Представление основных синтаксических конструкций объектного языка запросов (OQL) с помощью XML

П.П. Олейник

Аннотация. В статье описан способ представления синтаксических конструкций объектного языка запросов (OQL) в виде XML-документов. Проведён анализ существующих решений сходных задач и способов применения XML-представления конструкций языка.
Определены критерии, которым должна соответствовать оптимальная реализация. Описаны XML-схема и основные синтаксические конструкции разработанного языка запросов XOQL, который соответствует сформулированным критериям. Представлены примеры XOQL-запросов.

Ключевые слова: объектный язык запросов, XML-языки, базы данных, объектно-реляционные отображения.

Введение

Расширяемый язык разметки XML используется для представления слабоструктурированных данных, т.е. информации, структура которой либо неизвестна, либо в будущем претерпит кардинальные изменения [1]. Благодаря своей гибкости, наглядности и существованию большого количества вспомогательных технологий данный язык применяется в различных звеньях ИС (на сервере системы управления базами данных, на клиентском приложении, на сервере приложений). Одной из проблем, требующей сохранения информации, структура которой заранее неизвестна, является задача представления синтаксиса языка запросов (ЯЗ) к данным. Реализация подобного ЯЗ в виде XML-документов позволит унифицировать процесс манипулирования данными, который в этом случае не зависит от конкретного источника данных и позволяет тиражировать данное решение для различных программно-аппаратных платформ. Т.е. появляется возможность выполнения XML-запросов в гетерогенной среде с помощью создания механизма отображения синтаксических конструкций разработанного языка на конкретный ЯЗ, поддерживаемый источником данных. Примерами подобных языков является структурированный язык запросов SQL (Structured Query Language) и объектный язык запросов OQL (Object Query Language).

Классическая реализация предполагает представление синтаксических конструкций языка запросов в программном приложении в виде текстовых строк, содержащих ключевые слова [2] и соответствующих определённой грамматике. Подобной реализации присущи следующие недостатки:

1. Необходимость полной реализации синтаксического (семантического) анализа. Требуется разработать классы для управления строками, которые позволят проверить соответствие запроса определенной для него грамматике и строить синтаксическое дерево для введенных лексем языка запросов.

2. Необходимость реализации объектной модели, применяемой для программной генерации запроса. Современные приложения разрабатываются на объектно-ориентированных языках программирования (ООЯП) и манипу-

лируют экземплярами классов, вызывая их методы и присваивая значения свойствам. Оптимальной для использования является генерация запроса с помощью создания экземпляров классов, представляющих синтаксические конструкции ЯЗ.

Для любого языка необходимо также предусмотреть процедуру выполнения запроса и манипулирования данными. Мы не рассматриваем в данной статье этот вопрос, т.к. предполагается, что сформированный запрос будет трансформирован в язык запросов, поддерживаемый конкретной СУБД.

Использование XML для представления синтаксических конструкций языка запросов позволит избавиться от описанных недостатков и абстрагировать язык от конкретной СУБД, используемой для хранения информации. Основная часть синтаксических конструкций любого языка запросов предназначена для выборки информации (сохранённой в БД), удовлетворяющей заданным критериям, поэтому в данной статье рассматриваются только эти конструкции, которые описаны в виде XML-документов. Запросы, составленные с помощью таких документов, будем называть XOQL-запросы, а сам язык запросов (конкретное приложение языка XML, для которого объявлена схема, позволяющая выполнять валидацию) - XOQL (XML Object Query Language).

1. Анализ имеющихся инструментов и схожих работ

Все имеющиеся работы можно разделить на три крупные категории:

1. работы, описывающие реализацию языков запросов в инструментах объектно-реляционного отображения;

2. статьи, рассматривающие проблемно-ориентированные языки запросов, используемые в определённых предметных областях;

3. работы, описывающие способы представления отдельных синтаксических конструкций языка запросов в виде XML-документов.

Каждая из этих категорий подробно описана в следующих подразделах.

1.1. Языки запросов, реализуемые в инструментах ОРО

В настоящее время при разработке информационных систем всё чаще используют инструменты объектно-реляционного отображения (ОРО), позволяющие организовать объектную систему на основе реляционной СУБД. Практически каждый инструмент ОРО реализует собственный диалект объектного языка запросов. Так, программный продукт Hibernate поддерживает развитый язык HQL (Hibernate Query Language) и содержит в своём составе классы объектной модели, позволяющие генерировать запрос [14]. Запросы на этом языке могут быть представлены в виде XML-документа (Рис. 1).

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
       "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.hibernatebook.criteria">
  <class name="Product">
    <id name="id" type="int">
      <generator class="native"/>
    </id>
    <property name="name" type="string"/>
    <property name="description" type="string"/>
    <property name="price" type="double"/>
    <many-to-one name="supplier" class="Supplier" column="supplierId"/>
  </class>
  <query name="com.hibernatebook.criteria.Product.HQLpricing">
    <![CDATA[select product.price from Product product where product.price > 25.0]]>
  </query>
  <sql-query name="com.hibernatebook.criteria.Product.SQLpricing">
    <return-scalar column="price" type="double"/>
    <![CDATA[select product.price from Product as product where product.price > 25.0]]>
  </sql-query>
</hibernate-mapping>
```

Рис. 1. Пример HQL-запроса, извлекающего информацию о стоимости продуктов, цена которых превышает 25

Из Рис. 1 видно, что запрос представлен в виде строки текста, не использована возможность организации иерархии тегов для более чёткого представления структуры и синтаксических элементов. Разработчики языка HQL прибегли к классической реализации, о которой говорилось ранее и реализовали как синтаксический анализ и проверку соответствия введённых строк определённой грамматике, так и развитую иерархию классов, позволяющую программно генерировать запросы. В случае использования языка XML (и сопутствующих технологий) этих трудностей частично можно избежать.

Описанный инструмент в качестве хранилища информации использует реляционную базу данных для организации объектной системы и поддерживает метаинформацию о классах в клиентском приложении, но ключевым недостатком этого продукта (и аналогичных продуктов), по мнению автора, является формирование динамического запроса к СУБД на клиентском приложении (для извлечения экземпляров класса, хранящихся в различных таблицах), что негативно сказывается на скорости выполнения. Перемещение преобразований из объектного языка в ЯЗ, поддерживаемый конкретным источником данных (ИД), на серверную часть приложения, позволит повысить скорость выполнения запросов. К сожалению, данная возможность не предусматривается в подобных инструментах. В свою очередь при представлении запросов в виде XML-документов появляется возможность перенести процесс трансформации на сервер БД, например, в хранимую процедуру, т.к. многие современные СУБД имеют тип данных xml.

Технология Enterprise JavaBeans (EJB), которую можно рассматривать как инструмент ОРО, реализованный на языке Java, поддерживает язык запросов EJB-QL, позволяющий манипулировать объектами [12]. Имеется возможность записи EJB-QL-запроса в файл конфигурации (в дескрипторе поставки), который является XML-документом. На Рис. 2 представлен пример запроса, извлекающего информацию о структурных подразделениях организации.

Из рисунка видно, что для непосредственной записи запроса используется только один тег <ejb-ql>, значением которого выступает строка SQL-подобного ЯЗ, то есть сам язык представляется в виде текстовой строки, записанной в XML-узел. Этот подход аналогичен тому, что используется при представлении HQL-запросов в программном продукте Hibernate. Поэтому ему присущи недостатки, описанные ранее.

1.2. Проблемно-ориентированные языки запросов

Вопросы организации языка запросов затрагиваются в теоретических работах, посвященных методам объектно-реляционного отображения (ОРО). В [13] описана реализация ЯЗ, позволяющего возвратить экземпляры классов объектной системы, организованной на основе реляционной СУБД. На Рис. 3 представлен пример запроса, который может быть построен на основании XML-схемы, опубликованный в анализируемой работе.

Данная реализация позволяет извлекать, в основном, темпоральные данные и привязана к конкретной предметной области (медицинские

