SP-RMU
версия **01–08**
от 31.III.2003

МГУ им М.В. Ломоносова

спецификация **Проскурня М.О.**

Модуль морфологии русского языка

Спецификация на модуль морфологии русского языка.
Проскурня Максим Олегович

Данная спецификация распространяется в составе комплекта поставки модуля морфологии RMU, который распространяется на следующих условиях:
Данное программное обеспечение поставляется без каких-либо гарантий. Пользователь соглашается использовать программу на свой страх и риск. Авторы не несут никакой ответственности за любое использование программы и любые возможные последствия. Любое использование программы в коммерческих целях недопустимо и запрещено.

Дополнительную информацию по проекту RMU, а также последние изменения можно получить по адресу http://axofiber.org.ru/rm/

Торговые марки
Microsoft Windows 95/98, Windows NT, Visual C++ являются зарегистрированными торговыми марками компании Microsoft Corp.
Unix является зарегистрированной торговой маркой компании AT&T Bell Labs.

Другие, упомянутые в этом документе, торговые марки являются зарегистрированными торговыми марками своих обладателей.

Благодарности
Джулиану Р. Сьюарду (Julian R. Seward) за библиотеку libbzip2.

СОДЕРЖАНИЕ

Предмет настоящей спецификации

Данная спецификация входит в комплект сопроводительной документацией к модулю морфологии RMUnit, разработанного на кафедре Алгоритмических Языков факультета ВМиК МГУ им. М.В. Ломоносова. Архитектура, принципы функционирования и структуры данных модуля описаны в данном документе.

Содержание документа ориентировано на специалистов в области компьютерной лингвистики, владеющих объектно–ориентированным программированием (в частности, программированием на языке «C++»). В первой части дано общее описание модуля, алгоритма работы, пользовательского интерфейса и методов интеграции. Вторая часть посвящена описанию структуры компонентов модуля, его классов, методов и полей данных. В «Приложении A» приводится описание структуры файлов данных модуля морфологии, «Приложение B» описывает настройку работы модуля с помощью конфигурационных файлов.

Соглашение о нотации

Не смотря на то, что описываемый программный инструмент (ПИ) может функционировать самостоятельно, его приоритетным предназначением является работа в комплексе программ лингвистической обработки текстов на естественном языке. В связи с этим в данном документе делается упор на модульность исполнения, поэтому описываемый ПИ обозначается как «Модуль морфологии русского языка», а его составляющие программные модули (во избежание смыслового пересечения) называются компонентами.

Для представления архитектуры модуля используются диаграммы на языке UML (диаграммы классов). Также в тексте приняты следующие соглашения по начертанию:

Начертание	Значение
моноширинный	фрагменты программного кода
без засечек	обозначение программно-ориентированных терминов и объектов

Кроме того, приняты следующие соглашения по сокращению и толкованию обозначений и наименований:

Обозначение	Толкование
модуль морфологии	модуль морфологии русского языка
сервер морфологии	модуль морфологии русского языка в виде самостоятельного приложения с поддержкой сетевых сервисов
морфологическая база	набор словарей модуля морфологии
словоформа	форма некоторого слова русского языка
компонент	программный модуль, составляющий модуль морфологии русского языка
описание	совокупное понятие объявления и определения различного рода программных объектов

объявление	описание класса (или интерфейса), объявление импортируемых объектов, декларирование профиля функции
определение	определение (или инициализация) объекта, реализация функции (или метода)
представление объекта	набор полей данных соответствующего класса
поведение объекта	набор методов соответствующего класса
метод	функция–член класса
процедура	функция, не возвращающая значение
→ стр. 10	сокращённая ссылка «смотри страницу 10»

В программном коде модуля для облегчения восприятия контекста приняты следующие правила именования программных объектов:

Способ именования	**Тип объекта**
CName	Описание класса реализации
IName	Интерфейсный класс для объектов класса CName
RName	Класс описания представления для класса CName
XClassName	Класс объектов, описывающих исключительные ситуации
EName	Описание констант или перечислимых типов
AParameterName	Артикль "A" перед именем параметра ParameterName для разрешения конфликта с локальным именем (только в случае необходимости) или с ключевым словом языка «C++»

Часть I. Описание модуля морфологии

Морфологический модуль представляет собой набор готовых решений для проведения морфологического анализа или синтеза слов русского языка. Анализ заключается в определении морфологических признаков исследуемых словоформ, а синтез — в построении словоформы, удовлетворяющей заданным признакам.

Глава 1. Общее описание

В данной главе даётся общее описание ПИ: описание алгоритма работы, варианты поставки, схема взаимодействия.

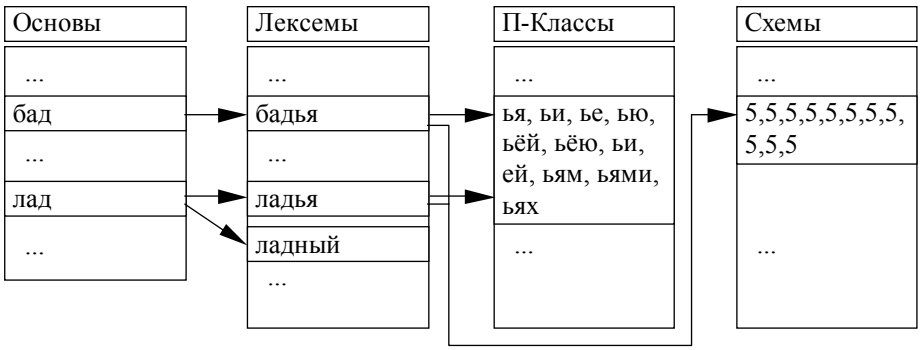
1.1. Описание алгоритма

Разработка алгоритма была основана на опыте предыдущих разработок [1], проводившихся на кафедре.

1.1.1. Основная идея

Главным фактором, влияющим на алгоритм анализа русских слов, является флективность русского языка, то есть значительная часть морфологических признаков определяется по окончанию словоформы. В качестве исходных морфологических данных был использован «Грамматический словарь» [2], морфологическая модель которого была переработана.

Разработанная модель состоит из 4 связанных словарей: словарь псевдооснов, лексем (слов), парадигматических классов (классы словоизменения), схем ударения. Схематически это можно представить следующим образом:



При анализе (синтезе) словоформы необходимо найти такую пару псевдооснова + флексия некоторого элемента П–класса, которые при конкатенации дают исходную словоформу. Если это удалось, то все необходимые морфологические признаки определяются по соответствующему элементу из словаря лексем и по номеру флексии в П–класса, а также по имени этого П–класса. Из связанного элемента схем ударений выбирается позиция ударной буквы, соответствующей номеру выбранной флексии.

При синтезе происходит дополнительная операция: генерация всех словоформ для заданной лексемы путём конкатенации соответствующей псевдоосновы со всеми флексиями выбранного П–класса.

Вследствие морфологической омонимии, вариативности флексий и позиций ударных букв алгоритм является недетерминированным, и в качестве ответа могут быть выданы

несколько правильных вариантов анализа (синтеза). Некоторые из вариантов могут быть отсеяны, если заранее известна дополнительная информация об ударных буквах.

1.1.2. Оценка эффективности

Сложность алгоритма анализа (синтеза) можно оценить требуемым для его проведения количеством сравнений строк.

$$N \le w \cdot (\log_2 S + f)$$

Где w — максимальная длина анализируемой словоформы, S — количество основ в отсортированном словаре основ, f — максимальное количество флексий в парадигматических классах. Для значений $w = 30$, $S = 70000$, $f = 60$ получаем, что $N \le O(3 \cdot 10^3)$.

Скорость работы модуля (в конфигурации Pentium III-700, 128 Мб ОЗУ) можно оценить следующими значениями:

анализ в одном процессе	—	3000 сл/с
анализ через сетевое соединение	—	500 сл/с

1.2. Варианты поставки

В базовом комплекте модуля морфологии поставляется исполняемый файл (для платформы Windows 9x/NT[1]), который представляет собой модуль морфологии с графическим пользовательским интерфейсом и поддержкой сетевых сервисов (сервер морфологии), и набор файлов морфологической базы данных: rm.stem.bin.data, rm.lexem.bin.data, rm.pclass.bin.data, rm.schema.bin.data.

Расширенный комплект включает в себя исполняемые файлы основного модуля и сетевого клиента, а также исходные тексты и сопроводительную документацию.

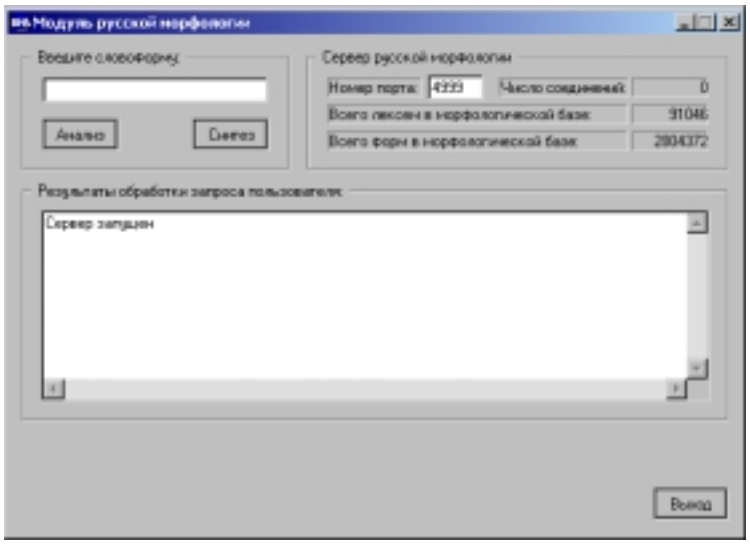
[1] Поставка модуля морфологии с графическим интерфейсом для платформы Unix пока находится в стадии рассмотрения.

Глава 2. Пользовательский интерфейс

В данной главе описывается пользовательский интерфейс самого модуля морфологии (который может выступать в качестве самостоятельного приложения или сервера), сетевого клиента[2] и html–интерфейса.

