Инструментальные средства для разработки систем извлечения информации из русскоязычных текстов

Большакова Е.И., Ефремова Н.Э., Шариков Г.Ф.
МГУ имени М.В. Ломоносова, факультет ВМК
eibolshakova@gmail.com, nvasil@list.ru, egos_@mail.ru

Аннотация. В работе представлен краткий обзор и сравнительный анализ языковых и программных средств пяти инструментальных систем, применяемых для построения приложений для извлечения информации из текстов на русском языке: систем *GATE* и *RCO Pattern Extractor,* программного комплекса *LSPL*, процессора *DSTL*, *Томита-парсера*. По результатам проведенного анализа намечаются наиболее важные направления развития языка *LSPL* и поддерживающих его программных средств.

Ключевые слова: извлечение информации из текстов, распознавание языковых выражений, шаблоны языковых конструкций, лингвистические правила, язык *LSPL*

1 Введение

К одной из наиболее актуальных прикладных задач в области компьютерной лингвистики относится извлечение информации из текстов на естественном языке (ЕЯ) – *Information Extraction* (*IE*). В рамках этой задачи реализуется автоматическое выделение в текстах именованных сущностей (персоналий, адресов, названий организаций и др.), их свойств и связанных с ними событий [Grishman, 2003]. Нередко к этой задаче относят также выявление терминов и их отношений из коллекций текстов определенной предметной области.

Разработка любых ЕЯ-приложений, в том числе по извлечению информации, является сложным и трудоемким процессом, существенную помощь в котором оказывает соответствующий инструментарий. Он включает средства представления лингвистической информации, средства отладки модулей разрабатываемого приложения, а также возможности расширения уже готового набора стандартных компонентов.

Наиболее известной из инструментальных систем, применяемых для разработки *IE*-приложений, является система *GATE* [GATE]. Однако используемый в ней внутренний язык *Jape* не имеет встроенных средств задания лингвистической информации, что требует определенной настройки системы на язык анализируемых текстов. Отметим, что для высокофлективных языков, таких как русский или немецкий, эта настройка весьма значительна. Для русского языка она была реализована при создании коммерческого продукта *ONTOS* [Хорошевский, 2004].

Для эффективной разработки приложений, работающих с текстами на русском языке, был создан ряд отечественных инструментальных систем:

RCO Pattern Extractor [Ермаков, 2003], система *Alex* [Жигалов, 2002], программный комплекс для языка *LSPL* [Большакова, 2007, 2010]. Появление в последние годы новых систем – процессора языка *DSTL* [Скатов, 2010] и системы [Томита-парсер] свидетельствует о том, что существующий инструментарий для разработки *IE*-приложений для русскоязычных текстов еще имеет потенциал развития.

Каждая из указанных выше инструментальных систем имеет свои особенности и ограничения, при этом общим является использование формального языка для описания лингвистических свойств распознаваемых и извлекаемых из текста языковых конструкций. Описание свойств осуществляется в форме специальных **шаблонов** и **правил**, с помощью которых уже готовые программные модули систем настраиваются для решения требуемой задачи. Распознавание и извлечение конструкций реализуется на базе частичного синтаксического анализа (*shallow analysis*).

В данной работе представлен краткий обзор и сравнительный анализ существующих в системах разработки *IE*-приложений инструментальных средств, к которым мы относим языки записи лингвистических шаблонов и правил, а также средства поддержки их разработки и включения в систему новых компонентов. В обзор включены инструментальные системы, которые вплоть до настоящего времени поддерживаются и применяются для обработки русскоязычных текстов: *GATE*, *RCO Pattern Extractor*, *LSPL*, *DSTL*, *Томита-парсер* (*GATE*, комплекс *LSPL* и *Томита-парсер* относятся к свободному ПО). По результатам проведенного анализа формулируются наиболее важные направления развития языка *LSPL* и поддерживающих его программных средств, повышающие их гибкость и мощность.

2 Характеристика инструментальных систем

2.1 Система *GATE*

Система *GATE* создавалась как языково-независимая среда разработки различных приложений по автоматической обработке ЕЯ-текстов [Bontcheva et al., 2003]. Основные ее архитектурные особенности:

- компонентный подход, позволяющий выполнять сборку конкретного ЕЯ-приложения из готовых программных модулей анализа текста (токенизация, морфологическое тегирование, сегментация, словарный модуль – *Gazetteer*, выделение именованных сущностей и др.);
- представление информации о промежуточных результатах анализа текста в форме **аннотаций**, приписываемых непрерывным текстовым фрагментам, но хранимых отдельно от самого обрабатываемого текста;
- выполнение анализа текста как последовательного создания и переработки его аннотаций (т.е. аннотирования), осуществляемого модулями системы, согласно записанным в них правилам.

Для записи правил в системе *GATE* используется язык *Jape* [Cunningham et al., 2000]. Правило на языке *Jape* состоит из двух частей: **левая часть** задает **шаблон**, определяющий некоторый фрагмент текста (языковую конструкцию) по его аннотациям, а **правая часть** описывает действия с аннотациями найденного фрагмента.

