УДК 004.4'242

*Н.Н. Глибовец, В.М. Федорченко*

# ИНФРАСТРУКТУРА ДЛЯ ТРАНСФОРМАЦИИ XML-МОДЕЛЕЙ

Предложен вариант упрощенной инфраструктуры для модель-ориентированной разработки больших программных систем. База построения – первичное представление модели в виде доменно-зависимого XML-формата. Модель имеет максимально компактное представление, допускает использование развитых средств изменения и расширения, позволяет легко определить трансформации для преобразования созданных доменно-зависимых моделей в любые другие модели (например, UML). Использование XSL для описания трансформаций дает возможность эффективно организовывать как вертикальные трансформации с произвольным количеством слоев абстракции, так и горизонтальные; создает оптимальные условия для командной работы.

## Введение

Как способ формализации описания разнообразных процессов программирование, по определению, зависит от конкретных технических ограничений и инфраструктуры, в рамках которой собственно эта формализация проводится. Прогресс в этой области привел к тому, что для многих типов задач приоритет программирования сместился с максимально эффективного (в контексте используемых технических ресурсов) кодирования на максимально быструю (в контексте человеческих ресурсов) и одновременно качественную (quality) разработку [1–4].

**Модель-ориентированная разработка.** Концепция MDD (Model Driven Development) построена на доменно-зависимом языке моделирования и трансформации моделей [5, 6]. В контексте MDD, модель – это абстракция программной системы или/и ее окружения. Согласно этому определению, программный код также является моделью как абстракция над машинным кодом, генерируемым компилятором. Метамодель определяет синтаксис и семантику DSM-языка через определение понятий и их отношений в конкретном домене (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 можно трансформировать в комбинацию исходных текстов, ресурсов и XML-конфигураций (поскольку в контексте MDA программа в любой форме является собственно моделью).

Консорциум OMG предложил свой вариант такой технологии под названием модель-ориентированная архитектура (MDA, model driven architecture) в виде ряда спецификаций и стандартов [7].

Существующие подходы к трансформации входной модели условно разделяют на генерирующие исходную модель в виде текста (обычно текста программы на языке программирования общего назначения: Java, C++) или структурированной формы (например, XML), отвечающей некоторой метамодели. Поскольку программный код также является моделью, отличие в подходах заключается лишь в использовании исходной метамодели. 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, основанных на сочетании нескольких подходов; типичный пример – стандарт QVT, на основе которого можно определять трансформации в MDA [7]. Другим обще-

доступным и мощным трансформатором является реализация стандарта XSLT, поскольку любую модель можно представить в виде XML с помощью формата XMI (XML Metadata Interchange [8]). Однако ввиду достаточной громоздкости формата XMI (для разработчика трансформации) приходим к сложным для разработки и поддержки XSL-правилам.

Для успешного внедрения MDD необходима инфраструктура, поддерживающая модель-ориентированную разработку и удовлетворяющая следующим требованиям: гибкое манипулирование параметрами жизненного цикла, определение и расширение моделей цели при необходимости (by demand), использование одновременно разных уровней абстракции, интеграция с уже существующими программными системами, минимизация расходов на интеграцию и поддержку MDD инфраструктуры.

**Упрощенная инфраструктура.** Рассмотрим вариант такой упрощенной инфраструктуры для модель-ориентированной разработки. Идея состоит в возвращении к первичному представлению модели в виде доменно-зависимого XML-формата, который избавляет от неостатков варианта XMI+XSL: модель имеет максимально компактное представление, причем читать и изменять модель можно без использования программистами специализированных инструментов (редакторов, визуализаторов). Это свойство приобретает первоочередное значение, когда доменная метамодель существенно меняется в процессе разработки. При этом сохраняется возможность быстро изменять и расширять метамодель, а когда доменно-специфическая метамодель стабилизируется, также определять трансформации для преобразования созданных доменно-зависимых моделей в любые другие модели (например, UML).

На практике также можно свести определение метамодели к созданию XML-схемы (XSD) доменно-зависимой модели (метамодели). Этот язык 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. Конечно, XSD имеет весьма ограниченные возможности для описания метамодели, но когда речь идет о максимально упрощенной инфраструктуре этот вариант приемлем как компромисс, поскольку в принципе есть возможность добавлять собственные метаданные к XML-схеме. Когда и этого будет недостаточно, определить метамодель можно с помощью стандарта MOF (MetaObject Facility) [7], допускающего определение любой метамодели.

В данном случае использование XSL для описания трансформаций более приемлемо, поскольку этот язык является мощным инструментом (содержит тьюринг-полный набор функций), особенно когда выходную модель можно также представить в формате XML. Это позволит организовывать как вертикальные трансформации с произвольным количеством слоев абстракции, так и горизонтальные. Широкая распространенность и общеизвестность этого языка создает условия для действительно командной работы, когда определение метамодели и одновременное создание моделей на ее основе выполняет одна команда разработчиков. Это важно для современного процесса разработки, когда необходимо обеспечить минимальный промежуток времени между началом разработки и первыми результатами (feedback).