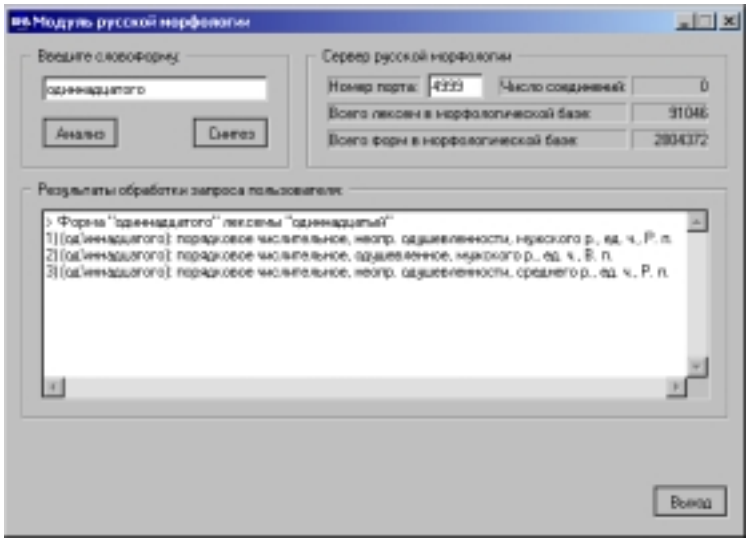
2.1. Главное окно

Главное окно модуля морфологии содержит поле ввода словоформы, кнопки «Анализ» и «Синтез», окно вывода результата обработки и различные индикаторы: номер сетевого порта, число сетевых соединений, количество лексем в базе и число порождаемых словоформ.



Номер сетевого порта можно изменить, введя новое значение в поле ввода «Номер порта». При этом надо учитывать, что все текущие сетевые соединения будут разорваны, и после смены номера сервер начнёт принимать запросы на соединения на новый указанный номер порта.

[2] Поставляется в расширенном комплекте.



Если результат обработки запроса пользователя не умещается в окне, используйте полосы прокрутки для перемещения по тексту вывода. К тому же результат можно выделить и скопировать в буфер обмена для последующей обработки в текстовом редакторе. Так в приведённом примере после копирования результата анализа в буфере обмена будет текст:

```
> Форма "одиннадцатого" лексемы "одиннадцатый"
1) (од'иннадцатого): порядковое числительное, неопр. одушевленности,
мужского р., ед. ч., Р. п.
2) (од'иннадцатого): порядковое числительное, одушевленное, мужского р., ед.
ч., В. п.
3) (од'иннадцатого): порядковое числительное, неопр. одушевленности,
среднего р., ед. ч., Р. п.
```

Если в результате анализа получилось несколько вариантов, относящихся к разным лексемам, то ответ разделяется строками вида:

```
> Форма "слофоворма" лексемы "лексема"
```

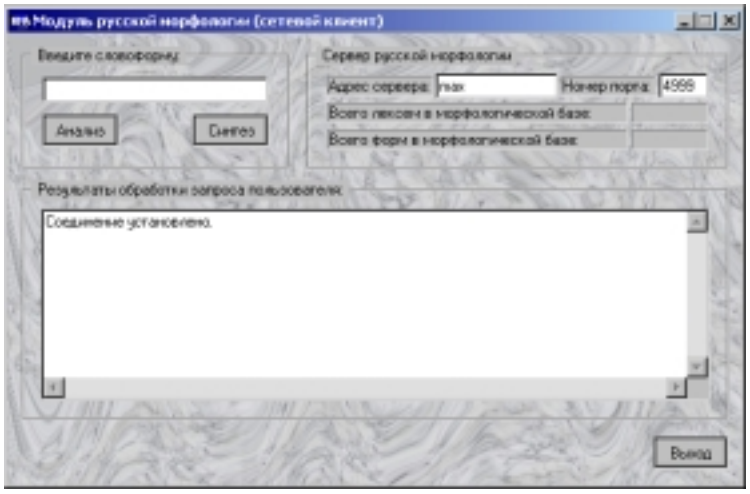
При синтезе разделительной строкой будет:

```
> Лексема "лексема"
```

Каждый вариант нумеруется и содержит повтор словоформы с указанием конкретной позиции ударной буквы с помощью апострофа «'».

2.2. Окно сетевого клиента

Пользовательский интерфейс сетевого клиента модуля морфологии по внешнему виду идентичен серверу морфологии, с той лишь разницей, что в нём необходимо указать адрес сетевого узла, на котором запущен сервер морфологии и его номер порта.

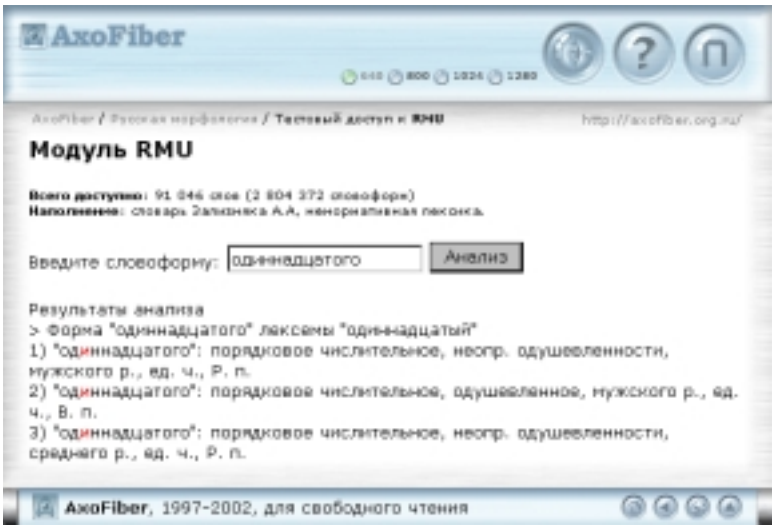


Результат обработки запроса пользователя может быть достаточно объёмным, поэтому если он не умещается в окне, используйте полосы прокрутки для перемещения по тексту вывода. К тому же результат можно выделить и скопировать в буфер обмена для последующей обработки в текстовом редакторе.

2.3. Пользовательский интерфейс на базе WWW-технологий

Для работы с сервером морфологии конечный пользователь не обязан иметь исполняемый файл клиентского модуля (→ 2.2, стр. 12) на своей машине. Пользователю достаточно иметь сетевой обозреватель (web–браузер), поддерживающий протокол HTTP 1.x и язык разметки HTML 2.0 (или выше).

Пример такого интерфейса представлен на странице http://axofiber.org.ru/rm/rmu.php:



Красным цветов выделены ударные буквы в каждом варианте.

Глава 3. Интеграция

В данной главе описываются особенности реализации модуля морфологии, интеграции, программный интерфейс и способ сетевого взаимодействия.

3.1. Особенности реализации

В качестве строкового типа выбран тип string из библиотеки STL.

Для операций с файлами используется класс CRMFileStream, который наследует интерфейс IRMStream и является обёрткой низкоуровневых системных вызовов для работы с файлами. Абстрактный класс IRMStream описывает базовые операции буферизованного чтения/записи и является интерфейсом для «потоков» CRMFileStream, CRMZipStream, CRMXMLStream.

Для компрессии внешнего представления морфологической базы используется библиотека Дж.Р. Сьюарда libbzip2 [4].

Сетевое взаимодействие на платформе Windows ведётся с помощью библиотеки winsock2 и реализовано на базе соответствующих классов MFC: CSocket и CAsyncSocket. Для передачи текстовых строк используется кодировка Windows (CP-1251).

Для представления динамических массивов объектов или указателей на объекты реализованы шаблоны CSCTypeArray<**class type**> и CSCPtrArray<**class type**>, на базе которых реализован упорядоченный по ключу динамический массив указателей CSCDictionary<**class type**>.

3.2. Программный интерфейс

В данном разделе приводится описание программного интерфейса (API) модуля морфологии, которое поможет при встраивании модуля на уровне исходных текстов в проекты MS VC++ 6.0[3].

3.2.1. Встраивание на уровне исходных текстов

При интеграции на уровне исходных текстов взаимодействие с модулем идёт через интерфейсный объект. Основной процесс может управлять загрузкой и выгрузкой словарей RMU, производить анализ/синтез словоформ, а также частичное пополнение и модификацию морфологической базы.

[3] Описание для платформы Unix будет размещено в следующей версии данного документа.

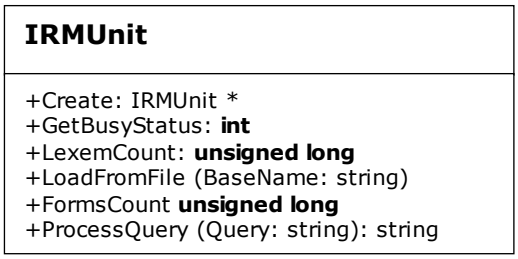
Подключение файлов в проект

Для подключения модуля морфологии в проект MS VC++ необходимо добавить в проект следующие файлы:

RMStock.cpp
RMStockManager.cpp
RMUnit.cpp
RMUtils.cpp
GeneralDefs.h
RMServer.h
RMStock.h
RMStockData.h
RMStockManager.h
RMU.h
RMUDlg.h
RMUnit.h
RMUtils.h
SCArray.h

Инициализация и работа с модулем

Логической единицей, представляющей интерфейс к модулю морфологии, является объект класса IRMUnit.



Для построения экземпляра морфологического анализатора необходимо воспользоваться производящей функцией Create:

```
IRMUnit* RMUnit = IRMUnit::Create ();
```

После конструирования необходимо произвести загрузку морфологической базы в оперативную память с помощью метода LoadFromFile (string& BaseName) (стр. 24).

```
RMUnit->LoadFromFile ( "rm" );
```

В случае успешного завершения этого метода в ОП будут построены все необходимые структуры, в противном случае выбрасывается исключение XException.

Так как процесс загрузки морфологических баз является трудоёмким, то рекомендуется производить вызов данного метода в параллельной нити для проектов, использующих графический пользовательский интерфейс. При этом с помощью метода GetBusyStatus можно отслеживать прогресс данной операции (→ стр. 23).

