema$; $Table$; $Column$; $PrimaryKey$; $ForeignKey$.

Для представления правил трансформации в графическом виде (рис. 4) использованы обозначения, взятые из работы [3]. Закрашенные вершины обозначают условие, необходимое для выполнения правила (на момент трансформации эти элементы должны существовать).

Элемент $Schema$ — исходный элемент, с которого начинается формирование модели, поэтому добавляем правило для создания этого элемента (рис. 4, *а*). Элемент связан отношением композиции с эле-

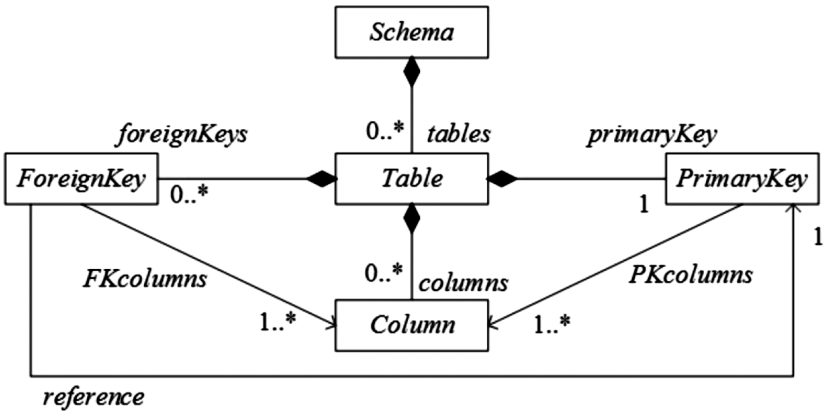

Рис. 3. Пример реляционной метамодели

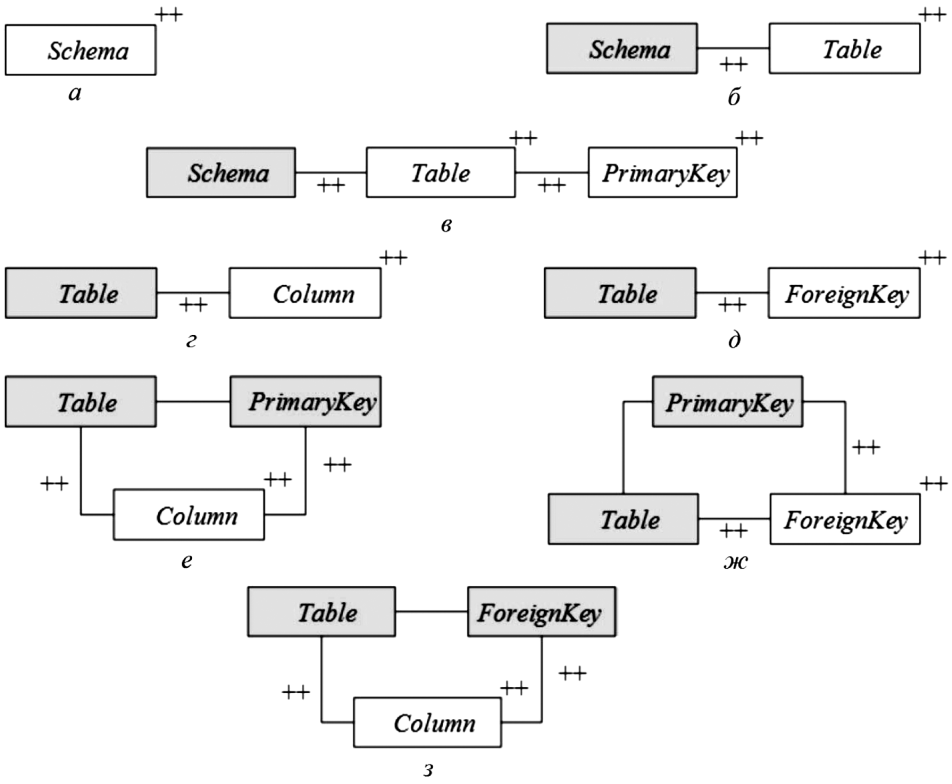

Рис. 4. Графический вид представления правил трансформации, полученных для реляционной метамодели, представленной на рис. 3

