ВОПРОСЫ СОЗДАНИЯ ПОРТАЛА «МОДЕЛИРОВАНИЕ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ СИСТЕМ: МЕТОДЫ И СРЕДСТВА»

А.О. Сухов[1]

Пермский государственный национальный
исследовательский университет

Sukhov_PSU@mail.ru

КЛАССИФИКАЦИЯ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ И ЯЗЫКОВЫХ ИНСТРУМЕНТАРИЕВ

Для того чтобы достичь промышленных масштабов в процессе производства программного обеспечения, необходимо разрабатывать языки моделирования, которые оперируют терминами предметной области. Такие языки моделирования позволяют сократить трудоёмкость разработки, т.к. в удобной форме описывать модели предметной области, для которой они предназначены, вместо того чтобы делать акцент на частной технологической проблеме, такой как программирование или обмен данными. Использование таких языков позволяет устранить семантический разрыв при разработке программного обеспечения.

Предметно-ориентированные языки позволяют приблизить процесс моделирования к экспертам предметной области, упростить обслуживание и развитие моделей, что повышает производительность и эффективность разработки программного обеспечения.

При разработке информационной системы важно решить, какой язык моделирования необходимо использовать при ее создании и эксплуатации, а также какой инструментарий следует применять для описания самого языка.

В статье приведена классификация предметно-ориентированных языков по способу представления конструкций языка, по принадлеж-

[1] Работа выполнена при поддержке Программы «Научный фонд НИУ ВШЭ» (проект № 12-09-0102)

ности языка основному приложению и парадигме языка. Инструментальные средства создания предметно-ориентированных языков классифицированы по следующим признакам: способ интерпретации данных, вид разрабатываемых языков.

Классификация предметно-ориентированных языков

Предметно-ориентированный язык (Domain Specific Language, DSL) – это язык программирования или моделирования, созданный для использования в рамках конкретной предметной области. С такими языками мы встречаемся практически каждый день: SQL, shell-скрипты, языки описания пользовательского интерфейса – это все примеры предметно-ориентированных языков, поскольку они создавались для решения определённых задач, работы в некоторой конкретной предметной области.

В отличие от языков общего назначения, DSL более выразительны, просты в применении и понятны различным категориям пользователей, поскольку они оперируют привычными для них терминами предметной области. Именно поэтому в настоящее время разработано большое число DSL, предназначенных для решения тех или иных задач.

Сегодня большое развитие получили визуальные DSL, поскольку диаграммы обладают большей наглядностью и понятностью не только для программистов, но и для экспертов в предметной области и пользователей системы.

Предметно-ориентированное моделирование (*Domain Specific Modeling, DSM*) является новым подходом к разработке программного обеспечения, основанным на моделях. DSM-подход позволяет разделить процесс создания системы на отдельные этапы и на каждом из них использовать предметно-ориентированный язык, который оптимальным образом решает задачи этого этапа.

В описании синтаксиса DSL можно выделить две основные части: абстрактный и конкретный синтаксис.

Абстрактный синтаксис – такое описание синтаксиса, которое не зависит от способа визуального представления данных, т.е. такой синтаксис не зависит от пиктограмм, используемых для визуализации объектов, от текстового представления команды. Для задания абстрактного синтаксиса используются различные метаязыки: расширенные формы Бэкуса-Наура, диаграммы классов UML в подходе MOF или другие более специализированные формализмы, например, интегрированное в Microsoft DSL Tools средство построение метамоделей.

Конкретный синтаксис – описание конструкций языка, с помощью конкретных пиктограммам и текстового представления. Конкретный синтаксис используется для того, чтобы улучшить удобство использования конструкций языка, сделать их наглядными и выразительными. Большое внимание определению конкретного синтаксиса уделяется при описании визуальных языков, поскольку при использовании таких языков пользователь должен оперировать интуитивно понятными изображениями конструкций. Определить конкретный синтаксис можно расширением форм Бэкуса-Наура или отображением множества конструкций языка на множество объектов (пиктограмм, шаблонов операторов), используемых для их визуализации.

Все предметно-ориентированные языки, как и другие языки моделирования и программирования, по способу представления конструкций можно разделить на две группы: визуальные и текстовые (рис. 1). *Текстовые* DSL позволяют описывать модель в текстовом виде, а *визуальные* – в графическом. Причем большую популярность получили именно визуальные DSL, поскольку диаграммы обладают большей наглядностью и понятностью не только для программистов, но и для экспертов в предметной области и пользователей системы.