Для проведения анализа или синтеза словоформ (выполнение запросов) предназначен метод ProcessQuery (→ стр. 24). Запрос представляет собой размеченную текстовую

строку в формате XML (точнее — SGML[4]), оканчивающуюся двумя символами перевода строки \n\n (→ Запрос к модулю морфологии, стр. 49). После обработки запроса данный метод возвратит ответ в виде размеченной текстовой строки (также в формате XML).

```
string Answer;
Answer = RMUnit->ProcessQuery ( "<analyze form=\"краткий\">\n\n" );
```

Сервисные утилиты

Для того чтобы проинтерпретировать ответ, нужно разобрать строку в формате XML. Для этого можно воспользоваться сервисными утилитами (→ Компонент RMUtils, стр. 37). С помощью процедуры ParseLine (string& Line, strings& Tags) из компонента RMUtils можно разбить размеченную строку на несколько строк, в каждой из которых будет размещено по одному XML дескриптору из исходной строки.

```
strings   Answers;
ParseLine ( Answer, Answers );
```

После выделения отдельных XML–дескрипторов необходимо выделить атрибуты, в которых представлена дополнительная информация. Это можно сделать с помощью процедуры ParseTag (string& Tag, CAttrArray& Attrs).

```
CAttrArray   Attrs;
for ( int i = 0; i < Answers.Count (); ++i )
{
       ParseTag ( Answers[i], Attrs );
       //...
}
```

Теперь можно получить доступ к XML–атрибутам очередного XML–дескриптора либо поимённо, либо по его индексу.

Примеры работы

Рассмотрим некоторые примеры работы с анализатором:

```
Answer = RMUnit->ProcessQuery ( "<analyze form=\"крючья\">\n\n" );
```

После этого вызова Answer может содержать приблизительно следующий текст.

```
<answer>
<analyzed form="крючья" lexem="крюк"><noun animated="no" gender="male"
number="plural" case="nominative" stress="3"><noun animated="no"
gender="male" number="plural" case="accusative" stress="3">
</answer>
```

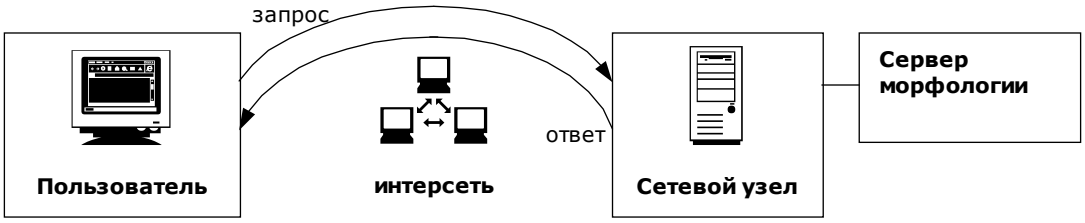
[4] На самом деле протокол взаимодействия отвечает стандарту SGML (ISO 8879), так как имеется чёткое описание DTD. А стандарт XML требует наличия дополнительных символов ("/>") в теле дескриптора, для которого допускается закрытие по умолчанию. Поэтому, строго говоря, протокол взаимодействия отвечает только стандарту SGML.

3.3. Интеграция с помощью сетевых сервисов

Морфологический сервер представляет собой исполняемое приложение, которое выполняет удалённые запросы к модулю морфологии. Общая схема взаимодействия выглядит следующим образом. На выделенном сетевом узле устанавливается сервер морфологии, а на соответствующие клиентские сетевые узлы — клиентский модуль морфологии. Для начала работы пользователя с удалённого узла необходимо запустить клиентский модуль, который производит установление сетевого соединения с сервером на весь сеанс работы.

3.3.1. Схема сетевого взаимодействия с сервером морфологии.

Сетевое взаимодействие с сервером морфологии происходит на базе механизма сетевых разъёмов, который реализован на основе стека протоколов TCP/IP. Это означает, что целевая компьютерная сеть должна поддерживать транспортные механизмы протоколов TCP/IP. Механизм сетевых разъёмов подразумевает, что помимо адреса сетевого узла, на котором установлен сервер морфологии, необходимо определить и номер порта, с которым будет ассоциирован сервер.



3.3.2. Каркас для сетевого взаимодействия

Для соединения с сервером морфологии необходимо создать сетевое соединение на базе сетевых разъёмов модели TCP/IP. Эту парадигму можно реализовать с помощью класса CSocket библиотеки MFC. Создав наследника этого класса (CRMServer), можно реализовать «посредника» между клиентским приложением и сервером морфологии, который будет скрывать сложности передачи данных в сети. Более того, обращение к нему идентично обращению к самому классу CRMUnit.

```
class CRMServer: public CSocket
{
    // Attributes
public:
    // Operations
    string ProcessQuery ( const string& Query );

    CRMServer ( string RMServerIP, unsigned long RMServerPort );
   ~CRMServer () {};
```

```
};

CRMServer::CRMServer ( string RMServerIP, unsigned long RMServerPort )
{
    if ( !Create ( 0, SOCK_STREAM ) )
        throw "Failure to open socket";
    else if ( !Connect ( RMServerIP.c_str(), RMServerPort ) )
        throw "Failure to connect the server" + RMServerIP;
}

string
CRMServer::SendQuery ( const string & Query )
{
    char   buffer[1024];
    int    MsgLength;
    string Answer;

    Query += "\n\n";
    Send ( Query, Query.length () );
    MsgLength = Receive ( &buffer, 1024 );
    Answer    = string ( buffer, MsgLength );
    while ( str.substr ( Answer.length() - 3, 2 ) != "\n\n" )
    {
        MsgLength = Receive ( &buffer, 1024 );
        Answer += string ( buffer, MsgLength );
    }
    return Answer;
}
```

Буферизация приёма ответа сделана для ускорения работы, размер буфера выбран минимальным, при котором задержка становиться приемлемой.

В качестве параметров конструктора CRMServer передаются сетевой адрес узла, на котором запущен сервер морфологии, и номер порта. При этом сеанс работы сетевого клиента можно описать следующим образом.

```
string     Answer;
strings    Answers;
CAttrArray Attrs;

CRMServer Server ( "myserver", 4999 );
Answer = Server.ProcesQuery ( "<analyze form=\"крючья\">" );
// Разобрать ответ на отдельные XML-дескрипторы
ParseLine ( Answer, Answers );
for ( int i = 0; i < Answers.Count (); ++i )
{
    // Выделить атрибуты из текущего XML-дескриптора
    ParseTag ( Answers[i], Attrs );
    // Обработать конкретные значения атрибутов
    // ...
}
```


Для разбора ответа на отдельные XML-дескрипторы и выделения их атрибутов в данном примере использованы вызовы сервисных функций ParseLine и ParseTag из компонента CRMUtils (→ стр. 37). Но в базовом комплекте поставки исходные тексты этого модуля не поставляются, поэтому для анализа ответа анализатора можно использовать сторонние SGML-фильтры, так как протокол взаимодействия, для которого DTD описан в приложении, основан на языке SGML.

3.3.3. Отладка с помощью терминала

Для отладки сетевого взаимодействия можно использовать текстовый сетевой клиент, например telnet. Для поддержки такого режима взаимодействия язык запросов расширен следующими командами:

<enableecho> — включает режим «отображения ввода», при этом обрабатывается символ "BackSpace" («забой») в пределах ввода одного запроса;

<disableecho> — выключает режим «отображения ввода»;

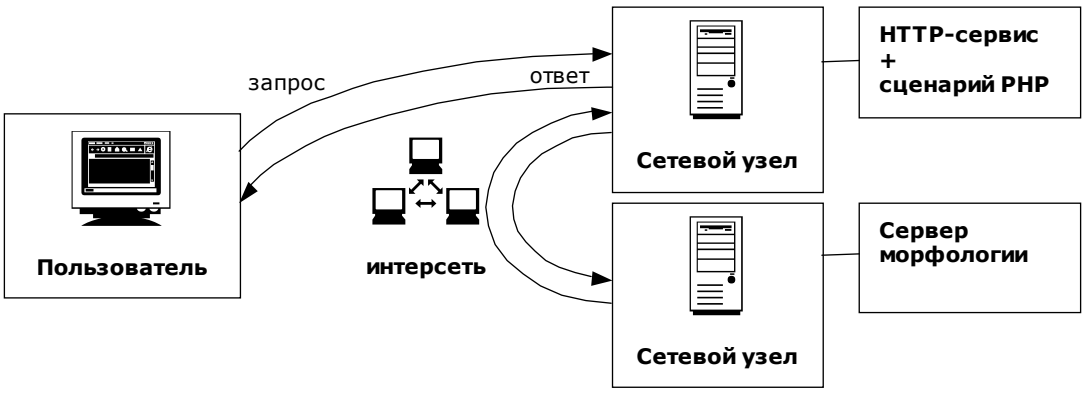
<select charset=" "> — выбирает текущую кодировку сеанса взаимодействия, чтобы избежать проблем с русскими буквами при тестировании в разных системах, допустимые значения параметра charset: cp866, cp1251.

3.4. WWW-сценарий

Для взаимодействия через интерсеть требуется обеспечить следующее два условия:

- наличие в сети выделенного узла с HTTP-сервисом, который поддерживает сценарии PHP;
- наличие в сети выделенного узла, на котором установлен сервер морфологии.

При выполнении этих условий на HTTP-сервер можно записать сценарий из расширенного комплекта поставки модуля морфологии, тогда после определенных настроек (→ 3.4.2, стр. 20) пользователь получит возможность работы с сервером морфологии через WWW. На приведённой ниже диаграмме схематически изображено данное взаимодействие пользователя с сервером морфологии.



В качестве пояснения можно отметить, что при необходимости сервер морфологии и HTTP-сервис могут быть установлены на одном сетевом узле.

3.4.1. Сценарий для HTTP-сервера

Серверный сценарий написан на языке PHP (для интерпретатора PHP версии 4.0 и выше). Поставляемый сценарий тестировался на HTTP-сервере Apache (версии 1.3.x и выше) на платформах Windows 9x/NT и Unix. В качестве основы для сценария был взят исходный текст пользовательского интерфейса модуля морфологии на языке C++, поскольку синтаксис PHP имеет много общего с языком C++.

Процесс взаимодействие сценария с сервером морфологии происходит аналогично процессу общения клиентского модуля (→ 2.2, стр. 12). Единственная разница заключается в том, что соединение на базе сетевых разъёмов устанавливается не на весь сеанс работы пользователя с сервером морфологии, а только на период обработки каждого из поданных запросов. Эта особенность сетевого соединения обусловлена спецификой сеансовой работы WWW-сервисов, а именно: отсутствие постоянного соединения с HTTP-сервером.

