УДК 004.4’242

Н.Н. ГЛИБОВЕЦ, В.М. ФЕДОРЧЕНКО

УПРОЩЕННАЯ ИНФРАСТРУКТУРА ДЛЯ ТРАНСФОРМАЦИИ XML-МОДЕЛЕЙ

Ключевые слова: *моделе-ориентированная разработка, предметно-зависимая модель, трансформация моделей, IoC-контейнер*.

ВВЕДЕНИЕ

Программирование, как способ формализации описания разнообразных процессов, зависит от конкретных технических ограничений и инфраструктуры, в рамках которой эта формализация проводится. Благодаря прогрессу в этой области приоритет в программировании для многих типов задач сместился с максимально эффективного (в контексте используемых технических ресурсов) кодирования на максимально быструю (в контексте человеческих ресурсов) и одновременно качественную (quality) разработку [1–4].

МОДЕЛЕ-ОРИЕНТИРОВАННАЯ РАЗРАБОТКА

Ключевые концепции MDD (Model Driven Development) — предметно-зависимый язык моделирования и трансформация моделей [5, 6]. В контексте MDD модель представляет абстракцию программной системы или/и окружения над ее программной реализацией. Так, согласно этому определению программный код также представляет модель, поскольку является абстракцией над машинным кодом, генерируемым компилятором. Метамодель (metamodel) описывает синтаксис и семантику DSM-языка через определение понятий и их отношений в конкретной предметной области (процесс создания DSML называют метамоделированием).

Трансформация моделей (Model Transformations) играет ключевую роль в MDD и отвечает за преобразование моделей, определенных с помощью DSML, в другие программные артефакты и формы представления. Например, модель может быть трансформирована в комбинацию исходных текстов, ресурсов и XML-конфигураций (поскольку в контексте MDA (Model Driven Architecture) программа в любой форме является собственно моделью). Консорциум OMG (Object Management Group) предложил собственный вариант такой технологии под названием моделе-ориентированная архитектура (MDA) в виде ряда спецификаций и стандартов [7].

Существующие подходы к трансформации входной модели условно разделяют на те, которые генерируют исходную модель в виде текста (обычно это текст программы на определенном языке программирования общего назначения, например Java, C++), и те, которые генерируют исходную модель в более структурированном виде (например, XML), которая отвечает некоторой метамодели. Поскольку программный код является также моделью, то подходы отличаются лишь способом использования исходной метамодели. В случае, когда генерируется текст, трансформатору не обязательно оперировать структурой метамодели языка программирования (побочным эффектом такого упрощения является вероятность генерации некорректного кода).

Достаточно распространена практика определения трансформаторов, которые базируются на сочетании нескольких подходов. Типичным примером является стандарт QVT, на основе которого можно определять трансформации в MDA [7]. Другим общедоступным и мощным трансформатором является реализация стандарта XSLT, поскольку любая модель может быть представлена в виде XML с помощью формата XMI (XML Metadata Interchange [8]). Однако при этом имеется су-

щественный недостаток: формат XMI достаточно громоздок (для разработчика трансформации), что приводит к созданию сложных для разработки и поддержки XSL-правил.

Для успешного внедрения MDD при разработке программных продуктов необходима инфраструктура, которая поддерживает моделе-ориентированную разработку и должна удовлетворять следующим требованиям: возможность гибкого манипулирования параметрами жизненного цикла, определение и расширение моделей по необходимости (by demand), возможность одновременного использования моделей разного уровня абстракции, интеграция с уже существующими программными системами, минимизация расходов на интеграцию и поддержку инфраструктуры для MDD.

УПРОЩЕННАЯ ИНФРАСТРУКТУРА

Рассмотрим вариант такой упрощенной инфраструктуры для моделе-ориентированной разработки, которая удовлетворяет описанным выше требованиям. Суть заключается в возвращении к первичному представлению модели в виде предметно-зависимого XML-формата, который позволит избавиться от недостатков варианта XMI+XSL. Модель будет иметь максимально компактное представление, при этом чтение и изменение модели программистами возможно без использования каких-либо специализированных инструментов (редакторов, визуализаторов). Это свойство приобретает первоочередное значение в случаях, когда в предметно-зависимую метамодель вносятся изменения в процессе разработки. При этом сохраняется возможность быстро изменять и расширять метамодель, а в случае стабилизации предметно-зависимой метамодели — определять трансформации для созданных предметно-зависимых моделей в любые другие (стандартизированные) модели (например, UML).