ментом $Table$ и согласно семантике композиции отвечает за его создание. Добавляем правило (рис. 4, *б*) и переходим к элементу $Table$. У него есть обязательная композиционная связь $primaryKey$ с элементом $PrimaryKey$, поэтому при создании элемента $Table$ должен автоматически создаваться элемент $PrimaryKey$. Дополняем предыдущее правило и получаем результат, показанный на рис. 4, *в*. Из необязательных связей у элемента $Table$ есть композиционные связи $columns$ и $foreignKeys$ с элементами $Column$ и $ForeignKey$. Добавляем еще два правила (рис. 4, *г*, *д*) и переходим к элементу $Column$. У него нет обязательных связей, а за его создание отвечает элемент $Table$ и соответствующее правило уже есть. Переходим к элементу $PrimaryKey$. При создании ему необходим один экземпляр $Column$, который на момент его создания уже должен существовать. Однако элемент $PrimaryKey$ может содержать ссылки на множество элементов $Column$, а согласно семантике трансформации сначала необходимо трансформировать контейнер, а потом его содержимое. В связи с этим добавляем еще одно правило (рис. 4, *е*). Данное правило служит для трансформации столбцов таблицы, которые входят в состав первичного ключа. Остался последний элемент

— $ForeignKey$. При создании ему требуется ссылка на $PrimaryKey$, поэтому правило, содержащее элемент $ForeignKey$, дополняем элементом $PrimaryKey$ (рис. 4, *ж*). Наконец, по аналогии с элементами $PrimaryKey$ и $Column$ добавляем еще одно правило (рис. 4, *з*). На этом составление списка правил трансформации завершено.

Доказательство полноты стратегии. Согласно описанной стратегии множества правил трансформации, доказательство полноты основывается на следующих соображениях.

1. Создаются элементы всех типов, определяемых метамоделью.

2. В соответствии с семантикой языка UML для каждого элемента формируются все возможные отношения.

3. Для создания правил трансформации рассматриваются все уровни отношений в порядке, необходимом для их использования.

4. Создание правил трансформации осуществляется до тех пор, пока все правила не будут созданы.

Заключение. Предложенные в статье принципы и алгоритмы анализа структуры метамоделей и генерации на этой основе правил трансформации позволяют существенно повысить эффективность разработки систем синхронизации независимо от используемых моделей, снижая временные затраты на тестирование. В отличие от известных подходов, в значительной степени базирующихся на интуитивных и эмпирических приемах обеспечения полноты и корректности трансформации, предложенная методика обеспечивает автоматическое порождение полных и корректных правил трансформации. Дальнейшее развитие предложенной методики позволит создать процедуру автоматической генерации систем синхронизации моделей для каждого конкретного случая используемых моделей.

ЛИТЕРАТУРА

1. *Den Haan Johan*. 15 reasons why you should start using Model Driven Development. Available at: http://www.theenterprisearchitect.eu/archive/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development (accessed: 20.01.2015).
2. *Девятков В.В.*, *Ошкало Д.В.* Разработка процессов синхронизации моделей и принципов проверки их корректности // Инженерный журнал: наука и инновации. 2013. Вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1052.html
3. *Kindler E.*, *Robert W.* Triple Graph Grammars: concepts, extensions, implementations, and application scenarios. Tech. Rep., no. tr-ri-07-284. Software Engineering Group, Department of Computer Science, University of Paderborn, 2007.
4. *Schurr A.* Specification of graph translators with triple graph grammars. Graph-Theoretic Concepts in Computer Science // 20th International Workshop. Herrsching, Germany, 1994. Vol. 903. P. 151–163.
5. *Biehl M.* Literature study on model transformations. Tech. Rep., Royal Institute of Technology, 2010.
6. *Stevens P.* Bidirectional model transformations in QVT: semantic issues and open questions // In International Conference on Model Driven Engineering Languages and Systems (MoDELS 2007). Springer, 2007. Vol. 4735. P. 1–15.

7. *Stevens P.* A landscape of bidirectional model transformations // In Generative and Transformational Techniques in Software Engineering II, International Summer School (GTTSE 2007). Springer, 2008. Vol. 5235. P. 408–424.
8. *Antkiewicz M.*, *Czarnecki K.* Design space of heterogeneous synchronization // In Generative and Transformational Techniques in Software Engineering (GTTSE 2007). Springer, 2008. Vol. 5235. P. 3–46.
9. *Ehrig H.*, *Ehrig K.*, *de Lara J.* Termination criteria for model transformation // International Conference on Fundamental Approaches to Software Engineering (FASE 2005). Springer, 2005. Vol. 3442. P. 49–63.
10. *Habel A.*, *Muller J.*, *Plump D*. Double push-out approach with injective matching // Springer. 2000. Vol. 1764. P. 103–117.
11. *Unified* Modeling Language. URL: http://www.uml.org (дата обращения: 20.01.2015).
12. *Eclipse* Modeling Framework. URL: http://eclipse.org/modeling/emf (дата обращения: 20.01.2015).
13. *Meta*-Object Facility. URL: http://www.omg.org/mof (дата обращения: 20.01.2015).
14. *Галкина В.А*. Дискретная математика: комбинаторная оптимизация на графах. М.: Гелиос АРВ, 2003. 232 с.