```
<query>
  <query-method>
    <method-name>findAll</method-name>
    <method-params/>
  </query-method>
  <ejb-ql>SELECT OBJECT(d) From Department d</ejb-ql>
</query>
<query>
  <query-method>
    <method-name>findByName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql>SELECT OBJECT(d) FROM Department d WHERE d.name = ?1</ejb-ql>
</query>
```

Рис. 2. Пример EJB-QL-запроса, извлекающего информацию о структурных подразделениях предприятия

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:DATA xmlns:ns=http://amd47.med.yale.edo/TrialDB_AdHocQuery
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://amd47.med.yale.edo/TrialDB_AdHocQuery
Schema.xsd">
  <ns:TIME_CRITERIA>
    <ns:TIME_ROW>
      <ns:TIME_SERIAL_NO>123</ns:TIME_SERIAL_NO>
      <ns:CRITERION_1>11</ns:CRITERION_1>
      <ns:QUALIFIER_1>Start</ns:QUALIFIER_1>
      <ns:TIMEOP>Equals</ns:TIMEOP>
      <ns:CRITERION_2>21</ns:CRITERION_2>
      <ns:QUALIFIER_2>Duration</ns:QUALIFIER_2>
      <ns:TIME_RELOP>within</ns:TIME_RELOP>
      <ns:TIME_VALUE>10.5</ns:TIME_VALUE>
      <ns:TIME_UNITS>mm</ns:TIME_UNITS>
    </ns:TIME_ROW>
  </ns:TIME_CRITERIA>
</ns:DATA>
```

Рис. 3. – Запрос, извлекающий информацию из приложения TrialDB

ИС), в которой функционирует система, поэтому в ней присутствуют соответствующие теги. Язык не ортогонален по отношению к предметной области и не может быть использован в качестве основного в информационных системах другого профиля. Кроме того, представленная спецификация языка позволяет создавать логические выражения, соединённые только логической операцией «И», что явно недостаточно для крупных программных продуктов.

К работам рассматриваемой категории можно отнести спецификацию языка CAML (Collaborative Application Markup Language), описанного в работах [25-26] и предназначенного для организации единого электронного документооборота в масштабах предприятия. На Рис. 4 представлен запрос на описываемом языке.

Для извлечения данных используется тег <Query>. Вложенный узел <Where> позволяет создавать сложные выражения с помощью комбинации операций сравнения и логических операций. В каждой операции сравнения используется тег <FieldRef> для указания ссылки на поле и узел <Value> для представления сравниваемого значения. Отметим, что в <Value> можно подставить либо константу, либо имя поля. При этом сложное арифметическое выражение придётся указать в виде текстовой строки. Более верно было бы сделать это с помощью организации вложенных тегов и тем самым предоставить возможность описания произвольных выражений. С помощью тега <OrderBy> указывается порядок сортировки результирующего набора данных посредством перечисления полей в узлах <FieldRef>. Для

```
<?xml version="1.0" encoding="UTF-8"?>
<Query xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="CAMLSchema.xsd">
  <Where>
    <And>
      <Gt>
        <FieldRef Name="FieldA"/>
        <Value Type="integer">123</Value>
      </Gt>
      <Eg>
        <FieldRef Name="FieldB"/>
        <Value Type="integer">1</Value>
      </Eg>
    </And>
  </Where>
  <OrderBy>
    <FieldRef Name="FieldA"/>
  </OrderBy>
  <GroupBy>
    <FieldRef Name="FieldA"/>
  </GroupBy>
</Query>
```

Рис. 4. Запрос, описанный с помощью языка CAML

```
<?xml version="1.0" encoding="UTF-8"?>
<KDD_QUERY name="generic_tree" par_list="perc,source,dest">
  <TREE_MINER xml_dest="#dest#" target_attribute="class_name">
    <PP_SAMPLING xml_dest="sampling.xml">
      <TABLE LOADER xml source="#source#"/>
      <ALGORITHM algorithm name="simple sampling">
        <PARAM name="percentage" value="#perc#"/>
        <PARAM name="with_replacement" value="false"/>
      </ALGORITHM>
    </PP_SAMPLING>
    <ALGORITHM algorithm_name="YADT">
      <PARAM name="num_instances_for_leaf" value="3"/>
    </ALGORITHM>
  </TREE MINER>
</KDD_QUERY>
```

Рис. 5. Пример запроса на языке KDDML

группировки данных используется узел <GroupBy>. Представление других синтаксических конструкций языка запросов (например, конструкций, позволяющих организовать естественное соединение данных) в CAML не предусмотрено, что связано со спецификой применения данного языка.

Рассмотрим ещё один предметно-ориентированный язык разметки. Язык KDDML (KDD Markup Language) используется в качестве промежуточного языка для извлечения информации в системах выявления (обнаружения) знаний (knowledge discovery) [27-28]. Пример запроса представлен на Рис. 5.

Как видно из Рис. 5, KDDML предназначен для определения полного жизненного цикла анализируемых данных и описывает процедуру загрузки информации в хранилище, «чистку данных» и применяемый алгоритм, используемый для извлечения информации о скрытых закономерностях, присутствующих в данных. Поэтому этот язык может быть использован только в системах принятия решений.

Анализ представленных работ позволяет сделать вывод, что ни один проблемно-ориентированный язык запросов не может быть использован в качестве унифицированного языка запросов данных.

1.3. Способы представления отдельных синтаксических конструкций языка запросов в виде XML

Работы этой категории наиболее близко соответствуют задачам, решаемым в данной статье, т.к. предлагают унифицированный подход к представлению отдельных синтаксических конструкций ЯЗ. Такая задача решалась в проекте XSQL [24], который предлагал разработку XML-языка, предназначенного для извлечения данных из реляционного источника.

Пример запроса представлен на Рис. 6.

Корневым элементом является тег <xsql>, содержащий один или несколько запросов, описанных в теге <query>. Прежде всего нас интересует набор тегов, используемых для извлечения данных, поэтому рассмотрим дочерние узлы тега <select>. В элементе <table_column> указывается имя поля (<column>) и имя таблицы, в котором оно присутствует (<table>). В XSQL отсутствует простой способ указания выражений, выступающих в качестве извлекаемых столбцов и указания псевдонима полю. Для указания условия фильтрации данных использован тег <conditional>, содержащий строковое выражение предиката. При этом невозможно создание условного выражения с помощью описания иерархии тегов, т.е. используется SQL-подобный подход.

Для создания сложных запросов и запросов, содержащих параметры, использован тег <statement> (Рис. 6, справа), в котором записана строка SQL-запроса. Такой подход к описанию запросов связан с рядом недостатков, которые рассмотрены ранее при обсуждении инструментов ОРО (разд. 1.1.).

К сожалению, в настоящий момент проект XSQL больше не развивается, и сейчас доступно лишь dtd-описание некоторых конструкций, которых явно недостаточно для описания полноценных запросов.

