УДК 004.435

АРХИТЕКТУРА ПРОЦЕССОРА МЕТАДАННЫХ

Д. В. Ковалев,
аспирант
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»
В. В. Шаповалов,
доктор техн. наук, профессор
Федеральное государственное научное учреждение «Научно-исследовательский конструкторско-технологический институт биотехнических систем»

Приводятся краткие сведения о методологии метауправления и описывается подход, обеспечивающий независимость реализации метауправления в информационных системах от конкретного языка представления и средства доступа к метаданным.

Введение

На заре развития компьютерных технологий разработка программного обеспечения (ПО) была искусством, которым владела горстка посвященных. Сейчас программирование относится к категории общепризнанных научных технологий [1]. Создание, внедрение и сопровождение программных систем — это набирающий обороты бизнес, в который вовлечены сотни тысяч квалифицированных специалистов.

Свободная конкуренция вынуждает производителей ПО выпускать качественные программы в максимально короткие сроки. Современные реалии таковы, что помимо первичных требований к программным системам, таких как полнота функциональных возможностей, надежность, производительность и дружественность интерфейса пользователя, все большую значимость приобретают требования, ранее бывшие вторичными. Речь идет об оперативности создания и модернизации программных продуктов, а также независимости успешного выполнения проекта от конкретных исполнителей. В таких условиях, как отмечает Эдвард Йордон [2], параметры (срок разработки, штат разработчиков и бюджет) большинства современных софтверных проектов отклоняются от нормальных значений на 50 % и больше. Подобные проекты Йордон относит к категории «безнадежных» и утверждает, что «безнадежные проекты являются нормой, а не исключением».

В целом проблема «выживания» в безнадежных проектах является весьма актуальной и только совершенствованием методов управления решена быть не может — необходимо использование прогрессивных технологий, таких как обобщенное и объектно-ориентированное программирование, программирование с использованием шаблонов (паттернов), мультипарадигменное программирование, рефакторинг, разработка через тестирование, методология метауправления функциональностью информационных систем и др. Применение этих технологий должно повысить сопровождаемость ПО и, как следствие, оперативность разработки, а также обеспечить читабельность исходного текста, полноту и актуальность документации, что позволит добиться независимости от исполнителей.

Таким образом, исследования, направленные на повышение гибкости и унифицированности прогрессивных технологий, всегда будут своевременными и актуальными. В настоящей статье рассматривается архитектура процессора метаданных MDE (Metadata Engine) как средства повышения гибкости реализации метауправления в информационных системах.

Методология метауправления

Метауправление (МУ) — это методология построения информационных систем (ИС), при которой пользователю предоставляется возможность варьировать структуру данных и содержание вычислительного процесса их обработки при изменениях в соответствующей предметной области [3]. Другими словами, в ИС с МУ реализована возможность метауправления функциональностью во время ее эксплуатации с использованием понятийной модели соответствующей предметной области.

Совокупность структур хранения данных и алгоритмов выполнения типовых операций их обработки можно рассматривать как алфавит ИС, который определим как язык реализации функциональности (ЯРФ). При метауправлении со сторо-

■ ***Рис. 1.*** *Использование языков описания и реализации функциональности*

ны пользователя алфавиту ставится в соответствие метаописание понятийной модели предметной области и типовых операций обработки предметных данных. Используя имеющийся алфавит, пользователь формирует (изменяет) метаописания, далее преобразуемые в описания структур баз данных (БД) и (или) определяющие содержание процессов обработки предметных данных.

Таким образом, суть МУ функциональностью информационной системы заключается в возможности задания метаописаний компонентов ИС, определяющих ее функциональность, на предметно-ориентированном дескриптивном языке. Выполнение метаописаний должно приводить к получению результатов решения прикладных задач, а при необходимости — и к изменению используемых структур хранения данных.

Алфавит γ ЯРФ образуют идентификаторы структур хранения данных в БД и заданных на них отношений γ_d, а также идентификаторы алгоритмов обработки данных γ_a. Синтаксис ЯРФ определяется его алфавитом и правилами ν композиции символов алфавита в логически правильные (синтаксически допустимые) цепочки (предложения), описывающие структуру БД (λ_d) и процессы обработки данных (λ_a). В свою очередь, правила ν — это правила, являющиеся результатом применения синтаксических правил инструментальных лингвистических средств, использованных при создании БД и функциональных приложений ИС к алфавиту γ. Для любой ИС можно определить следующие понятия:

$\gamma = \gamma_d \cup \gamma_a$ — алфавит ЯРФ;

$G_R = (\gamma, \nu)$ — грамматика, описывающая синтаксис ЯРФ;

$L_R(G_R)$ — собственно ЯРФ как множество всех терминальных цепочек, порождаемых грамматикой G_R;

$\lambda = \lambda_d \cup \lambda_a$ — совокупность предложений ЯРФ ($\lambda \subset L_R(G_R)$), реализованных в ИС и определяющих ее функциональность.

