УДК 004.04 DOI: 10.14529/ctcr150402

ФОРМАЛЬНОЕ ПРЕДСТАВЛЕНИЕ МОДЕЛЕЙ ПРИКЛАДНЫХ ПРЕДМЕТНЫХ ОБЛАСТЕЙ В ПОНЯТИЯХ УНИФИЦИРОВАННОЙ МЕТАМОДЕЛИ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

П.П. Олейник

ОАО «Астон»; Шахтинский институт (филиал) Южно-Российского государственного политехнического университета им. М.И. Платова, г. Ростов-на-Дону

Представлено формальное описание объектных моделей в понятиях унифицированной метамодели объектно-ориентированных приложений баз данных, разрабатываемой автором и независящей от предметной области. Описанная метамодель реализована автором в собственной среде разработки SharpArchitect RAD Studio. В начале статьи подробно проанализированы имеющиеся работы сходной тематики. Анализ показал, что чаще всего авторы используют теорию множеств и исчисление предикатов для представления элементов объектно-ориентированных языков программирования и проектирования объектных моделей. Этот же подход использован в данной статье.

В работе представлена метамодель, разработанная автором для проектирования и реализации объектно-ориентированных приложений баз данных. Уделено внимание иерархии выделенных классов, присутствующих в прикладных предметных областях. Также подробно описаны состав и структура классов, позволяющих представить как атомарные литеральные типы, так и классы атрибутов, значениями которых выступают экземпляры классов и коллекцию экземпляров. Уделено внимание реализации динамической составляющей с помощью описания методов (с поддержкой параметров) и событий классов.

На основе представленной метамодели с применением теории множеств предложен формальный подход к описанию моделей прикладных предметных областей. Для анализа применимости предложенного подхода была разработана унифицированная модель тестирования инструментов разработки объектно-ориентированных приложений. Описаны критерии оптимальности этой модели, а также её реализация на языке C# с представлением всех ключевых классов, присутствующих в ней. Предложенный формальный подход был протестирован на описанной модели. Были описаны классы предметной области, атрибуты, ассоциации и образованные иерархии. Это позволило подтвердить корректность предложенного формального решения.

В заключение статьи сделаны предположения о направлениях дальнейшего развития предложенного подхода.

Ключевые слова: объектно-ориентированное программирование, объектно-ориентированные базы данных, метамодель объектной системы, формальное описание объектных моделей.

Введение

В настоящее время доминирующей методологией, используемой при разработке приложений, является объектно-ориентированный подход. В данной статье представлено формальное описание объектных моделей прикладных предметных областей в понятиях унифицированной метамодели объектно-ориентированных приложений баз данных. При этом описываются все виды метаклассов, которые позволяют описать все виды сущностей, имеющихся в системе, а также состав и структура каждого метакласса. Кроме того, в статье уделено внимание ограничениям, наложенным на формальную модель, и приведён список допустимых значений, в том случае, где это возможно. В формальной модели отсутствуют недостатки многочисленных имеющихся работ, краткая характеристика которых представлена в начале статьи. Описанная метамодель реализована автором в собственной среде разработки SharpArchitect RAD Studio.

Также уделено внимание текущей версии унифицированной метамодели, представлены основные типы метаклассов для представления классов, атрибутов, правил валидации и визуализации, а также методов и событий сущностей предметных областей.

1. Обзор имеющихся работ

Объектно-ориентированная парадигма наиболее популярный подход, применяемый сегодня при разработке различных приложений, в том числе и приложений баз данных (БД). Эта парадигма появилась довольно давно, но в отличие от других подходов (например, от реляционной модели данных) не имеет точного формального описания. Однако существуют множество попыток формализации.

В работе [1] авторы предлагают формальную модель языка Z в виде набора операторов, представляющих собой нотацию для объектно-ориентированного языка. Авторы вводят технологию, основанную на модели спецификаций, описывающих трансформации из входной схемы в выходную. При этом указано множество шагов и правил подобного преобразования. В статье [2] представлен подход, на основании которого выполняется формализация UML диаграмм классов с применением нотации Z. Ключевой особенностью работы является то, что кроме статической составляющей (значений свойств экземпляров класса) уделено внимание описанию поведения объектов. То есть дана формальная модель динамической составляющей, представляемой в объектно-ориентированной парадигме в виде методов класса, отображающих множество параметров на множество результирующего значения. В работе [3] авторы представляют собственную алгебру, применяемую для систем объектно-ориентированных баз данных. Особенностью ООБД является идентификация объектов, основанная на объектных идентификаторах. Так как эти идентификаторы уникальны, то по ним можно однозначно найти связанный с идентификатором объект. Именно этот подход используется авторами, которые рассматривают множество идентификаторов как основной элемент модели.

Другая алгебраическая теория объектно-ориентированных приложений описана в работе [4]. Эта работа основывается на понятии абстрактного типа данных (ADT), состоящего из вложенных объектов, соединенных с внешним классом ассоциациями. В качестве ключевых понятий выступают сорты, которые позволяют описать множество всех существующих объектов (экземпляров классов) и множество операций, манипулирующих элементами сорта. На основе этих понятий авторы делают ряд определений, которые позволяют им конструировать как общепринятые структуры, такие как множества, списки, массивы, так и их комбинации неограниченной вложенности.

В работе [5] сделана попытка представить объектные модели в виде теорий. Теоретико-базируемый подход, предложенный авторами, предполагает описание различных структур данных (например, массивов) в виде спецификаций, в которых имеются различные секции для описания статической (полей данных) и поведенческой составляющих (операций, применяемых над специфицируемой структурой). Авторам удалось представить даже множественное наследование и некоторое подобие событий, концепция которых соответствовала времени написания статьи (1995 г.).

Еще один теоретико-базируемый подход к формализации объектно-ориентированных моделей для прикладных предметных областей представлен в работе [6]. Эта работа значительно расширяет результаты работы [5] и вводит ряд дополнительных секций в описание АТД (который в понятиях авторов напоминает класс языка программирования). В работе уделено внимание формализации описания операций, методов и событий, основой которых выступает исчисление предикатов. Этот подход позволил авторам представить только самые простые операции, однако этого достаточно в подавляющем большинстве случаев. Для описания поведенческой составляющей авторы используют конечные автоматы с детерминированным набором состояний, описываемых в определенной секции спецификации.