3.4.2. Описание серверного сценария RMU

Для настройки сценария RMU.php необходимо изменить значения двух переменных ($RMServerAddress и $RMServerPortNumber), которые содержат сетевое имя узла (с установленным сервером морфологии) и номер порта, на фактические значения. Например:

```
$RMServerAddress = "morpho.server.ru"; // Set the real server network address here
$RMServerportNumber = "4999";          // Set the real server port number here
```

В целом сценарий содержит три функции, которые аналогичны функциям модуля морфологии, описанным в первой части документа.

Функция ParseLine

Профиль

```
array function ParseLine ( string $Line )
```

Данная функция производит разбиение строки на массив строк, который содержат выделенные из входной строки XML-элементы.

Функция ParseTag

Профиль:

```
array of array function ParseTag ( string $Tag )
```

Данная функция производит разбиение XML-элемента на ассоциативный массив массивов строк, в которые размещает выделенные имена атрибутов XML-элемента и их значения.

Процедура ProcessQuery

Профиль:

```
void function ProcessQuery ( string $Act, string $Form )
```

Данная процедура производит выполнение запроса пользователя. Параметр `$Act` указывает тип операции: анализ или синтез, а параметр `$Form` — входную словоформу. Результатом работы процедуры направляется в стандартный вывод. Для выполнения запроса процедура производит соединение с сетевым узлом, адрес которого указан в переменной `$RMServerAddress`. В случае неудачного соединения процедура выводит в стандартный вывод сообщение об ошибке.

Часть II. Описание компонентов модуля

Данная часть документа предназначена для разработчиков RMU, так как содержит подробное описание программной архитектуры модуля.

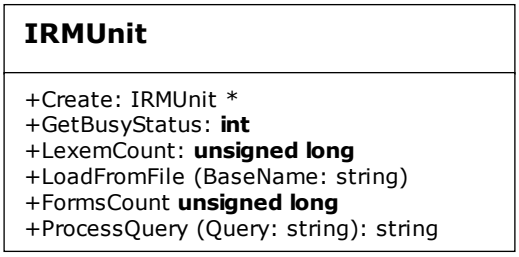
Глава 4. Описание главных компонент

В данной главе приводится описание основных компонентов (или модулей в терминах языка C++), входящих в ядро RMUnit. Почти все реализации основных компонент имеют интерфейс, описанный отдельным (одноименным, с префиксом "I") классом, и представление, описанное отдельным (одноименным, с префиксом "R") классом. По определенным причинам в описании не упоминаются некоторые приватные методы или поля данных классов, раскрытие функциональности или семантики которых не является необходимым.

Ядро модуля морфологии написано на языке C++ с использованием стандартной библиотеки ввода/вывода языка C, а также стандартной библиотеки шаблонов STL. Графический интерфейс модуля для ОС MS Windows 9x/NT написан с использованием библиотеки MFC.

4.1. Компонент RMUnit

Компонент RMUnit представляет собой реализацию встраиваемого ядра модуля морфологии (embedded), интерфейс которого описан с помощью класса IRMUnit.



Экземпляр RMUnit конструируется с помощью производящей функции Create.

Функция GetBusyStatus

Профиль:

```
int GetBusyStatus () const;
```

Данная функция возвращает прогресс выполнения модулем морфологии трудоёмкой операции, например, загрузка словарей в оперативную память. Эта функция полезна при встраивании модуля морфологии в многопоточные приложения. Чем меньше возвращаемое значение, тем меньше объём оставшихся необходимых вычислений. Максимальное значение — 100, минимальное — 0 (означает, что трудоёмкая операция закончена).

Функция FormsCount

Профиль:

```
unsigned long FormsCount () const;
```

Данная функция возвращает число словоформ в базе модуля морфологии. Этот метод является «обёрткой» метода IRMStockManager::FormsCount (→ стр. 25).

Функция LexemCount

Профиль:

```
unsigned long LexemCount () const;
```

Данная функция возвращает число лексем в базе модуля морфологии. Этот метод является «обёрткой» метода IRMStockManager::LexemCount (→ стр. 26).

Процедура LoadFromFile

Профиль:

```
void LoadFromFile ( string BaseName );
```

Данная процедура является «обёрткой» метода IRMStockManager::LoadFromFile (→ стр. 26).

Функция ProcessQuery

Профиль:

```
string ProcessQuery ( string Query );
```

Данная процедура обрабатывает запрос к модулю морфологии. Запрос и результат работы функции представляется в виде строки в XML-формате (→ Запрос к модулю морфологии, стр. 49). Данная функция является ключевой в программном интерфейсе модуля морфологии.

Процедура SaveToFile

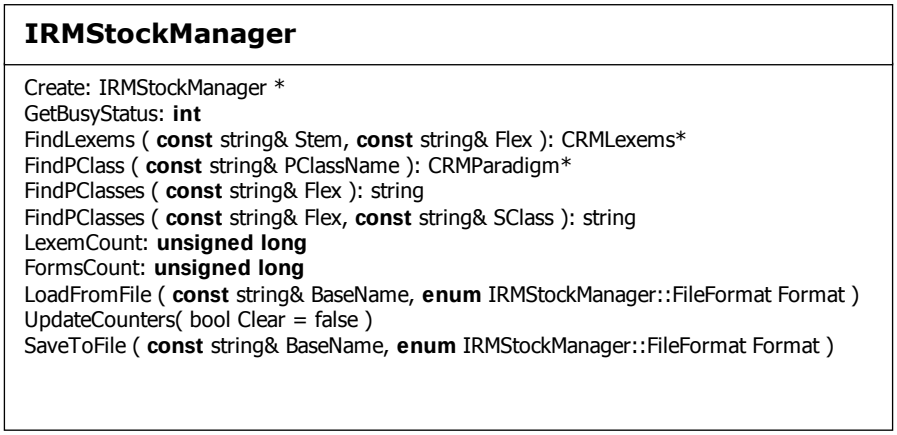
Профиль:

```
void SaveToFile ( string BaseName );
```

Данная процедура является обёрткой метода IRMStockManager::SaveToFile (→ стр. 26).

4.2. Компонент RMStockManager

Компонент RMStockManager представляет собой реализацию интерфейса IRMStockManager, обеспечивая ядру анализатора функциональность работы с грамматическими словарями в оперативной памяти с возможностью загрузки и сохранения во внешнюю память в формате XML.



Экземпляр RMStockManager конструируется с помощью производящей функции Create.

Функция GetBusyStatus

Профиль:

```
int GetBusyStatus () const;
```

Данная функция возвращает прогресс выполнения модулем морфологии трудоёмкой операции, например, загрузка словарей в оперативную память (→ Функция GetBusyStatus, стр. 23).

Функция FindLexems

Профиль:

```
CRMLexems * FindLexems ( const string& Stem, const string& Flex );
```

Данная функция формирует список лексем, которые имеют заданную основу и заданную флексию.

Функция FindPClass

Профиль:

```
CRMParadigm * FindPClass ( const string& PClassName );
```

Данная функция возвращает указатель на П-класс с заданным именем.

Функция FindPClasses

Профиль:

```
string FindPClasses ( const string& Flex );
```

Данная функция строит XML-строку, содержащую перечень всех П-классов, содержащих заданную флексию.

Функция FindPClasses

Профиль:

```
string FindPClasses ( const string& Flex, const string& SClass );
```

Данная функция строит XML-строку, содержащую перечень всех П-классов, содержащих заданную флексию и относящихся к заданному морфосинтакическому классу.

Функция FormsCount

Профиль:

```
unsigned long FormsCount () const;
```

Данная функция возвращает число словоформ в базе модуля морфологии. Этот метод используется (в паре с LexemCount) для формирования статистики о базе при работе модуля морфологии в комплектации «Сервер русской морфологии».

Функция FindLexems

Профиль:

```
CRMLexems * FindLexems ( string Stem, string Flex );
```

Данная функция производит поиск всех лексем, содержащих флексию Flex и имеющих основу Stem. Результатом является список указателей на лексемы (класс CRMLexems).

Функция LexemCount

Профиль:

```
unsigned long LexemCount () const;
```

Данная функция возвращает число лексем в базе модуля морфологии. Этот метод используется (в паре с FormsCount) для формирования статистики о базе при работе модуля морфологии в комплектации «Сервер русской морфологии».

Процедура LoadFromFile

Профиль:

```
void LoadFromFile ( string BaseName,
                    enum IRMStockManager::FileFormat Format );
```

Данная процедура производит загрузку в память грамматических словарей из файлов с заданным именем в качестве префикса и в заданном формате (бинарный или XML). Файлы морфологической базы имеют аффиксы (affixes): [".lexem.data" | ".stem.data" | ".pclass.data" | ".schema.data"]. В зависимости от указанного формата при загрузке будут считаны файлы BaseName + affixes или BaseName + "bin." + affixes, а также файлы пользовательского словаря BaseName + "user." + affixes. При возникновении неустранимой ошибки в процессе выполнения данной процедуры генерируется исключительная ситуация XException (→ Класс XException, стр. 41).

Процедура SaveToFile

Профиль:

```
void SaveToFile ( string BaseName,
                  enum IRMStockManager::FileFormat Format );
```

Данная процедура производит сохранение во внешнюю память грамматических словарей в заданном формате (бинарный или XML). В зависимости от указанного формата при сохранении будут записаны файлы BaseName + affixes или BaseName + "bin." + affixes, а также файлы пользовательского словаря BaseName + "user." +

affixes. При возникновении неустранимой ошибки в процессе выполнения данной процедуры генерируется исключительная ситуация XException (→ Класс XException, стр. 41).

Процедура UpdateCounters

Профиль:

```
void UpdateCounters ( bool Clear );
```

Данная процедура обновляет счётчики частотности лексем.

4.3. Компонент RMStock

В компоненте RMStock описаны классы объектов, которые представляют собой элементы грамматических словарей псевдооснов (CRMStem), лексем (CRMLexem), словоизменительных классов (CRMParadigm) и схем ударений (CRMStressSchema). Основные типы, необходимые для представления грамматическо-морфологической информации, описаны в компоненте RMStockData (→ стр. 32).

4.3.1. Класс CRMStem

Класс CRMStem описывает элемент словаря псевдооснов.

```
class CRMStem
{
public:
    string                UID;
    CSCTypeArray<string>  LexemIDs;
    CSCPtrArray<CRMLexem> Lexems;

                          CRMStem ();
                          CRMStem ( const CRMStem& Stem );
};
```

Поле UID

Данное поле хранит уникальный идентификатор элемента.