Реализация свойства синтаксической вариантности (возможности варьирования алфавита) в ИС предполагает введение нового языка — языка описания функциональности (ЯОФ), предоставляемого пользователю для задания (описания) γ_d, γ_a и λ_d, λ_a (рис. 1). По своей сути это должен быть предметно-ориентированный язык декларативного типа, синтаксис и семантика которого определяются при создании конкретной ИС (на рис. 1 этот язык обозначен L_c, а его грамматика — G_c).

Поскольку, манипулируя предложениями на ЯОФ, пользователь осуществляет управление синтаксисом ЯРФ и (или) описаниями на ЯРФ, а предложения на ЯРФ определяют структуру БД и возможные процессы содержательной обработки данных в интересах решения прикладных задач, в синтаксически вариантной информационной системе реализуется двухуровневая схема управления. Верхний уровень управления по данной схеме является уровнем метауправления.

Совокупность предложений на ЯОФ U_c (см. рис. 1) можно рассматривать как совокупность компонентов метаинформации (МИ), имеющейся в ИС. Каждый компонент метаинформации u_c является одним предложением на ЯОФ L_c ($u_c \in L_c$).

Таким образом, метауправление функциональностью ИС заключается в разнесении предметно-ориентированного описания компонентов, определяющих функциональность ИС, и программной реализации этих компонентов так, что изменение описания приводит к формально вычисляемым изменениям в указанной выше реализации.

Особенности реализации метауправления в информационных системах

Для представления МИ необходима соответствующая модель. В настоящее время во множестве приложений с МУ используется так называемая базовая модель представления метаинформации (БМПМ), организующая знания в виде дерева следующих понятий: сущностей, атрибутов, экземпляров сущностей, значений атрибутов и отношений.

В качестве языка представления метаинформации во многих ИС с МУ используется расширяемый язык разметки XML (Extensible Markup Language), соответствующий БМПМ (таблица).

Однако такое решение ограничивает разработчиков возможностями языка и средств доступа к сформированным на этом языке компонентам МИ. XML как языку представления МИ присущи следующие недостатки:

1) отсутствие типизации атрибутов;

2) отсутствие поддержки для указания отношений между сущностями, кроме отношения «предок — потомок», что вызвано отсутствием поддержки для идентификации:

- экземпляров сущностей;
- xml-документа как категории сущностей;

3) большой объем xml-документа.

Что касается средств доступа к компонентам МИ, описанным на XML, то их недостатками являются:

1) низкая производительность, что обуславливается:

- необходимостью загрузки всего xml-документа в оперативную память;
- возможностью только последовательного доступа к экземплярам сущностей;

2) отсутствие приемлемого контроля семантической целостности (действительности) xml-документа.

Кроме того, метаданные в свою очередь могут выступать в роли данных, подлежащих учету в другой ИС. В этом случае метаданные должны храниться в сетевой или локальной БД, а для доступа должны использоваться средства соответствующей системы управления базами данных (СУБД), которые для разных СУБД принципиально различны.

Таким образом, использование при реализации МУ в информационных системах конкретного языка представления метаинформации и, следовательно, соответствующих средств доступа к метаданным в общем случае представляется неэффективным.

Наиболее предпочтительным выглядит подход, обеспечивающий независимость реализации МУ в ИС от конкретного языка представления, что достигается путем внесения дополнительной прослойки между средствами формирования, перевода и интерпретации компонентов МИ и метаданными. При этом метаданные целесообразно представлять в виде БМПМ. В качестве упомянутой прослойки можно предложить процессор метаданных MDE (Metadata Engine).

■ *Соответствие понятий БМПМ и XML*

Понятие БМПМ	Понятие XML
Категория сущностей	Документ
Сущность	Описание тега (например, в определении типа документа DTD)
Экземпляр сущности	Тег
Атрибут	Описание атрибута (например, в определении типа документа DTD)
Значение атрибута	Значение атрибута
Отношение	Вложенность тегов

Процессор метаданных MDE

MDE — это унифицированный механизм доступа к метаданным (рис. 2).