2. Унифицированная метамодель ОО-приложений

В данном разделе кратко рассмотрим метамодель, используемую в унифицированной среде быстрой разработки корпоративных информационных систем SharpArchitect RAD Studio [7]. В работах [8–14] была представлена полная диаграмма классов метамодели и подробно описано назначение классов. Здесь рассмотрим лишь значимые для данной статьи фрагменты. На рис. 1 представлен фрагмент унифицированной метамодели объектной системы с отображением ключевых ассоциаций, важных для дальнейшего обсуждения.

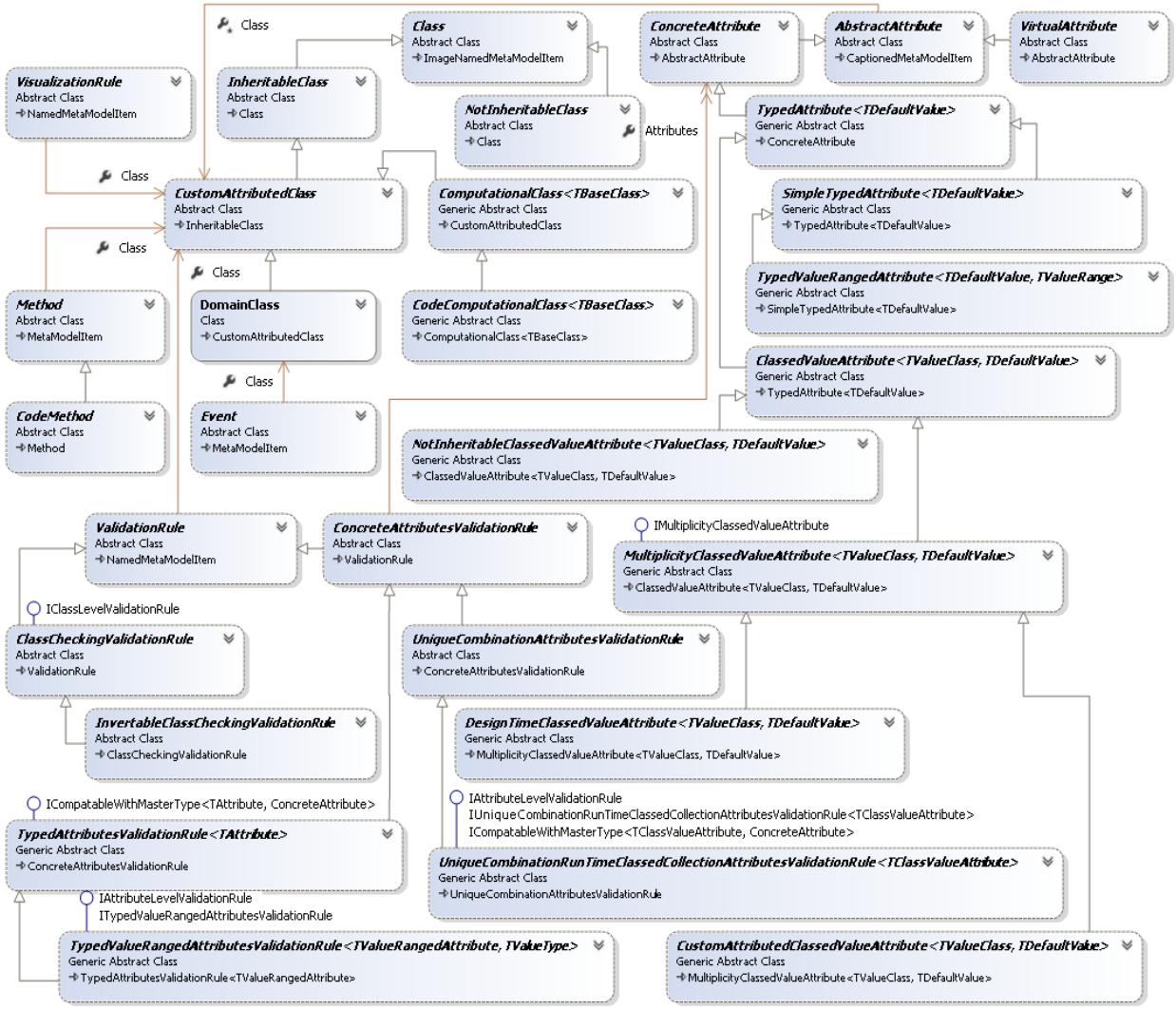

Рис. 1. Фрагмент унифицированной метамодели объектной системы

Корнем иерархии, используемой для представления сущностей предметной области, является абстрактный метакласс Class, имеющий два унаследованных: 1) InheritableClass – используется для представления метаклассов, которые могут быть наследуемыми, т.е. поддерживают наследование; 2) NotInheritableClass – применяется для представления метаклассов, которые не могут быть унаследованы. Матакласс Enum позволяет представить перечисление или множество значений одного простого типа.

Абстрактный базовый метакласс CustomAttributedClass используется для представления метаклассов, которые имеют атрибуты. Метакласс DomainClass применяется для представления классов предметной области. Экземпляры класса предметной области позволяют описывать персистентные классы сущностей (например, Клиент, Продукт, Продажа), объекты которых (например, Иванов, Хлеб) сохраняются в БД. Для упрощения описания будем называть экземпляры класса предметной области классами предметной области (если не предполагается иное).

Абстрактный метакласс ComputationalClass<TBaseClass> является базовым для всех вычисляемых метаклассов, т. е. тех классов, экземпляры которых не сохраняются в БД, а вычисляются в момент выполнения программы (транзиентные). Например, оборотно-сальдовая ведомость не хранится непосредственно в БД, а вычисляется на основе инвентаризации, прихода и расхода (которые являются классами предметной области и представляются экземплярами метакласса DomainClass).

Метакласс MethodParameterClass используется для представления класса-параметра методов. В SharpArchitect RAD Studio реализуется шаблон проектирования (паттерн), называемый Объект-

параметр (Parameter object), суть которого в передаче множества параметров в метод в виде единого объекта вместо множества переменных, имеющих атомарный тип данных.

Абстрактный метакласс CodeComputationalClass<TBaseClass> является базовым для вычисляемых метаклассов, реализуемых с помощью программного кода на языке C#. QueryClass представляет метакласс запроса, позволяющий формировать результат на основе запросов к БД (чаще всего на основе Linq-запросов к объектам, но возможна и прямая посылка SQL-строк). HelperClass используется для представления вспомогательных метаклассов, которые могут как отображаться в интерфейсе пользователя, так и использоваться для внутренних целей при реализации бизнес-логики.