Последний шаг формирования для модель-ориентированной разработки – определение формата модели низшего уровня, доступного в рамках инфраструктуры. Желательно, чтобы модель имела естественное представление в формате XML, при этом была достаточно простой для превращения в машинный код и одновременно достаточно удобной для выражения концептов из моделей более высоких уровней абстракции (безусловно, эти два условия противоречивы).

XML-представление (SpringFramework [9], Winter.NET [10]) обычно имеет конфигурация IoC-контейнера, реализуемого как паттерн программирования «инверсия управления» (inversion of control),

также известного как «инстанциация зависимостей» (dependency injection) [11]. Суть этого паттерна состоит в абстракции создания и инициализации одних объектов другими, путем делегирования 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 определения зависимостей в виде конкретных экземпляров объектов, реализующих нужные интерфейсы, полагается также на IoC-контейнер. В контексте MDD эта специфика дает уникальную возможность выполнить оба условия для модели низшего уровня, поскольку способ связывания объектов фиксирован (и чрезвычайно прост), а сами объекты могут реализовать достаточно сложные концепты. Создание такого графа по XML-конфигурации является достаточно быстрой и тривиальной задачей [9, 10].

В терминах MDA IoC-конфигурация – это PSM самого низкого уровня (поскольку состоит из ссылок на классы и свойства вполне конкретной платформы), а доменно-зависимые XML модели, если они не имеют привязки к особенностям платформы, относятся к PIM. Конечно, такое деление достаточно условно. В этом варианте можно регулировать уровень абстракции модели от PSM, в зависимости от конкретной ситуации.

**Пример создания упрощенной MDD инфраструктуры.** Для иллюстрации рассмотрим создание упрощенной MDD инфраструктуры для трансформации моделей из домена автоматизации бизнес процессов.

Определение базовых понятий для такой модели можно позаимствовать из многочисленных вариантов «корпоративной онтологии» (enterprise ontology), например Core Enterprise Ontology (CEO [12]), как одной из самых простых. Результат трансформации – конфигурация Winter IoC контейнера [10] для применения на платформе Microsoft .NET.

CEO предложена как домен 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 информации. Их важное свойств – идентифицируемость как необходимое условие включения пассивных сущностей в определение домена данных ИС;
- активные сущности (active entity) – активные элементы из корпоративной среды, которые собственно принимают решения и инициируют те или иные действия. Это могут быть как люди, так и более сложные образования (офис, департамент и т. п.). По отношению к активным сущностям определяются обязанности и права доступа;
- трансформации (transformation) – любые действия, операции и процессы в корпоративной среде, отображенные в ИС. Обычно в трансформациях участвуют как пассивные, так и активные сущности;
- условия – сложные выражения, которые могут быть вычислены с целью определения, выполняются ли они. На этом базовом концепте базируются такие понятия, как «цель», «правило», «ограничение», «состояние».

Отметим, что метамодель на основе этих понятий позволяет описать разнообразные бизнес-процессы. При этом даже простейшее описание этих понятий достаточно объемно, поэтому для иллюстрации упрощенного подхода к трансформации моделей в деталях, рассмотрим лишь определение понятия «сущности».

```
<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 некоторой сущности, например, «заказ», является:

```
<entity name="purchaseOrder">
 <storage versions="true">

<sourcename>purchase_orders</sour
cename>
 </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>
```

Итак, информационная система (ИС) пополняется еще одной пассивной сущностью «purchaseOrder» с характеристиками «id» (уникальный идентификатор), «contact_id», «sum», «delta» и «expired_date». Эти характеристики находятся в хранилище данных (например, БД) под именем «purchase_orders». Эту сущность по определению также можно сохранять, изменять и загружать из хранилища данных; более того, маркировка versions=«true» обязывает ИС хранить все версии этой сущности.

Для определения трансформации этой модели к конфигурации компонентов IoC-контейнера необходимо представить концепт пассивной сущности с описанным выше набором свойств в виде одного или больше связанных объектов. Допустим, в классе, 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, имеем класс DbActiveRecord (реализующий интерфейс IActiveRecord), для функционирования которого необходимо инициализировать свойства XmlSchema, SourceName и VersionsEnabled, причем XmlSchema пред-

ставлена в формате, необходимом классу System.Data.DataSet (ReadXmlSchema). Тогда трансформация модели сущности к IoC-конфигурации класса ActiveRecord будет иметь такой такой 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 шаг – определение конфигурации IoC-контейнера, чтобы трансформация выполнялась автоматически при загрузке конфигурации контейнера (пусть модели сущностей сохранены в файле entityModel.xml.config, а описание XSL-трансформации – в entityModelToIoCTransform.xsl):