Поле LexemIDs

Данное поле представляет собой массив строковых идентификаторов элементов типа CRMLexem (элементы списка лексем), с которыми связана данная основа. Данный массив заполняется во время загрузки с внешней памяти и после прямого связывания указателями очищается (→ Поле Lexems).

Поле Lexems

Данное поле представляет собой список указателей на элементы типа CRMLexem (элементы списка лексем), с которыми связана данная основа. Данный массив заполняется после загрузки с внешней памяти во время процедуры «жёсткого» связывания элементов словаря основ с элементами списка лексем.

4.3.2. Класс CRMLexem

Класс CRMLexem описывает семантическую единицу языка (лексему, слово), её постоянные грамматические признаки, а также имеет ссылки на изменяемые признаки ее различных словоформ. Для более точного толкования принятого понятия «лексема» обращайтесь к описанию используемой модели русского языка (...).

Класс CRMLexem является абстрактным базовым типом для конкретных типов лексем (→ стр. 30), поскольку в нём объявлено несколько чисто виртуальных функций.

```
class CRMLexem
{
public:
  string           UID;
  string           ParadigmID;
  CRMParadigm*     Paradigm;
  string           StressSchemaID;
  CRMStressSchema* StressSchema;
  unsigned long    UsageCounter;

  string           GetDerivedProperties () const;
  virtual ERMMorphoClass GetSynClass () const = 0;
  string           IdentifyParticiple ( unsigned long ParticipleFlex,
                                        const string& Preamble ) const;
  virtual string   IdentifyFlex ( unsigned long FlexPosition,
                                  const string& Form ) const = 0;
  virtual string   ToXML () const = 0;
                   CRMLexem ();
                   CRMLexem ( const CRMLexem& Lexem );
  virtual         ~CRMLexem ();
};
```

Поле UID

Данное поле хранит уникальный идентификатор элемента.

Поле ParadigmID

Данное поле хранит идентификатор элемента словоизменительного класса CRMParadigm. После загрузки с внешней памяти и прямого связывания, данное поле очищается.

Поле Paradigm

Данное поле хранит указатель на элемент CRMParadigm словоизменительного класса (П-класс).

Поле StressSchemaID

Данное поле хранит идентификатор элемента схемы ударений CRMStressSchema. После загрузки с внешней памяти и прямого связывания, данное поле очищается.

Поле StressSchema

Данное поле хранит указатель на элемент схемы ударений CRMStressSchema.

Поле SynClass

Данное поле хранит морфологический (синтаксический) класс формы.

Функция GetDerivedProperties

Профиль:

```
string GetDerivedProperties () const;
```

Данная функция возвращает базовую информацию для всех типов лексем: значения полей UID и UsageCounter в виде атрибутов в формате XML.

Функция GetSynClass

Профиль:

```
virtual ERMMorphoClass GetSynClass () const;
```

Данная функция возвращает тип данной лексемы (→ Тип ERMMorphoClass, стр.33).

Функция IdentifyParticiple

Профиль:

```
string IdentifyParticiple ( unsigned long ParticipleFlex,
                            const string& Preamble ) const;
```

Данная функция формирует строку в формате XML, описывающую образованное от данной лексемы причастие или деепричастие в форме, соответствующей номеру флексии ParticipleFlex (→ Функция Contains, стр. 32). Эта функция вызывается при анализе глагольной группы.

Функция IdentifyFlex

Профиль:

```
virtual string IdentifyFlex ( unsigned long FlexPosition,
                              const string& Form ) const = 0;
```

Данная функция формирует строку в формате XML, описывающую образованную от данной лексемы форму, которая соответствует номеру флексии FlexPosition.

Функция ToXML

Профиль:

```
virtual string ToXML () const = 0;
```

Данная функция возвращает статическое представление лексемы в формате XML. Эта функция используется при сохранении морфологической базы во внешнее представление в формате XML.

4.3.3. Классы, описывающие лексемы

Классы, представляющие описания конкретных типов лексем, являются наследниками класса CRMLexem (→ стр. 28). Помимо реализации интерфейса, описанного в CRMLexem, данные классы реализуют ещё два метода: GetStaticProperties, GetDynamicProperties, которые возвращают статическую и вычисляемую (динамическую) информацию при анализе словоформы.

4.3.4. Класс CRMParadigm

Класс CRMParadigm представляет описание словоизменительного класса (парадигматического класса).

```
class CRMParadigm
{
public:
    struct ParadigmItem
    {
        int       ItemType;
        bool      Is ( string Type );
    };

    struct Flex: public ParadigmItem
    {
        string   ID;
    };

    struct Variation: public ParadigmItem
    {
        CSCTypeArray<string> Flexes;
        CSCTypeArray<string> NestedIDs;
    };

    struct Flexes: public ParadigmItem
    {
        CSCTypeArray<string>      Items;
        CSCPtrArray<CRMParadigm>  Nested;
                                  Flexes ();
    };

    struct FlexPositions
    {
        CSCTypeArray<unsigned long>   Items;
    };

    // Attributes
    string                    UID;
    CSCPtrArray<ParadigmItem> Items;

    // Operations
    FlexPositions*        Contains      ( const string& Flex );
    bool                  ContainsFlex  ( const string& Flex );
    strings*              GetAllFlexes  () const;
    FlexPositions*        GetAllFlexPositions ()  const;
    string                GetFirstFlex  () const;
                          CRMParadigm ();
                          CRMParadigm ( const CRMParadigm& Paradigm );
}
```

Поле UID

Данное поле хранит уникальный идентификатор элемента.

Поле Items

Данное поле хранит список флексий, относящихся к данному словоизменительному классу. Поле представляет собой массив указателей на полиморфные объекты, являющиеся наследниками от класса CRMParadigm::ParadigmItem. Это означает, элементами этого списка могут быть объекты классов CRMParadigm::Flex, либо CRMParadigm::Variation, либо CRMParadigm::Flexes. На этапе загрузки (до операции связывания) вариативность флексии представляется объектом класса CRMParadigm::Variation. Затем, после связывания, все элементы CRMParadigm::Variation заменяются на CRMParadigm::Flexes, которые содержат прямые ссылки на соответствующие вложенные парадигматические классы.

Функция ContainsFlex

Профиль:

```
bool ContainsFlex ( const string& Flex )
```

Данная функция определяет, содержит ли данная парадигма указанную флексию.

Функция Contains

Профиль:

```
FlexPositions Contains ( const string& Flex )
```

Данная функция возвращает номера флексий данного парадигматического класса, совпадающих с указанной строкой. В случае вложенных парадигматических классов, применяется следующее кодирование номера флексии:

$$FlexPosition_i = \sum_{k=0} SubFlexPosition_{i,k} \cdot 512^k,$$

то есть имеется ограничение на длину набора флексий лексемы в пределах одного уровня вложенности парадигматического класса (512 флексий), и на количество вложенных подклассов (MAX_INT / 512). Но, судя по всему, такое ограничение не затрагивает существо решаемой задачи.

4.3.5. Класс CRMStressSchema

Класс CRMStressSchema представляет описание схемы ударения.

4.4. Компонент RMStockData

В данном компоненте описаны основные типы данных для представления морфологической информации, а также некоторые константные объекты.

4.4.1. Перечислимые типы

Для кодификации различных состояний и ситуаций в компоненте RMStock описаны соответствующие перечислимые типы, которые широко используются в других компонентах модуля морфологии.