Упрощенная реализация отдельных элементов языка XML-запросов описана в работах [15, 16], где рассмотрено выражение, состоящее из критериев выборки данных, т.е. логических критериев, эквивалентных по функционалу

```
<?xml version="1.0"?>
<!DOCTYPE xsql SYSTEM
"xsql.dtd">
<xsql>
  <query>
    <select>
      <table_column>
        <ta-
ble>people</table>
        <col-
umn>name</column>
      </table column>
      <table>people</table>
      <conditional>
        name = 'Smith'
      </conditional>
    </select>
  </query>
</xsql>
```

```
<?xml version="1.0"?>
<!DOCTYPE xsql PUBLIC "xsql.dtd">
<xsql>
  <query>
    <statement>
      select name from people
      where pid = <parameter name="pid">Personal
ID</parameter>
    </statement>
  </query>
  <query>
    <statement>
      select id, name, title from people
      where name = <parameter name="name">Last
Name</parameter>
       and title like <parameter name="title">Job Ti-
tle</parameter>
    </statement>
    <param_values>
      <value parameter="name">Smith</value>
      <value parameter="title">%President%</value>
    </param_values>
  </query>
</xsql>
```

Рис. 6. Запросы, описанные с помощью языка XSQL: извлечение информации о лицах по фамилии Smith (слева) и сложный запрос с параметрами (справа)

выражению, записываемому в конструкции where языка SQL. Там же предложен алгоритм оптимизации выражений и рассмотрены принципы построения синтаксического дерева и дальнейшей его обработки на СУБД. Пример реализованных XML-элементов представлен на Рис. 7.

Основной идеей является размещение каждого элемента, представляющего критерий фильтрации данных (например, a=5) внутри элемента, определяющего ту логическую операцию, с помощью которой он объединяется с другими критериями.

На Рис. 7 (слева) видно, что в представленном языке рассмотрены только конструкции, позволяющие организовать логические выражения. При этом было сделано допущение, что логические выражения не содержат булевых операций («И», «ИЛИ»), т.е. листовые узлы синтаксического дерева являются терминаль-

```
<?xml version="1.0" encoding="UTF-8"?>
<Criterias>
  <Or>
    <Or>
      <And>
        <Criteria Name="a" Operator="equal">
          <Item Value="5"/>
        </Criteria>
        <Criteria Name="b" Operator="equal">
          <Item Value="4"/>
        </Criteria>
      </And>
      <Criteria Name="c" Operator="notequal">
        <Item Value="17"/>
      </Criteria>
    </Or>
    <And>
      <Criteria Name="d" Operator="like">
        <Item Value="abc%"/>
      </Criteria>
      <Criteria Name="e" Operator="in">
        <Item Value="1"/>
        <Item Value="4"/>
        <Item Value="6"/>
      </Criteria>
    </And>
  </Or>
</Criterias>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Criterias>
  <Or>
    <And>
      <Criteria Name="a" Operator="equal">
        <Item Value="5"/>
      </Criteria>
      <Criteria Name="b" Operator="equal">
        <Item Value="4"/>
      </Criteria>
    </And>
    <Criteria Name="c" Operator="notequal">
      <Item Value="17"/>
    </Criteria>
    <And>
      <Criteria Name="d" Operator="like">
        <Item Value="abc%"/>
      </Criteria>
      <Criteria Name="e" Operator="in">
        <Item Value="1"/>
        <Item Value="4"/>
        <Item Value="6"/>
      </Criteria>
    </And>
  </Or>
</Criterias>
```

Рис. 7. Представление логического выражения (a=5 and b=4) or c<>17 or (d like 'abc%' and e in (1,4,6)) с помощью XML: исходное представление (слева); оптимизированное представление (справа)

```
<?xml version="1.0" encoding="UTF-8"?>
<SIML>
  <Var id="i">
    <Root>
      <server>localhost</server>
    </Root>
  </Var>
  <SQLML input="i" output="v">
    <param>
      <connection>example</connection>
      <title>Product</title>
      <sql>
           SELECT ProductID,ProductName,QuantityPerUnit,UnitPrice,SKU
             FROM Products
      </sql>
      <filters>
        <query>
          <field>ProductType</field>
          <value>toys</value>
          <label>y</label>
        </query>
      </filters>
    </param>
  </SQLML>
  <Return value="v" valuetype="variable" />
</SIML>
```

Рис. 8. Запрос, описанный с помощью языка SIML: извлечении информации о товарах, для которых в поле ProductType указано значение toys

ными символами. В некоторых случаях это неверно и нарушается при формировании сложных предикатов, у которых в операциях сравнения в качестве операндов используются не константные значения, а арифметические выражения.

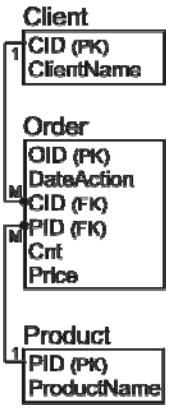
Ещё один подход, разработанный для представления условий фильтрации данных языка SQL (для представления конструкций директивы where), описан в работе [29], посвящённой языку SIML (Software Integration Markup Language). Пример запроса, извлекающего информацию о товарах, представлен на Рис. 8.

В работе [29] описано применение данного языка только для описания логических выражений с операций сравнения на равенство. Построение предикатов с другими операциями и принципы описания сложных логических выражений (с несколькими операциями) в этой работе не рассматривались. Идея применения двух противоположных подходов к описанию извлекаемых полей (в виде строки SQL-запроса) и условий фильтрации данных (в виде набора вложенных тегов) имеет серьёзный недостаток, который проявляется в невозможности написания запроса выборки данных из нескольких таблиц и выполнении join-операций. Наличие этих недостатков обусловлено тем, что SIML разрабатывался как язык интеграции данных, сохранённых в различных источниках, а не как ЯЗ.

Наиболее полной работой (и соответствующей реализацией), посвящённой принципам представления синтаксических конструкция языка SQL, можно назвать проект ZsqlML (Zenark's XML for SQL) [30]. Данный язык позволяет описать различные SQL-конструкции с помощью предопределённых тегов, но в рамках данной статьи мы рассмотрим только элемент <select>, позволяющий извлекать данные и представленный на Рис. 9.

В примере на Рис. 9 показано, как извлечь информацию о клиентах, наименовании купленных продуктов, а также количестве товара и итоговой сумме сделки. Анализируются только заказы, сделанные до 01.03.2009 и только те продукты, которые были куплены более чем на 1000 рублей.