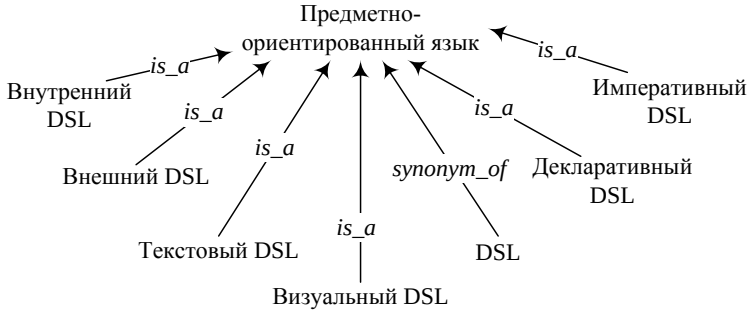

Рис. 1. Фрагмент онтологии классификации DSL

В своей статье [3] Мартин Фаулер выделяет два вида языков предметной области: внешние и внутренние DSL.

Внешними называются такие DSL, которые написаны отдельно от основного приложения. После создания такие языки встраиваются в основное приложение в качестве компилятора или интерпретатора. Основным преимуществом внешних DSL является то, что они могут наилучшим образом отразить концепты предметной области и требования заказчика. Пользователи при работе с такими DSL оперируют

привычными для них терминами предметной области. К DSL этого вида относятся, например, малые языки Unix, конфигурационные XML-файлы, язык запросов SQL.

Внутренние DSL – это DSL, которые написаны на языке основного приложения и встроены в этот язык. Внешние DSL намного проще языков общего назначения, набор команд в них ограничен и специфицирован под некоторую конкретную предметную область, поэтому изучать эти языки гораздо легче. Пример DSL такого вида – фреймворк Rails, написанный на языке программирования Ruby.

При работе с внутренними DSL можно пользоваться всеми возможностями языка основного приложения. Внутренние DSL ограничены лишь синтаксисом и семантикой основного языка. Но при этом излишняя «перегруженность» этих DSL конструкциями базового языка усложняет их использование.

Внутренние DSL являются лишь дополнением базового языка программирования, поэтому остается нерешенной проблема адекватного представления концептов предметной области языками общего назначения.

Преимущество внешнего DSL для использования программистами-непрофессионалами состоит в том, что есть возможность отказаться от сложности базового языка приложения, и создать DSL, который будет понятен и удобен пользователям.

Помимо этого, все предметно-ориентированные языки можно разделить на две группы:

- декларативные языки,
- императивные языки.

Декларативными являются языки, в которых программист указывает лишь «что» необходимо сделать, не описывая пошаговый алгоритм «как» это следует сделать. Примерами языков такого вида являются Prolog, SQL. Достоинства декларативных языков являются очевидными: пользователю для решения задачи достаточно лишь некоторым образом описать результат, который он хочет получить в итоге, не описывая алгоритм получения этого результата.

Императивными языками называются языки, в основу которых положен принцип описания последовательности действий для решения задачи. К языкам такого типа относятся C++, C#, Java.

При создании DSL важно решить, необходимо ли в нем использовать императивные конструкции. Руководствуясь здравым смыслом, можно сказать, что императивные конструкции целесообразно применять тогда, когда это продиктовано спецификой предметной области. Например, если в постановке задачи часто употребляется слово «кол-

лекция» и говорится о необходимости добавления, удаления, поиска элементов коллекции, то введение в DSL понятия «цикл» существенно упростит написание программы. Тем не менее, предпочтение всё-таки стоит отдавать декларативным конструкциям, т.к. их проще использовать, они легче воспринимаются различными категориями пользователей и несут больше информации о решаемой задаче.

При всех достоинствах DSL у них есть один большой недостаток – сложность разработки. Если языки общего назначения позволяют создавать программы безотносительно предметной области, то в случае DSL для каждой предметной области, а в некоторых случаях, даже для каждой задачи приходится создавать свой предметно-ориентированный язык. Если предметная область достаточно проста и язык несложен, то создать транслятор будет нетрудно. Более сложная предметная область и язык потребуют больших усилий, несмотря на то, что в настоящее время существуют и построители лексических анализаторов, и другие инструменты для создания компиляторов, облегчающие работу программиста.

Еще одним недостатком предметно-ориентированных языков является то, что при их разработке приходится создавать полнофункциональные среды для работы с ними, т.е. удобный графический редактор или редактор кода с возможностью подсветки синтаксиса языка, автоматического дополнения лексем, выделения ошибок, а также удобный отладчик программ. В связи с этим в последнее время стал активно развиваться новый тип программного обеспечения – языковые инструментарии.