Абоненты метаданных — это объекты ИС с МУ, осуществляющие формирование, перевод и интерпретацию компонентов МИ. Для доступа к метаданным абоненты используют MDE, а точнее — провайдеры служб через интерфейсы оболочки.

Процессор метаданных MDE предоставляет абонентам доступ к метаданным, выраженным с использованием конкретного средства представления, в виде БМПМ. Таким образом, MDE обеспечивает независимость абонентов от средств представления метаданных. MDE состоит из оболочки доступа к метаданным и провайдеров служб.

Оболочка доступа к метаданным представляет метаданные в виде БМПМ. Оболочка является набором полиморфных интерфейсов семейства IMetaDataEngine. Под интерфейсом понимается абстрактный класс, состоящий только из открытых «чисто» виртуальных функций. Здесь и далее используются понятия объектно-ориентированного программирования (ООП), подробнее о котором можно узнать, например, из работы [4].

Провайдеры служб представляют метаданные, выраженные с использованием конкретных средств представления, в виде БМПМ. Доступ к метаданным осуществляется через провайдеры метаданных. Провайдеры служб являются классами, реализующими интерфейсы семейства IMetaDataEngine. Под реализацией интерфейса понимается наследование и реализация всех его методов. Для каждого средства представления метаданных должна быть своя группа провайдеров служб.

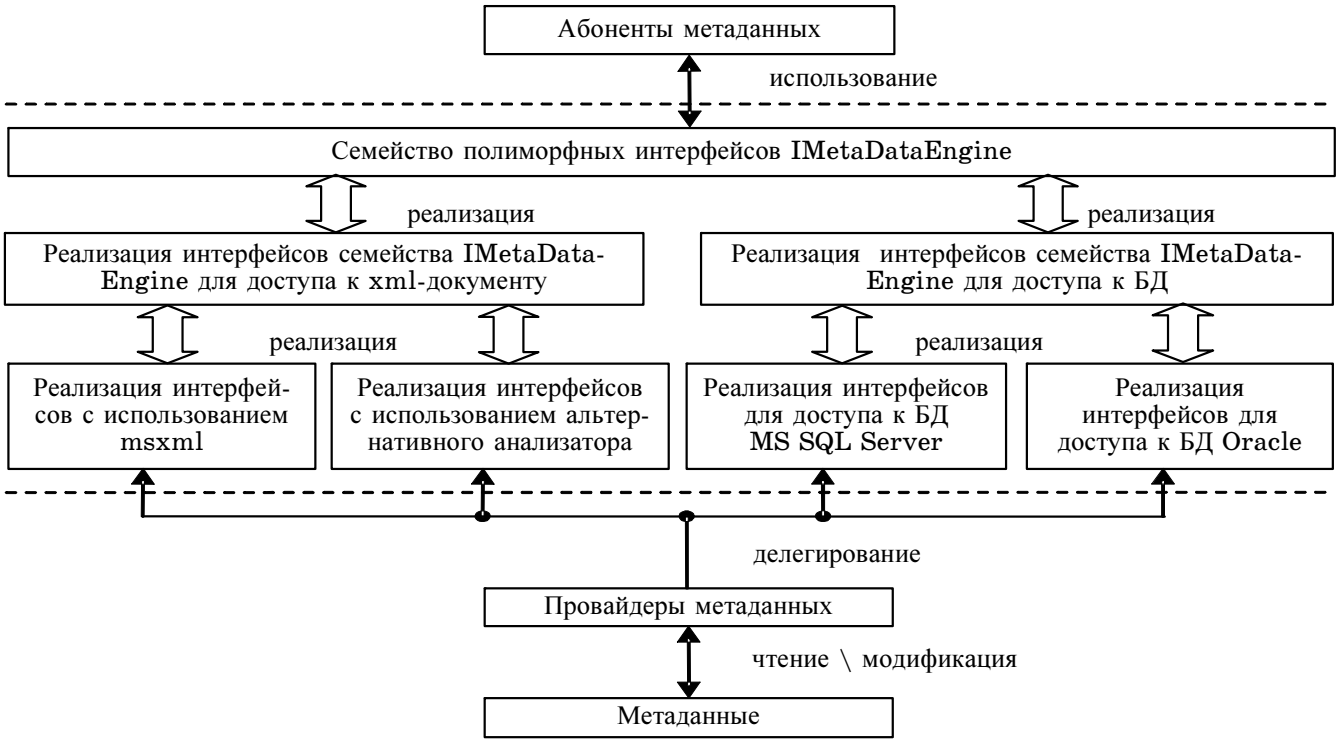
Провайдеры метаданных обеспечивают доступ к метаданным, выраженным с использованием конкретного средства представления.