На практике допустимо свести определение метамодели к созданию XML-схемы (XSD) предметно-зависимой модели. Этот язык хорошо известен программистам и имеет инструментальную поддержку в любой современной платформе. При этом можно обеспечить приемлемый на практике уровень валидации моделей с минимальными расходами времени на расширение метамодели. Естественно, XSD имеет достаточно ограниченные возможности для описания метамодели, но в случае максимально упрощенной инфраструктуры этот вариант является приемлемым компромиссом, поскольку при необходимости есть возможность добавлять собственные метаданные к XML-схеме. Если и этого будет недостаточно, дополнить метамодели можно с помощью стандарта MOF (MetaObject Facility [7]), поскольку эта спецификация допускает определение любой метамодели.

В данном случае использование XSL для описания трансформаций представляется наиболее приемлемым вариантом, поскольку этот язык является мощным инструментом (содержит тьюринг-полный набор функций), особенно когда выходная модель также может быть представлена в формате XML. Это позволит организовывать как вертикальные трансформации с произвольным количеством слоев абстракции, так и горизонтальные. Широкая распространенность и общеизвестность такого языка создает необходимые условия для командной работы, когда определение метамодели и одновременное создание моделей на основе этой метамодели выполняется одной группой разработчиков. Этот фактор чрезвычайно важен для современного процесса разработки, когда необходимо обеспечить минимальный промежуток времени между началом разработки и первыми результатами.

Последним этапом в формировании упрощенной инфраструктуры для моделе-ориентированной разработки является определение формата модели самого низкого уровня, доступного в рамках используемой инфраструктуры. Как отмечено выше, целесообразно, чтобы эта модель имела естественное представление в формате XML, при этом была достаточно простой для тривиального преобразования в машинный код и одновременно достаточно удобной для выражения концепций из моделей более высоких уровней абстракции (хотя эти два условия являются несколько противоречивыми).

В последние годы широко используются IoC-контейнеры, конфигурация которых обычно имеет XML-представление (SpringFramework [9], Winter.NET [10]). IoC-контейнеры являются реализацией паттерна программирования «инверсия управления» (inversion of control), также известного под названием «инстанциация зависимостей» (dependency injection), который стал особенно популярным после статьи Мартина Фаулера [11]. Суть этого паттерна заключается в абстрагировании создания и инициализации одних объектов другими путем делегирования определенных действий особым типам объектов (IoC-контейнерам). При этом все другие объекты лишь декларируют минимально необходимые для функционирования зависимости (через интерфейсы). Создание графа объектов и соответственно определение зависимостей в виде конкретных экземпляров объектов, которые реализуют нужные интерфейсы, полагается также на IoC-контейнер. В контексте MDD эта специфика дает уникальную возможность удовлетворить оба условия для модели самого низкого уровня, поскольку способ связывания объектов является фиксированным (и чрезвычайно простым), а сами объекты могут реализовывать достаточно сложные концепты. Создание такого графа по XML-конфигурации является достаточно быстрой и тривиальной задачей [9, 10].

В терминах MDA IoC-конфигурация — это модель PSM (Platform Specific Model), зависимая от платформы самого низкого уровня (поскольку включает ссылки на классы и свойства вполне конкретной платформы), а предметно-зависимые XML-модели, если они не имеют привязки к особенностям платформы, относятся к PIM. Естественно, это достаточно условное деление. В предложенном варианте инфраструктуры имеется возможность регулировать степень абстракции PSM в зависимости от конкретной ситуации.

ПРИМЕР СОЗДАНИЯ УПРОЩЕННОЙ MDD ИНФРАСТРУКТУРЫ

Для иллюстрации описанного подхода рассмотрим создание упрощенной MDD инфраструктуры для трансформации моделей в сфере автоматизации бизнес-процессов.

Определение базовых понятий для такой модели можно позаимствовать из многочисленных вариантов корпоративных онтологий (enterprise ontology), например CEO (Core Enterprise Ontology [12]), как одной из самых простых. В результате трансформации получим конфигурацию Winter IoC-контейнера [10] для применения на платформе Microsoft .NET.

CEO предназначена для описания бизнес-процессов и моделирования в виде информационной системы (ИС). В упрощенном виде (минимально необходимом для построения метамодели) эта онтология состоит из таких базовых понятий [12].

- Пассивные сущности (passive entity) — представляют бизнес-объекты, пассивные элементы корпоративной среды, которые создаются, изменяются, пересматриваются с целью получения какой-то информации. Важным свойством пассивных сущностей является идентифицируемость, что является необходимым условием для включения пассивных сущностей к определению схемы данных информационной системы.
- Активные сущности (active entity) — это активные элементы из корпоративной среды, которые имеют возможность принимать решения и инициировать определенные действия. Ими могут быть как люди, так и организации (офис, департамент, отдел). Именно по отношению к активным сущностям определяются обязанности, права доступа и т.д.
- Трансформации (transformation) — это любые действия, операции и процессы, которые существуют в корпоративной среде и имеют отображение в ИС. Обычно в трансформациях принимают участие как пассивные, так и активные сущности.
- Условия — это предикаты, которые могут быть вычислены с целью определения их истинности. На этом базовом концепте основаны такие понятия, как цель, правило, ограничение, состояние.