Классификация и структура языковых инструментариев

Языковой инструментарий, или DSM-платформа, – это инструментальное программное обеспечение, предназначенное для поддержки разработки и сопровождения языков предметной области [1]. Использование при создании DSL языкового инструментария значительно упрощает процесс создания языков.

DSM-платформы можно классифицировать по различным признакам. В зависимости от способа интерпретации данных языковые инструментарии можно разделить на DSM-платформы с генерацией кода и DSM-платформы с интерпретацией метаданных (см. рис. 2).

DSM-платформа с генерацией кода на основе метамодели предметной области и модели редактора позволяет сгенерировать исходный код редактора для работы с языком. При этом существует возможность внести изменения и дополнения в сгенерированный код «вручную». К

системам такого класса относят: Microsoft DSL Tools, Eclipse Graphical Modeling Framework, QReal.

DSM-платформа с интерпретацией метаданных обеспечивает максимальные возможности адаптации. Метамодель и модель редактора находятся «внутри» языкового инструментария. В этом случае DSM-платформы позволяют создавать метамодели DSL и интерпретировать их в процессе эксплуатации системы, что предоставляет возможность модификации языка во время работы системы без перегенерации исходного кода и без перезапуска системы. Примерами систем такого типа являются MetaEdit+, XMF Mosaic [4], MetaLanguage.

В зависимости от вида разрабатываемых DSL языковые инструментарии можно разделить на инструментарии, предназначенные для создания текстовых DSL, например, Meta Programming System [2] и инструментарии, предназначенные для создания визуальных DSL, например, MetaEdit+.

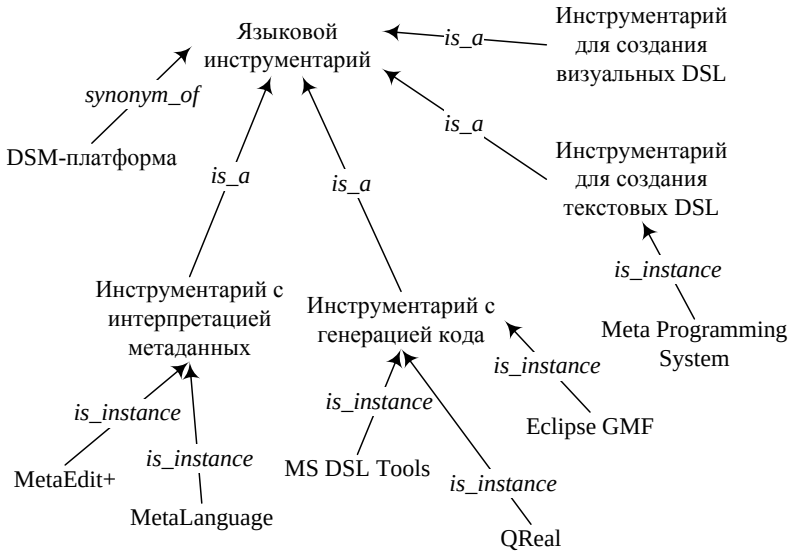

Рис. 2. Фрагмент онтологии классификации DSM-платформ

Языковой инструментарий состоит из следующих компонентов: редактор, браузер объектов, репозиторий, валидатор, генератор (см. рис. 3).

Назначением *редактора* является построение, изменение, удаление моделей, а также установление взаимосвязей между различными моделями. Каждая конструкция языка представлена некоторым графическим символом или текстовой командой. Графический редактор предоставляет пользователю возможность располагать на рабочем листе различные фигуры (экземпляры конструкций языка и отношений), применять к этим фигурам наборы операций, задавать для них различные графические свойства.

Рис. 3. Структура языкового инструментария

Браузер объектов – компонент DSM-плафтормы, предназначенный для просмотра и редактирования информации об элементах языка и связях между ними. Браузер также предоставляет возможность экспорта/импорта моделей из других систем. Данный компонент необязательно должен входить в состав языкового инструментария.

Единым хранилищем всей информации о системе является *репозитарий*. Он содержит информацию о метамоделях, моделях, элементах, отношениях, атрибутах, ограничениях, пиктограммах, используемых для отображения элементов и отношений.

Валидатор позволяет проверить соответствие построенных моделей ограничениям, заданным пользователем.

Генератор позволяют на основе имеющихся моделей сгенерировать исходный код редактора языка, документацию или файлы для экспорта модели в стороннюю систему.

Трансформация моделей

Использование DSL и инструментальных средств их создания также затрагивает проблему трансформации, поскольку необходимо экспортировать созданные модели во внешние системы.