Метаданные — это выраженные с использованием конкретного средства представления компоненты МИ.

■ ***Рис. 2.*** *Структурная схема доступа к метаданным посредством MDE*

Таким образом, при использовании MDE разработчик абонентов метаданных оперирует не с понятиями языка представления МИ, такими как «xml-тег», «xml-атрибут», «таблица БД», «столбец таблицы БД» и т. п., а с понятиями БМПМ.

Рассмотрим MDE подробнее. На рис. 3 показана структура гипотетического MDE, обеспечивающего доступ к метаданным, одна часть которых хранится в виде xml-документов, вторая — в БД MS SQL Server, а третья — в БД Oracle. До-



■ ***Рис. 3.*** *Структура гипотетического процессора метаданных MDE*

ступ к xml-документам осуществляется использованием анализатора MSXML (Microsoft XML Core Services) и некоторого альтернативного гипотетического анализатора.

К семейству IMetaDataEngine относятся следующие интерфейсы:

— интерфейсы для доступа к структуре МИ:
  IMDEntity (сущность);
  IMDAttribute (атрибут сущности);
  IMDRelationship (отношение);
— интерфейсы для доступа к МИ:
  IMDInstance (экземпляр сущности);
  IMDProperty (значение атрибута сущности).

IMDEntity соответствует понятию «сущность» БМПМ и представляет ее (сущности) структуру, а именно: наименование, сведения об атрибутах (IMDAttribute) и сведения о сущностях, находящихся в отношении с данной сущностью (IMDRelationship).

IMDAttribute соответствует понятию «атрибут сущности» БМПМ и представляет о нем (атрибуте) следующие сведения: наименование, тип данных, ограничения целостности данных.

IMDRelationship соответствует понятию «отношение» БМПМ и представляет о нем (отношении) следующие сведения: сведения об объектах отношения (IMDEntity), вид отношения (зависимость, ассоциация, агрегация, композиция или обобщение). Виды отношений заимствованы из унифицированного языка моделирования UML (Unified Modeling Language) [5].

IMDInstance соответствует понятию «экземпляр сущности» БМПМ и инкапсулирует следующие данные: ссылку на сведения о структуре сущности (IMDEntity), значения атрибутов (IMDProperty), а также сведения об экземплярах сущностей, находящихся в отношении с экземпляром данной сущности (IMDInstance, IMDRelationship).

IMDProperty соответствует понятию «значение атрибута сущности» БМПМ и инкапсулирует следующие данные: ссылку на сведения об атрибуте (IMDAttribute), данные атрибута.

Соответствующие диаграммы классов на языке UML представлены на рис. 4, *а—в*.

