УДК 004.4’236

АЛГОРИТМЫ ГОРИЗОНТАЛЬНОЙ ТРАНСФОРМАЦИИ МОДЕЛЕЙ В СИСТЕМЕ METALANGUAGE

Сухов А.О.

Пермский филиал ФГАОУ ВПО «Национальный исследовательский университет «Высшая школа экономики», Пермь, e-mail: sukhov_psu@mail.ru

В процессе создания и сопровождения информационных систем все чаще используется модельно-ориентированный подход к разработке программного обеспечения, позволяющий сместить фокус с написания кода на языке общего назначения к построению моделей и автоматической генерации исходного кода системы на основе созданных моделей. Однако при использовании такого подхода возникает потребность преобразования моделей, построенных на различных этапах разработки системы различными категориями специалистов с использованием различных языков моделирования. В статье рассмотрены алгоритмы горизонтальной трансформации моделей, описанных с использованием одного визуального предметно-ориентированного языка моделирования, в модели, построенные в иной графической (текстовой) нотации. Компонент системы MetaLanguage, реализующий данные алгоритмы, позволяет выполнять трансформации визуальных моделей вида «модель-текст» и «модель-модель».

Ключевые слова: трансформация моделей, графовые грамматики, визуальные языки моделирования, метамоделирование

ALGORITHMS FOR HORIZONTAL TRANSFORMATION OF MODELS IN METALANGUAGE SYSTEM

Sukhov A.O.

National Research University Higher School of Economics, Perm, e-mail: sukhov_psu@mail.ru

In the process of information systems creation and maintaining the model-based approach to the software development, which allows to shift the focus from code writing on general purpose language to models creation and automatic generation of the system source code on the basis of the created models, is increasingly used. However at usage of this approach there is a need of conversion of models constructed at various stages of system development by various categories of users with usage of different modeling languages. In paper the algorithms for horizontal transformation of models, which are described with usage of one domain-specific modeling language, to models, which are constructed in other graphical (textual) notation is considered. The component of MetaLanguage system, which implements these algorithms, allows to fulfill the transformations of visual models of the type «model-text» and «model-model».

Keywords: model transformation, graph grammars, visual modeling languages, meta-modeling

Создание информационных систем (ИС) с использованием современных инструментальных средств основано на разработке различных моделей, описывающих предметную область ИС, определяющих структуры данных и алгоритмы функционирования системы. Основная идея такого модельно-ориентированного подхода – систематическое использование моделей на различных этапах разработки ИС, что позволяет сместить фокус с написания кода на языке общего назначения к построению моделей и автоматической генерации исходного кода системы и других необходимых артефактов.

Существуют реализации модельно-ориентированного подхода, которые применяют языки моделирования общего назначения для построения моделей ИС. Так, язык моделирования UML вместе со стандартом MOF (Meta-Object Facility) формирует основу для концепции MDA (Model-Driven Architecture). Другие реализации модельно-ориентированного подхода применяют для построения моделей визуальные *предметно-ориентированные языки* (Domain-Specific Language, DSL), предназначенные для решения определенного класса задач в конкретной предметной области. В отличие от языков моделирования общего назначения DSL более выразительны, просты в применении и понятны различным категориям специалистов, поскольку они оперируют привычными для них терминами предметной области. Для поддержки процесса разработки и сопровождения DSL используется специальный вид программного обеспечения – *языковой инструментарий*.

В процессе создания и эксплуатации ИС принимают участие различные категории специалистов (программисты, бизнес-аналитики, эксперты в предметной области и др.), поэтому достаточно часто появляется потребность изменения описания языка моделирования, его настройки на возможности конкретного пользователя, а также необходимость преобразования моделей, построенных различными специалистами на разных этапах создания ИС с использованием различных DSL. Для реализации этих возможностей необходимо, чтобы языковой инструментарий позволял строить

целую иерархию моделей: модель, метамодель, мета-метамодель и т.д., где *модель* – это абстрактное описание на некотором формальном языке характеристик системы (процесса), важных с точки зрения цели моделирования, *метамодель* – модель языка, используемого для создания моделей, а *мета-метамодель* – язык, на котором описываются метамодели. Кроме того, языковой инструментарий должен содержать средства трансформации, позволяющие выполнять преобразование визуальных моделей как между различными уровнями иерархии (*вертикальные* трансформации), так и внутри одного уровня (*горизонтальные* трансформации).

