УДК 004.434

ИСПОЛЬЗОВАНИЕ ПОРОЖДАЮЩЕГО ПРОГРАММИРОВАНИЯ ПРИ РЕАЛИЗАЦИИ ЯЗЫКА ОПИСАНИЯ ДИАГРАММ

Ф. А. Новиков,
канд. физ.-мат. наук, заведующий лабораторией
Институт прикладной астрономии РАН
К. Б. Степанян,
начальник отдела управления операционного обслуживания
УК «Арсагера»

Язык описания диаграмм DiaDeL позволяет формально определить графический синтаксис (нотацию) диаграмм заданного типа и связать нотацию с семантикой, заданной в форме набора классов. В статье обсуждается применение порождающего программирования при реализации этого языка.

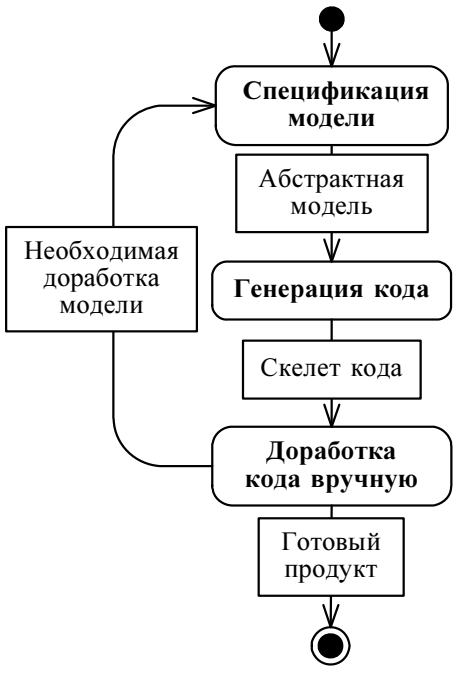
Ключевые слова *— порождающее программирование, метамоделирование, графический язык, абстрактный синтаксис.*

Идея порождающего программирования

В современной компьютерной индустрии порождающее программирование [1] находит все большее применение, так как позволяет автоматизировать и ускорить процесс создания программ. Возможно, наиболее известный пример — это визуальное конструирование приложений [2], которое основывается на генерации кода приложения по заданной визуальной спецификации. В настоящее время порождающее программирование (и, в частности, визуальное конструирование приложений) не способно полностью заменить существующий традиционный способ разработки кода вручную. Это связано с отсутствием такого стандарта визуального конструирования, который обладал бы ясностью, однозначностью и простотой, с одной стороны, и с рядом проблем визуального конструирования, эффективного решения для которых на данный момент не предложено, с другой стороны. Например, создание полной и непротиворечивой визуальной спецификации поведения приложения зачастую более трудоемко, чем реализация этого поведения в виде программы на обычном языке программирования.

Несмотря на то что порождающее программирование не может пока вытеснить написание кода вручную, оно может с успехом использоваться в сочетании с ним — часть приложения автоматически генерируется по спецификации, а часть программируется вручную обычным образом. В этом случае процесс разработки можно представить в виде диаграммы (рис. 1).

Уточним термины, используемые в статье. Набор сущностей/классов/интерфейсов, специфи-



■ ***Рис. 1.*** *Цикл разработки с использованием порождающего программирования*

цирующий предметную область, мы называем моделью этой области. Набор объектов сущностей и/или экземпляров классов и/или объектов, реализующих интерфейсы, составляющие модель, мы называем экземпляром модели предметной области.

Модель описывается на языке спецификации, понятном генератору. Далее генерируется скелет кода на целевом языке. Затем код дорабатывается вручную и/или используется в других частях приложения.

Модели можно разделить на два класса — статические и динамические. Статические модели формализуют и описывают структуру предметной области, для этого наиболее часто используются диаграммы классов UML [3].

Динамические модели описывают поведение системы, для этого используются диаграммы состояний, диаграммы деятельности (блок-схемы) и т. д. Вопросы генерации кода по динамическим моделям освещены, например, в работах [4, 5].

