УДК 004.4’244
https://doi.org/10.36906/AP-2020/02

РЕАЛИЗАЦИЯ ТРАНСФОРМАЦИИ МОДЕЛЕЙ НА ОСНОВЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Черкашин Е. А.
канд. техн. наук,
Институт динамики систем
и теории управления им. В.М. Матросова СО РАН
г. Иркутск, Россия

Исследование поддержано проектом ФАНО № АААА-А17-117032210079-1

Аннотация. Рассматривается задача представления трансформации моделей программного обеспечения на объектно-ориентированном логическом языке Logtalk, преимущества использования этого языка для манипулирования системой знаний. Приводится сравнение с существующими популярными технологиями и пример реализации фрагмента процедуры трансформации.

Ключевые слова. Model driven architecture, логическое программирование, объектно-ориентированное программирование, манипулирование базой правил.

Существуют предметные области, обладающие высокой динамикой и неопределенностью свойств и структур, пример, ведение данных медицинской диагностики (электронной карты пациента), топливно-энергетический комплекс страны. Даже в случае стабильной структуры предметной области существуют схемы организации разработки, где необходим постоянный возврат на ранние этапы жизненного цикла программного обеспечения, например, при применении экстремального и agile-программирования («проворного» программирования). В таких условиях очень важно быстро распространять изменения, внесенные в дизайн системы, на уровень исходного кода подсистем, причем необходимо, чтобы изменения в разные подсистемы не противоречили друг другу. Анализ процесса реализации проектных решений для стандартных нотаций моделирования программных систем (ПС), например, UML, для конкретных программных платформ показывает, что творческая процедура реализации, в целом, повторяется согласно одному и тому же общему сценарию. Повторяющиеся операции имеет смысл автоматизировать, однако автоматизация творческой деятельности является нетривиальной задачей.

Методы автоматизации анализа абстрактных представлений моделей ПС, и реализация программного кода основываются на анализе структур объектов и отношений между ними в исходных моделях и последующим порождением производных конкретизированных моделей, и, затем уже, исходного кода. Такие трансформации являются предметом изучения *инженерии программного обеспечения (ПО), основанного на моделировании* (Model Driven Engineering, MDE), относительно нового направления разработки ПО. Одной из основных и сложных задач, решаемой в MDE, является распространение изменений (change propagation). В интуитивной постановке задача состоит в том, чтобы сравнить две версии модели и внести соответствующие изменения в остальные модели. В идеале, изменения человеком могут быть внесены в модель любого уровня

абстракции, включая сгенерированный ранее исходный код, и должны быть автоматически распространены (propagated) в остальные модели.

Более простая задача в MDE — это порождение исходного кода подсистем по набору исходных абстрактных моделей. В качестве исходных моделей выступают:

1. структуры базы данных,
2. ER-диаграммы,
3. наборы диаграмм UML.

Для баз, данных (1) и их конкретных структур (2) разрабатываются методы интерпретации, *основанные на данных* (Data Driven Engineering, DDE), для абстрактных моделей (3) — архитектуры, основанные на моделировании (Model Driven Architecure, MDA). MDA является предметом нашего НИРОКР.

MDA реализует трансформацию поэтапно. Исходный код подсистем порождается на основе *платформонезависимой модели* (Platform Independent Model, PIM), допускающей его порождение алгоритмически. PIM порождается из модели программной системы, где не представлены свойства целевой платформы реализации, *платформонезависимой модели* (Platform Independent Model, PIM), ее задача — представлять структуры данных и вычислительный процесс в абстрактном виде. Эта модель, в свою очередь, полностью или частично порождается из *вычислительно-независимой модели* (Computationally Independent Model, CIM), которая, по сути, абстрактная модель предметной области. Трансформации реализуются на основе *модели платформы* (Platform Model, PM), представляющей свойства платформы, методики реализации и традиции программирования конкретной группы разработчиков. Именно наличие модели платформы отличает MDA от использованного в 90-х годах CASE-подхода (Computer-Aided Software Engineering). В основе парадигмы MDA находится возможность совершенствования моделей трансформации параллельно с основной разработкой программной системы.

Основная задача исследования — разработка инструментальных средств представления и трансформации CIM, PIM, PSM и PM, позволяющие создавать каркасы программных систем и, в частности, информационных систем (ИС), разработка последних весьма популярна и также наиболее стандартизирована плоть до отдельных поведенческих сценариев разработчиков. В данной статье представлена методика использования объектно-ориентированного логического программирования и средств Семантического Веба для описания и реализации сценариев трансформации в MDA.