Система MetaLanguage – это языковой инструментарий, предназначенный для создания визуальных динамически настраиваемых предметно-ориентированных языков моделирования [1]. Архитектура данной системы описана в работе [3].

В процессе создания и сопровождения DSL часто появляется потребность экспорта созданных пользователями моделей во внешние системы, которые, как правило, используют стандартные языки моделирования, отличные от DSL, поэтому одним из центральных компонентов системы MetaLanguage является трансформатор.

Анализ наиболее развитых на сегодняшний день языков и инструментальных средств трансформации моделей, приведенный в работе [4], показал, что наиболее подходящим для реализации в системе MetaLanguage является алгебраический подход с одинарным выталкиванием [7], основанный на графовых грамматиках.

Опишем представление графовых грамматик в соответствии с целью создания системы MetaLanguage и математической моделью, лежащей в основе реализации данного языкового инструментария, которая рассмотрена в работах [5, 6].

Горизонтальные трансформации моделей в системе MetaLanguage

Базовым понятием при описании трансформации графов в соответствии с алгебраическим подходом является *продукционное правило*, которое имеет вид $p: L \to R$, где p – имя правила, L – *левая часть правила*, которая называется *паттерном*, а R – *правая часть правила*, которая называется *графом замены*.

Правила применяются к исходному графу, называемому *хост-графом*. Если граф из левой части правила был найден в исходном графе, то правило может быть применено.

Графовая грамматика – это пара $GG = (P, G_0)$, где P – набор продукционных правил; G_0 – начальный граф грамматики.

Пусть дан исходный граф G, а также графы L и R, которые представляют собой левую и правую части продукционного правила $p: L \to R$, причем граф L является подграфом графа G. Тогда *применением правила p* к исходному графу G называется замена в графе G подграфа L на граф R, результатом замены является граф H, причем граф R – подграф графа H.

Графовая трансформация – это последовательное применение к исходному помеченному графу G_0 конечного набора правил

$$P = (p_1, p_1, \ldots, p_n): \; G_0 \overset{p_1}{\to} G_1 \overset{p_2}{\to} \ldots \overset{p_n}{\to} G_n,$$

где G_n – результат выполнения трансформации.

Горизонтальные трансформации в системе MetaLanguage описываются на уровне метамоделей, что предоставляет возможность определять преобразования, которые могут быть применены ко всем моделям, построенным с использованием исходной и целевой метамоделей. Для задания трансформации необходимо указать исходную и целевую метамодели и определить правила, описывающие преобразование [3].

Продукционные правила применяются в соответствии с порядком, заданным пользователем. Пусть система выбрала очередное продукционное правило трансформации и пытается его применить, для этого ей необходимо выполнить два алгоритма: алгоритм поиска паттерна в исходном хост-графе и алгоритм замены левой части правила на правую. Рассмотрим эти алгоритмы подробнее.

Алгоритм поиска паттерна в исходном графе

Существуют различные алгоритмы поиска графа, изоморфного заданному паттерну, наиболее распространенными на практике являются: алгоритм Ульмана, алгоритм Шмидта и Дрюфелла, алгоритм Венто и Фоггиа, Nauty-алгоритм и др. Отличием предлагаемого подхода от решения классической задачи сопоставления графов является то, что в данном случае требуется провести сопоставление графов, принадлежащих различным уровням иерархии моделей, при этом необходимо учитывать типы вершин и дуг графа.

Реализованный в системе MetaLanguage алгоритм поиска паттерна в графе модели является разновидностью алгоритма перебора с возвратом, который имеет экспоненциальную сложность. Алгоритм состоит из трех этапов. При выполнении первого этапа алгоритма будут найдены все экземпляры некоторого произвольного отношения из графа-паттерна, т.е. осуществляется

поиск дуги, с которой начнется выполнение второго этапа алгоритма. На втором этапе требуется найти одно из возможных вхождений экземпляров всех отношений графа-паттерна G_P в граф исходной модели G_S. На третьем этапе в результирующий граф G_T необходимо добавить те вершины графа G_S, прообразы которых принадлежат графу G_P. Далее следует выполнить замену левой части правила на правую. Если исходный граф G_S содержит еще несколько вхождений экземпляров всех отношений графа-паттерна G_P, то для каждого такого вхождения необходимо выполнить данный алгоритм, начиная со второго этапа. Рассмотрим этапы алгоритма поиска паттерна в исходном графе более подробно.

