Использование графовых грамматик для трансформации моделей

Александр Сухов[1], Александр Серый[2]

[1]ПГНИУ, Пермь, Россия. Sukhov.PSU@gmail.com
[2]ПГНИУ, Пермь, Россия. SerAlexandr@bk.ru

Аннотация. В статье рассматриваются вопросы использования графовых грамматик для трансформации моделей предметной области из одной нотации в другую. Предложен подход к реализации инструментального средства для создания языков описания трансформаций визуальных моделей.

Ключевые слова: графовые грамматики, трансформации визуальных моделей, предметно-ориентированные языки.

Введение

В любую развитую CASE-систему встроен целый набор языков моделирования различных уровней: языки описания моделей предметных областей, схем баз данных, бизнес-логики и др. При этом широко используются визуальные языки, т.к. графические нотации позволяют строить наглядные, легко воспринимаемые различными категориями пользователей модели.

В настоящее время не существует единого универсального визуального языка создания программного обеспечения [1]. Сейчас активно используются на практике такие языки визуального моделирования, как UML (Unified Modeling Language) и ERD (Entity Relationship Diagram) – для моделирования предметных областей; IDEF (Integrated DEFinition), DFD (Data Flow Diagram), EPC (Event-Driven Process Chain),

Выходные данные сборника.

BPEL (usiness Process Execution Language) и BPML (Business Process Modeling Language) – для моделирования бизнес-процессов на разных уровнях и т.п. Для многих областей деятельности ставится задача создания информационных систем, которые допускали бы участие экспертов (не программистов) как в разработке системы, так и в настройке ее с учетом меняющихся условий эксплуатации, потребностей бизнес-процессов и конкретных пользователей. При этом каждому специалисту необходимо обеспечить возможность работы в привычных для него терминах знакомой ему предметной области [2].

Предметно-ориентированные языки и *языковые инструментарии*, предназначенные для их создания, позволяют решить эту задачу, однако при этом остро встает проблема трансформации построенных моделей, т.к. большинство наиболее развитых на сегодняшний день языковых инструментариев не позволяет производить трансформацию моделей из одной нотации в другую. Эта проблема возникает, когда появляется необходимость определения одной и той же графической модели с помощью различных нотаций, например, представление модели предметной области с помощью диаграмм «Сущность-Связь» и диаграмм классов UML. Проблема трансформации актуальна, когда с одной и той же моделью работают сразу несколько человек, причем каждый из них желает использовать наиболее удобное для него представление модели. Кроме того, различные CASE-средства используют разные языки моделирования, поэтому для экспорта модели из одного CASE-средства в другое необходимо выполнить преобразование модели.

Трансформации моделей

Для задания синтаксиса визуальных языков используются различные формализмы: автоматные модели, алгоритмические сети, но наиболее распространенными являются графовые грамматики [3]. Графовые грамматики – обобщение грамматик Хомского на графы.

Графовой грамматикой называется четверка $\left(T, N, P, S\right)$,

где T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил вида $L \to R$, причем L – непустая последовательность терминальных и нетерминальных символов, содержащая хотя бы один нетерминальный символ, R – любая последовательность терминальных и нетерминальных символов,

$S \in N$ – стартовый (начальный) символ.

Как видно из определения графовая грамматика содержит правила, которые состоят из левой и правой частей. Правила применяются для

исходного графа, называемого хостом. Если подграф в левой части правила был найден в исходном графе, то правило может быть применено. Применение правила означает замену найденного в хосте подграфа графом, расположенным в правой части правила.

Пусть даны четыре помеченных графа G, H, L, R, причем граф L является подграфом графа G, а граф R является подграфом графа H. *Применением правила* $L \to R$ к исходному графу G называется замена в графе G подграфа L на граф R, результатом замены будет граф H.

Список правил замены упорядочивается в соответствии с выбранным способом, например, по приоритетам. При этом после выполнения какого-либо правила поиск следующего применимого правила начинается с начала списка, где расположены более приоритетные правила. Процесс заканчивается, когда в списке правил не осталось ни одного правила, которое может быть применено. Существуют и другие способы упорядочивания правил, так в некоторых системах используется механизм управления, чтобы явно указать какое из правил должно быть применено следующим. Для этого каждому правилу присваивается уникальное имя [4].

Если обобщить классическое определение графовой грамматики, то в качестве правой части правила может выступать не только помеченный граф, но и код на каком-либо языке программирования, а также фрагмент визуальной модели, описанной в другой нотации. Именно поэтому графовые грамматики могут использоваться как для порождения синтаксически правильных моделей, так и для рефакторинга существующих моделей, генерации кода и трансформации моделей с одного языка моделирования в другой [5].