Корневым абстрактным метаклассом, представляющим атрибут, является AbstractAttribute. Унаследованные от VirtualAttribute классы применяются для представления атрибутов, которые не были созданы разработчиком ИС для прикладной предметной области, а были представлены системой. Они необходимы для понимания метамодели и упрощения процесса разработки программного обеспечения. SystemAttribute позволяет описать атрибуты, которые являются системными и присутствуют в языке C#. Класс GeneratedAttribute применяется для представления атрибутов, которые автоматически генерируются системой. Например, при наследовании от базового древовидного класса автоматически добавляется атрибут Node, позволяющий получить дочерние узлы и таким образом образовать иерархическую структуру.

Для представления атрибутов, значения которых может задавать пользователь, используется абстрактный базовый метакласс ConcreteAttribute. Так как система реализуется на языке C#, то при сохранении значений в базе данных используются типы данных этого языка. Для описания этого момента добавлен параметрированный метакласс TypedAttribute<TDefaultValue>. TypeAttribute используется для представления свойств, значения которых могут сохранять значения типа данных языка C#.

3. Формальное представление объектных моделей

Для описания математической модели, позволяющей представить объектную модель информационной системы любой прикладной предметной области, будем использовать формальный аппарат теории множеств. Дальнейшим развитием данного решения может быть модельно-ориентированный подход, при котором понятие типа основано на теоретико-множественной модели, то есть с помощью описания, x: T, «x имеет тип T», что эквивалентно описанию принадлежности ($x \in T$, «x является членом множества T»).

При этом все операции программы можно моделировать как набор манипуляций с множествами.

При проектировании информационных систем необходимо построить модель предметной области, содержащей различные типы сущностей. Например, сущность Продукт, Приход и Расход являются сохраняемыми сущностями, а сущность Оборотно-сальдовая ведомость является вычисляемой, т. е. её экземпляры не сохраняются в БД непосредственно, а рассчитываются на основании других сохраняемых сущностей. С формальной точки зрения, формальная модель модели предметной области, построенной в понятиях унифицированной метамодели объектно-ориентированного приложения БД ($ФММПО_{УММООПБД}$), представляет собой кортеж, состоящий из множеств и представляемый как

$$DMFM = (DC, HC, QC, MPC, EC), \quad (1)$$

где DC (DomainClass) – непустое множество классов предметной области, т. е. $dmfm \in DMFM$, $DC \neq \varnothing$;

HC (HelperClass) – множество вспомогательных классов;

QC (QueryClass) – множество классов-запросов;

MPC (MethodParameterClass) – множество классов-параметров методов;

EC (EnumClass) – множество классов-перечислений и классов-множеств элементов.

Рассмотрим описание каждого перечисленного множества. Классы предметной области (КПО, Domain Class, DC) – это ключевой элемент, используемый для моделирования сущностей предметной области, экземпляры которых сохраняются в БД. В общем случае для описания всех

классов предметной области (DC) используется множество, элементами которых является кортеж и описываемое как

$$DC = \{(ATT, BC_{dc}, M, E, VLR, VSR, BHC, R)\}, \quad (2)$$

где элементы кортежа каждого КПО заданы следующим образом:

ATT (Attribute) – множество атрибутов класса предметной области, $ATT \neq \varnothing \vee BC \neq \varnothing$;

BC_{dc} (BaseClass) – множество базовых классов предметной области, от которого унаследован данный, $DC \notin BC_{dc}$;

M (Method) – множество методов класса, позволяющих реализовать поведение экземпляров классов, т. е. динамическую составляющую;

E (Event) – множество обработчиков событий, возникающих в жизненном цикле объекта класса предметной области;

VLR (ValidationRule) – множество предикатов, представляющих валидационные правила, которым должен отвечать каждый объект;

VSR (VisualizationRule) – множество визуализационных правил, которые управляют видимостью, доступностью, цветом отдельных атрибутов;

BHC (BehaviorController) – множество контроллеров поведения, позволяющих управлять как поведением объектов, так и пользовательским интерфейсом приложения;

R (Report) – множество отчетов системы, позволяющих выводить экземпляры класса (объекта) в удобном для пользователя виде с возможностью распечатки данных.

При этом выполняются следующие ограничения:

$dc \in DC, \forall\, bc_{dc} \in BC_{dc} \subsetneq DC \Rightarrow dc \notin BC_{dc}$.

Для описания свойств (характеристик) экземпляров классов (объектов) необходимо описать множество атрибутов ATT, формально описываемое как кортеж вида

$$ATT = \{(Name, AttributeKind, Multiplicity, dc, hc, ec)\}, \quad (3)$$

где Name – уникальное название атрибута (правильный идентификатор);

AttributeKind – вид атрибута;

Multiplicity – множественность атрибута (минимальное и максимальное количество связанных объектов);

dc – класс предметной области ($dc \in DC$), указывается когда AttributeKind = DomainClassAttribute или AttributeKind = GeneratedAttribute;

hc – вспомогательный класс ($hc \in HC$), указывается когда AttributeKind = HelperClassAttribute или AttributeKind = GeneratedAttribute;

ec – класс-перечисление / класс-множество ($ec \in EC$), указывается когда AttributeKind = = EnumAttribute.

Система предоставляет разработчику множество различных видов атрибутов, которые можно представить как

$$\begin{aligned} AttributeKind = \{&BuiltInClassAttribute, ColorAttribute, DateTimeAttribute,\\ &DecimalAttribute, DomainClassAttribute, EnumAttribute, FileDataAttribute,\\ &GeographyAttribute, GeometryAttribute, HelperClassAttribute,\\ &HyperLinkAttribute, ImageAttribute, IntAttribute, LogicalAttribute,\\ &MetaModelClassAttribute, MoneyAttribute, ObjectAttribute, StringAttribute,\\ &SymbolAttribute, TextAttribute, TimeAttribute, TypeAttribute\}, \end{aligned} \quad (4)$$

где BuiltInClassAttribute – встроенно-классовый атрибут, используется для сохранения экземпляра метамодели;

ColorAttribute – цветовой атрибут, используется для представления целочисленной константы описывающей цвет;

DateTimeAttribute – дата-временной атрибут, используется для представления даты и времени;

DecimalAttribute – дробный атрибут, значением которого является дробное число;

DomainClassAttribute – предметно-классовый атрибут, значением которого является экземпляр (объект) класса предметной области. Часто используется для организации отношений ассоциаций;

EnumAttribute – перечисляемый / множественный атрибут, используется для хранения значения перечисления / множества;

FileDataAttribute – файловый атрибут, используется для сохранения содержимого файла;

GeneratedAttribute – сгенерированный атрибут, применяется для представления атрибутов, которые автоматически генерируются системой;

GeographyAttribute – географический атрибут, используется для представления географических координат;