Первый этап – процедура *FindPattern*:

1.1. Очистить множество вершин исходного графа, просмотренных в ходе поиска, – *VisitedEntities*.

1.2. Выбрать в графе-паттерне G_P одно из отношений, обозначим его *rel*. Если таких отношений не оказалось, то перейти к выполнению процедуры *AddNodes* добавления вершин в граф G_T.

1.3. Найти все экземпляры отношения *rel* в исходном графе модели G_S. Множество всех этих отношений обозначим *FoundRelations*.

1.4. Для каждого экземпляра отношения *relI* из множества *FoundRelations* выполнить процедуру *FindSubGraph* для поиска подграфа G_T в графе исходной модели G_S, соответствующего паттерну и содержащего этот экземпляр отношения.

Второй этап – процедура поиска подграфа, содержащего указанный экземпляр отношения, *FindSubGraph* состоит из следующих шагов:

2.1. Добавить в множество дуг результирующего графа G_T дугу *relI*.

2.2. Если после добавления дуги оказалось, что количество дуг графа G_T совпадает с количеством дуг графа-паттерна G_P, то следует перейти к выполнению процедуры добавления вершин в граф G_T (*AddNodes*), а затем вернуться, удалить из графа G_T дугу *relI* и перейти к шагу 1.4, т.к. в исходном графе могут существовать и другие экземпляры отношений того же типа. Иначе перейти к шагу 2.3.

2.3. Приступить к рассмотрению первой вершины $entI_1$, инцидентной ребру *relI*, если она ранее не была рассмотрена:

a. Добавить вершину $entI_1$ в множество *VisitedEntities*.

b. Рассмотреть все входящие в $entI_1$ дуги графа G_S, если прообраз некоторой из них $fr^{-1}(r\,I_i^I)$ принадлежит графу-паттерну G_P и он ранее не рассматривался, то следует произвести поиск подграфа, содержащего экземпляр отношения rI_i^I, начиная со второго этапа описываемого алгоритма.

c. Рассмотреть все исходящие из $entI_1$ дуги графа G_S, если прообраз некоторой из них $fr^{-1}(r\,I_i^O)$ принадлежит графу-паттерну G_P и он ранее не рассматривался, то следует произвести поиск подграфа, содержащего экземпляр отношения rI_i^O, начиная со второго этапа данного алгоритма.

2.4. Приступить к рассмотрению второй вершины $entI_2$, инцидентной ребру *relI*, если она ранее не была рассмотрена. Рассмотрение производится аналогично тому, как это было описано в шаге 2.3.

2.5. Перейти к выполнению процедуры добавления вершин в граф G_T.

Третий этап – процедура добавления вершин *AddNodes* – состоит из трех шагов:

3.1. Рассмотреть все дуги графа G_T. Если прообраз какой-либо инцидентной дуге вершины принадлежит графу-паттерну G_P, то ее следует добавить в множество вершин графа G_T.

3.2. Найти в графе G_S множество вершин, прообразы которых в графе G_P являются изолированными, и добавить их в граф G_T.

3.3. Вызвать процедуру замены левой части правила на правую *LHSReplacement*. Она определяется видом правила трансформации.

Алгоритмы замены левой части правила на правую

После того как в исходном графе был найден подграф из левой части правила, необходимо выполнить его замену на правую часть продукционного правила, при этом алгоритм замены зависит от вида трансформации: имеет трансформация вид «модель-текст» или «модель-модель».

Трансформация вида «модель-текст» предоставляет пользователю системы MetaLanguage возможность по заданным им шаблонам генерировать на основе построенных моделей исходный код на некотором целевом языке программирования, а также другое текстовое представление модели, например, ее описание в формате XML. Левой частью продукционного правила в этом случае является некоторый фрагмент исходной метамодели, а правой честью – текстовый шаблон, содержащий как статические элементы, которые не зависят от найденного паттерна, так и динамические элементы, которые меняются в зависимости от найденного фрагмента исходной модели.

Описание шаблона производится на целевом языке. Для выделения динамической

части используются специальные метасимволы: «<<» (двойные открывающиеся угловые скобки) – для обозначения начала динамической части; «>>» (двойные зарывающиеся угловые скобки) – для обозначения окончания динамической части. Поскольку имена некоторых сущностей и отношений могут совпадать, то для обозначения сущности перед ее именем указываются префикс «С.», а для обозначения отношения – префикс «О.».