Представление процедуры трансформации

Трансформация моделей осуществляется согласно сценариям, последовательностям логически связанных этапов. Например, при разработке ИС [2], первым делом порождается логическая и физическая модели базы данных (БД). Затем создается собственный уровень ORM, сопрягающий реляционные структуры БД с объектами Zope-2.0. Третьим этапом является синтез специализированных методов, интерпретирующих, например, отношения некоторых таблиц многие-к-одному со справочниками: lookup-функции, фильтры. Параллельно синтезируются текстовый шаблон XML для представления данных из БД унаследованной версии ПС, и программа на языке C для импорта структур на основе этого шаблона. Данный формат кроме импорта обеспечивал и транспортный уровень при обмене информации с подсистемами. Затем, БД заполнялась данными об объектной структуре приложения, метаинформацией, которая потенциально могла бы быть использована триггерами. Также объектная модель БД представлена в виде объектов Zope для использования в представлении форм ввода первичной информации. Таким образом,

сценарий порождал одиннадцать логически зависимых подсистем, 110 классов приложения с их отображением в реляционные таблицы.

По сути, сценарии — это иерархические древовидные структуры, где каждый уровень отражает определенный уровень абстракции процесса трансформации. В качестве языка представления сценариев используется объектно-ориентированная надстройка, макропакет, Logtalk, порождающая программу на языке Prolog. Каждый узел сценария — это объект Logtalk, инициирующий действия на нижних уровнях. Последовательность этапов задается последовательностью правил-методов узла-объекта. В листовых вершинах дерева-сценария осуществляется анализ структуры исходной модели и порождение производных структур.

Язык Пролог с его ООП надстройкой оказался мощным инструментом для представления PM. Благодаря его простой декларативной структуре и возможности реализовывать различные интерпретации этих структур, он позволяет представлять все аспекты трансформации, начиная с импорта и хранения в оперативной памяти исходных моделей, распознавания и синтеза структур, до порождения текстов исходного кода. И, в отличие от стандартных подходов, базирующихся на использовании специальных языков трансформации таких, как ATL и QVT [4], Prolog позволяет не только выполнять трансформации, но и выходить за их рамки: использовать внешние программные библиотеки, не имеющие прямого отношения к MDA. Logtalk выводит все на новый уровень, позволяя при помощи:

1. инкапсуляции создавать фасадные объекты к исходным данным, выполняющие функции адаптеров, скрывающих сложные алгоритмы за их спецификациями в интерфейсе;
2. наследования манипулировать наборами правил баз знаний;
3. композиции реализовывать типичные сценарии решения задач в листовых вершинах;
4. перехвата сообщений фильтровать распознанные объекты в зависимости от контекста трансформации.

К такому достаточно универсальному варианту представления мы пришли в результате исследований различных подходов к заданию сценариев и правил трансформации, комбинируя различные языки программирования. Интересным решением был вариант [2], где правила представлялись комбинацией распознавателя, реализованного в виде — пролог-предиката, а процедура трансформации реализовалась на языке Python как тело соответствующего метода. Параметры метода предавались в параметры предиката. Тело метода запускалось заново для каждого ответа распознающего предиката. Сценарии реализовались в виде списков объектов и методов, последовательный запуск которых реализовывал MDA. Для манипулирования наборами правил использовалась композиция на основе подмешивания (mixin) классов, не имеющих своего состояния. Такой механизм реализовывал подход на основе типовых конфигураций (pattern-directed programming).

Использование, Prolog-а в качестве языка задания распознающей процедуры позволяло абстрагироваться от некоторых технических деталей, в частности, рассмотрения «правых» и «левых» отношений между объектами диаграммы классов, анализировать только имеющие практический смысл комбинации свойств объектов, а не все возможные комбинации. В Python синтезировались структуры PIM и исходный код. Анализ полученного кода показал, что текст тела правила по своей сути тоже декларативная конструкция, и по большому счету, Python не давал значительных выразительных инструментов, за исключением богатого набора механизмов шаблонов. Много в тексте занимали различные отображения простых типов, данных одних моделей и подсистем в другие. В результате было принято решение полностью перейти на логический язык, но при этом хотелось

сохранить объектную структуру, что нам и позволил сделать макропакет Logtalk. Переход на один язык также позволил решить проблему конфликтов в менеджменте динамической части оперативной памяти процесса-трансформатора: и Python и Prolog являются динамическими языками, выделяющим память под новые объекты и освобождая ненужные.