```
<components>
 <import

file="config/entityModel.xml.conf
ig"
    xsl-
file="config/xslt/entityModelToIo
CTransform.xsl"/>
</components>
```

Теперь определение «заказа» доступно для связывания с другими компонентами (которое, например, также можно сгенерировать из модели активности).

Этот пример иллюстрирует простейшее превращение доменно-зависимой PIM (описывает абстрактную пассивную сущность «заказа») к PSM в виде конфигурации Winter IoC-контейнера (полный набор исходных текстов этого примера см. http://www.winter4.net/download/LightweightTransformSample.zip).

Аналогично строятся более сложные превращения, например, класс DbActiveRecord мог бы и не иметь свойства VersionsEnabled; тогда этот аспект необходимо было бы 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. То есть общая стратегия заключается в конфигурации сложных концептов из метамодели PIM в виде композиции более примитивных концептов, доступных в PSM.

## Выводы

Предложенный вариант упрощенной инфраструктуры для модель-ориентированной разработки выполняет современные требования создания больших программных систем. Базой построения есть возврат к первичному представлению модели в виде доменно-зависимого XML-формата, избавленного от недостатков варианта XMI+XSL: модель имеет максимально компактное представление, при этом просмотр и изменение модели доступны программистам без использования специальных инструментов. Сохраняется возможность быстро изменять и расширять метамодель. Когда доменно-специфическая метамодель стабилизируется, можно также определить трансформации для преобразования созданных доменно-зависимых моделей в любые другие (стандартизированные) модели (например, UML).

Использование XSL для описания трансформаций приемлемо, поскольку этот язык – мощный инструмент с

тьюринг-полным набором функций. Эффект достижим, когда выходную модель можно представить в формате XML и организовать как вертикальные трансформации с произвольным количеством слоев абстракции, так и горизонтальные.

Распространенность XML создает условия для командной работы, когда определение метамодели и одновременное создание моделей на ее основе выполняет одна команда разработчиков. Последний фактор чрезвычайно важен для современного процесса разработки, когда необходимо обеспечить минимальный промежуток времени между началом разработки и первыми результатами (feedback).

Предложенный подход к модель-ориентированной разработке апробирован в фирме “NewtonIdeas” (http://www.newtonideas.com), специализирующейся на создании большого объема схожих веб-проектов (custom development) для систем управления бизнес-процессами (BPMS, Business Process Management Systems). Позитивный эффект получен уже после первых двух месяцев внедрения, в течение которых сформированы основные доменно-специфические концепты в виде метамоделей и определения трансформаций (на данный момент определено около 900 понятий, повторно используемых в разных проектах; для реализации этих понятий используется библиотека из более 1000 классов). В целом за полтора года использования вышеописанный подход полностью подтвердил свою эффективность: без существенных затрат на внедрение, а коэффициент повторного использования кода (и трансформаций) увеличился до 10 раз.


1. *Dijkstra, Edsger W.* On the role of scientific thought", in Dijkstra, Edsger W., Selected writings on Computing: A Personal Perspective. – New York, NY, USA: Springer-Verlag New York, Inc., 1982. – P. 60–66.
2. *Перевозчикова О.Л.* Основи системного аналізу об’єктів і процесів комп’ютеризації. – К.: Вид. Дім Києво-Могилянської академії, 2003.
3. *Andrew Hunt, David Thomas.* Pragmatic Programmer, The: From Journeyman to Master. – Addison Wesley, 1999.
4. *ДСТУ* 3918-99 (ISO/IEC 12207:1995) Інформаційні технології. Процеси життєвого циклу програмного забезпечення.
5. *Krzysztof Czarnecki.* Model-Driven Software Development: Technology, Engineering, Management. – Wiley, 2006.
6. *Dragan Gasevic, Dragan Djuric, Vladan Devedzic.* Model Driven Architecture and Ontology Development .– Springer, 2006.
7. *Anneke Kleppe, Jos Warmer, Wim Bast.* MDA Explained: The Model Driven Architecture™: Practice and Promise. – Addison Wesley, 2006.
8. *ISO/IEC* 19503:2005 Information technology – XML Metadata Interchange (XMI).
9. *Craig Walls, Ryan Breidenbach.* Spring in Action, Second Edition. – Manning, 2007.
10. Lightweight .NET "Inversion of Control" container – Winter4Net, 2006. – http://www.winter4.net/
11. *Martin Fowler.* 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, OES-SEO 2001, Int. Workshop on Open Enterprise Solutions: Systems, Experiences, and Organizations, Rome, September 14–15, 2001.
13. *Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford.* Patterns of Enterprise Application Architecture. – Addison Wesley, 2002.


Получено 31.03.2011

***Об авторах:***


*Глибовец Николай Николаевич,*
доктор физико-математических наук,
профессор,
декан факультета информатики,

*Федорченко Виталий Михайлович,*
аспирант.

***Место работы авторов:***
Национальний университет
«Киево-Могилянская Академия».
04071, Киев, ул. Г. Сковороды, 2.
Тел.: (067) 209 0740
glib@ukma.kiev.ua