GeometryAttribute – геометрический атрибут, используется для хранения геометрических объектов;

HelperClassAttribute – вспомогательно-классовый атрибут, используется для сохранения экземпляров вспомогательных классов, для реализации вычислений;

HyperLinkAttribute – гиперссылочный атрибут, значением которого является гиперссылка;

ImageAttribute – графический атрибут, используется для сохранения изображений;

IntAttribute – целочисленный атрибут, применяется для хранения целочисленных значений;

LogicalAttribute – логический атрибут, используется для хранения булева значения (0 или 1);

MetaModelClassAttribute – метамодельно-классовый атрибут, применяется для сохранения экземпляра описания класса;

MoneyAttribute – денежный атрибут, применяется для сохранения значений в валюте;

ObjectAttribute – объектный атрибут, используется для сохранения объекта любого типа;

StringAttribute – строковый атрибут, применяется для сохранения строки текста;

SymbolAttribute – символьный атрибут, используется для сохранения символа;

TextAttribute – текстовый атрибут, применяется для сохранения текста неограниченной длины с форматированием;

TimeAttribute – временной атрибут, используется для хранения времени;

TypeAttribute – типовый атрибут, применяется для сохранения названия типа данных.

Допустимыми значениями для множественности являются:

Multiplicity = {0..1, 1..1, 0..*, 1..*}.

Так как предполагается организация двунаправленных ассоциаций, реализуемых с помощью описания двух атрибутов на разных краях, то подобным способом можно описать и другие виды множественности, например, многие-ко-многим (*..*). Отметим, что обязательными элементами в выражении (4) являются лишь Name, AttributeKind, т. е. имя и вид атрибута. Остальные компоненты необязательны и могут отсутствовать при описании атрибута.

Множество M позволяет реализовать поведение экземпляров классов в виде методов, т. е. динамическую составляющую и представимо в виде

$$M = \{(Name, mpc, Body)\}, \quad (5)$$

где Name – название метода (правильный идентификатор);

mpc – параметр метода, mpc $\in$ MPC;

Body – строки программного кода, реализующего метод.

Каждый метод представляет собой процедуру (функцию, не возвращающую результат). При этом функция может иметь параметр, представляемый элементом множества MPC. То есть реализуется шаблон (паттерн) проектирования объект-параметр (parameter object), когда в качестве параметра передается экземпляр класса.

Множество E представляет собой множество обработчиков событий, возникающих в жизненном цикле объекта (экземпляра) сущности, описываемой классом предметной области и представляемыми в виде

$$E = \{(Name, EventKind, Body)\}, \quad (6)$$

где Name – название обработчика события(правильный дентификатор);

EventKind – вид события для которого создается обработчик;

Body – строки программного кода, реализующего обработчик события.

Для описания видов событий используется перечисление вида:

EventKind = {AfterChangedAttributeValueEvent, AfterDeletedEvent,
AfterLoadedEvent, AfterSavedEvent, BeforeDeletingEvent, (7)
BeforeSavingEvent, InitializationEvent},

где AfterChangedAttributeValueEvent – событие, вызываемое после изменения значения атрибута объекта;

AfterDeletedEvent – событие, вызываемое после удаления объекта;
AfterLoadedEvent – событие, вызываемое после загрузки объекта;
AfterSavedEvent – событие, вызываемое после сохранения объекта;
BeforeDeletingEvent – событие, вызываемое перед удалением объекта;
BeforeSavingEvent – событие, вызываемое перед сохранением объекта;
InitializationEvent – событие инициализации объекта.

Множество предикатов, представляющих валидационные правила, которым должен отвечать каждый объект, представимо множеством VLR, которое описывается как

$$VLR = \{(cr, ATT_{VLR})\}, \quad (8)$$

где cr – предикат, имеющий параметры, которыми выступают атрибуты классов, которому должен соответствовать экземпляр сущности предметной области;

ATT_{VLR} – множество атрибутов, из которого построено валидационное правило.

Для улучшения анализа данных и упрощения представления используется множество визуализационных правил VSR, представляющих собой неупорядоченные тройки вида

$$VSR = \{(cr, vrk, ATT_{vsr})\}, \quad (9)$$

где cr – предикат, определяющий применимость визуализационных правил;
vrk – определяет вид визуализационного правила;
ATT_{vsr} – множество атрибутов, на которое распространяется визуализационное правило.
При этом выполняется:
$ATT_{vsr} \subseteq ATT$;
$vrk \in VRK$,
где VRK = {HideProperty, DisableProperty, SetFontColor, SetBackgroundColor},
где HideProperty – данный вид правила скрывает атрибуты;
DisableProperty – данный вид правила делает неактивными атрибуты;
SetFontColor – данный вид правила устанавливает цвет шрифта в редакторах атрибутов;
SetBackgroundColor – данный вид правила устанавливает цвет фона в редакторах атрибутов.

Множество контроллеров поведений BHC состоит из элементов, каждый из которых представляет собой класс языка программирования, реализующий требуемый функционал.

Множество отчетов R представляет набор отчетов, каждый из которых описывается расширяемым языком разметки XML и содержит данные, интерпретируемые прикладной программой.

Перейдем к рассмотрению других видов классов, представляющих отдельные элементы метамодели.

Формальное описание множества вспомогательных классов (HC):

$$HC = \{(ATT, bc_{hc}, PC, M, VSR, BHC, R)\}, \quad (10)$$

где ATT (Attribute) – множество атрибутов класса предметной области;
bc_{dc} (BaseClass) – базовый вспомогательный класс, от которого унаследован данный;
PC (ProgramCode) – программный код реализации вспомогательного класса, представляемый в виде множество строк языка C#;
M (Method) – множество методов класса, позволяющих реализовать поведение экземпляров классов, т. е. динамическую составляющую;
VSR (VisualizationRule) – множество визуализационных правил, которые управляют видимостью, доступностью цветом отдельных атрибутов;
BHC (BehaviorController) – множество контроллеров поведения, позволяющих управлять как поведением объектов, так и пользовательским интерфейсом приложения;
R (Report) – множество отчетов системы, позволяющих выводить экземпляры класса (объекта) в удобном для пользователя виде с возможностью распечатки данных.

Пи этом выполняются следующие ограничения:
$\forall$ umm $\in$ UMM, $\forall$ hc $\in$ HC, $ATT \neq \varnothing \vee bc_{dc} \neq \varnothing \vee PC \neq \varnothing$,
$\forall$ hc $\in$ HC, $\forall$ $bc_{hc} \in HC \Rightarrow hc \neq bc_{hc}$.