Графовая трансформация – это последовательное применение к исходному помеченному графу G_0 конечного набора правил $P = (p_1, p_2 \dots p_n) : G_0 \Rightarrow_{p_1} G_1 \Rightarrow_{p_2} \dots \Rightarrow_{p_n} G_n$.

Можно выделить два основных вида трансформации моделей: вертикальные и горизонтальные. *Вертикальные* трансформации управляют ходом преобразования моделей при переходе от одного уровня абстракции к другому, например, при отображении объектов метамодели на объекты модели предметной области.

Для создания модели всей системы недостаточно проанализировать все части в отдельности, нужно рассматривать и анализировать систему в целом. Для этого в многопарадигмальном моделировании все части модели должны конвертироваться в общий формализм, в котором и будет происходить анализ. Такая конвертация называется *горизонтальной* и для ее создания нужно описать правила трансформации графов-моделей. Причем для описания вертикальных и горизонтальных транс-

формаций могут быть использованы как разные методы и средства, так и общие.

Построение трансформаций, удовлетворяющих заданным требованиям, само по себе является нетривиальным процессом, который может занимать довольно длительное время. Для проверки правильности описанных трансформаций должна существовать возможность их отладки. Причем отладка должна быть наглядной и простой. Наиболее логичным решением в данном случае является управляемое пошаговое исполнение трансформации правило за правилом.

Существуют различные подходы к трансформации моделей, некоторые из них имеют формальную основу, так технология GReAT использует для трансформации тройные графовые грамматики, а некоторые – применяют технологии из других областей программной инженерии, например, метод программирования на примерах.

Графовые грамматики являются визуальным средством описания трансформаций, что расширяет границы их применимости. В основе графовых грамматик лежит мощный математический аппарат – аппарат графов и аппарат формальных грамматик. Это дает возможность автоматической верификации и валидации построенных трансформаций.

Язык трансформации ATL (ATLAS Transformation Language) является частью архитектуры управления моделью ATLAS. ATL – язык, позволяющий описывать трансформации любой исходной модели в указанную целевую модель. Преобразование производится на уровне метамоделей – моделей языка моделирования. Интеграция в рамках одного языка декларативного и императивного подхода позволяет производить преобразования для сложных предметных областей, при этом декларативный подход скрывает сложность алгоритмов трансформации. К недостаткам данного подхода следует отнести высокие требования, предъявляемые к разработчику преобразования. Кроме того, использование императивных конструкций элиминирует преимущества декларативного подхода. Еще одним недостатком является однонаправленность правил трансформации.

GReAT (Graph REwriting And Transformation) – язык описания преобразований модели, базирующийся на подходе тройных трансформаций графа. GReAT является языком описания трансформаций моделей, который использует диаграммы классов UML и язык OCL для представления преобразований. Данный подход позволяет производить трансформацию сразу для нескольких исходных метамоделей, что является значительным преимуществом по сравнению с другими средствами трансформации. Однако это средство трансформации моделей обладает также и недостатками: у пользователя отсутствует возможность выбора

языка спецификации метамоделей; трансформации в GReAT однонаправлены; отсутствует единая математическая основа описания графов.

AGG (Attributed Graph Grammar) – средство описания графовых грамматик для типизированных атрибутных графов, которое поддерживает трансформацию графов. AGG использует для описания исходной и целевой моделей ориентированные типизированные атрибутные графы, что позволяет применять данный инструментарий практически в любой предметной области. Благодаря использованию в качестве формальной основы алгебраического подхода к трансформации графов существует возможность парсинга графа, проверки графовой модели на противоречивость. Расширение возможностей графовых грамматик средствами языка Java позволяет приблизить формальную модель к предметной области и реализовать сложные функции поведения системы. Преобразования модели задаются правилами перезаписи графа, которые применяются недетерминировано до тех пор, пока ни одно из них не может быть больше выполнено. Если необходимо явно указать порядок применения правил, то пользователь может сгруппировать их по уровням.

VIATRA – основанный на правилах и паттернах, язык преобразования для управления графовыми моделями, который комбинирует в единую парадигму спецификации два подхода: математический формализм, основанный на правилах трансформации графа, для описания моделей и абстрактные конечные автоматы, предназначенные для описания потока управления. Благодаря использованию конструкций конечных автоматов разработчикам удалось значительно повысить семантику стандартных языков описания паттернов и преобразования графов. Кроме того, мощные конструкции языка позволяют производить многоуровневое метамоделирование предметных областей. К преимуществам VIATRA следует отнести, во-первых, возможность использования универсальных метапреобразований, которые позволяют производить многоуровневое метамоделирование и повторное использование уже созданных алгоритмов трансформаций. Во-вторых, основанный на шаблонах метод генерации кода для преобразований вида «модель−код» позволяет генерировать файлы, содержащие как статичные части, так и изменяемые данные. Существенным недостатком VIATRA является отсутствие возможности задания двунаправленных трансформаций. Интеграция реализации данного подхода в систему Eclipse накладывает на него ряд ограничений, присутствующих у этой платформы.