В данной статье рассматриваются статические модели и генерация кода по ним. Подобный подход был использован при разработке системы автоматического представления диаграмм на языке DiaDeL (Diagram Definition Language) [6, 7]. А именно, методами порождающего программирования была реализована синтаксическая модель языка описания диаграмм DiaDeL.

Порождение модели языка DiaDeL

Язык DiaDeL основан на предположении, что семантика каждой отдельной диаграммы задана в виде набора конкретных программных объектов определенных классов. Набор всех возможных классов объектов, которые могут появляться на диаграммах данного типа, называется семантической моделью, а набор объектов, соответствующих конкретной диаграмме, называется экземпляром ее семантической модели.

Основным назначением языка DiaDeL является:

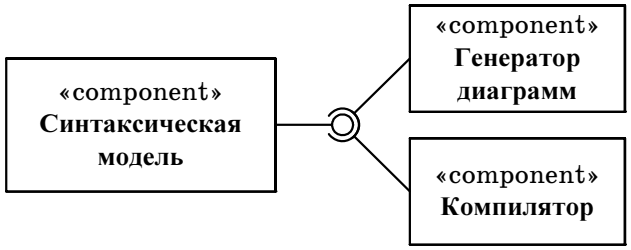
• описание нотации (графического синтаксиса) диаграмм и

• описание связи нотации с существующей семантикой.

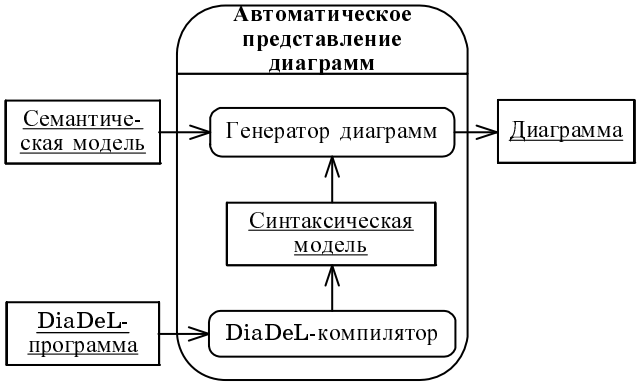
Нотация диаграммы задается в виде описания графических конструкций и графических отношений между ними. Связывание нотации с семантикой является ключевым моментом в описании диаграммы. Для этого элементам из семантической модели, которые должны быть представлены визуально, сопоставляются графические конструкции.

Одним из возможных использований языка DiaDeL является построение системы автоматического представления диаграмм. Основные компоненты системы показаны на рис. 2.

Данная система строит и отображает диаграмму, используя описание на языке DiaDeL и экземпляр семантической модели (рис. 3).



■ ***Рис. 2.*** *Основные компоненты системы автоматического представления диаграмм*



■ ***Рис. 3.*** *Схема работы системы автоматического представления диаграмм*

Синтаксическая модель является программным представлением языка DiaDeL, иными словами, абстрактным синтаксисом. Используя переданное описание диаграммы на языке DiaDeL, компилятор строит экземпляр синтаксической модели. Затем генератор диаграмм (визуализатор), получая построенный экземпляр синтаксической модели и переданный на вход экземпляр семантической модели, строит диаграмму.

Поскольку синтаксическая модель является «фундаментом» в данной системе, то к ней предъявляются жесткие требования по гибкости и связности, а именно:

• синтаксическая модель не должна налагать необоснованных ограничений и должна быть удобна в использовании;

• зависимость других компонентов должна быть сведена к минимуму, т. е. изменения реализации не должны приводить к изменениям в зависимых компонентах.

Для создания системы автоматического построения диаграмм выбрана платформа Eclipse [8]. Eclipse предоставляет множество полезных сервисов и библиотек, но в данной статье мы остановимся только на одной из них — EMF (Eclipse Modeling Framework) [9]. Данная библиотека является основой для построения моделей (сама является метамоделью) и позволяет использовать по-

рождающее программирование для генерации готовых к использованию работоспособных компонентов.