Формальное описание множества классов-запросов (QC):

$$QC = \{(ATT, PC, VSR, BHC, R)\}, \quad (11)$$

где ATT (Attribute) – множество атрибутов класса-запроса;

PC (ProgramCode) – программный код реализации класса-запроса, представляемый в виде множество строк языка C#;

VSR (VisualizationRule) – множество визуализационных правил, которые управляют видимостью, доступностью, цветом отдельных атрибутов;

BHC (BehaviorController) – множество контроллеров поведения, позволяющих управлять как поведением объектов, так и пользовательским интерфейсом приложения;

R (Report) – множество отчетов системы, позволяющих выводить экземпляры класса (объекта) в удобном для пользователя виде с возможностью распечатки данных.

При этом выполняются следующие ограничения:

$\forall$ umm $\in$ UMM $\forall$ qc $\in$ QC, ATT $\neq \varnothing \vee$ PC $\neq \varnothing$.

В (11) представлено формальное описание множества классов-параметров методов (MPC):

$$MPC = \{(ATT, bc_{mpc}, UM, M, VLR, VSR, BHC, R)\}, \quad (12)$$

где ATT (Attribute) – множество атрибутов класса-параметра метода;

bc_{mpc} (BaseClass) – базовый класс-параметр методов, от которого унаследован данный;

UM (UsingMethod) – множество методов, которые используют данный класс в качестве параметра;

M (Method) – множество методов класса, позволяющих реализовать поведение экземпляров классов, т.е. динамическую составляющую;

VLR (ValidationRule) – множество предикатов, представляющих валидационные правила, которым должен отвечать каждый объект;

VSR (VisualizationRule) – множество визуализационных правил, которые управляют видимостью, доступностью, цветом отдельных атрибутов;

BHC (BehaviorController) – множество контроллеров поведения, позволяющих управлять как поведением объектов, так и пользовательским интерфейсом приложения;

R (Report) – множество отчетов системы, позволяющих выводить экземпляры класса (объекта) в удобном для пользователя виде с возможностью распечатки данных.

При этом выполняются следующие ограничения:

$\forall$ umm $\in$ UMM $\forall$ mpc $\in$ MPC, ATT $\neq \varnothing \vee bc_{mpc} \neq \varnothing$,

$\forall$ mpc $\in$ MPC, $\forall bc_{mpc} \in$ MPC $\Rightarrow$ mpc $\neq bc_{mpc}$.

Формальное описание множества классов-перечислений (EC), представляющих именованные константы с присвоенными целыми значениями:

$$EC = \{(Name, ek, \{val_i = 2^i - 1\})\}, \quad (13)$$

где $i \in 0..n$;

ek $\in$ EnumKind;

EnumKind = {Enum, Set} – тип класса перечисления, Enum – описывает перечисление, а Set – множество.

Из представленных формул видно, что выделены все ключевые моменты объектной метамодели, поэтому можно перейти к применению формального аппарата к тестовой модели.

4. Применение формального описания для представления унифицированной модели тестирования

В данном разделе будем рассматривать применение формальной модели на унифицированной модели тестирования инструментов разработки ОО-приложений [18].

Перед проектированием были выдвинуты следующие требования (критерии оптимальности (КО)):

1. Необходимо наличие глубоких иерархий наследования.
2. Присутствие нескольких иерархий наследования.
3. Наличие абстрактных классов в иерархии.
4. Присутствие множественных (n-арных) ассоциаций.
5. Наличие ассоциаций с атрибутами.
6. Присутствие композиции между классами.
7. Наличие рекурсивных ассоциаций.
8. Наличие ассоциаций между классами, входящими в одну иерархию наследования.

9. Присутствие класса-ассоциации.

10. Наличие ассоциаций между классом-ассоциацией и другим классом.

11. Присутствие в модели перечислений.

Все описанные выше КО были реализованы в тестовой модели и графически представлены на рис. 2 [18].

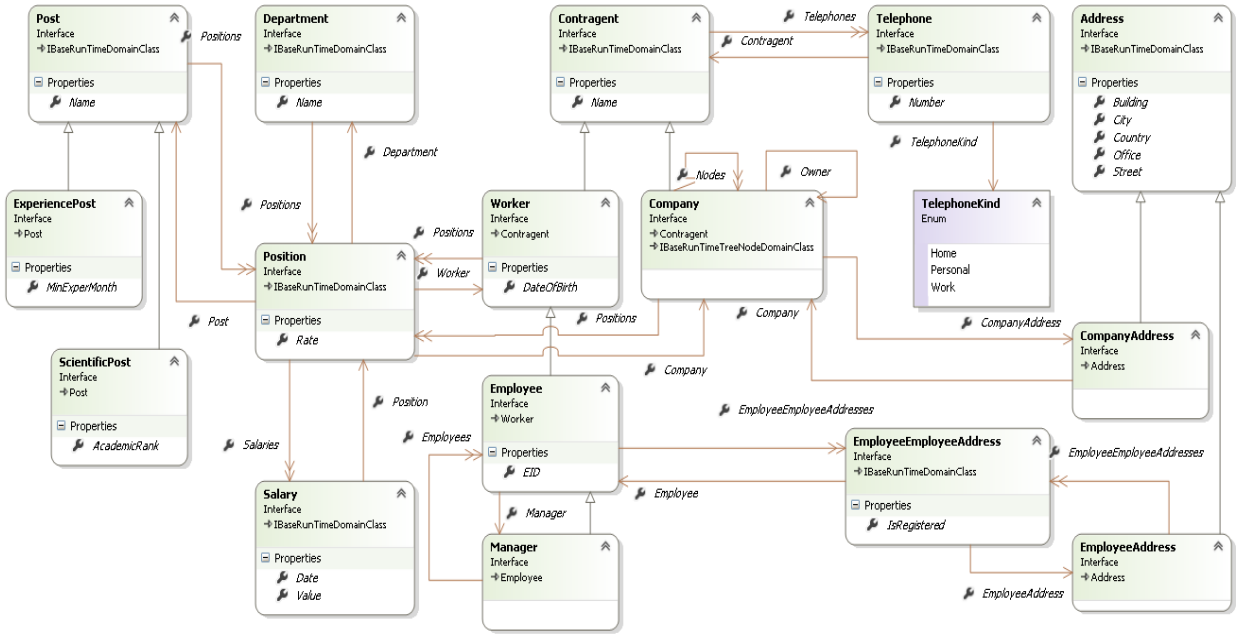

Рис. 2. Унифицированная модель тестирования инструментов разработки объектно-ориентированных приложений, реализованная в SharpArchitect RAD Studio на языке C# [18]