Подход трансформации модели на примере (MTBE) основан на подходах программирования на примере (PBE) и формировании запросов на примере (QBE). Основная цель MTBE – автоматическая генерация правил трансформации на основе начального набора обучающих примеров. Основное преимущество MTBE заключается в том, что для

получения правил преобразования пользователю не требуется знание какого-либо языка трансформации моделей, будь то графовые трансформации, ATL и др., поскольку в качестве такого языка выступают концепты исходной и целевой моделей, т.е. генерировать правила преобразования может конечный пользователь, владеющий информацией лишь о нотации языка моделирования. Текущие реализации подхода MTBE позволяет выполнять лишь полные эквивалентные отображения, не учитывая сложные преобразования атрибутов. Еще одним существенным недостатком большинства реализаций MTBE является то, что в них не затрагивается проблема трансформации ограничений, налагаемых на концепты и отношения модели.

Система графовых грамматик может представляться в компьютере в виде набора пар графов, описывающих левую и правую части продукции. Однако, возможен и другой подход, реализованный, например, в системе GReAT. Обе части продукции объединяются в один граф, вершины и дуги которого имеют специальные служебные метки, которые сообщают системе, относится ли эта вершина/дуга к левой части правила (т.е. будет удаляться при работе системы), к правой (т.е. будет создаваться) или же она входит в обе части правила (т.е. не подвергнется изменениям). Такой подход избавляет от дублирования информации.

При применении одного из правил грамматики к имеющемуся хост-графу G система должна найти в нем подграф, изоморфный левой части правила. Однако нахождение такого подграфа ничего не говорит о возможности применения правила: часть элементов, которые будут удалены из графа, может быть связана с элементами, не вошедшими в подграф. Тогда встанет вопрос – что делать с ребрами, связывающими две части графа? Эта проблема получила название проблемы висячих ссылок. Для ее решения предложено два подхода:

1. Удалять эти ребра вместе с вершинами.
2. Запрещать применять правило.

Однако, эти два подхода не универсальны, возможны ситуации, когда для корректной работы системы необходима их комбинация.

Графовые грамматики позволяют наглядно описывать преобразования, которые должны производиться в системе при выполнении над ней заданных в грамматике операций. На практике чаше встречается использование данного подхода для задания гомогенных преобразований, однако ничто не мешает применять его для трансформации моделей из одной нотации в другую.

Для поиска подграфа изоморфному искомому графу в предлагаемом подходе используется генетический алгоритм Венто и Фоггиа [6].

Выводы

Подводя итог, можно говорить о том, что с каждым годом интерес научного сообщества к проблеме трансформации моделей растет, однако существующие методы преобразования моделей обладают значительными недостатками. Одним из подходов к решению этих проблем является разработка языка трансформаций, который оперирует конструкциями понятными пользователям – непрофессиональным программистам. Такой язык должен позволять изменять свое описание во время работы системы. В статье рассмотрен подход, используемый в инструментарии для создания языков описания трансформаций предметно-ориентированных языков моделирования. Созданный инструментарий планируется интегрировать в инструментальное средство описания визуальных предметно-ориентированных языков MetaLanguage [7].

Список источников

1. Сухов А.О. Решение проблем традиционного подхода к разработке информационных систем на основе использования предметно-ориентированных языков // Материалы конференции «Технологии Microsoft в теории и практике программирования». – 2009. – С. 180-181.

2. Сухов А.О. Предметно-ориентированный язык в адаптируемых информационных системах // Материалы конференции «Технологии Microsoft в теории и практике программирования». – 2008. – С. 25-26.

3. Сухов А.О. Анализ формализмов описания визуальных языков моделирования // Современные проблемы науки и образования. – 2012. – № 2; URL: www.science-education.ru/102-5655 (дата обращения: 01.08.2012).

4. Rekers J., Schuerr A. A Graph Grammar approach to Graphical Parsing // Visual Languages, Proceedings, 11th IEEE International Symposium. – 1995. – P. 195-202.

5. Courcelle B. Graph Rewriting: An Algebraic and Logic Approach // Handbook of Theoretical Computer Science. – 1990. – Vol. B. – P. 193-242.

6. Vento M., Foggia P., Sansone C. An Improved Algorithm for Matching Large Graphs // IAPR-TC-15 International Workshop on graphbased Representations. – 2001. – №3. – P. 193-212.

7. Сухов А.О. Среда разработки визуальных предметно-ориентированных языков моделирования // Математика программных систем: Межвузовский сборник научных трудов. – 2008. – С. 84-94.