Несмотря на ряд имеющихся достоинств (реализуемых с помощью специализированных тегов), языку ZsqlML (как и всем остальным языкам, рассмотренным ранее в этом разделе и предназначенным для представления SQL-конструкций) присущ очень большой недостаток, связанный с тем, что в языке SQL отсутствуют многие объектные расширения, часто используемые при разработке объектно-ориентированных приложений. В частности,




```
select c.ClientName, p.ProductName
       ,count(o.Cnt) as Count
       ,sum(o.Price*o.Cnt) as Summa
  from Client c
  inner join Order o on o.CID=c.CID
  inner join Product p on
p.PID=o.PID
where o.DateAction<'01.03.2009'
group       by       c.ClientName,
p.ProductName
  having sum(o.Price*o.Cnt)>1000
order       by       c.ClientName,
p.ProductName
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE zsql:select SYSTEM "zsql.dtd">
<zsql:select>
  <zsql:field column="ClientName"  table="c"/>
  <zsql:field column="ProductName" table="p"/>
  <zsql:field alias="Count">
    <zsql:value>count(o.Cnt)</zsql:value>
  </zsql:field>
  <zsql:field alias="Summa">
    <zsql:value>sum(o.Price*o.Cnt)</zsql:value>
  </zsql:field>
  <zsql:source table="Client" alias="c"/>
  <zsql:join type="inner">
    <zsql:source table="Order" alias="o"/>
    <zsql:eq>
      <zsql:field column="CID" table="o"/>
      <zsql:field column="CID" table="c"/>
    </zsql:eq>
  </zsql:join>
  <zsql:join type="inner">
    <zsql:source table="Product" alias="p"/>
    <zsql:eq>
      <zsql:field column="PID" table="p"/>
      <zsql:field column="PID" table="o"/>
    </zsql:eq>
  </zsql:join>
  <zsql:where>
    <zsql:lt>
      <zsql:field column="DateAction" table="o"/>
      <zsql:value>01.03.2009</zsql:value>
    </zsql:lt>
  </zsql:where>
  <zsql:group-by>
    <zsql:field column="ClientName"  table="c"/>
    <zsql:field column="ProductName" table="p"/>
    <zsql:having>
      <zsql:gt>
        <zsql:value>Sum(o.Price*o.Cnt)</zsql:value>
        <zsql:value>1000</zsql:value>
      </zsql:gt>
    </zsql:having>
  </zsql:group-by>
  <zsql:order-by>
    <zsql:field column="ClientName"  table="c"/>
    <zsql:field column="ProductName" table="p"/>
  </zsql:order-by>
</zsql:select>
```

Рис. 9. Тестирование функциональных возможностей языка ZsqlML: логическая модель БД

отсутствует возможность описания путевых выражений, являющихся ключевой концепцией OQL и позволяющих извлекать данные из агрегированных классов.

Подведя итог анализу работ всех трёх выделенных категорий, можно утверждать, что каждому из рассмотренных подходов (и инструментов) присущ ряд недостатков, которые не позволяют успешно применять существующий язык при разработке крупной информационной системы в объектно-ориентированной парадигме. Проектированию и реализации языка запросов, свободного от многих перечисленных недостатков, посвящены последующие разделы данной статьи.

2. Реализация языка запросов XOQL

Современные корпоративные информационные системы (КИС) разрабатываются с применением объектно-ориентированной парадигмы (ООП). При этом основные принципы ООП используются как при реализации клиентской части приложения, так и при реализации серверной части (на сервере СУБД).

При разработке объектной модели программного приложения, позволяющего сохранить бизнес-сущности в долговременной памяти, чаще всего используются следующие подходы:

1. Использование в качестве хранилища информации объектно-ориентированной СУБД.

Стандартом, определяющим функциональные возможности данных СУБД, является спецификация ODMG 3.0 [18-19].

2. Реализация бизнес-логики приложения в среде объектно-реляционной СУБД. Основным стандартом, регламентирующим объектные расширения, присутствующие в реляционной СУБД, является стандарт SQL:2003 [10].

3. Организация объектной системы в среде реляционной СУБД (объектно-реляционное отображение, ОРО, object-relational mapping, ORM) [13, 17].

Каждый инструмент, реализующий один из перечисленных подходов, предоставляет разработчику определённый язык запросов, основную часть которого составляют синтаксические конструкции, позволяющие извлекать экземпляры классов и значения атрибутов.

Наиболее проработанным, с нашей точки зрения, является объектный язык запросов (OQL, Object Query Language), предложенный консорциумом ODMG. Последней спецификацией, детально описывающей синтаксические конструкции языка, выступает документ «The Object Data Standard:ODMG 3.0» [18]. При этом каждая фирма-производитель объектно-ориентированной СУБД (ООСУБД) поддерживает некоторое подмножество (собственный диалект) языка OQL, что снижает переносимость разработанного приложения с одной СУБД на другую [19].

Отметим, что производители инструментов объектно-реляционного отображения также предоставляют возможность выборки экземпляров классов с помощью ЯЗ, похожего по своей структуре на OQL. В языке SQL:2003 также присутствует набор объектных расширений, эквивалентных возможностям языка OQL.

Исходя из всего вышесказанного, при разработке синтаксиса языка запросов, представляемого в виде XML, и определении критериев оптимальности будем придерживаться в данной статье структуры языка OQL.

2.1. Критерии оптимальности для разрабатываемой реализации языка запросов

Основным достоинством языка XML является наличие большого количества взаимосвязанных технологий, позволяющих задавать и контролировать структуру приложения (языка), созданного на его основе, что частично снимает необходимость проведения синтаксического анализа, который в данном случае выполняется парсером [3]. В современных объектно-ориентированных языках программирования (ООЯП) присутствует библиотека классов, представляющих собой парсер языка XML [3-6]. Кроме того, последние версии популярных СУБД, такие как Oracle, MS SQL Server, DB2, поддерживают встроенный тип данных XML, который в настоящее время включен в стандарт SQL 2003 [7-10].

Для задания допустимой структуры ЯЗ и последующего выполнения синтаксического анализа могут использоваться различные технологии, например, создание DTD-описания или объявление XSD-(XDR-)схем. Если этого недостаточно, то для выполнения синтаксического (и семантического) анализа можно использовать запросы на языке XQuery, позволяющем возвращать отдельные элементы XML-документа [11].

Классы, имеющиеся в ООЯП, создают отдельные XML-узлы и атрибуты с помощью вызова предопределенных методов. Это частично избавляет от второго недостатка – необходимости реализации объектной модели ЯЗ.

Для формирования семантической структуры языка XOQL выделим критерии оптимальности, которым должна соответствовать полученная реализация ($КО_{ЯЗ}$):

1. Независимость от предметной области, в которой функционирует ИС. Это позволит унифицировать ЯЗ и применить его практически в любой информационной системе. Поэтому необходимо выделить в структуре языка запросов общие XML-узлы с такими названиями, как <Select>, <From>, <Where> и т.п. Из названия видно, что выделенные теги не относятся к какой-либо предметной области, но позволяют описать различные элементы ЯЗ.

2. Чёткая структура запроса, организованная с помощью набора вложенных XML-узлов. Предполагает активное использование вложенности XML-узлов друг в друга для наиболее точного отображения иерархии подчинённости элементов ЯЗ. Например, элемент <Where>

должен располагаться внутри элемента <Select>, так как он указывает условие фильтрации данных, а Select определяет тип запроса (выборка данных). Проверка правильности вложения XML-узлов может быть выполнена на этапе синтаксического анализа запроса с помощью технологий, описанных ранее.

3. Возможность расширения синтаксиса языка запросов за счёт введения новых конструкций, представляемых узлами и атрибутами. Так как сама структура запроса не зависит от источника данных ($КО_{ЯЗ}4$), то добавление новых элементов (и атрибутов) никак не повлияет на приложения, использующие старые версии синтаксиса ЯЗ, т.е. расширение будет происходить в соответствии с принципами обратной совместимости. Так как запрос выполняется на источнике данных (например, на РСУБД), необходимо разработать процедуры трансформации XML-запроса в ЯЗ конкретного источника. При добавлении нового элемента, представляющего определённую синтаксическую конструкцию языка запросов целевой СУБД, потребуется корректировка процедуры трансформации.

4. Наличие основных синтаксических конструкций, присутствующих в объектном языке запросов OQL. Одними из ключевых особенностей объектного языка запросов являются: путевые выражения, позволяющие описать сложные отношения между классами и возможность выборки в качестве элементов проекции не только отдельных атомарных атрибутов класса, а целых объектов. Реализация этого (и некоторого нерассмотренного в данной статье) функционала предоставит разработчику ИС широкие возможности по выборке, обновлению и удалению данных, аналогичные возможностям современных ЯЗ, реализованных во многих популярных ООСУБД, инструментах ОРО, технологиях построения объектных и распределённых приложений.