Аннотации имеют тип и представляют собой набор атрибутов, которые могут иметь определенные значения. В общем случае в шаблоне левой части задаются условия-ограничения на типы и значения атрибутов аннотаций. К примеру, простой шаблон для описания числа: *{Token.kind==number}* содержит указание типа аннотации (*Token*), ее атрибута (*kind*) и его значения (*number*). В шаблоне можно использовать также операторы регулярных выражений (*, +, ?). Порядок следования аннотаций соответствует порядку элементов в описываемой конструкции.

В правой части правил обычно записывают новые аннотации, присваиваемые всему выделенному фрагменту или его части, с указанием их конкретных атрибутов. Если аннотации должны приписываться только к части фрагмента, в шаблоне используются метки (они записываются после двоеточия).

Следующее *Jape*-правило служит для выявления в тексте конструкций вида «Иван родился в Самаре», т.е. определения места (города) рождения:

Rule: BornPlace
(({Token.kind == word, Token.orth == upperInitial}): person
{Token.string == "родился"} {Token.string == "в"}
({Lookup.majorType == "City"}): city)
--> person.Name ={BirthCity = city.Token.string}

Левая часть этого правила описывает фрагмент текста, первое слово которого – имя, начинающееся с заглавной буквы (*Token.kind == word, Token.orth == upperInitial*), второе и третье – конкретные словоформы «родился» и «в», а четвертое – название города, описанное в словаре названий городов (аннотацией *Lookup* помечаются слова, найденные в словаре). Правая часть правила создает новую аннотацию *Name* с атрибутом *BirthCity*, значением которого берется *Token.string* (строка-название города). Метки *person* и *city* ссылаются на части выявленной конструкции.

Важная особенность языка *Jape* – возможность ввести новые типы и атрибуты аннотаций, поскольку в нем отсутствуют встроенные типы и атрибуты, в том числе лингвистические (лексические, морфологические, синтаксические): предполагается, что нужные типы и атрибуты определяются при разработке конкретного приложения. Подобная гибкость и универсальность имеет и отрицательные стороны. Хотя архитектура *GATE* предполагает ее применимость для обработки текстов на любом ЕЯ, в реальности для флективных языков, в том числе для русского, средств языка *Jape* явно недостаточно. К примеру, элемент

{Token.string == "родился"} из рассмотренного выше правила учитывает лишь одну из возможных в рассматриваемом контексте словоформ; чтобы учесть другие словоформы, их необходимо явно указать в правиле.

Еще один недостаток языка – невозможность сравнить значения атрибутов различных аннотаций, что существенно затрудняет запись условия грамматического согласования слов описываемой конструкции (в частности, именного словосочетания). Кроме того, в языке отсутствует встроенная возможность извлечения распознанной в тексте конструкции или ее части: для этого требуется программирование модуля, отбирающего из текста фрагменты с определенными аннотациями.

2.2 *RCO Pattern Extractor*

Тем не менее успешное применение системы *GATE* для построения различных *IE*-приложений для текстов на разных ЕЯ способствовало ее широкому распространению и попыткам построения новых инструментальных средств, сохраняющих определенные принципы этой системы и преодолевающих ряд ее ограничений и недостатков.

Отечественная коммерческая система *RCO Pattern Extractor* [Ермаков, 2003], в отличие от *GATE*, изначально разрабатывалась для задач извлечения информации из текстов на русском языке. Эта система реализована с учетом ряда архитектурных принципов *GATE* и использует для описания распознаваемых в тексте конструкций язык, подобный *Jape*. Фактически язык *Jape* был расширен встроенными средствами задания морфологических характеристик слов выделяемых языковых конструкций (около 20 предопределенных атрибутов для аннотаций *Token* и *Morph*). В шаблонах *RCO* можно также использовать разные виды сравнения на равенство: тождественное равенство (==), совпадение с точностью до регистра букв (=^), равенство с учетом морфологии (=|). Последнее позволяет описывать конструкции со словами в произвольной форме. Так, элемент шаблона *{Token.text =| "родиться"}* служит для выделения всех словоформ глагола «родиться».

В тоже время язык шаблонов системы *RCO* сохраняет некоторые недостатки исходного языка *Jape*, в частности, невозможность прямого задания в них условия грамматического согласования. Кроме того, в отличие от *GATE*, состав программных модулей и последовательность этапов анализа в *RCO* фиксирована: токенизация (графематический анализ), морфологический анализ, лексический анализ (распознавание словарных единиц), выделение объектов текста по шаблонам и правилам.

2.3 *LSPL* и программные средства его поддержки

Язык *LSPL* [Большакова, 2007; 2010] разрабатывался как средство декларативного и компактного описания конструкций русского языка на принципах, отличных от языка *Jape*.