REFERENCES

[1] Den Haan Johan.15 reasons why you should start using Model Driven Development. Available at: http://www.theenterprisearchitect.eu/archive/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development (accessed: 20.01.2015).
[2] Devyatkov V.V., Oshkalo D.V. Development of model synchronization processes and principles of their verification. *Jelektr. nauchno-tehn. Izd. "Inzhenernyy zhurnal: nauka i innovacii"* [El. Sc.-Techn. Publ. "Eng. J.: Science and Innovation"], 2013, iss. 11. URL: http://engjournal.ru/catalog/it/hidden/1052.html
[3] Kindler E., Robert W. Triple Graph Grammars: concepts, extensions, implementations, and application scenarios. Tech. Rep., no. tr-ri-07-284. Software Engineering Group, Department of Computer Science, University of Paderborn, 2007.
[4] Schurr A. Specification of graph translators with triple graph grammars. Graph-Theoretic Concepts in Computer Science. *20th International Workshop*. Herrsching, Germany, 1994, vol. 903, pp. 151–163.
[5] Biehl M. Literature study on model transformations. Tech. Rep., Royal Institute of Technology, 2010.
[6] Stevens P. Bidirectional model transformations in QVT: semantic issues and open questions.*In International Conference on Model Driven Engineering Languages and Systems* (MoDELS 2007). *Springer*, 2007, vol. 4735, pp. 1–15.
[7] Stevens P. A landscape of bidirectional model transformations. *In Generative and Transformational Techniques in Software Engineering II, International Summer School* (GTTSE 2007). *Springer*, 2008, vol. 5235, pp. 408–424.
[8] Antkiewicz M., Czarnecki K. Design space of heterogeneous synchronization. *In Generative and Transformational Techniques in Software Engineering* (GTTSE 2007). *Springer*, 2008, vol. 5235, pp. 3–46.
[9] Ehrig H., Ehrig K., de Lara J. Termination criteria for model transformation. *International Conference on Fundamental Approaches to Software Engineering* (FASE 2005). *Springer*, 2005, vol. 3442, pp. 49–63.
[10] Habel A., Muller J., Plump D. Double push-out approach with injective matching. *Springer*, 2000, vol. 1764, pp. 103–117.
[11] Unified Modeling Language. URL: http://www.uml.org (accessed: 20.01.2015).
[12] Eclipse Modeling Framework. URL: http://eclipse.org/modeling/emf (accessed: 20.01.2015).

[13] Meta-Object Facility. URL: http://www.omg.org/mof (accessed: 20.01.2015).
[14] Galkina V.A. Diskretnaya matematika: kombinatornaya optimizatsiya na grafakh [Discrete mathematics: combinatorial optimization on graphs]. Moscow, Gelios ARV Publ., 2003. 232 p.

Статья поступила в редакцию 26.02.2015

Девятков Владимир Валентинович — д-р техн. наук, профессор, заведующий кафедрой "Информационные системы и телекоммуникации" МГТУ им. Н.Э. Баумана. Автор более 120 научных работ (в том числе трех монографий) в области искусственного интеллекта, распознавания образов, принятия решений, логических исчислений, представления знаний, теории конечных автоматов, логического синтеза и анализа дискретных устройств и систем.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

Devyatkov V.V. — Dr. Sci. (Eng.), professor, head of the Information Systems and Telecommunications Department of the Bauman Moscow State Technical University. Author of more than 120 publications (including three monographs) in the fields of logical control, computer systems and complexes of technical cybernetic.
Bauman Moscow State Technical University, 2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Ошкало Дмитрий Владиславович — аспирант кафедры "Информационные системы и телекоммуникации" МГТУ им. Н.Э. Баумана. Автор ряда научных работ в области моделеориентированной разработки программного обеспечения и мультиагентных систем.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

Oshkalo D.V. — postgraduate of the Information Systems and Telecommunications Department of the Bauman Moscow State Technical University. Author of several publications in the fields of model-driven software development and multiagent systems.
Bauman Moscow State Technical University, 2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Просьба ссылаться на эту статью следующим образом:

Девятков В.В., Ошкало Д.В. Принципы полной и корректной трансформации синхронизируемых моделей // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2015. № 3. С. 79–91.

Please cite this article in English as:

Devyatkov V.V., Oshkalo D.V. The principles of complete and correct transformation of the synchronized models. *Vestn. Mosk. Gos. Tekh. Univ. im. N.E. Baumana, Priborostr.* [Herald of the Bauman Moscow State Tech. Univ., Instrum. Eng.], 2015, no. 3, pp. 79–91.