5. Независимость от источника данных (ИД), модели данных (МД) и архитектуры приложения. Несмотря на то, что разрабатываемый язык запросов должен поддерживать синтаксические конструкции, присущие объектному языку запросов (OQL), один и тот же запрос может быть использован для манипулирования (выборки, обновления, удаления) информацией, физически хранимой в различных источниках данных (ООСУБД, инструмент ОРО, ОРСУБД и т.п.). Для каждого источника разрабатываются процедуры трансформации в диалект поддерживаемого ЯЗ. Учитывая масштабы современных информационных систем, и тот факт, что клиентские приложения могут одновременно работать с несколькими СУБД, реализация описанного критерия позволит сократить затраты на разработку приложения, функционирующего в гетерогенной среде. При этом синтаксис языка не привязан к реализованной архитектуре приложения и трансформация в ЯЗ, поддерживаемый ИД, может выполняться на любом звене (слое) программного приложения.

2.2. Синтаксические конструкции языка XOQL

С целью определения допустимой семантики синтаксических конструкций разработанного объектного ЯЗ описана XML-схема, графическое представление которой изображено на Рис. 10.

Корневым элементом любого XOQL-запроса является тег <XOQL>, в котором имеется атрибут version, предназначенный для описания версии языка запросов. Значение данного атрибута анализируется программой, выполняющей трансформацию XOQL-запроса в ЯЗ, поддерживаемый целевой СУБД. Таким способом организована возможность поддержки обратной совместимости, которая позволяет приложению работать с различными версиями языка. Дочерний элемент <Select> указывает на то, что запрос извлекает информацию из хранилища. Т.к. другие типы запросов (вставка новых данных, удаление и модификация существующей информации) в данной статье не рассматриваются, поэтому соответствующие синтаксические конструкции (представленные XML-тегами) отсутствуют на Рис. 10.

Внутри элемента <Select> может присутствовать необязательный тег <SelectOptions>, предназначенный для указания различных опций, влияющих на выборку данных. Например, с помощью вложенного узла <TopRowCount> можно ограничить результирующий набор первыми N записями, где N – положительное целое число.

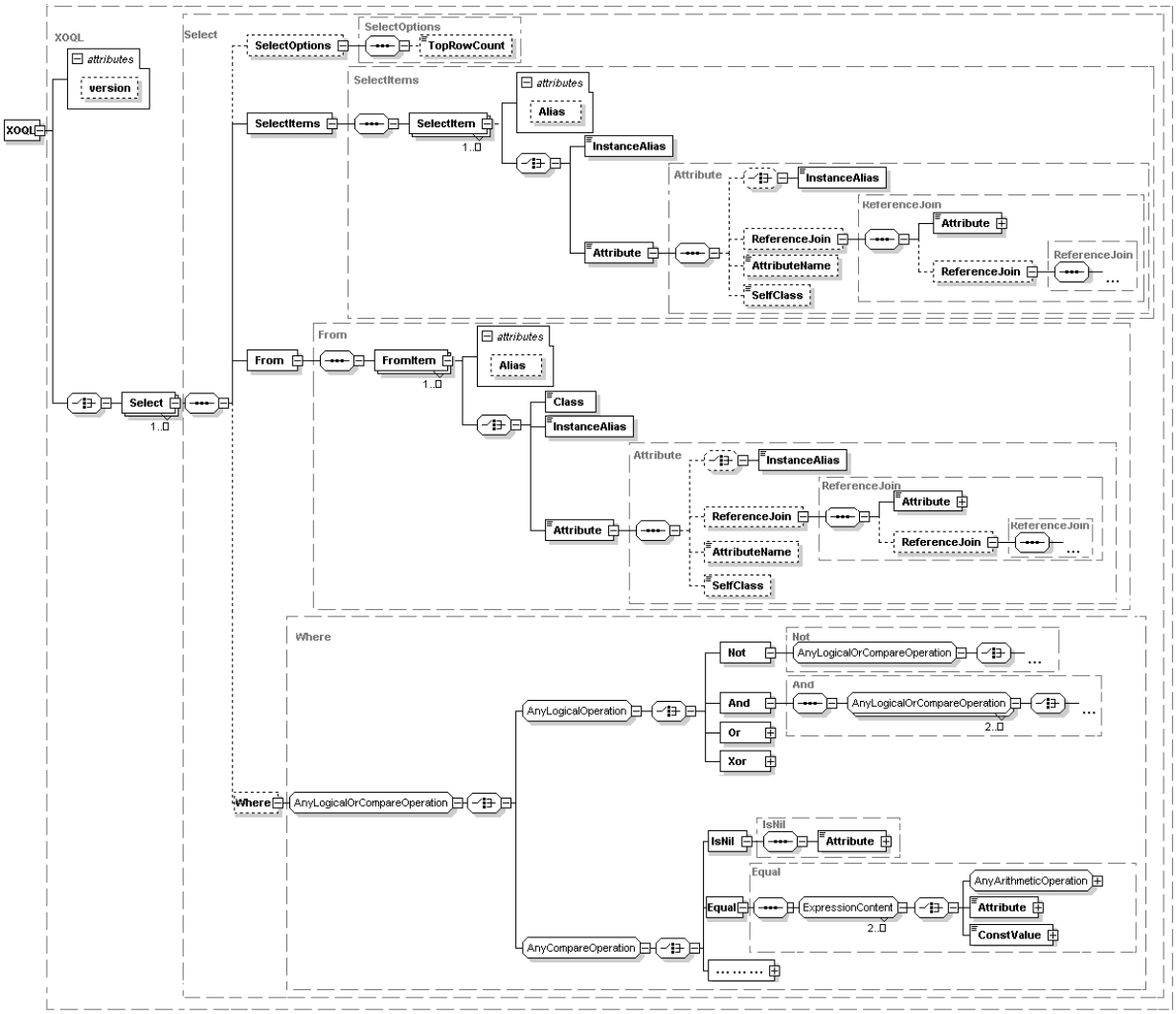


Рис. 10. XML-схема, описывающая основные синтаксические конструкции языка XOQL

Состав и структура опций зависят от языка запросов, поддерживаемого целевой СУБД, используемой в качестве хранилища информации для разрабатываемого приложения.

Для описания элементов проекции возвращаемого набора данных используется тег <SelectItems>. Каждый элемент определяется в XML-узле <SelectItem> и может содержать псевдоним (указанный в качестве значения атрибута Alias), под которым он будет возвращён в клиентское приложение, т.е. применение атрибута Alias эквивалентно использованию директивы as при выборке данных (с помощью оператора select) в языке SQL.

Ключевой особенностью объектного ЯЗ OQL является возможность указать в списке выборки не только атрибуты атомарного литерного типа, но и атрибуты, тип которых представлен классом. Эта возможность (отсутствующая в SQL, который предполагает извлечение только атомарных значений в директиве select) упрощает процесс извлечения требуемых данных. Для реализации данного функционала используется точечная нотация, применяемая для представления путевых выражений.

Для извлечения объектов (экземпляров класса) в качестве элементов проекции используется тег <InstanceAlias> (вложенный в <SelectItem>), содержащий псевдоним коллекции, объявленной с помощью узла <FromItem>.