Тип ERMMorphoClass

Перечислимый тип ERMMorphoClass описывает морфологический (синтаксический) класс лексем русского языка. Данное разбиение является спецификой модели русского языка, реализованной в данном модуле морфологии.

```
enum ERMMorphoClass
{
    ERM_PERSONAL_PRONOUN,                       // личное местоимение                                                    (3-1)
    ERM_REFLEXIVE_PRONOUN,                      // возвратное местоимение                                                (3-2)
    ERM_PRONOUN,                                // местоимение                                                           (3-3)
    ERM_NUMBER_ONE,                             // числительное один                                                     (4-1)
    ERM_NUMBER_TWO,                             // числительное два, оба, полтора                                        (4-2)
    // числительное три, четыре сколько, несколько, столько, много, немного и собирательные числительные                 (4-3)
    ERM_NUMBER_THREE,
    ERM_NUMBER,                                 // количественное числительное                                           (4-4)
    ERM_NUMBER_BIFORM,                          // числительное с двумя формами                                          (4-5)
    ERM_NOUN_MALE_INANIMATE,                    // существительное неодушевленное мужского рода                          (7-1)
    ERM_NOUN_MALE_ANIMATE,                      // существительное одушевленное мужского рода                            (7-2)
    ERM_NOUN_MALE_OR_FEMALE_INANIMATE,          // существительное неодушевленное мужского/женского рода                 (7-9)
    ERM_NOUN_MALE_OR_FEMALE_ANIMATE,            // существительное одушевленное мужского/женского рода                   (7-10)
    ERM_ADJECTIVE,                              // прилагательное                                                        (8-1)
    ERM_PRONOUN_ADJECTIVE,                      // местоименное прилагательное                                           (8-2)
    ERM_NUMBER_ORDINAL,                         // порядковое числительное                                               (8-3)
    ERM_PARTICIPLE,                             // причастие                                                             (8-4)
    ERM_POSSESIVE_ADJECTIVE,                    // притяжательное прилагательное                                         (8-5)
    // глагол невозвратный непереходный несовершенного вида (не многократный и не безличный)                             (9-1)
    ERM_VERB_IRREFLEXIVE_INTRANSITIVE_IMPERFECTIVE,
    // глагол невозвратный непереходный совершенного вида (не многократный и не безличный)                               (9-2)
    ERM_VERB_IRREFLEXIVE_INTRANSITIVE_PERFECTIVE,
    // глагол невозвратный переходный несовершенного вида (не многократный и не безличный)                               (9-3)
    ERM_VERB_IRREFLEXIVE_TRANSITIVE_IMPERFECTIVE,
    // глагол невозвратный переходный совершенного вида (не многократный и не безличный)                                 (9-4)
    ERM_VERB_IRREFLEXIVE_TRANSITIVE_PERFECTIVE,
    // глагол возвратный непереходный несовершенного вида (не многократный и не безличный)                               (9-5)
    ERM_VERB_REFLEXIVE_IMPERFECTIVE,
    // глагол возвратный непереходный совершенного вида (не многократный и не безличный)                                 (9-6)
    ERM_VERB_REFLEXIVE_PERFECTIVE,
    // глагол многократный невозвратный непереходный несовершенного вида                                                 (9-7)
    ERM_VERB_FREQUENTATIVE_IRREFLEXIVE_INTRANSITIVE_IMPERFECTIVE,
    // глагол многократный невозвратный переходный несовершенного вида                                                   (9-8)
    ERM_VERB_FREQUENTATIVE_IRREFLEXIVE_TRANSITIVE_IMPERFECTIVE,
    // глагол безличный невозвратный непереходный несовершенного вида                                                    (9-9)
    ERM_VERB_IMPERSONAL_IRREFLEXIVE_INTRANSITIVE_IMPERFECTIVE,
    // глагол безличный невозвратный непереходный совершенного вида                                                      (9-10)
    ERM_VERB_IMPERSONAL_IRREFLEXIVE_INTRANSITIVE_PERFECTIVE,
    // глагол безличный невозвратный переходный несовершенного вида                                                      (9-11)
    ERM_VERB_IMPERSONAL_IRREFLEXIVE_TRANSITIVE_IMPERFECTIVE,
    // глагол безличный невозвратный переходный совершенного вида                                                        (9-12)
    ERM_VERB_IMPERSONAL_IRREFLEXIVE_TRANSITIVE_PERFECTIVE,
    // глагол безличный возвратный непереходный несовершенного вида                                                      (9-13)
    ERM_VERB_IMPERSONAL_REFLEXIVE_INTRANSITIVE_IMPERFECTIVE,
    // глагол безличный возвратный непереходный совершенного вида                                                        (9-14)
    ERM_VERB_IMPERSONAL_REFLEXIVE_INTRANSITIVE_PERFECTIVE,
    // глагол невозвратный непереходный двувидовый                                                                       (9-15)
    ERM_VERB_IRREFLEXIVE_INTRANSITIVE_BIFORM,
    // глагол невозвратный переходный двувидовый                                                                         (9-16)
    ERM_VERB_IRREFLEXIVE_TRANSITIVE_BIFORM,
    // глагол возвратный непереходный двувидовый                                                                         (9-17)
    ERM_VERB_REFLEXIVE_INTRANSITIVE_BIFORM,
    // глагол возвратный непереходный двувидовый (только в наст/буд)                                                     (9-18)
    ERM_VERB_REFLEXIVE_INTRANSITIVE_BIFORM_EXT,
    // глагол "быть" невозвратный непереходный                                                                           (9-19)
    ERM_VERB_BE_IRREFLEXIVE_INTRANSITIVE,
    ERM_PREPOSITION,                            // предлог
    ERM_CONJUNCTION,                            // союз
```

```
    ERM_INTERJECTION,                   // междометие
    ERM_PREDICATIVE,                    // предикатив
    ERM_PARTICLE,                       // частица
    ERM_PARENTHESIS,                    // вводное слово
    ERM_ADVERB,                         // наречие
};
```

Тип ERMCasesNumbers

Данный тип кодифицирует основные падежи русского языка.

```
enum ERMCasesNumbers
{
    ERMNominativeCase,    // именительный
    ERMGenitiveCase,      // родительный
    ERMDativeCase,        // дательный
    ERMAccusativeCase,    // винительный
    ERMInstrumentalCase,  // творительный
    ERMPrepositionalCase, // предложный
};
```

4.4.2. Константные строковые массивы

Аналогично описанным выше перечислимым типам в компоненте RMStock описаны глобальные строковые константные массивы. В основном они используются при формировании ответа модуля морфологии в формате XML.

Массив ERMCases

Данный массив хранит строковое представление английских названий падежей русского языка.

```
const char ERMCases[6][16] =
{
    "nominative",     // именительный падеж
    "genitive",       // родительный падеж
    "dative",         // дательный падеж
    "accusative",     // винительный падеж
    "instrumental",   // творительный падеж
    "prepositional"   // предложный падеж
};
```

Массив ERMAnimates

Данный массив хранит строковое представление английских названий вариантов одушевлённости.

```
const char ERMAnimates[3][4] =
{
    "yes",      // одушевлённое
    "no",       // неодушевлённое
```



```
    "any"       // обобщённая одушевлённость
};
```

Массив ERMGenders

Данный массив хранит строковое представление английских названий родовой принадлежности.

```
const char ERMGenders[4][8] =
{
    "male",     // мужской род
    "female",   // женский род
    "neuter",   // средний род
    "any"       // обобщённый род
};
```

Массив ERMNumbers

Данный массив хранит строковое представление английских названий численной принадлежности.

```
const char ERMNumbers[3][8] =
{
    "single",   // единственное число
    "plural",   // множественное число
    "any"       // неопределённое число
};
```

Массив ERMPersons

Данный массив хранит строковое представление английских названий принадлежности к соответствующему лицу.

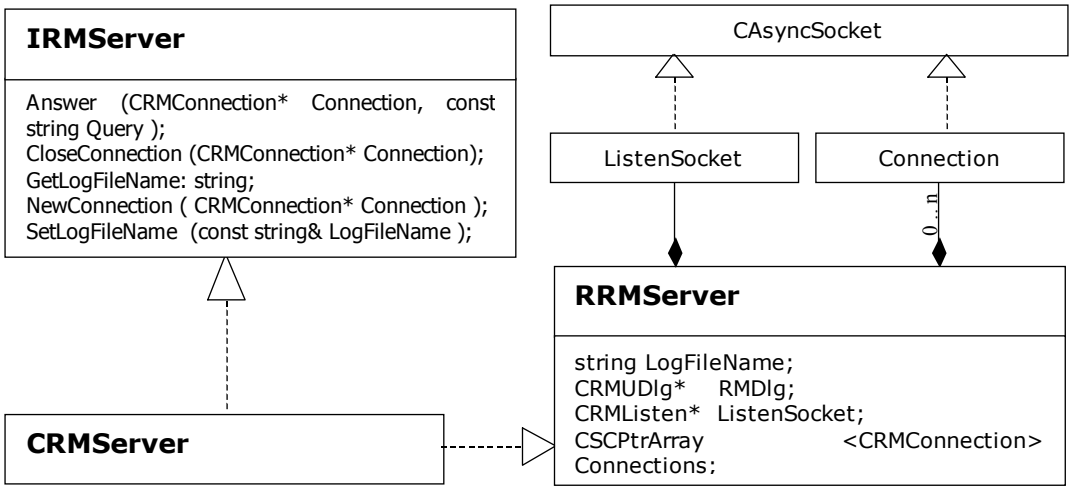
```
const char ERMPersons[4][4] =
{
    "1",        // первое лицо
    "2",        // второе лицо
    "3",        // третье лицо
    "any"       // четвёртое лицо
};
```

4.5. Компонент RMServer

Компонент RMServer реализует механизм сетевого взаимодействия модуля морфологии в режиме сервера с сетевыми клиентами.

Класс CRMServer реализует основной интерфейс сетевого взаимодействия. Для приёма заявок на соединение с сервером используется асинхронный сетевой разъём CRMListenSocket (наследник от CAsyncSocket). После обработки события CAsyncSocket

::OnAccept создаётся новое соединение CRMConnection (наследник от CAsyncSocket), которое заносится в массив активных соединений RRMServer::Connections. По завершении сеанса соединения (инициатива сетевого клиента или ошибка сети) активное подключение удаляется из списка.



В конструктор класса передаётся указатель на диалог приложения для того, чтобы своевременно выводить в окно текущее количество подключений.

```
class CRMServer: public IRMServer, protected RRMServer
{
public:
    void     Answer ( CRMConnection* Connection, string& Query );
    void     CloseConnection ( CRMConnection* Connection );
    string   GetLogFileName  () const;
    void     NewConnection   ( CRMConnection* Connection );
    void     SetLogFileName  ( const string& ALogFileName );

             CRMServer       ( CRMUDlg* RMUDlg );
    virtual ~CRMServer       ();
};
```

Функция GetLogFileName

Профиль:

```
string GetLogFileName () const;
```

Данная функция возвращает имя текущего файла протокола установленных соединений.

Процедура SetLogFileName

Профиль

```
void SetLogFileName ( const string& ALogFileName );
```

Данная процедура устанавливает имя файла протокола установленных соединений. Если ALogFileName — пустая строка, протоколирование соединений производиться не будет.

Глава 5. Дополнительные инструментальные компоненты

В данной главе приводятся описания дополнительных компонентов, которые входят в состав модуля русской морфологии.

5.1. Компонент RMUtils

Для удобства и наглядности некоторые полезные вспомогательные инструментальные классы и процедуры модуля морфологии, не входящие в его ядро, вынесены в данный компонент.

5.1.1. Класс CAttr

Класс CAttr представляет собой структуру из двух полей: Name — имя атрибута, и Value — его значение. Данный класс используется в качестве элемента массива атрибутов (→ Класс CAttrArray).

```
struct CAttr
{
    string  Name;
    string  Value;
    CAttr () {};
    CAttr ( const CAttr& Attr );
};
```

5.1.2. Класс CAttrArray

Класс CAttrArray предназначен для представления проанализированной размеченной XML-строки в виде массива атрибутов, которые представлены классом CAttr. Интерфейс класса позволяет получать доступ к атрибутам либо по номеру, либо по имени. Доступ по имени можно проводить двумя способами: простой по имени (метод Get), либо двухэтапный оптимизированный. Во втором случае доступ происходит в два этапа: сначала проверяется наличие требуемого атрибута и его порядковый номер (метод Exists), а затем производится обычный доступ по индексу. Остальные методы класса идентичны одноимённым методам класса CSCPtrArray (→ стр. 39).

```
class CAttrArray
{
    CSCTypeArray<CAttr> Array;
public:
    CAttr         operator[] ( int Index );
    unsigned long Add ( CAttr Attr );
    unsigned long Count ();
```

```
        bool        Exists ( const string& AttrName,
                             unsigned long& AttrIndex ) const;
        string      Get ( const string& AttrName ) const;
        void        RemoveAll ();

                    CAttrArray () {};
                    CAttrArray ( const CAttrArray& AttrArray )
    };
```

Функция Exists

Профиль:

```
bool Exists ( const string& AttrName, unsigned long& AttrIndex ) const;
```

Данная функция проверяет наличие атрибута с указанным именем. Если он существует, то параметр AttrIndex принимает значение, равное порядковому номеру атрибута.