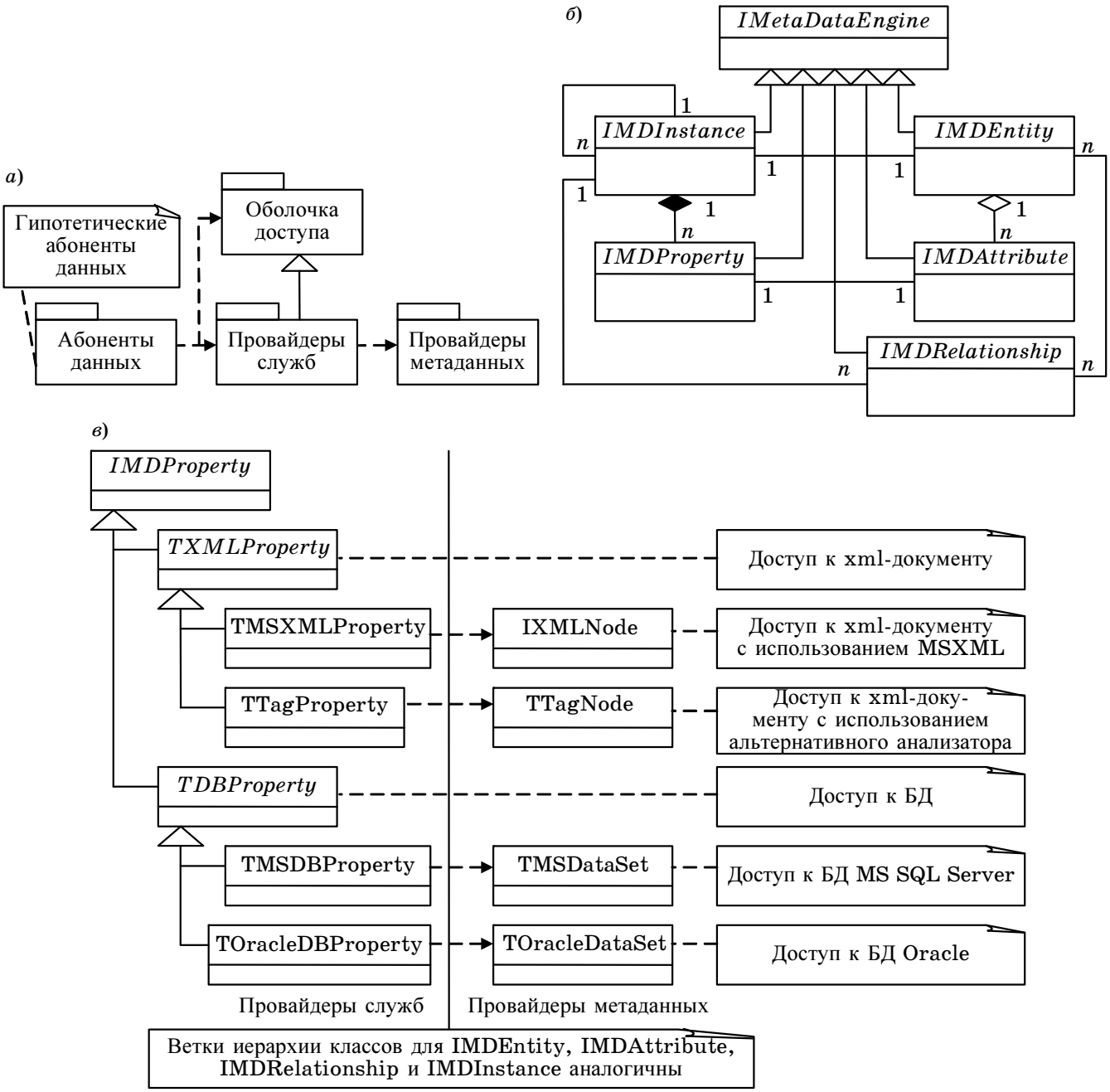
В качестве примера реализации MDE в листинге приведен исходный текст части интерфейса IMDProperty на языке C++, детально описанном в работе [6].

Листинг. Интерфейс IMDProperty

```
#define interface struct
//-----------------------------
interface IMetaDataEngine
{
 …
}
//-----------------------------
interface IMDAttribute : public IMetaDataEngine
{
 // Получить наименование атрибута
 virtual string getName () = 0;
 …
}
//-----------------------------
interface IMDProperty : public IMetaDataEngine
{
 // Получить указатель на соответствующий атрибут
 virtual IMDAttribute* getAttr() = 0;
 // Чтение/установка данных
 virtual Variant getData() = 0;
 virtual void setData( const Variant vValue) = 0;
 …
}
//-----------------------------
class TXMLProperty : public IMDProperty
{
 virtual IMDAttribute* getAttr() {…}
 …
}
//-----------------------------
class TMSXMLProperty : public TXMLProperty
{
 private:
 IXMLNode* FXMLNode; // Провайдер метаданных

 public:
 // Чтение/установка данных
 virtual Variant getData() = 0;
 virtual void setData( const Variant vValue) = 0;
};
//-----------------------------
class TTagXMLProperty : public TXMLProperty
{
 private:
 TAttr* FAttr; // Провайдер метаданных

 public:
 // Чтение/установка данных
 virtual Variant getData() = 0;
 virtual void setData( const Variant vValue) = 0;
};
//-----------------------------
class TDBProperty : public IMDProperty
{
 virtual IMDAttribute* getAttr() {…}
 …
}
//-----------------------------
class TMSDBProperty : public TDBProperty
{
 private:
 TMSDataSet* FDataSet; // Провайдер метаданных

 public:
 // Чтение/установка данных
 virtual Variant getData() = 0;
 virtual void setData( const Variant vValue) = 0;
};
//-----------------------------
class TOracleDBProperty : public TDBProperty
{
 private:
 TOracleDataSet* FDataSet; // Провайдер метаданных

 public:
 // Чтение/установка данных
 virtual Variant getData() = 0;
 virtual void setData( const Variant vValue) = 0;
};
//-----------------------------
Variant TMSXMLProperty::getData()
{
 return FXMLNode->NodeValue;
}
//-----------------------------
void TMSXMLProperty::setData( const Variant vValue )
```

```
{
 FXMLNode->NodeValue = vValue;
}
//-----------------------------
Variant TTagXMLProperty::getData()
{
 return FAttr->Value;
}
//-----------------------------
void TTagXMLProperty::setData( const Variant vValue )
{
 FAttr->Value = vValue;
}
```

```
//-----------------------------
Variant TMSDBProperty::getData()
{
 return FDataSet->FieldByName(
 getAttr()->getName()
 )->AsVariant;
}
//-----------------------------
void TMSDBProperty::setData( const Variant vValue )
{
 FDataSet->FieldByName(
 getAttr()->getName()
 )->AsVariant = vValue;
```