При выборке значения атрибута используется тег <Attribute>. Если атрибут представлен

атомарным литеральным типом данных (строка, целое число, число с фиксированной точкой и т.п.) и однозначно определяется его принадлежность какому-либо типу элемента коллекции, то достаточно указать лишь его имя (в качестве значения рассматриваемого XML-элемента). Если по имени атрибута невозможно однозначно определить его принадлежность типу элемента коллекции, что происходит при наличии нескольких атрибутов с одинаковыми именами в одной области видимости, то необходимо с помощью тега <InstanceAlias> указать псевдоним требуемой коллекции. Имя атрибута класса в этом случае задаётся в качестве значения XML-элемента <AttributeName>.

Для извлечения значения атрибута агрегированного класса необходимо описать путевое выражение, которое организуется с помощью чередования вложенных друг в друга тегов <Attribute> и <ReferenceJoin>.

Если извлекаемое значение необходимо привести к какому-либо конкретному классу (при извлечении экземпляров производного класса через ссылку, типом которой является базовый класс), то его имя указывается в качестве значения узла <SelfClass>.

Тег <From> позволяет указать различные коллекции (с помощью вложенных узлов <FromItem>), декартовое произведение которых является источником выборки данных. Коллекцией могут выступать как экстент, содержащий объекты определённого класса, так и класс, указанный с помощью путевого выражения, начиная от ранее объявленной коллекции. В первом случае используется тег <Class>, содержащий имя класса, объекты которого необходимо извлечь, а во втором - либо XML-узел <InstanceAlias>, содержащий имя псевдонима, заданное ранее для конкретной коллекции, либо элемент <Attribute>, извлекающий атрибут, типом которого является класс. Атрибут Alias используется для задания псевдонима коллекции с целью последующей ссылки на неё с помощью указания значения в теге <InstanceAlias>.

Для описания предиката, налагающего условия фильтрации на извлекаемые данные, используется тег <Where>. Из Рис.10 видно, что внутри данного тега указываются как логические операции (and, or, not, xor), так и операции сравнения (>, =, < и т.п.). При разработке программного приложения на объектно-ориентированном языке программирования и решении задачи описания схожих объектов (в нашем случае операций) чаще всего применяется наследование классов. Подобная задача может быть решена путем использования полиморфизма, реализуемого в XML-схеме атрибутом base тега extension (дочерний тег узла <complexContent>), который содержит имя XML-типа (описанного с помощью тега <complexType>), выступающего родительским по отношению к объявляемому [31]. Несмотря на кажущееся сходство, наследование XML-типов и наследование классов имеют ключевое различие. При наследовании тегов в XML-документе (т.е. в XOQL-запросе) указывается имя тега базового класса, а в атрибуте xsi:type указывается имя унаследованного XML-типа.

С позиций нашей задачи для построения предиката в элементе <Where> при наследовании XML-узлов необходимо разместить несколько одинаковых XML-узлов, вложенных друг друга, представляющих каждую логическую операцию и операцию сравнения. Такой подход затрудняет написание логических выражений и снижает их наглядность из-за необходимости поддержки множества различных операций, поэтому не может быть использован в нашей задаче.

В качестве альтернативного решения выступает объявление группы AnyLogicalOrCompare Operation (с помощью стандартного тега XML-схемы <group>), в которой с помощью элемента <choice> указаны ссылки на группу AnyLogicalOperation, представляющую множество всех логических операций, и на группу AnyCompareOperation, описывающую множество всех операций сравнения. При этом для всех логических операций был выделен базовый тип LogicalOperation (не показан на Рис. 10), от которого унаследованы два подтипа: UnaryLogicalOperation – объявлен в качестве родительского для всех тегов, представляющих унарные логические операции и позволяет описывать в качестве значения теги, соответствующие группе AnyLogicalOrCompareOperation; MultipleLogicalOperation – предназначен для

описания множественных логических операций и позволяет описывать в качестве значения два и более тега, соответствующих группе AnyLogicalOrCompareOperation. Многие логические операции, такие как or, and, xor, являются бинарными. При написании сложных запросов появляется необходимость соединить одной операцией более двух операндов. Чтобы избавиться от повторения одной и той же операции был описан XML-тип MultipleLogicalOperation, от которого унаследованы типы конкретных операций (теги <And>, <Or>, <Xor>). Т.е. бинарные операции в общем случае являются n-арными [15-16]. При этом значения операций вычисляются слева направо, что соответствует последовательному извлечению тегов с помощью таких программных интерфейсов (API), как DOM или SAX [3].

При разработке тегов, позволяющих описать операции сравнения, использован схожий подход: для унарных операций выделен тип UnaryCompareOperation, а для бинарных – тип MultipleCompareOperation. Оба типа унаследованы от базового CompareOperation. При этом бинарные операции так же как и логические, являются n-арными. Единственной унарной операцией, объявленной в схеме, является операция IsNull, проверяющая переданный атрибут (типом которого выступает класс) на наличие пустого значения. Из Рис. 10 видно, что для остальных операций сравнения в качестве дочерних может использоваться набор тегов, соответствующих группе ExpressionContent, представляющей последовательность из арифметических операций (группа AnyArithmeticOperation), значений атрибутов (тег <Attribute>) и константных значений, записанных в теге <ConstValue>. Так как значением последнего XML-узла является строка, то с помощью атрибута Type можно указать необходимый тип данных результирующего значения (представленного строкой), соответствующий встроенному, или пользовательскому типу данных, присутствующему в языке запросов целевой СУБД, в который будет трансформироваться XOQL-запрос.

В любой операции сравнения может быть использован тег <Attribute>, что предполагает потенциальную возможность описания путевого выражения. Этот подход демонстрирует унифицированность реализованного подхода представления точечной нотации.

Из описания способов организации выражений видно, что используется префиксная форма записи, которая подразумевает запись сначала операции, а затем операндов (слева на право). Учитывая возможность организации иерархии из вложенных друг в друга XML-узлов, данная система записи выражений позволяет указать операнды (и другие операции), над которыми выполняется указанная операция, в качестве дочерних тегов [15].

В языке MathML для решения сходных задач используется набор вложенных тегов <apply>, в каждом из которых в качестве дочерних узлов указывается и/или другой узел <apply>, операция, операнды [21]. Т.е. операнды и выполняемая над ними операция имеют один уровень вложенности, что отличается от представленного подхода. Описанный в статье подход, по мнению автора, позволяет более наглядно (чем в MathML) представить структуру выражения благодаря вложенности тегов. Однако такой подход увеличивает время, требуемое на синтаксический анализ (и последующую интерпретацию) выражения, и усложняет разработку XML-схемы.

Из детального описания синтаксических конструкций видно, что структура разработанного ЯЗ полностью соответствует критериям оптимальности, выделенным в предыдущем разделе данной статьи. Так как при рассмотрении объявленных тегов не указана конкретная предметная область, в которой может найти своё применение данный язык, можно говорить об унифицированости предложенного подхода, что соответствует требованию $КО_{ЯЗ}1$. Организация синтаксических конструкций в виде набора вложенных друг в друга тегов позволяет более чётко структурировать XOQL-запрос, что соответствует требованию $КО_{ЯЗ}2$. Представленный ЯЗ соответствует $КО_{ЯЗ}3$, т.к. в нем описаны лишь основные конструкции, не привязанные к синтаксическим директивам языка запросов конкретной целевой СУБД. При этом дополнительные XML-узлы (и атрибуты) необходимо добавить при разработке процедуры трансформации XOQL-запросов в ЯЗ целевой СУБД. Соответствие $КО_{ЯЗ}4$ также не вызывает

сомнения, т.к. часть описанных тегов предназначены для представления синтаксических конструкций, присутствующих в объектном языке запросов. Например, для представления путевых выражений и при выборке атрибутов, типом которых является класс, использованы теги <Attribute>, <ReferenceJoin> и <InstanceAlias>.