Среди основных преимуществ EMF можно отметить следующие:

1) EMF позволяет моделировать на уровне сущностей, атрибутов и отношений между ними;

2) сгенерированная модель поддерживает целостность в указанных рамках;

3) EMF генерирует модель, используя образец проектирования «Абстрактная фабрика» (Abstract Factory) [10];

4) EMF генерирует работоспособный код, не требующий отладки;

5) повторная генерация учитывает изменения, внесенные в реализацию вручную;

6) сгенерированная модель поддерживает механизм оповещений;

7) EMF предоставляет средства для сериализации/десериализации созданных экземпляров моделей;

8) EMF позволяет сгенерировать визуальный редактор для создания экземпляров моделей.

Рассмотрим некоторые из перечисленных пунктов подробней. Первый пункт является одним из самых значительных, поскольку обеспечивает достаточный уровень абстракции для проектирования. Нет необходимости на этапе моделирования учитывать ограничения целевого языка (в нашем случае это Java). Например, EMF обеспечивает поддержку множественного наследования сущностей. Второй пункт, в частности, выражается в обеспечении поддержки двунаправленных ассоциаций между сущностями. Корректно обрабатывается ситуация, когда изменение состояния одного из полюсов ассоциации требует изменения состояния другого (иначе экземпляр модели станет несостоятельным). Хорошим примером может служить отношение один ко многим типа «часть—целое» между двумя сущностями.

Генерация модели с применением образца проектирования «Абстрактная фабрика» позволяет добиться требуемой гибкости и слабой связности. Модель в программе представляется в виде набора интерфейсов, реализующих их классов и фабрики. Важными аспектами являются четвертый и пятый пункты. После генерации не требуется вносить какие-либо изменения для того, чтобы начать использовать модель. Вносимые изменения обычно связаны со спецификацией поведения, отличающегося от стандартного. Если выполняется повторная генерация, внесенные изменения остаются незатронутыми. В силу того, что модель в ходе разработки проекта обычно сама эволюционирует, эта особенность EMF становится бесценной.

В рамках реализации модели языка DiaDeL оставшиеся пункты не имеют большого значения. Однако в общем случае могут быть одними из первых. Например, библиотека EMF использовалась нами для генерации семантических моделей визуальных языков IDEF0 [11] и ERD [12]. Эти модели были необходимы для создания тестовых примеров. В рамках задачи создания и использования конкретных диаграмм возможность создавать, редактировать, сохранять и подгружать экземпляры модели, используя пользовательский интерфейс, выходит на первый план.

Процесс генерации модели с использованием EMF состоит из следующих шагов.

1. **Спецификация модели**. Модель может быть задана в разных форматах, в том числе в виде аннотированного Java-кода, Ecore-формата, формата Rational Rose, формата UML- и XML-схемы. Для спецификации модели DiaDeL использовался Ecore-формат как средство, являющееся частью EMF.

2. **Задание свойств генерации**. Строится модель генерации, в которой каждой сущности модели сопоставляется целевая единица (пакет, класс, перечисление и т. д.) и задаются свойства генерации как для всей модели, так и для отдельных сущностей.

3. **Генерация реализации модели**. Используя построенную модель и заданную модель генерации, EMF генерирует (возможно, повторно) интерфейсы, их реализацию и фабрику.

4. **Генерация редактора**. Используя построенную модель и заданную модель генерации, EMF генерирует подключаемый модуль для создания и редактирования экземпляров заданной модели.

Спецификация модели — это очень распространенная задача, которую решают архитекторы при построении приложения. Отнюдь не всегда модель прямо связана с разрабатываемым приложением. Обычно целью спецификации является проектирование и документирование. Разумно было бы использовать спецификацию не только как документ или наглядное представление, но и как входные данные для генерации кода модели. Таким образом, проделав всю ту же работу по моделированию и еще немного (шаги 2 и 3), можно получить работающий код. Альтернатива — написать его вручную. Однако подобная задача может быть весьма и весьма трудоемкой. Например, синтаксическая модель языка DiaDeL содержит более чем 50 сущностей. EMF автоматически генерирует по ним более 120 классов и интерфейсов. Объем сгенерированного кода составляет более двух третей от всего проекта. Автоматически сгенерированный код не всегда является оптимальным с точки зрения объема и быстродействия. Но зачастую это не играет важной роли в разработке бизнес-приложений.