Функция Get

Профиль:

```
string Get ( const string& AttrName ) const;
```

Данная функция возвращает значение атрибута с указанным именем. Если атрибут не существует, генерируется исключение.

5.1.3. Процедуры компонента RMUtils

Помимо описаний вспомогательных классов в данном компоненте определены инструментальные функции.

Функция IntToStr

Профиль:

```
inline int IntToStr ( string& Str );
```

Данная функция преобразует строку, представляющую целое число, в целый тип int.

Процедура ParseLine

Профиль:

```
void ParseLine ( string& Line, strings& Tags );
```

Данная процедура предназначена для выделения из размеченной строки описаний отдельных XML-элементов. Параметр Line является входной XML-строкой, а массив strings заполняется по мере выполнения процедуры.

Процедура ParseTag

Профиль:

```
void ParseTag ( string& Line, CAttrArray& Attrs );
```

Данная процедура предназначена для выделения из размеченной строки списка имен и значений атрибутов XML-элемента. Параметр Line является входной XML-строкой

описывающей один XML-элемента, а массив Attrs заполняется по мере выполнения процедуры.

Функция StrToInt

Профиль:

```
inline char* IntToStr ( int Value );
```

Данная функция преобразует целое число в строковое представление.

5.2. Компонент SCArray

Данный компонент содержит описания шаблонов классов SCPtrArray, SCPtrHugeArray и SCTypeArray, которые реализуют функциональность динамических массивов. Данные шаблоны классов идентичны друг другу с точностью некоторых деталей. Первые два шаблона предназначены для хранения типизированных указателей на объекты, причем SCPtrHugeArray оптимизирован для работы с большими массивами. Шаблон SCTypeArray предназначен для реализации динамического массива объектов параметризованного типа. Он позволяет сэкономить на хранении дополнительных указателей на объекты в массиве.

5.2.1. Шаблон класса CSCPtrArray

Данный класс реализует функциональность динамического массива указателей на объекты типа ArgType, который задается параметром шаблона. Нумерация элементов производится с 0.

```
template <class ArgType> class CSCPtrArray
{
protected:
    // Attributes
    unsigned    long        ItemsQnt;
                ArgType**   Array;
                bool        OwnsItems;
public:
    // Operations
    unsigned    long        Add             ( ArgType* );
    unsigned    long        Add             ( ArgType*, unsigned long );
    unsigned    long        Count           () const;
                void        ForceNotToOwn   ();
                ArgType*    operator[]      ( unsigned long ) const;
                void        operator=       ( const CSCPtrArray& );
                void        Replace         ( ArgType*, unsigned long );
                void        Remove          ( unsigned long );
                void        RemoveAll       ();

                            CSCPtrArray     ();
                           ~CSCPtrArray     ();
};
```

Функция Add

Профиль:

```
unsigned long Add ( ArgType* );
```

Данная функция производит добавление нового элемента в конец массива. В качестве возвращаемого значения выступает позиция добавленного элемента.

Профиль:

```
unsigned long Add ( ArgType*, unsigned long );
```

Данная функция производит добавление нового элемента в указанную позицию. В качестве возвращаемого значения выступает позиция добавленного элемента.

Функция Count

Профиль:

```
unsigned long Count () const;
```

Данная функция возвращает количество элементов динамического массива.

Процедура ForceNotToOwn

Профиль:

```
void ForceNotToOwn ();
```

Данная процедура должна вызываться для того, чтобы проинформировать объект класса CSCPtrArray о том, что при разрушении массива не следует производить удаление из памяти самих элементов массива.

Процедура Replace

Профиль:

```
void Replace ( ArgType*, unsigned long );
```

Данная процедура производит замену указателя в указанной позиции на новый. Если поле OwnsItems==true, то есть считается, что динамический массив указателей «владеет» адресуемыми объектами, то данная процедура также производит удаление старого объекта, чтобы исключить появление «висячих» объектов.

Процедура Remove

Профиль:

```
void Remove ( unsigned long );
```

Данная процедура производит удаление из массива элемента в указанной позиции. Если поле OwnsItems==true, то есть считается, что динамический массив указателей «владеет» адресуемыми объектами, то данная процедура также производит разрушение старого объекта, чтобы исключить появление «висячих» объектов.

Процедура RemoveAll

Профиль:

```
void RemoveAll ();
```


Данная процедура производит удаление всех элементов из массива. Если поле OwnsItems==true, то есть считается, что динамический массив указателей «владеет» адресуемыми объектами, то данная процедура также производит разрушение всех адресуемых объектов, чтобы исключить появление «висячих» объектов.

5.3. Компонент общих определений

Компонент GeneralDefs представляет собой заголовочный файл с базовыми определениями для всего проекта модуля морфологии. В частности, в нём содержится описание класса объектов, используемых при обработке исключительных ситуации во время исполнения кода модуля морфологии.

5.3.1. Класс XException

Данный класс представляет собой базовый тип объектов, которые используются при обработке различного рода исключительных ситуаций.

```
class XException
{
public:
    string  GetNotification ();
    void    Notify ();
            XException ( string Msg = "", string Method = "undescribed",
                         string Class = "undescribed",
                         string Module = "undescribed" )
            XException ( EExceptions ExceptionType, string Msg,
                         string Method, string Class, string Module )
};
```

Функция GetNotification

Профиль:

```
string GetNotification ();
```

Данная функция возвращает текст сообщения об ошибке, составленного из параметров, заданных в конструкторе XException.

Процедура Notify

Профиль:

```
void Notify ();
```

Данная процедура выводит окно с текстом сообщения об ошибке, составленного из параметров, заданных в конструкторе XException.

Дополнительная литература

[1] Волкова И.А. Адаптация и обучение системы общения с ЭВМ на естественном языке. Канд. диссертация. М. 1982. — http://axofiber.org.ru/rm/volkova-dissertation.pdf
[2] Зализняк А.А. Грамматический словарь русского языка. — М., Русский язык, 1980.
[3] Страуструп Б., Язык C++, специальное издание. — М., Бином, 2001.
[4] Seward J., Bzip2 and libbzip2. — http://sources.redhat.com/bzip2/
[5] Stig Sæther Bakken, Egon Schmid и другие, PHP Manual. — http://www.php.net/manual/
[6] W3C, XML. — http://www.w3.org/...

Приложение А. Типовая структура файлов данных

В данном приложении рассматривается структура файлов данных, в которых морфологический анализатор перзистентно хранит свои грамматические словари во внешней памяти.

Представление основ

Класс основ (CRMStem).

```
<!ELEMENT stem       - 0 EMPTY >
<!ATTLIST stem
              UID           CDATA                       #REQUIRED
              lexemID[]     CDATA                       #REQUIRED
>
```

Представление лексем

Базовый класс для лексем (CRMLexem).

```
<!ENTITY %lexem
              uid           CDATA                       #REQUIRED
              paradigmID    CDATA                       #REQUIRED
              semclassID    CDATA                       #IMPLIED
              stress        NUMBER                      #REQUIRED
              auxstress     NUMBER                      #IMPLIED
>
```

Описание элемента «Личное местоимение» (CRMPersonalPronounForm).

```
<!ELEMENT personalpronoun          - 0 EMPTY >
<!ATTLIST personalpronoun          (%lexem;)
              animate       (yes|no|any)                #REQUIRED
              gender        (male|female|neuter|any)    #REQUIRED
              number        (single|plural)             #REQUIRED
              person        (1|2|3)                     #REQUIRED
              case          CDATA                       #REQUIRED
>
```

Описание элемента «Возвратное местоимение» (CRMReflexivePronoun).

```
<!ELEMENT reflexivepronoun         - 0 EMPTY >
<!ATTLIST reflexivepronoun         (%lexem;)
              animate       (yes|no|any)                #REQUIRED
              gender        (male|female|neuter|any)    #REQUIRED
              number        (single|plural)             #REQUIRED
              case          CDATA                       #REQUIRED
>
```

Описание элемента «Местоимение» (CRMPronoun).

```
<!ELEMENT pronoun           - 0 EMPTY >
<!ATTLIST pronoun           (%lexem;)
              animate       (yes|no|any)              #REQUIRED
              gender        (male|female|neuter|any)  #REQUIRED
              number        (single|plural)           #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Числительное один» (CRMNumberOne)

```
<!ELEMENT numberone         - 0 EMPTY >
<!ATTLIST numberone         (%lexem;)
              animate       (yes)                     #REQUIRED
              gender        (male|female|neuter|any)  #REQUIRED
              number        (single|plural)           #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Числительное два» (CRMNumberTwo)

```
<!ELEMENT numbertwo         - 0 EMPTY >
<!ATTLIST numbertwo         (%lexem;)
              animate       (yes|no|any)              #REQUIRED
              gender        (male|female|neuter|any)  #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Числительное три» (CRMNumberThree)

```
<!ELEMENT numberthree       - 0 EMPTY >
<!ATTLIST numberthree       (%lexem;)
              animate       (yes|no|any)              #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Количественное числительное» (CRMNumber)

```
<!ELEMENT number            - 0 EMPTY >
<!ATTLIST number            (%lexem;)
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Числительное с двумя формами» (CRMNumberBiform)

```
<!ELEMENT numberbiform      - 0 EMPTY >
<!ATTLIST numberbiform      (%lexem;)
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Существительное неодушевленное мужского рода» (CRMNounMaleInanimate)

```
<!ELEMENT noun_m_i          - 0 EMPTY >
<!ATTLIST noun_m_i          (%lexem;)
              animate       (yes|no|any)              #REQUIRED
```



```
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное одушевленное мужского рода» (CRMNounMaleAnimate)