При реализации использовались интерфейсы языка C#, поэтому невозможно курсивом выделить абстрактные классы. Двунаправленные ассоциации показаны соответствующими стрелками, соединяющими классы. При реализации ассоциаций использовался следующий подход. Со стороны «один» создавалось свойство (property), типом которого является список (C# тип IList()), содержащий элементы, типом которых является класс, расположенный со стороны «ко-многим». При этом со стороны «ко-многим» в классе объявляется свойство, типом которого является класс, расположенный со стороны «один». Ассоциация типа «многие-ко-многим» (без атрибутов) может быть представлена двумя списками, объявленными в противоположных классах. В среде разработки SharpArchitect RAD Studio имеется ряд базовых классов, в которых реализован наиболее общий функционал. Например, класс IBaseRunTimeDomainClass является корневым для всех классов предметной области. Для реализации древовидной структуры достаточно унаследоваться от IBaseRunTimeTreeNodeDomainClass. В момент кодогенерации автоматически будут генерироваться дополнительные атрибуты Nodes и Owner, позволяющие сохранять ссылки на вложенные и родительский узлы соответственно. Именно таким способом реализуются рекурсивные ассоциации. Для представления перечислений и множеств используется синтаксическая конструкция enum.

Перейдем к формальному описанию представленной модели. На основании формулы (1) получаем следующее:

dmfm = (DC),

т. е. унифицированная модель тестирования состоит только из классов предметной области, которыми являются элементы множества DC, определенного как

DC = {Post, ExperiencePost, ScientificPost, Department, Position, Salary,
Contagent, Company, Worker, Employee, Manager, Telephone, Address,
CompanyAddress, EmployeeAddress, EmployeeEmployeeAddress}.

На основании формулы (2) в нашем случае будем использовать формулу вида

$DC = \{(ATT, BC_{dc})\}$.

То есть рассмотрим только формальные описания атрибутов и базовых классов, при этом либо атрибуты, либо базовые классы могут быть пустыми (но не оба одновременно).

Post = ({(Name, StringAttribute), (Positions, DomainClassAttribute, 0..*, Position)})

ExperiencePost = ({(MinExperMonth, IntAttribute)}, {Post})

ScientificPost = ({(AcademicRank, StringAttribute)}, {Post})

Department = ({(Name, StringAttribute)
,(Positions, DomainClassAttribute, 0..*
, Position)})

Position = ({(Rate, DecimalAttribute), (Post, DomainClassAttribute, 1..1, Post), (Department, DomainClassAttribute, 1..1, Department), (Company, DomainClassAttribute, 1..1, Company), (Worker, DomainClassAttribute, 1..1, Worker), (Salaries, DomainClassAttribute, 0..*, Salary)})

Salary = ({(Date, DateTimeAttribute), (Value, DecimalAttribute), (Position, DomainClassAttribute, 1..1, Position)})

Contragent = ({(Name, StringAttribute), (Telephones, DomainClassAttribute, 0..*, Telephone)})

Telephone = ({(Number, StringAttribute), (Contragent , DomainClassAttribute, 1..1, Contragent), (TelephoneKind, EnumAttribute, TelephoneKind)})

Company = ({(CompanyAddress, DomainClassAttribute, 1..1, CompanyAddress), (Owner, GeneratedAttribute, 0..1, Company), (Nodes, GeneratedAttribute, 0..*, Company), (CompanyAddress, DomainClassAttribute, 1..1, CompanyAddress), (Positions, DomainClassAttribute, 0..*, Position)}, {Contragent, IBaseRunTimeTreeNodeDomainClass})

Worker = ({(DateOfBirth, DateTimeAttribute), (Positions, DomainClassAttribute, 0..*, Position)}, {Contragent})

Employee = ({(EID, StringAttribute), (Manager, DomainClassAttribute, 0..1, Manager), (EmployeeEmployeeAddresses, DomainClassAttribute, 0..*, EmployeeEmployeeAddress)}, {Worker})

Manager = (Employees, DomainClassAttribute, 0..*, Employee)}, {Employee})

Address = ({(Country, StringAttribute), (City, StringAttribute), (Street, StringAttribute), (Building, StringAttribute), (Office, StringAttribute)})

CompanyAddress = ({(Company, DomainClassAttribute, 1..1, Company)}, {Address})
EmployeeAddress = ({(EmployeeEmployeeAddresses, DomainClassAttribute, 0..*, EmployeeEmployeeAddress)}, {Address})
EmployeeEmployeeAddress = ({(IsRegistered, LogicalAttribute), (Employee, DomainClassAttribute, 1..1, Employee), (EmployeeAddress, DomainClassAttribute, 0..*, EmployeeAddress)})

Для описания перечислений / множеств будем использовать запись вида:

EC = {(TelephoneKind, Enum, {Home = 0, Personal = 1, Work = 2})}.

Из представленного выше видно, что мы получили полное формальное описание унифицированной модели тестирования инструментов разработки объектно-ориентированных приложений, описываемой в понятиях метамодели.

Выводы и дальнейшие исследования

Итогом данной статьи явилось формальное описание унифицированной модели тестирования инструментов разработки объектно-ориентированных приложений с помощью подхода, представленного выше и основанного на теории множеств. Несмотря на возможность формального описания практически любых объектных моделей на основе имеющихся метаклассов, в будущем предполагается расширить формальный аппарат с учетом развития самой метамодели, а также добавить различные опции, присутствующие в настоящий момент. Например, необходимы опции для указания абстрактных классов, а также опции для указания необходимости сохранения экземпляров классов в БД. В будущих статьях предполагается разработать формальные описания ограничений, налагаемых на описываемые модели, и тем самым реализовать механизм валидации для моделей прикладных предметных областей.

Литература

1. Periyasamy, K. Deriving test cases for composite operations in Object-Z specifications / K. Periyasamy, V.S. Alagar, S. Subramanian // Proc. Technology of OO Languages and Systems (TOOLS 26), Santa Barbara, CA, August 1999. – P. 429–441.

2. Shroff, M. Towards a Formalization of UML Class Structures in Z / M. Shroff, R. France // Proceedings, 21st International Computer Software and Applications Conference (COMPSAC'97), Washington DC, August 1997. – P. 646–651.

3. Shugang Wang. Object identity set algebra for object-oriented database systems / Shugang Wang // 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2012). – 2012. – P. 1–6.

4. Yu, X.M. An Algebraic Theory of Object-Oriented Systems / X.M. Yu, T.S. Dillon // IEEE Transactions on Knowledge and Data Engineering archive. – June 1994. – Vol. 6, iss. 3. – P. 412–419.