Всё описанное выше позволяет утверждать, что XOQL соответствует всем выдвинутым критериям оптимальности языка запросов, рассмотренным в разд. 2.1.

2.3. Тестовые примеры XOQL-запросов

Рассмотрим несколько XOQL-запросов, наглядно демонстрирующих принципы описания объектных запросов с помощью XML. Предполагается, что данные извлекаются из тестовой объектной модели, разработанной в [20].

На Рис. 11 оба представленных запроса извлекают информацию обо всех служащих, имеющихся в БД (в элементе <FromItem> указан класс Employee). Результатом выполнения запроса является коллекция, элементы которой представляют собой проекцию, содержащую табельный номер служащего (EID), ФИО (Name), должность (Post) и информацию о руководителе (Chief). Особенность запроса состоит в том, что элементами проекции выступают как атрибуты атомарного литерального типа данных (EID, Name, Post), так и атрибуты, типом которых является класс предметной области (Chief). Реализовано это с помощью унифицированного подхода, требующего указания имени атрибута в теге <Attribute>.

В XOQL поддерживается информация о наследовании классов, поэтому запрос вернёт объекты не только класса Employee, но и объекты всех производных классов (например, Manager).

Рассмотрим более сложный пример (Рис. 12), в котором извлекается название, контактный телефон и адрес регистрации Московских компаний.

В операторе from (тег <FromItem>) указано, что выборка данных выполняется из двух коллекций (экстентов), содержащих экземпляры классов Company и Address соответственно. Экстентам присвоены псевдонимы («c» и «a» соответственно), для чего применён атрибут Alias. Затем это новое имя использовано в теге <InstanceAlias> при объявлении элементов проекции результирующей выборки данных (тег <Select>) и при ссылке на ранее объявленную коллекцию в XML-узле <FromItem>. На Рис. 12 справа продемонстрировано указание псевдонима CompanyName элементу проекции, под которым он будет присутствовать в результирующей выборке данных, что реализовано с помощью присвоения значения атрибуту Alias (тега <SelectItem>).

Предикат, налагающий ограничения на результирующую проекцию (выбираются только организации, зарегистрированные в Москве),

```
select EID, Name, Post, Chief
from Employee
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XOQL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XOQL.xsd">
  <Select>
    <SelectItems>
      <SelectItem>
        <Attribute>EID</Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>Name</Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>Post</Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>Chief</Attribute>
      </SelectItem>
    </SelectItems>
    <From>
      <FromItem>
        <Class>Employee</Class>
      </FromItem>
    </From>
  </Select>
</XOQL>
```

Рис. 11. Объектный запрос, извлекающий подробную информацию о служащих: слева на языке OQL; справа на XOQL

```
select c.Name as CompanyName,
c.Phone
      ,c.LegalAddress
from Company c, c.LegalAddress a
where a.City='Москва'
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XOQL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="XOQL.xsd">
  <Select>
    <SelectItems>
      <SelectItem Alias="CompanyName">
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>Name</AttributeName>
        </Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>Phone</AttributeName>
        </Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>LegalAddress</AttributeName>
        </Attribute>
      </SelectItem>
    </SelectItems>
    <From>
      <FromItem Alias="c">
        <Class>Company</Class>
      </FromItem>
      <FromItem Alias="a">
        <Attribute>
          <InstanceAlias>c</InstanceAlias>
          <AttributeName>LegalAddress</AttributeName>
        </Attribute>
      </FromItem>
    </From>
    <Where>
      <Equal>
        <Attribute>
          <InstanceAlias>a</InstanceAlias>
          <AttributeName>City</AttributeName>
        </Attribute>
        <ConstValue>Москва</ConstValue>
      </Equal>
    </Where>
  </Select>
</XOQL>
```

Рис. 12. Объектный запрос, извлекающий информацию о компаниях, зарегистрированных в Москве: слева на языке OQL; справа на XOQL

записан с помощью дескриптора <Where>, в котором указана комбинация различных логических, арифметических операций и операций сравнения. Для сравнения значения атрибута City класса CompanyAddress на равенство константе применён тег <Equal>, в котором имеются два вложенных дескриптора (<Attribute>, <ConstValue>), используемые для представления имени атрибута и константы «Москва» соответственно.

Продемонстрируем реализацию возможности представления путевых выражений в языке XOQL (Рис. 13).

В данном примере используются два путевых выражения: первое - для выборки значения атрибута City, а второе - в предикате для указания требуемого значения атрибуту Name. Так как в обоих случаях путевое выражение составлено для извлечения значения атрибута, использован тег <Attribute>. Для ссылки на ранее объявленную (в теге <FromItem>) коллекцию в элементе <InstanceAlias> указан её псевдоним. Затем с помощью рекурсивного вложения тегов <ReferenceJoin> и <Attribute> описаны промежуточные составляющие (от псевдонима коллекции до имени атрибута) путевого выражения.

Представленные примеры XOQL-запросов (Рис. 11- Рис.13) демонстрируют широкие возможности разработанного ЯЗ и способы применения синтаксических конструкций, представленных в виде тегов и атрибутов.

```
select e.Name, e.HomeAddresses.City
from Employee e
where e.Chief.Name ='Иванов И.И.'
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XOQL xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="XOQL.xsd">
  <Select>
    <SelectItems>
      <SelectItem>
        <Attribute>
          <InstanceAlias>e</InstanceAlias>
          <AttributeName>Name</AttributeName>
        </Attribute>
      </SelectItem>
      <SelectItem>
        <Attribute>
          <InstanceAlias>e</InstanceAlias>
          <ReferenceJoin>
            <Attribute>HomeAddresses</Attribute>
            <ReferenceJoin>
              <Attribute>City</Attribute>
            </ReferenceJoin>
          </ReferenceJoin>
        </Attribute>
      </SelectItem>
    </SelectItems>
    <From>
      <FromItem Alias="e">
        <Class>Employee</Class>
      </FromItem>
    </From>
    <Where>
      <Equal>
        <Attribute>
          <InstanceAlias>e</InstanceAlias>
          <ReferenceJoin>
            <Attribute>Chief</Attribute>
            <ReferenceJoin>
              <Attribute>Name</Attribute>
            </ReferenceJoin>
          </ReferenceJoin>
        </Attribute>
        <ConstValue>Иванов И.И.</ConstValue>
      </Equal>
    </Where>
  </Select>
</XOQL>
```

Рис. 13. Объектный запрос, извлекающий информацию об именах работников (и названиях городов проживания), подчиняющихся менеджеру Иванову И.И.: слева на языке OQL; справа на XOQL

Заключение

В данной статье предложен способ представления основных синтаксических конструкций объектного языка запросов в виде XML-документов. При этом, в отличие от аналогов, предложен подход, основанный на организации чёткой иерархии вложенности тегов, что позволяет применять ряд взаимосвязанных технологий для выполнения проверки синтаксиса и семантики запросов. Кроме того, представлен значительный набор тегов, позволяющий описать наиболее часто используемые синтаксические конструкции выборки данных, что выгодно отличает описанную реализацию от имеющихся проанализированных работ.