При описании трансформации можно задать ограничение на вхождение паттерна. Такое ограничение позволяет определить контекст выполнения правила. Оно содержит условия, которым должен удовлетворять каждый из найденных фрагментов исходной модели. Если какое-либо из условий не выполняется, то правило для данного фрагмента исходной модели не применяется.

Трансформация вида «модель-модель» позволяет произвести преобразование модели, созданной с использованием одного визуального языка моделирования, в модель, описанную на другом визуальном языке.

Левая часть продукционного правила трансформаций такого вида является некоторым фрагментом исходной метамодели, а правая часть правила – некоторым фрагментом целевой метамодели. При описании продукционного правила также необходимо задать правила преобразования атрибутов элементов метамоделей.

Для того чтобы создаваемая модель не содержала висячих ссылок, процесс выполнения правила трансформации начинается с создания экземпляров сущностей и лишь потом создаются экземпляры отношений.

Трансформацию вида «модель-модель» можно декомпозировать на следующие элементарные преобразования: «сущность → сущность», «отношение → отношение», «сущность → отношение», «отношение → сущность».

Будем полагать, что в исходной модели уже найдены экземпляры сущностей и/или отношений паттерна по ранее описанному алгоритму.

При выполнении *преобразования* «*сущность → сущность*» *ee*: $Ent_L \rightarrow Ent_R$ необходимо для каждого вхождения в исходную модель экземпляра сущности левой части правила $EntI_L$ создать в целевой модели новый экземпляр сущности правой части правила $EntI_R$ и выполнить заданные правила преобразования атрибутов. При этом созданный экземпляр сущности будет иметь то же имя, что и найденный по паттерну экземпляр сущности.

Для выполнения *преобразования* «*отношение → отношение*» *rr*: $Rel_L \rightarrow Rel_R$ сначала необходимо в исходной модели найти все экземпляры сущностей $RelI_L.SEI$ и $RelI_L.TEI$, которые соединяет экземпляр отношения $RelI_L$, далее для каждого вхождения таких пар экземпляров сущностей следует найти их образы в целевой модели *fe*($RelI_L.SEI$), *fe*($RelI_L.TEI$) и провести между ними экземпляр отношения из правой части правила $RelI_R$. После этого следует выполнить правила преобразования атрибутов.

При выполнении *преобразования* «*сущность → отношение*» *er*: $Ent_L \rightarrow Rel_R$ для каждого вхождения в исходную модель экземпляра сущности левой части правила $EntI_L$ необходимо найти в этой модели вершины $EntI_S$, $EntI_T$, смежные с этим экземпляром сущности. Обозначим через *Source* и *Target* образы этих вершин в целевой модели. Для каждой пары вершин *Source*, *Target* в целевой модели необходимо провести экземпляр отношения $RelI_R$. Далее следует выполнить заданные правила преобразования атрибутов.

Трансформация «*отношение → сущность*» *re*: $Rel_L \rightarrow Ent_R$ преобразует найденный в исходной модели экземпляр отношения паттерна $RelI_L$ в экземпляр сущности целевой модели $EntI_R$. При выполнении преобразования для каждого вхождения экземпляра отношения левой части правила в исходную модель необходимо в целевой модели создать экземпляр сущности $EntI_R$, выполнить заданные правила преобразования атрибутов. Имя экземпляра сущности $EntI_R$ будет совпадать с именем экземпляра отношения $RelI_L$. На следующем шаге следует для каждого вхождения в исходную модель экземпляра отношения левой части правила найти в этой модели экземпляры сущностей $RelI_L.SEI$, $RelI_L.TEI$, соединяемые этим экземпляром отношения. Далее необходимо создать экземпляры отношений, соединяющие экземпляр сущности $EntI_R$ с вершинами *Source* и *Target*, которые являются образами вершин $RelI_L.SEI$ и $RelI_L.TEI$ в целевом графе, с сохранением направления экземпляра отношения $RelI_L$.

Сложность алгоритмов замены левой части продукционного правила на правую полиномиальна.