Детальный анализ CEO выходит за рамки данной статьи. Метамодель на основе этих понятий позволяет описывать разнообразные бизнес-процессы. Отметим,

что даже самое простое описание таких понятий достаточно объемно, поэтому для иллюстрации упрощенного подхода к трансформации моделей в деталях рассмотрим лишь определение понятия «сущность».

Опишем метамодель, которая в рамках упрощенной инфраструктры представляется в виде XML-схемы:

```
<xsd:element  name="entity">
  <xsd:complexType>
    <xsd:sequence  maxOccurs="unbounded">
      <xsd:element  name="storage">
        <xsd:complexType>
          <xsd:sequence  maxOccurs="unbounded">
      <xsd:element name="sourcename" type="xsd:string" minOccurs="1"/>
          </xsd:sequence>
     <xsd:attribute name="versions" type="xsd:boolean" use="optional"
default="false"  />
        </xsd:complexType>
      </xsd:element>
      <xsd:element  name="schema">
        <xsd:complexType>
          <xsd:sequence  maxOccurs="unbounded">
            <xsd:element  name="field"  minOccurs="1">
              <xsd:complexType>
        <xsd:attribute name="uid" type="xsd:boolean" use="optional"
default="false"  />
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="type" type="xsd:string" use="required"/>
          <xsd:attribute name="nullable" type="xsd:boolean" use="optional"
default="false"  />
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute  name="name"  type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>
```

На основе этой метамодели модель некой сущности, например «заказ», будет иметь такой вид:

```
<entity  name="purchaseOrder">
  <storage  versions="true">
    <sourcename>purchase_orders</sourcename>
  </storage>
  <schema>
    <field  uid="true"  name="id"  type="int"/>
    <field  name="contact_id"  type="int"/>
    <field  name="sum"  type="decimal"/>
    <field  name="delta"  type="decimal"/>
    <field  name="expired_date"  type="dateTime"  nullable="true"/>
  </schema>
</entity>
```

Таким образом, информационную систему пополняет еще одна пассивная сущность «purchaseOrder» с характеристиками «id» (уникальный идентификатор), «contact_id», «sum», «delta» и «expired_date». Эти характеристики должны сохраняться в хранилище данных (например, БД) под именем «purchase_orders». Эта сущность по определению должна также иметь возможность сохраняться, изменяться и загружаться из хранилища данных. Более того, маркировка versions=«true» обязывает ИС хранить все версии этой сущности, которые сохранялись в хранилище данных.

Для определения трансформации этой модели к конфигурации компонентов IoC-контейнера необходимо иметь возможность представить концепт пассивной сущности с описанным выше набором свойств с помощью одного или композиции нескольких объектов. Предположим, что существует некоторый класс, в котором в соответствии с паттерном «активная запись» (Active Record [13]) реализованы свойства хранения, загрузки и изменения записи в хранилище данных (C#):

```
public  interface  IActiveRecord  {
       string  XmlSchema  { get;  set; }
       string  SourceName  { get;  set; }
       bool  VersionsEnabled  { get;  set; }
       bool  IsPersisted  { get; }
       object[]  Uid  {get;  set;}
       object  this[string  fieldName]  { get;  set; }
       void  Save();
       void  Load(object[]  uid);
       void  Delete();
       void  Disconnect();
}
```

Допустим также, что имеется класс DbActiveRecord (реализующий интерфейс IActiveRecord), для функционирования которого необходимо инициализировать свойства XmlSchema, SourceName и VersionsEnabled, причем свойство XmlSchema должно быть представлено в формате, который необходим методу ReadXmlSchema класса System.Data.DataSet. Тогда трансформация модели сущности к IoC-конфигурации класа ActiveRecord будет иметь вид (XSL)

```
<xsl:template  match='entity'>
  <component  name="{@name}"
        type="LightweightTransformSample.DbActiveRecord"
        singleton="false">
    <property  name="XmlSchema">
      <xml>
        <xsl:apply-templates  select="schema"/>
      </xml>
    </property>
    <property  name="SourceName">
        <value>
          <xsl:value-of  select="storage/sourcename"/>
        </value>
    </property>
    <property  name="VersionsEnabled">
        <value>
          <xsl:value-of  select="storage/@versions"/>
        </value>
    </property>
  </component>
</xsl:template>
```

Последний этап — это определение конфигурации IoC-контейнера таким образом, чтобы трансформация выполнялась автоматически при загрузке конфигурации контейнера (модели сущностей сохранены в файле entityModel.xml.config, а описание XSL-трансформации — в файле entityModelToIoCTransform.xsl):

```
<components>
  <import
        file="config/entityModel.xml.config"
        xsl-file="config/xslt/entityModelToIoCTransform.xsl"/>
</components>
```

В результате определение «заказ» доступно для связывания с другими компонентами. Этот пример иллюстрирует простейший случай преобразования модели PIM (которая описывает абстрактную пассивную сущность «заказ») в PSM в виде конфигурации Winter IoC-контейнера.