Представление исходных данных CIM, PIM

Другой задачей является представление исходных данных моделей. В [2] исходная модель в формате XMI-1.2 обрабатывалась специальной библиотекой, в результате работы, которой создавались Python-объекты — модели. Затем, при помощи рекурсивной процедуры структуры этих объектов преобразовывалась в набор фактов Prolog. Предикаты распознавания анализировали комбинации фактов. Синтезирующая часть правила пользовалась как объектным представлением модели, так и ссылками на объекты, распознанные предикатами. Новые структуры добавлялись и к объектам, и к набору фактов пролога. Исходный код порождался на основе окончательной структуры объектов при помощи шаблонов текста.

В новой версии инструмента MDA объектная структура, как дублирующая, потеряла смысл, т.к. Logtalk предоставляет средства реализации фасадов. Исходная модель PIM, в формате XMI-2.0 процедурой обхода в глубину преобразуется в набор троек <субъект, отношение, объект>, хранимых в графе Семантического Веба. Эффективный доступ к тройкам обеспечивается при помощи низкоуровневого слоя библиотеки SWI-Prolog rdflib, реализованного на языке C. PSM в настоящее время представляется в виде объектов Logtalk, хранящих состояние (stateful objects). Использование объектов, сохраняющих состояние снижает позитивный эффект от Logtalk-макропакета, т.к. такой код не может быть статически проанализирован и скомпилирован в Prolog. Поэтому в следующих версиях системы необходимо постепенно избавляться от таких объектов и хранить PSM в виде троек.

Абстрактная модель CIM сначала не находила достойного места в системе моделей, с которыми мы имели дело на практике пока не появились задачи моделирования технических объектов, результатом решения которых является синтез экспертной системы [5]. В другой задаче CIM представляет структуру модулей биоинформатического прикладного пакета Mothur. CIM также представляется семантическим графом, заполняемого тройками, представляющими структуру Mothur. Тройки создаются программой анализа кода пакета, функционирующего на основе распознавания текстовых структур регулярными выражениями.

Преимущества нового подхода следующие. Модели CIM и PIM теперь могут ссылаться на внешние объекты, хранимые в распределенных базах данных и знаний, таких как, DBPedia и Wordnet. Это, в свою очередь, позволяет использовать эти данные в процессе трансформации. Например, из DBPedia можно получить локализованные названия ресурса, подсказки для полей ввода в виде частей статей Wikipedia. Объекты, осуществляющие трансформацию, получают данные с этих ресурсов при помощи SPARQL-запросов. Теперь не важно в какой версии формата XMI представлена исходная модель, т. к. реализация адаптера-фасада, благодаря наследованию, задача достаточно простая.

Программирование трансформации

Ограничения объема доклада позволяют разобрать только один пример представления процедуры трансформации в PM, причем это только абстрактный пример, не учитывающий какую-либо специфику платформы и приемов программирования.

% ?- tr_package(sample,lp,cp)::tr(class,person,ID).

- **object**(tr_package(_Package_, _Local_, _Global_)).
tr(class, Class, ClassID):-
% Метод - трансформатор класса
query(_Package_)::class(Name, ClassID), *% Распознавание класса* ::new(Class, [instantiates(class)]), *% создание экземпляров* -
::new(Attributes,[instantiates(attributes)]), *% списков элементов*
::new(Methods, [instantiates(methods)]), *% класса*
Class::name(Name), *% Установить название класса.*
forall(::tr(attribute, Attribute, ClassID, _), *% Трансформировать*
Attributes::append(Attribute)), *% его атрибуты* **forall**(::tr(method, Method, ClassID,_), *% и методы*
Methods::append(Method)),
Class::attributes(Attributes), *% Установить структурные*
Class::methods(Methods). *% элементы класса* :- **end_object**.
:- **object**(query(_Graph_)). *% Часть объекта-фасада, распознающего* class (Name, ID):-
% элементы модели как классы
_Graph::rdf(ID, rdf:type, uml:*'Class'*), _Graph_::rdf(ID, rdfs:label, literal(Name)).
:- **end_object**.

Здесь tr_package/3 — параметризованный объект, задающий шаг сценария трансформации tr/_. Объекты Logtalk (классы, экземпляры, протоколы, категории) — это символы, находящиеся друг с другом в определенных отношениях. При помощи параметров объектов можно задавать контекст для его методов. В объекте tr_package/3 контекст задается тремя моделями _Package_ – PIM программной системы, _Local_ — перечень типов, данных проекта (локальная библиотека), _Global_ — набор стандартных типов данных. Эти три параметра задаются согласно спецификации UML-2.4. Параметризованный объект query/1 получает в качестве параметра граф, где хранятся тройки и где следует распознавать структуры модели. Объекты с параметрами позволяют обходиться без создания отдельных экземпляров класса, но при этом учитывать контекст в реализации методов. Языковая структура obj::method — это посылка сообщения (в терминологии Smalltalk) объекту obj или самому себе::method. В объекте query/1 реализован метод распознавания класса как структуры типа uml:Class с внутренними UML-идентификатором ID и именем (названием) Name. Методы класса::new/2 используются для создания экземпляров классов class, attributes и methods, которые, соответственно, помещаются в переменные Class, Attributes, Methods. Для всех структур, представляющих PSM, заданы процедуры перевода в исходный код.