Как основное направление развития представленного исследования рассматривается задача расширения предложенного синтаксиса дополнительными конструкциями, используемые при выборке данных, применяемыми для группировки и сортировки данных. Особый интерес представляют способы организации и выполнения вложенных запросов, часто используемых в сложных приложениях, автоматизирующих различные прикладные предметные области. Кроме того, необходимы конструкции добавления новых данных, редактирования и удаления существующих. Необходимо также разработать и реализовать алгоритм трансформации запросов на разработанном языке в определённый диалект языка запросов (например, в SQL), поддерживаемый конкретной системой управления базами данных. Реализация подобного функционала сопряжена с

рядом сложностей, в частности, с возникновением проблемы оптимизации запросов, что приводит к необходимости введения ряда эквивалентных преобразований, используемых в процессе трансформации.

Немаловажен вопрос формального описания синтаксиса сформированного в данной статье языка запросов. Несмотря на трудоёмкость этой задачи, её решение позволит в последствии упростить процесс трансформации (за счёт использования алгебры (или исчисления) языка запросов, поддерживаемого источником данных) и облегчить процесс оптимизации запросов, т.к. появится возможность использовать уже накопленные знания и зарекомендовавшие себя алгоритмы.

Кроме того, представляет интерес вопрос разработки методов графического моделирования запросов и результатов выполнения с помощью графических нотаций, используемых при проектировании и моделировании объектных систем. Решение этой и всех обозначенных ранее задач должно быть построено на принципах расширения предложенного в данной работе синтаксиса языка запросов.

Литература

1. Грейв М. Проектирование баз данных на основе XML, Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 640 с.: ил. – Парал. тит. англ.
2. Альфред В. Ахо, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. — М.: Издательский дом "Вильяме", 2003. — 768 с. : ил. — Парал. тит. англ
3. Холзнер С. XML. Энциклопедия, 2-е изд. – СПб.: Питер, 2004. – 1101с.: ил.
4. Робинсон С. и др. C# для профессионалов, в 2-х т., Пер. с англ. – М.: Лори, 2003. – 1002 с. :ил. – Парал. тит. лист.
5. Перроун П., Венката С., Чаганти Р. Создание корпоративных систем на основе Java 2 Enterprise Edition. Руководство разработчика, Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 1184 с.: ил. – Парал. тит. англ.
6. Wood K. Delphi Developer’s Guide to XML, Wordware Publishing, 2001, 545p.
7. Чанг Б., Скардина М. и др., Oracle9i XML. Разработка приложений электронной коммерции с использованием технологии XML, М.: Лори, 2003. - 492., ил.
8. Klein S. Professional SQL Server 2005 XML, Wiley Publishing, 2006, 549p.
9. IBM Corp. Книга по DB2 и XML, http://www.ibm.com/developerworks/ru/library/db2xmlintro/contents.html?S_TACT=105AGX99&S_CMP=GR01GR01
10. Кузнецов С. Наиболее интересные новшества в стандарте SQL:2003, http://citforum.ru/database/sql/sql2003/
11. Katz H., Chamberlin D., Draper D. XQuery from the Experts: A Guide to the W3C XML Query Language, Addison Wesley, 2003, 512p.
12. Keith M., Schincariol M. Pro EJB 3. Java Persistence API, Apress, 2006, 480 p.
13. Nadkarni P. M., Brandt C. A., Morse R., Matthews K., Sun K., Deshpande A. M., Gadagkar R., Cohen D. B., Miller P. L. Temporal query of attribute-value patient data: utilizing the constraints of clinical studies // International Journal of Medical Informatics 70, 2003,p.59-77
14. Minter D., Linwood J. Pro Hibernate 3, Apress, 2005, 264p.
15. Олейник П.П. Технология представления логических выражений с помощью XML-документов // Информационные технологии. Теоретический и прикладной научно-технический журнал. 2008. № 2.
16. Олейник П.П. Унифицированный механизм представления логических выражений // Моделирование. Теория, методы и средства: Материалы VII Междунар. науч.-практ. конф., г.Новочеркасск, 6 апр. 2007 г.: В 3 ч. / Юж.-Рос. гос. техн. ун-т (НПИ). – Новочеркасск: ЮРГТУ, 2007. – Ч.3. С. 67-71.
17. Флауер М., Архитектура корпоративных программных приложений, Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 544 с.: ил. – Парал. тит. англ.
18. Cattell R.G., Barry D.K. The Object Data Standard:ODMG 3.0, Morgan Kaufmann Publishers, 2000, 288p.
19. Джордан Д. Обработка объектных баз данных на C++. Программирование по стандарту ODMG. : Пер. с англ.: Уч. пос. - М.:Издательский дом "Вильямс", 2001.-384с.:ил.-Парал. тит. англ.
20. Олейник П.П. Пример объектной модели для выполнения тестовых OQL-запросов // Теоретические и прикладные вопросы современных информационных технологий: Материалы IX Всероссийской научно-технической конференции: в 2 ч. - Улан-Удэ: Изд-во ВСГТУ, 2008. - Ч. II. С. 264-266.
21. Mathematical Markup Language (MathML) Version 2.0 (Second Edition), 4.4.3 Arithmetic, Algebra and Logic, http://www.w3.org/TR/MathML2/chapter4.html#id.4.4.3
22. Олейник П.П. Трансформация и обработка языка запросов, представленного в виде XML // Технологии Microsoft в теории и практике программирования: труды V-ой Всероссийской конференции студентов, аспирантов и молодых учёных. Южный регион, Таганрог, 13-14 марта 2008 г. - Таганрог: Из-во ТТИ ЮФУ, 2008. С. 63-66.
23. Гранд М. Шаблоны проектирования в Java, Пер. с англ. С. Беликовой. – М.: Новое знание, 2004.-559с.: ил.-Парал. тит. англ.
24. XSQL - Combining XML and SQL, http://xsql.sourceforge.net/manual.php
25. Collaborative Application Markup Language (CAML) Structure Specification, http://download.microsoft.com/download/8/5/8/858F2155-D48D-4C68-9205-29460FD7698F/%5BMS-WSSCAML%5D.PDF
26. Лондер О., Бликер Т., Ковентри П., Иделен Д. Службы Microsoft Windows SharePoint. Практ. Пособ. Серия

«Шаг за шагом»:/Пер. с англ. – М.:«СП ЭКОМ», 2005.-384с.: ил.
27. KDDML: a middleware language and system for knowledge discovery in databases, http://kdd.di.unipi.it/kddml/papers/kddml.pdf
28. KDDML Language: Reference Guide, http://kdd.di.unipi.it/kddml/downloads/documentazione/Specifiche/kddml_specification_2_0_16.pdf.
29. Yujie Xu, Minhua Shi, SQL Markup Language for Enterprise Integration, Proceedings of the 2004 IEEE International Conference on Services Computing (SCC'04)
30. ZsqlML (Zenark's XML for SQL), http://sourceforge.net/projects/zsqlml/
31. Russell Butek, Web services tip: Use polymorphism as an alternative to xsd:choice, http://www.ibm.com/_developerworks/webservices/library/ws-tip-xsdchoice.html.

Олейник Павел Петрович. Системный архитектор программного обеспечения Группы Компаний «Астон» (г. Ростов-на-Дону). Окончил Шахтинский институт (филиал) Южно-Российского государственного технического университета (Новочеркасского политехнического института) в 2004 году. Кандидат технических наук (с 2007г.). Автор 35 публикаций. Имеет 2 свидетельства об официальной регистрации программы для ЭВМ. E-mail: xsl@list.ru.