Заключение

Рассмотренные в статье алгоритмы трансформации визуальных моделей, реализованные в системе MetaLanguage, позволяют выполнять горизонтальные преобразования моделей, описанных с использованием одного визуального языка моделирования в иную текстовую/графическую нотацию. Трансформатор предоставляет возможность не только генерировать

код на целевом языке программирования на основе моделей, созданных пользователем с использованием предметно-ориентированных языков, но и выполнять преобразование этих моделей в стандартные нотации с целью их экспорта в сторонние системы для дальнейшей обработки, например, для проведения имитационного моделирования. В работе [2] приведен пример описания и выполнения трансформаций моделей систем массового обслуживания, выполненных с помощью предметно-ориентированного языка имитационного моделирования, в код на языке GPSS.

Работа подготовлена при финансовой поддержке РФФИ (*проекты* № 12-07-00763-*а,* № 14-07-31330-*мол_а*).

Список литературы

1. Замятина Е.Б., Лядова Л.Н., Сухов А.О. Мультиязыковое моделирование с использованием DSM платформы MetaLanguage / Информатизация и связь. – 2013. – № 5. – С. 11-14.

2. Замятина Е.Б., Лядова Л.Н., Сухов А.О. О подходе к интеграции систем моделирования и информационных систем на основе DSM-платформы MetaLanguage / Технологии разработки информационных систем ТРИС-2013: материалы IV межд. научно-технической конференции. – Таганрог: Изд-во Технол. инст. ЮФУ, 2013. – Т. 1. – С. 61 –70.

3. Сухов А.О. Инструментальные средства создания визуальных предметно-ориентированных языков моделирования / Фундаментальные исследования. – 2013. – № 4 (ч. 4). – С. 848-852.

4. Сухов А.О. Методы трансформации визуальных моделей / Технологии разработки информационных систем ТРИС-2012: материалы III межд. научно-технической конференции. – Таганрог: Изд-во Технол. инст. ЮФУ, 2012. – Т. 1. – С. 120-124.

5. Сухов А.О. Теоретические основы разработки DSL-инструментария с использованием графовых грамматик // Информатизация и связь. – 2011. – № 3. – С. 35-37.

6. Сухов А.О. Формальное описание метаязыка системы MetaLanguage / Современные проблемы математики и ее прикладные аспекты: труды всерос. научно-практической конференции. – Пермь: Изд-во Перм. гос. ун-та, 2010. – С. 154-159.

7. Algebraic Approaches to Graph Transformation. Part II: Single Pushout Approach and Comparison with Double Pushout Approach // H. Ehrig, R. Heckel, M. Korff [et al.] / Handbook of Graph Grammars and Computing by Graph transformation. – 1997. – Vol. 1. – P. 247-312.

References

1. Zamyatina E.B., Lyadova L.N., Sukhov A.O. O podkhode k integratsii sistem modelirovaniya i informatsionnykh sistem na osnove DSM-platformy MetaLanguage / Informatization and Communication, 2013, no. 5, pp. 11-14.

2. Zamyatina E.B., Lyadova L.N., Sukhov A.O. Metody transformatsii vizualnykh modeley. Proc. of the 4th International Conference “Information Systems Development Technologies”. Gelendzhik, 2013, pp. 61-70.

3. Sukhov A.O. The Language Workbench for Visual Domain-Specific Modeling Languages Creation / Fundamental research, 2013, no. 4 (Part 4), pp. 848-852.

4. Sukhov A.O. Metody transformatsii vizualnykh modeley. Proc. of the 3th International Conference «Information Systems Development Technologies». Gelendzhik, 2012, pp. 120-124.

5. Sukhov A.O. Teoreticheskiye osnovy razrabotki DSL-instrumentariya s ispol’zovaniyem grafovykh grammatik / Informatization and Communication, 2011, no. 3, pp. 35-37.

6. Sukhov A.O. Formalnoye opisaniye metayazyka sistemy MetaLanguage. Proc. of the All-Russian Scientific Conference «Modern problems of mathematics and its applications». Perm, 2010, pp. 154-159.

7. Algebraic Approaches to Graph Transformation. Part II: Single Pushout Approach and Comparison with Double Pushout Approach // H. Ehrig, R. Heckel, M. Korff [et al.]/ Handbook of Graph Grammars and Computing by Graph transformation, 1997, Vol. 1, pp. 247-312.

Рецензенты:

Пенский О.Г., д.т.н., доцент, профессор кафедры процессов управления и информационной безопасности Пермского государственного национального исследовательского университета, г. Пермь;

Румянцев А.Н., д.ф.-м.н., профессор, генеральный директор ООО «Информационные бизнес-системы Пермь», г. Пермь.

Работа поступила в редакцию 27.01.2014.