Здесь необходимо уточнить, что EMF не имеет визуального редактора, представляющего модель в виде диаграммы, однако существует бесплатное (или условно бесплатное) программное обеспечение, которое может быть для этого использовано (например, Omondo Eclipse UML [13]).

Заключение

В статье обсуждается использование порождающего программирования для построения синтаксической модели языка описания диаграмм DiaDeL. Приведена общая архитектура разработанной системы автоматического представления диаграмм. Описаны преимущества библиотеки EMF и ее применение в качестве генератора статических моделей.

В основе архитектуры разработанной системы лежит синтаксическая модель языка DiaDeL, от которой зависят другие компоненты системы. Как следствие, к реализации модели предъявляются довольно жесткие требования по гибкости и связности. Также реализацию модели требовалось получить, используя подход порождающего программирования. Для решения поставленных задач была выбрана библиотека EMF.

Опыт применения EMF показал, что генерируемые модели удовлетворяют поставленным требованиям и имеют ряд других преимуществ (работоспособность, поддержка целостности, механизм оповещений и т. д.). Помимо самих моделей, EMF также позволяет генерировать полнофункциональный редактор для них. Подводя итог, можно смело утверждать, что библиотека EMF является эффективным и удобным средством, реализующим идею порождающего программирования. Использование подхода, представленного на рис. 1, вкупе с EMF позволяет автоматизировать процесс разработки программного обеспечения и повысить эффективность работы, не теряя при этом гибкости кода. Обратной стороной являются незначительные потери оптимальности в смысле требуемой памяти и скорости работы. Однако при разработке бизнес-приложений это незначительная плата. А если учитывать существующую скорость развития аппаратных средств, то в ближайшем будущем на это можно будет и вовсе не обращать внимания.

Поскольку общая архитектура системы автоматического представления диаграмм имеет типовую структуру, описанный случай является достаточно распространенным. Авторы более чем уверены в эффективности применения порождающего программирования.

Литература

1. **Чарнецки К., Айзенекер У.** Порождающее программирование: методы, инструменты, применение. СПб.: Питер, 2005. 736 с.
2. **Новиков Ф. А.** Визуальное конструирование программ // Информационно-управляющие системы. 2005. № 6. С. 9–22.
3. **Буч Г., Якобсон А., Рамбо Д.** UML. 2-е изд. СПб.: Питер, 2006.
4. **Канжелев С., Шалыто А.** Автоматическая генерация кода программ с явным выделением состояний // Paths to Competitive Advantage: Software Engineering Conference. М., 2006. С. 60–63.
5. **Канжелев С., Шалыто А.** Преобразование графов переходов, представленных в формате MS Visio, в исходные коды программ для различных языков программирования (инструментальное средство MetaAuto). 102 с. http://is.ifmo.ru/projects/metaauto
6. **Степанян К. Б.** Язык описания диаграмм // Научно-технические ведомости СПбГПУ. 2006. № 6-1. С. 36–41.
7. **Новиков Ф. А., Степанян К. Б.** Язык описания диаграмм // Информационно-управляющие системы. 2007. № 4. С. 28–36.
8. **http://www.eclipse.org**
9. **http://www.eclipse.org/modeling/emf/**
10. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2004.
11. **National Institute of Standards and Technology.** Integration Definition For Function Modeling (IDEF0). Washington: Draft Federal Information, 1993. http://www.itl.nist.gov/fipspubs/idef02.doc
12. **Chen P. P-S.** The Entity-Relationship Model — Toward a Unified View Of Data // ACM Transactions on Database Systems. 1976. Vol. 1. N 1.
13. **Omondo**. Eclipse UML. http://www.omondo.com