Для того чтобы выполнить трансформацию моделей необходимо чтобы они были описаны на некотором языке моделирования (напри-

мер, на языке UML). Синтаксис и семантика языка моделирования описывается метамоделью.

В зависимости от языка, на котором описаны исходная и целевая модели, трансформации можно разделить на два вида: эндогенные и экзогенные (рис. 4).

Эндогенная трансформация – это преобразование моделей, описанных на одном языке моделирования. *Экзогенная трансформация* – это преобразование моделей, которые описаны на различных языках моделирования [5].

Рис. 4. Фрагмент онтологии классификации видов трансформаций моделей

Типичными примерами экзогенных трансформаций являются:

- компиляция кода, когда исходный код на высокоуровневом языке программирования преобразуется в байт-код или исполняемый код;
- миграция, которая позволяет преобразовать программу, написанную на одном языке программирования, в другой язык, но при этом сохраняя тот же уровень абстракций.

Типичными примерами эндогенных трансформаций являются:

- оптимизация – преобразования, направленные на улучшение определенных эксплуатационных качеств (например, производительность) и сохраняющие семантику системы;
- рефакторинг – изменение внутренней структуры программного обеспечения для улучшения определенных качественных характеристик (таких как понятность, переиспользование, модульность, адаптивность) без изменения его внешнего поведения;
- упрощение и нормализация позволяют уменьшить синтаксическую сложность программы, например, путем перевода синтаксического сахара в более примитивные языковые конструкции.

В зависимости от направления выполнения трансформации их можно разделить на горизонтальные и вертикальные.

Горизонтальная трансформация – это преобразование, при котором исходная и целевая модель принадлежат одному уровню абстракций. Типичным примером является рефакторинг.

Вертикальная трансформация – это преобразование, при котором исходная и целевая модель принадлежат различным уровням абстракций. Примером вертикальной трансформации является рафинирование (уточнение) модели, при котором спецификация постепенно перерабатывается в полноценную реализацию, путем последовательных шагов уточнения, которые добавляет конкретные детали.

В зависимости от способа представления исходной и целевой модели трансформации можно разделить на несколько видов:

- трансформация *модель-модель*, например, преобразование модели с языка ERD в модель на языке диаграмм классов UML;
- трансформация *модель-текст*, например, генерация кода по созданной визуальной модели;
- трансформация *текст-модель*, например, построение визуальной модели на основе ее XML-описания;
- трансформация *текст-текст*, например, преобразование программы с языка Pascal в программу на языке C++.

Заключение

При создании моделей информационных систем и DSL важно определить, какой тип языка необходимо описать для решения поставленной задачи, какой инструментарий для этого необходимо использовать, требуется ли выполнять трансформацию моделей и каким подход при этом воспользоваться. Представленная классификация предметно-ориентированных языков, языковых инструментариев и подходов к трансформации моделей может быть использована как для описания DSL, разработки языковых инструментариев, так и для осуществления семантического поиска в сети Интернет информации, необходимой для разработчиков и исследователей, занимающихся вопросами проектирования информационных систем, создания инструментальных средств разработки.

Разрабатываемые онтологии предназначены для реализации средств интеллектуального поиска информации для учебно-исследовательского портала. Портал ориентирован на решение задач поддержки работы преподавателей и студентов, изучающих методы и средства, используемые при создании информационных систем, исследователей, работающих в этой области.

Библиографический список

1. *Сухов А.О.* Сравнение систем разработки визуальных предметно-ориентированных языков // Математика программных систем: межвузовский сборник научных статей / Перм. гос. нац. исслед. ун-т. – Пермь, 2012. – С. 84-111.
2. *Dmitriev S.* Language Oriented Programming: The Next Programming Paradigm: [Электронный ресурс]. URL: http://www.onboard.jetbrains.com/is1/articles/04/10/lop/index.html (дата обращения: 11.12.2012).
3. *Fowler M.* DomainSpecificLanguage [Электронный ресурс]. URL: http://martinfowler.com/bliki/DomainSpecificLanguage.html (дата обращения: 11.12.2012).
4. *Chatterjee S.* MDA Tools Evaluation - Part III: MIA & XMF-Mosaic [Электронный ресурс]. URL: http://www.jaxmag.de/itr/column/psecom,id,6,nodeid,354.html (дата обращения: 11.12.2012).
5. *Mens T., Czarnecki K., Gorp P.V.* A Taxonomy of Model Transformations // Electronic Notes in Theoretical Computer Science. – Amsterdam, 2006. – Vol. 152. – P. 125-142.