5. DeLoach, S. Representing object models as theories / S. DeLoach, P. Bailor, T. Hartrum // Proceedings 10th Knowledge-Based Software Engineering Conference. – Nov. 1995. – P. 28–35.

6. DeLoach, S.D. A Theory-Based Representation for Object-Oriented Domain Models / S.D. DeLoach, T.C. Hartrum // IEEE Transactions on Software Engineering. – June 2000. – Vol. 26, iss. 6. – P. 500–517.

7. Унифицированная среда быстрой разработки корпоративных информационных систем SharpArchitect RAD Studio: свидетельство о государственной регистрации программы для ЭВМ № 2013618212 / П.П. Олейник. – Опубл. 04.09.2013.

8. Олейник, П.П. Иерархия классов метамодели объектной системы / П.П. Олейник // Объектные системы – 2012: материалы VI Междунар. науч.-практ. конф. (Ростов-на-Дону, 10–12 мая 2012 г.) / под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – С. 37–40. – http://objectsystems.ru/files/2012/Object_Systems_2012_Proceedings.pdf

9. Олейник, П.П. Иерархия классов представления валидационных правил объектной системы / П.П. Олейник // Объектные системы – 2013: материалы VII Междунар. науч.-практ. конф. (Ростов-на-Дону, 10–12 мая 2013 г.) / под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ (ф) ЮРГТУ (НПИ), 2013. – С. 14–17. – http://objectsystems.ru/files/2013/Object_Systems_2013_Proceedings.pdf

10. Oleynik, P.P. Domain-driven design the database structure in terms of metamodel of object system / P.P. Oleynik // Proceedings of 11th IEEE East-West Design & Test Symposium (EWDTS'2013), Institute of Electrical and Electronics Engineers (IEEE). Rostov-on-Don, Russia, September 27–30, 2013. – Rostov-on-Don, 2013. – P. 469–472.

11. Олейник, П.П. Элементы среды разработки программных комплексов на основе организации метамодели объектной системы / П.П. Олейник // Бизнес-информатика. – 2013. – № 4 (26). – С. 69–76. – http://bijournal.hse.ru/data/2014/01/16/1326593606/1BI%204(26)%202013.pdf

12. Олейник, П.П. Предметно-ориентированное проектирование структуры базы данных в понятиях метамодели объектной системы / П.П. Олейник // Объектные системы – 2014: материалы VIII Междунар. науч.-практ. конф. (Ростов-на-Дону, 10–12 мая 2014 г.) / под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, 2014. – С. 41–46. – http://objectsystems.ru/files/2014/Object_Systems_2014_Proceedings.pdf

13. Oleynik P.P. Using metamodel of object system for domain-driven design the database structure // Proceedings of 12th IEEE East-West Design & Test Symposium (EWDTS'2014), Kiev, Ukraine, September 26–29, 2014. DOI: 10.1109/EWDTS.2014.7027052

14. Олейник, П.П. Концепция создания обслуживающей корпоративной информационной системы экономического производственно-энергетического кластера / П.П. Олейник, Ю.И. Кураков // Прикладная информатика. – 2014. – № 6. – С. 5–23.

15. Олейник, П.П. К вопросу о необходимости проектирования иерархии атомарных литеральных типов для объектной системы, организованной в РСУБД / П.П. Олейник // Информационно-вычислительные технологии и их приложения. IX Междунар. науч.-техн. конф.: сб. ст. – Пенза: РИО ПГСХА, 2008. – С. 201–205.

16. Олейник, П.П. Организация иерархии атомарных литеральных типов в объектной системе, построенной на основе РСУБД / П.П. Олейник // Программирование. – 2009. – № 4. – С. 73–80.

17. Oleynik, P.P. Implementation of the Hierarchy of Atomic Literal Types in an Object System Based of RDBMS / P.P. Oleynik // Programming and Computer Software. – 2009. – Vol. 35, no. 4. – P. 235–240.

18. Олейник, П.П. Унифицированная модель тестирования инструментов разработки объектно-ориентированных приложений / П.П. Олейник // Объектные системы – 2014 (Зимняя сессия): материалы IX Междунар. науч.-практ. конф. (Ростов-на-Дону, 10–12 декабря 2014 г.) / под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ (ф) ЮРГПУ (НПИ) им. М.И. Платова, 2014. – С. 23–32. – http://objectsystems.ru/files/2014WS/Object_Systems_2014_Winter_session_Proceedings.pdf

Олейник Павел Петрович, канд. техн. наук, системный архитектор программного обеспечения, ОАО «Астон»; доцент, Шахтинский институт (филиал) Южно-Российского государственного политехнического университета им. М.И. Платова, г. Ростов-на-Дону; xsl@list.ru.

Поступила в редакцию 30 июня 2015 г.

DOI: 10.14529/ctcr150402

FORMAL REPRESENTATION MODELS APPLIED OF DOMAINS IN TERMS OF UNIFIED OBJECT-ORIENTED DATABASE APPLICATIONS METAMODEL

***P.P. Oleynik**, xsl@list.ru*

Aston JSC; Shakhty Institute (branch) of Platov South Russian State Polytechnic University (NPI), Rostov-on-Don, Russian Federation

This article presents a formal description of object models in terms of a unified object-oriented metamodel for database applications, developed by the author and independent of the domain. Described metamodel is implemented by the author in own development environment which called SharpArchitect RAD Studio. In the beginning of the article the available works of similar subjects are thoroughly analyzed. Analysis have revealed that most authors use the set theory and predicate calculus for representation of the elements of object-oriented programming languages and design of object models. The same approach is used in this article.

This article presents a metamodel, developed by the author for the design and implementation for object-oriented database applications. Attention is paid to the hierarchy of the classes presented in the application domains. Also the composition and structure of classes that allows to present both atomic literal types and classes of attributes which values are the instances of classes and collection of copies are described in detail. Attention is paid to the implementation of the dynamic component by description of methods (with support of the parameters) and present classes events.

The formal approach to the description of the models for any application domains are proposed on the basis of the metamodel with using of set theory. The unified model of testing of object-oriented applications development tools was developed for the analysis of the applicability of the proposed approach. Optimality criteria of this model as well as its implementation in C# with the representation of all main classes presented in it are described. The proposed formal approach has been tested on the described model. Domain classes, attributes, associations and whole hierarchies have been described. It allowed to confirm the validity of the proposed formal solution.

In conclusion the assumptions about the directions of further development of the proposed approach are made.