```
<!ELEMENT noun_m_a          - 0 EMPTY >
<!ATTLIST noun_m_a          (%lexem;)
            animate     (yes|no|any)            #REQUIRED
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_f_i          - 0 EMPTY >
<!ATTLIST noun_f_i          (%lexem;)
            animate     (yes|no|any)            #REQUIRED
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_f_a          - 0 EMPTY >
<!ATTLIST noun_f_a          (%lexem;)
            animate     (yes|no|any)            #REQUIRED
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_n_i          - 0 EMPTY >
<!ATTLIST noun_n_i          (%lexem;)
            animate     (yes|no|any)            #REQUIRED
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_n_a          - 0 EMPTY >
<!ATTLIST noun_n_a          (%lexem;)
            animate     (yes|no|any)            #REQUIRED
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_mf_a         - 0 EMPTY >
<!ATTLIST noun_mf_a         (%lexem;)
            animate     (yes|no|any)            #REQUIRED
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_m            - 0 EMPTY >
<!ATTLIST noun_m            (%lexem;)
            animate     (yes|no|any)            #REQUIRED
            case        CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_m_f_i        - 0 EMPTY >
<!ATTLIST noun_m_f_i        (%lexem;)
              animate       (yes|no|any)            #REQUIRED
              case          CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_m_f_a        - 0 EMPTY >
<!ATTLIST noun_m_f_a        (%lexem;)
              animate       (yes|no|any)            #REQUIRED
              case          CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_f            - 0 EMPTY >
<!ATTLIST noun_f            (%lexem;)
              animate       (yes|no|any)            #REQUIRED
              case          CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_n_f_i        - 0 EMPTY >
<!ATTLIST noun_n_f_i        (%lexem;)
              animate       (yes|no|any)            #REQUIRED
              case          CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_n            - 0 EMPTY >
<!ATTLIST noun_n            (%lexem;)
              animate       (yes|no|any)            #REQUIRED
              case          CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_n_m_a        - 0 EMPTY >
<!ATTLIST noun_n_m_a        (%lexem;)
              animate       (yes|no|any)            #REQUIRED
              case          CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_n_c_i        - 0 EMPTY >
<!ATTLIST noun_n_c_i        (%lexem;)
              animate       (yes|no|any)            #REQUIRED
              case          CDATA                   #REQUIRED
>
```

Описание элемента «Существительное » ()

```
<!ELEMENT noun_n_m_i        - 0 EMPTY >
```



```
<!ATTLIST noun_n_m_i        (%lexem;)
              animate       (yes|no|any)              #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Прилагательное» (CRMAdjective)

```
<!ELEMENT adjective         - 0 EMPTY >
<!ATTLIST adjective         (%lexem;)
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Местоимённое прилагательное» (CRMPronounAdjective)

```
<!ELEMENT pronounadjective          - 0 EMPTY >
<!ATTLIST pronounadjective          (%lexem;)
              animate       (any)                     #REQUIRED
              gender        (male|female|neuter|any)  #REQUIRED
              number        (single|plural)           #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Порядковое числительное» (CRMNumberOrdinal)

```
<!ELEMENT numberordinal     - 0 EMPTY >
<!ATTLIST numberordinal     (%lexem;)
              animate       (any)                     #REQUIRED
              gender        (male|female|neuter|any)  #REQUIRED
              number        (single|plural)           #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Притяжательное прилагательное» (CRMPossesiveAdjective)

```
<!ELEMENT possesiveadjective        - 0 EMPTY >
<!ATTLIST possesiveadjective        (%lexem;)
              animate       (any)                     #REQUIRED
              gender        (male|female|neuter|any)  #REQUIRED
              number        (single|plural)           #REQUIRED
              case          CDATA                     #REQUIRED
>
```

Описание элемента «Глагол невозвратный непереходный несовершенного вида (не многократный и не безличный)» (CRMVerbIrreflexiveIntransitiveImperfective)

```
<!ELEMENT v_i_i_i           - 0 EMPTY >
<!ATTLIST v_i_i_i           (%lexem;)
              reflexive     (no)                      #REQUIRED
              transitive    (no)                      #REQUIRED
              perfective    (no)                      #REQUIRED
              mode          (participle               #IMPLIED
              ...
>

v_i_i_p |
v_i_t_i | v_i_t_p | v_r_i | v_r_p | v_f_i_i_i |
v_f_i_t_i | v_i_i_i_i | v_i_i_i_p | v_i_i_t_i |
```

```
v_i_i_t_p | v_i_r_i_i | v_i_r_i_p | v_i_i_b |
v_i_t_b | v_r_i_b | v_r_i_b_e | v_b_i_i |
```

Описание элемента «Предлог» (CRMPreposition)

```
<!ELEMENT preposition       - 0 EMPTY >
<!ATTLIST preposition       (%lexem;)
```

Описание элемента «Союз» (CRMConjunction)

```
<!ELEMENT conjunction       - 0 EMPTY >
<!ATTLIST conjunction       (%lexem;)
```

Описание элемента «Наречие» (CRMAdverb)

```
<!ELEMENT adverb            - 0 EMPTY >
<!ATTLIST adverb            (%lexem;)
```

Описание элемента «Предикатив» (CRMPredicative)

```
<!ELEMENT predicative       - 0 EMPTY >
<!ATTLIST predicative       (%lexem;)
```

Описание элемента «Вводное слово» (CRMParenthesis)

```
<!ELEMENT parenthesis       - 0 EMPTY >
<!ATTLIST parenthesis       (%lexem;)
```

Представление парадигматических классов

Класс парадигм (CRMParadigm), поддерживающий вариативность флексий и вложенность парадигматических классов.

```
<!ELEMENT flex              - 0 EMPTY >
<!ATTLIST flex
            ID              CDATA                   #REQUIRED
            nestedID        CDATA                   #IMPLIED
>

<!ELEMENT f                 - - ( flex )* >

<!ELEMENT paradigm          - - ( f )* >
<!ATTLIST paradigm
            uid             CDATA                   #REQUIRED
>
```

Представление схем ударений

Класс схем ударений (CRMStressSchema), поддерживающий вариативность и многопозиционность ударных букв.

```
<!-- Атрибут aux представляет возможную вторую ударную букву -->
<!ELEMENT pos               - 0 EMPTY >
<!ATTLIST pos
            id              NUMBER                  #REQUIRED
```



```
            aux         NUMBER                  #IMPLIED
>

<!ELEMENT nested        - 0 EMPTY >
<!ATTLIST nested
            id          IDREF                   #REQUIRED
>

<!ELEMENT flex          - - ( pos )* | nested >

<!-- Смысл аналогичен элементу f в представлении парадигмы -->
<!ELEMENT f             - - ( flex )* >

<!ELEMENT schema        - - ( f )* >
<!ATTLIST schema
            uid         CDATA                   #REQUIRED
>
```

Запрос к модулю морфологии

В данном пункте описано DTD запросов к модулю морфологии, посылаемых либо через прямой вызов функции ProcessQuery, либо через сетевое соединение на базе сетевых разъёмов (→ Часть I.3.2 Программный интерфейс, стр. 14). Возможные запросы к модулю морфологии описываются следующими элементами.

Элемент tellinfo предназначен для получения краткой информации о текущей версии модуля.

```
<!ELEMENT tellinfo      - 0  ( EMPTY ) >

<!ELEMENT analyze       - 0  ( EMPTY ) >

<!ATTLIST analyze
            form        CDATA                   #REQUIRED
            match       (spell)                 #IMPLIED
>

<!ELEMENT synthesize    - 0  ( EMPTY ) >
<!ATTLIST synthesize
            form        CDATA                   #REQUIRED
            match       (spell)                 #IMPLIED
>
```

Ответ модуля морфологии

Результат обработки запроса представляется элементом answer. Элемент unrecognized описывает ситуацию, возникающую при подаче некорректного запроса к модулю морфологии. Элемент notfound означает отрицательный результат анализа или синтеза (форма не найдена в словаре).

```
<!ELEMENT unrecognized  - 0  ( EMPTY ) >
```

```
<!ELEMENT notfound          - 0  ( EMPTY ) >

<!ELEMENT info              - 0  ( EMPTY ) >
<!ATTLIST info
              version       CDATA                       #IMPLIED
              build         CDATA                       #IMPLIED
>

<!ELEMENT analyzed          - 0  ( personalpronoun | reflexivepronoun |
                                   pronoun | numberone | numbertwo | numberthree |
                                   number | numberbiform | noun_m_i | noun_m_a |
                                   noun_f_i | noun_f_a | noun_n_i | noun_n_a |
                                   noun_mf_a | noun_m | noun_m_f_i | noun_m_f_a |
                                   noun_f | noun_n_f_i | noun_n | noun_n_m_a |
                                   noun_n_c_i | noun_n_m_i | adjective |
                                   pronounadjective | numberordinal |
                                   possesiveadjective | v_i_i_i | v_i_i_p |
                                   v_i_t_i | v_i_t_p | v_r_i | v_r_p | v_f_i_i_i |
                                   v_f_i_t_i | v_i_i_i_i | v_i_i_i_p | v_i_i_t_i |
                                   v_i_i_t_p | v_i_r_i_i | v_i_r_i_p | v_i_i_b |
                                   v_i_t_b | v_r_i_b | v_r_i_b_e | v_b_i_i |
                                   preposition | conjunction | adverb |
                                   predicative | parenthesis )* >
<!ATTLIST analyzed
              form          CDATA                       #REQUIRED
              lexem         CDATA                       #REQUIRED
>

<!ELEMENT form              - 0    ( personalpronoun | reflexivepronoun |
                                   pronoun | numberone | numbertwo | numberthree |
                                   number | numberbiform | noun_m_i | noun_m_a |
                                   noun_f_i | noun_f_a | noun_n_i | noun_n_a |
                                   noun_mf_a | noun_m | noun_m_f_i | noun_m_f_a |
                                   noun_f | noun_n_f_i | noun_n | noun_n_m_a |
                                   noun_n_c_i | noun_n_m_i | adjective |
                                   pronounadjective | numberordinal |
                                   possesiveadjective | v_i_i_i | v_i_i_p |
                                   v_i_t_i | v_i_t_p | v_r_i | v_r_p | v_f_i_i_i |
                                   v_f_i_t_i | v_i_i_i_i | v_i_i_i_p | v_i_i_t_i |
                                   v_i_i_t_p | v_i_r_i_i | v_i_r_i_p | v_i_i_b |
                                   v_i_t_b | v_r_i_b | v_r_i_b_e | v_b_i_i |
                                   preposition | conjunction | adverb |
                                   predicative | parenthesis )* >
<!ATTLIST form
              value         CDATA                       #REQUIRED
>

<!ELEMENT synthesized       - 0    ( form )* >
<!ATTLIST synthesized
              lexem         CDATA                       #REQUIRED
>
```



```
<!ELEMENT answer            - -  unrecognized | notfound | info |
                                   ( analyzed )* | ( synthesized )* >
```