Аналогичным образом строятся и значительно более сложные преобразования. Так, если класс DbActiveRecord не имеет свойства VersionsEnabled, то этот аспект необходимо отобразить через определение еще одной сущности для сохранения версий и генерации конфигурации для соответствующего триггера, который бы создавал новую версию при каждом изменении сущности purchaseOrder. Иными словами, общая стратегия заключается в конфигурации сложных концептов из метамодели PIM в виде композиции более примитивных концептов, которые доступны в PSM.

ЗАКЛЮЧЕНИЕ

Рассмотренный вариант упрощенной инфраструктуры для моделе-ориентированной разработки удовлетворяет современным требованиям создания больших программных систем. Следует отметить, что первичное представление моделей в виде предметно-зависимого XML-формата избавлено от недостатков варианта XMI+XSL.

Модель имеет максимально компактное представление, при этом просмотр и ее изменение доступны программистам без использования каких-либо специализированных инструментов. Сохраняется возможность быстро изменять и расширять метамодель. После стабилизации предметно-зависимой метамодели можно определить трансформации для преобразования используемых предметно-зависимых моделей в любые другие модели (например, UML).

Использование XSL для описания трансформаций приемлемо, поскольку этот язык является мощным инструментом (содержит тьюринг-полный набор функций). Такой подход особенно эффективен в случае, когда выходная модель может быть представлена в формате XML. Это позволяет организовывать как вертикальные трансформации с произвольным количеством слоев абстракции, так и горизонтальные.

В условиях современного процесса разработки, когда необходимо обеспечить минимальный промежуток времени между началом разработки и первыми результатами (feedback), использование упрощенной инфраструктуры (на основе стандартов XML) позволяет создать оптимальные условия для эффективной командной работы программистов.

Предложенный подход к внедрению процесса моделе-ориентированной разработки был апробирован в фирме «NewtonIdeas» (http://www.newtonideas.com), которая специализируется на создании большого количества однородных веб-проектов в области систем управления бизнес-процессами (BPMS, Business Process Management Systems). Позитивный эффект получен уже после первых двух месяцев внедрения, в течение которых формировались основные метамодели и трансформации (на данный момент определено около 900 понятий, которые повторно используются в различных проектах; библиотека из более 1000 классов используется для реализации этих понятий). Менее чем за два года после начала внедрения вышеописанный подход полностью подтвердил свою эффективность: без каких-либо существенных затрат на внедрение степень повторного использования кода (и трансформаций) увеличилась на порядок.

СПИСОК ЛИТЕРАТУРЫ

1. Dijkstra E.W. On the role of scientific thought // Selected Writings on Computing: A Personal Perspective. — N.-Y.: Springer-Verlag, 1982. — 362 p.
2. Перевозчикова О.Л. Основи системного аналізу об'єктів і процесів комп'ютеризації. — К.: Видавничий дім «КМ Академія», 2003. — 432 с.
3. Hunt A., Thomas D. Pragmatic programmer, The: From journeyman to master. — Addison Wesley, 1999. — 352 p.
4. ISO/IEC 12207:1995 Information technology — Software life cycle processes (ДСТУ 3918-99 - Інформаційні технології. Процеси життєвого циклу програмного забезпечення).
5. Czarnecki K., Stahl T., Volter M. Model-driven software development: Technology, engineering, management. — John Wiley&Sons, 2006 — 444 p.
6. Gasevic D., Djuric D., Devedzic V. Model driven architecture and ontology development. — N.-Y.: Springer, 2006. — 312 p.
7. Kleppe A., Warmer J., Bast W. MDA explained: The model driven architecture™: Practice and promise. — Addison-Wesley Professional. — 2003. — 192 p.
8. ISO/IEC 19503:2005 Information technology — XML Metadata Interchange (XMI) (standard).
9. Walls C., Breidenbach R. Spring in action. — Manning Publications; 2 ed. — 2007. — 650 p.
10. Lightweight .NET. Inversion of Control container — Winter4Net (http://www.winter4.net)
11. Fowler M. Inversion of control containers and the dependency injection pattern, 2004 (http://martinfowler.com/articles/injection.html)
12. Bertolazzi P., Krusich C., Missikoff M. An approach to the definition of a core enterprise ontology: CEO / Intern. Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations (OES-SEO 2001). — Luiss Publications, 2001. — 175 p.
13. Fowler M. at el. Patterns of enterprise application architecture. — Addison-Wesley Professional, 2002. — 560 p.

Поступила 22.12.2008