Keywords: object-oriented programming, object-oriented databases, object system metamodel, formal description of object models.

References

1. Periyasamy K., Alagar V.S., Subramanian S. Deriving Test Cases for Composite Operations in Object-Z Specifications. Proc. Technology of OO Languages and Systems (TOOLS 26), Santa Barbara, CA, August 1999, pp. 429–441.

2. Shroff M., France R., Towards a Formalization of UML Class Structures in Z. Proceedings, 21st International Computer Software and Applications Conference (COMPSAC'97), August 1997, Washington DC, pp. 646–651. DOI: 10.1109/cmpsac.1997.625087

3. Shugang Wang. Object Identity Set Algebra for Object-Oriented Database Systems. 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2012), 2012, pp. 1–6. DOI: 10.1109/SOCA.2012.6449439

4. Yu X.M., Dillon T.S. An Algebraic Theory of Object-Oriented Systems. IEEE Transactions on Knowledge and Data Engineering archive, vol. 6, iss. 3, June 1994, pp. 412–419.

5. DeLoach S., Bailor P., Hartrum T. Representing Object Models as Theories. Proceedings 10th Knowledge-Based Software Engineering Conference, Nov 1995, pp. 28–35. DOI: 10.1109/kbse.1995.490116

6. DeLoach S.D., Hartrum T.C. A Theory-Based Representation for Object-Oriented Domain Models. IEEE Transactions on Software Engineering, vol. 26, iss. 6, June 2000, pp. 500–517.

7. Oleynik P.P. *Programma dlya EVM "Unifitsirovannaya sreda bystroy razrabotki korporativnykh informatsionnykh sistem SharpArchitect RAD Studio": svidetel'stvo o gosudarstvennoy registratsii № 2013618212 ot 04 sentyabrya 2013 g.* [Computer program "The Unified Environment of Rapid Development of Corporate Information Systems SharpArchitect RAD Studio": the certificate on the state registration № 2013618212/ 04, September 2013].

8. Oleynik P.P. [Class Hierarchy of Object System Metamodel]. Object Systems – 2012: Proceedings of the Sixth International Theoretical and Practical Conference. Rostov-on-Don, Russia, 10–12 May, 2012, pp. 37–40. Available at: http://objectsystems.ru/files/2012/Object_Systems_2012_Proceedings.pdf (in Russ.)

9. Oleynik P.P. [Class Hierarchy for Presentation Validation Rules of Object System]. Object Systems – 2013: Proceedings of the Seventh International Theoretical and Practical Conference (Rostov-on-Don, 10–12 May, 2013). Rostov-on-Don, SI (b) SRSTU (NPI), 2013, pp. 14–17. Available at: http://objectsystems.ru/files/2013/Object_Systems_2013_Proceedings.pdf (in Russ.)

10. Oleynik P.P. Domain-Driven Design the Database Structure in Terms of Metamodel of Object System. Proceedings of 11th IEEE East-West Design & Test Symposium (EWDTS'2013), Institute of Electrical and Electronics Engineers (IEEE), Rostov-on-Don, Russia, September 27–30, 2013, pp. 469–472.

11. Oleynik P.P. [The Elements of Development Environment for Information Systems Based on Metamodel of Object System]. *Business Informatics*, 2013, no. 4 (26), pp. 69–76. Available at: http://bijournal.hse.ru/data/2014/01/16/1326593606/1BI%204(26)%202013.pdf (in Russ.)

12. Oleynik P.P. [Domain-Driven Design of the Database Structure in Terms of Object System Metamodel]. Object Systems – 2014: Proceedings of the Eighth International Theoretical and Practical Conference (Rostov-on-Don, 10–12 May, 2014). Rostov-on-Don, SI (b) SRSPU (NPI), 2014, pp. 41–46. Available at: http://objectsystems.ru/files/2014/Object_Systems_2014_Proceedings.pdf (in Russ.)

13. Oleynik P.P. Using Metamodel of Object System for Domain-Driven Design the Database Structure. Proceedings of 12th IEEE East-West Design & Test Symposium (EWDTS'2014), Kiev, Ukraine, September 26–29, 2014. DOI: 10.1109/EWDTS.2014.7027052

14. Oleynik P.P., Kurakov Yu.I. [The Concept Creation Service Corporate Information Systems of Economic Industrial Energy Cluster]. *Applied Informatics*, 2014, no. 6, pp. 5–23. (in Russ.)

15. Oleynik P.P. [To a Question of Necessity of Design of Atomic Iiteral Types Hierarchy for the Object System Organized in Relational Database Control System]. *Informatsionno-vychislitel'nye tekhnologii i ikh prilozheniya. IX Mezhdunarodnaya nauchno-tekhnicheskaya konferentsiya: sbornik statey* [Information Technologies and their Applications. Proc. of IX International Scientific and Technical Conference]. Penza, RIO PGSKhA, 2008, pp. 201–205. (in Russ.)

16. Oleynik P.P. [The Organization of Atomic Literal Types Hierarchy in the Object System Constructed on the Basis of Relational Database Control System]. *Programmirovanie* [Programming], 2009, no. 4, pp. 73–80. (in Russ.)

17. Oleynik P.P. Implementation of the Hierarchy of Atomic Literal Types in an Object System Based of RDBMS. *Programming and Computer Software*, 2009, vol. 35, no.4, pp. 235–240. DOI: 10.1134/S0361768809040070

18. Oleynik P.P. [Unified Model for Testing of Tools for Object-Oriented Application Development]. Object Systems – 2014 (Winter session): Proceedings of IX International Theoretical and Practical Conference (Rostov-on-Don, 10–12 December, 2014). Rostov-on-Don: SI (b) SRSPU (NPI), 2014, pp. 25–35. Available at: http://objectsystems.ru/files/2014ws/Object_Systems_2014_Winter_session_Proceedings.pdf (in Russ.)

Received 30 June 2015

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Олейник, П.П. Формальное представление моделей прикладных предметных областей в понятиях унифицированной метамодели объектно-ориентированных приложений баз данных / П.П. Олейник // Вестник ЮУрГУ. Серия «Компьютерные технологии, управление, радиоэлектроника». – 2015. – Т. 15, № 4. – С. 12–25. DOI: 10.14529/ctcr150402

FOR CITATION

Oleynik P.P. Formal Representation Models Applied of Domains in Terms of Unified Object-Oriented Database Applications Metamodel. *Bulletin of the South Ural State University. Ser. Computer Technologies, Automatic Control, Radio Electronics*, 2015, vol. 15, no. 4, pp. 12–25. (in Russ.) DOI: 10.14529/ctcr150402