Обсуждение

Сначала кратко приведем примеры реализации трансформации и использование MDA. Наиболее популярным и стандартными подходом к заданию трансформаций являются языки ATL (ATLAS Transformation Language) и его предшественник QVT [4]. Эти системы задают построение новой модели в метаязыке XMI из другой модели этого же метаязыка. Трансформация задается набором правил, где асцендент — это распознаваемая структура, а консеквент — сценарий трансформации. Для ATL созданы среды визуального конструирования, инструментарий интегрирован в среду

Eclipse IDE. В ИДСТУ СО РАН по этому пути пошли авторы [5], где рассматривается язык

TMRL (Transformation Model Representation Language). Область применения — концептуальное моделирование технических систем, включая визуальное моделирование

соответствующей отологии. MDA используется для анализа построенной структуры и порождения набора правил в языках CLIPS и OWL. Визуальное представления правил трансформации использовано в [1]. В [3] ATL использован в анализе CIM, представленной в виде модели BPMN. CMI преобразуется в набор диаграмм UML (PIM). Компактный язык для задания трансформаций представлен в [7], как и в ATL язык не позволяет выходить за рамки метамодели UML. Менее популярным подходом является использование XSLT для преобразования XMI, например, в OWL [6].

В прикладном аспекте данного исследования необходимо привести пример — разработку инструментальной надстройки над современными средствами ORM (Object-relation mapping), позволяющей не только визуально представлять модель, но и дополнительно абстрагировать модель от реляционных особенностей уровня хранения информации. Популярные ORM требуют, чтобы структура ORM-модели соответствовала свойствам реляционной модели. Например, синтетический ключевой атрибут, обычно называемый id, обязан присутствовать во всех определениях классов модели. В ООП подобная проблема избыточности решается при помощи наследования атрибутов у родительского класса. Однако, не все ORM способны интерпретировать наследование, в том числе, и популярная в среде Python библиотека SQLAlchemy. Трансформации моделей решается эту задачу через реализацию наследования от класса, помеченного стереотипом “abstract”. Трансформация интерпретирует наследование через копирование унаследованных атрибутов и методов.

Заключение

В докладе представлены общие идеи использования ообъектно-ориентированного логического программирования в качестве языка формального представления сценариев трансформации моделей программных систем согласно методики MDA (Model-driven architecture). В настоящее время ведется разработка системы объектов распознавания элементов исходных моделей трансформаций и методики их конфигурирования, позволяющей разработчику средств MDA адаптировать общие процедуры к программным технологиям, используемым в конкретном НИРОКР. Тестирование получаемых инструментов осуществляется в нескольких проектах, в том числе, в системе визуального моделирования ORM для популярных библиотек и сред разработки web-приложений, отображения модулей прикладных пакетов на визуальную dataflow-модель, моделирование процессов в медицине.

Литература

1. Belghiat A., Bourahla M. From UML Class Diagrams to OWL Ontologies: A Graph Transformation Based Approach // ICWIT. 2012. P. 330-335.

2. Cherkashin E. A., Terehin I. N., Paramonov V. V., Tertychniy V. S. New transformation approach for Model Driven Architecture // 2012 Proceedings of the 35th International Convention MIPRO. IEEE, 2012. P. 1082-1087.

3. Rhazali Y., Hadi Y., Mouloudi A. Model Transformation with ATL into MDA from CIM to PIM Structured through MVC // Procedia Computer Science. 2016. Vol. 83. P. 1096-1101. https://doi.org/10.1016/j.procs.2016.04.229

4. The MOF Query/View/Transformation Specification Version 1.1. https://clck.ru/SxWrY

5. Yurin A. Y., Dorodnykh N. O., Nikolaychuk O. A., Grishenko M. A. Designing rule-based expert systems with the aid of the model-driven development approach //expert systems. 2018. Vol. 35. №5. P. e12291. https://doi.org/10.1111/exsy.12291

6. UMLtoOWL: Converter from UML to OWL. URL: https://clck.ru/SxWsy

7. Кузнецов М. Б. Трансформация UML-моделей и ее использование в технологии MDA // Программирование. 2007. Т. 33. №1. С. 65-79.