■ ***Рис. 4.*** *Диаграмма классов гипотетического MDE: а — уровень 1; б — уровень 2, пакет «Оболочка доступа»; в — уровень 2, пакеты «Провайдеры служб» и «Провайдеры метаданных»*

```
}
//-----------------------------
Variant  TOracleDBProperty::getData()
{
 return FDataSet->FieldByName(
 getAttr()->getName()
 )->AsVariant;
}
//-----------------------------
void TOracleDBProperty::setData( const Variant vValue )
{
 FDataSet->FieldByName(
 getAttr()->getName()
 )->AsVariant = vValue;
}
//-----------------------------
void SetVals( IMDProperty* Prop )
{
 Prop->SetData(666);
}
//-----------------------------
void main()
{
 IMDProperty* Prop1 = new TMSXMLProperty;
 IMDProperty* Prop2 = new TTagXMLProperty;
 IMDProperty* Prop3 = new TMSDBProperty;
 IMDProperty* Prop4 = new TOracleDBProperty;
 SetVals( Prop1 );
 SetVals( Prop2 );
 SetVals( Prop3 );
 SetVals( Prop4 );
 delete Prop1;
 delete Prop2;
 delete Prop3;
 delete Prop4;
}
```

Пример демонстрирует независимость абонента данных (функция SetVals от конкретного представления метаданных).

К достоинствам предложенного подхода следует отнести гибкость и унифицированность, а к недостаткам — необходимость дополнительных накладных расходов на реализацию MDE.

Заключение

В настоящей работе рассмотрен процессор метаданных MDE как средство повышения гибкости реализации метауправления в информационных системах путем обеспечения независимости объектов информационных систем с метауправлением, осуществляющих формирование, перевод и интерпретацию компонентов метаинформации, от средств представления метаданных.

Литература

1. **Дейкстра Э.** Ремесленник или ученый? // Компьютеры. 2002. № 41. http://offline.computerra.ru/2002/466/21138/
2. **Йордон Э.** Путь камикадзе. М.: Лори, 2003. 272 с.
3. **Шерстюк Ю. М.** Основы метауправления функциональностью в информационных системах. СПб.: СПИИРАН, 2000. 156 с.
4. **Буч Г.** Объектно-ориентированный анализ и проектирование. 2-е изд. СПб.: Бином, 2001. 560 с.
5. **Буч Г., Рамбо Д., Джекобсон А.** Язык UML. Руководство пользователя. СПб.: Питер, 2004. 430 с.
6. **Страуструп Б.** Язык программирования C++. Специальное издание. СПб.: Бином, 2001. 1099 с.

ПАМЯТКА ДЛЯ АВТОРОВ

Поступающие в редакцию статьи проходят обязательное рецензирование.

При наличии положительной рецензии статья рассматривается редакционной коллегией. Принятая в печать статья направляется автору для согласования редакторских правок. После согласования автор представляет в редакцию окончательный вариант текста статьи.

Процедуры согласования текста статьи могут осуществляться как непосредственно в редакции, так и по e-mail (80x@mail.ru).

При отклонении статьи редакция представляет автору мотивированное заключение и рецензию, при необходимости доработать статью — рецензию. Рукописи не возвращаются.

Редакция журнала напоминает, что ответственность за достоверность и точность рекламных материалов несут рекламодатели.