Распознаваемая конструкция специфицируется в виде **лексико-синтаксического шаблона**, который представляет собой ее структурный **образец**, дополненный морфологическими характеристиками входящих в нее слов и словосочетаний и условиями их грамматического согласования.

В процессе развития языка он был расширен рядом средств, и в настоящий момент *LSPL*-шаблон имеет следующий вид:

шаблон_распознавания =*text*> шаблон_извлечения_текста
или шаблон_распознавания =*pattern*> синтезируемый_шаблон .

LSPL-шаблон является по сути правилом преобразования конструкции, найденной по **шаблону распознавания**, в извлекаемый из нее текст или же в генерируемый новый шаблон.

В шаблоне распознавания указывается последовательность элементов-слов, для каждого из которых могут быть конкретизированы лексема и морфологические характеристики (часть речи, падеж, род, число и т.п.) и заданы условия их грамматического согласования. Например, шаблон *N V<t=pres> Av <N=V>* описывает конструкцию, состоящую из существительного (*N*), следующего за ним глагола (*V*) в прошедшем времени (*t=pres*) и наречия (*Av*), причем существительное и глагол должны быть грамматически согласованы. В шаблоне можно задавать повторения элементов, опциональные элементы, альтернативы – для этого используются известные метасимволы (*{, }, [,],* |), эквивалентные операторам регулярных выражений.

Встроенные средства *LSPL* позволяют определять имя шаблона и его параметры, к примеру, шаблон *NP = {A} N1 <A=N1> [N2<c=gen>] (N1)* задает именную группу *NP* из прилагательных (*A*), согласованного с ними существительного (*N1*) и опционального существительного (*N2*) в родительном падеже (*c=gen*), параметрами этого шаблона установлены морфологические характеристики первого существительного. Уже определенные шаблоны можно применять для задания шаблонов более сложных ЕЯ-выражений, используя при этом их параметры для конкретизации или согласования элементов описываемой конструкции. Например, шаблон *NP* можно применить для задания частных случаев именной группы, когда первое ее существительное употребляется во множественном числе: *NP<n=plur>*.

Таким образом, набор взаимосвязанных шаблонов фактически задает КС-грамматику, расширенную условиями, для выявляемой в тексте языковой конструкции.

Шаблон извлечения текста в правой части *LSPL*-правила позволяет выделить фрагменты распознанной конструкции и сформировать из них нужную текстовую строку. К примеру, для извлечения места (города) рождения из конструкций вида «Иван родился в Самаре» или «Нина родилась в Москве» служит *LSPL*-правило:

BornPlace = N V<родиться> "в" City =text> #City

Заметим, что *N* здесь задает любое существительное без указания того, что оно должно начинаться с большой буквы, т.к. в *LSPL* нет встроенных средств проверки регистра букв (и других неграмматических свойств слов текста). Шаблон с именем *City* задает перечень всех названий городов. Символ # обозначает операцию *нормализации* извлекаемого из конструкции элемента (*City*), т.е. приведение его к словарной форме.

Если же в правиле *BornPlace* вместо шаблона извлечения текста *=text> #City* написать *=pattern> A N <$City.b> <A=N>*, то результатом применения правила к фразе «Нина родилась в Москве» станет построение шаблона *A N <Москва> <A=N>*, с помощью которого можно затем искать в тексте все согласованные прилагательные-эпитеты слова «Москва».

Программные средства, реализованные в поддержку языка *LSPL*, включают компонент распознавания в тексте конструкций по их шаблонам. Как и в системе *RCO Pattern Extractor*, последовательность этапов обработки текста жестко фиксирована: графематический анализ, морфологический анализ, поверхностно-синтаксический анализ по заданным шаблонам (лексический анализ не выделен в отдельный этап и проводится одновременно с синтаксическим). Для специалистов, участвующих в создании и тестировании *LSPL*-шаблонов, был реализован также пользовательский интерфейс для визуализации результатов анализа текстов по шаблонам.

2.4 *DSTL* и *DictaScope Tokenizer*

Язык *DSTL* [Скатов, 2010] является инструментальным языком коммерческой *IE*-системы *DictaScope Tokenizer*.

Правило *DSTL* состоит из трех секций. Первая секция (с именем *T*), содержит шаблон распознаваемой конструкции, который в общем случае представляет собой регулярное выражение относительно элементов этой конструкции. Свойства элементов этого шаблона указываются в секции критериев (с именем *C*) – там записывается логическое выражение, составленное из доступных в *DSTL* логических, арифметических и строковых функций. Действие правила (секция *A*) выполняется, если критерий оказался истинным. Приведем пример правила с именем *BornYear* для распознавания конструкций вида «Петр родился в 1986 г.» и извлечения из них года рождения:

```
Year {  T := X "родился" "в" Y "г." ?;
        C := HasGrammarForm( X, {Subtype: Name}) & IsNumeric(Y) &
             Length(Y) = 4;
        A := { year := Y; }; };
```

Шаблон *T* задает последовательность элементов распознаваемой конструкции (оператор регулярных выражений *?* означает, что элемент может отсутствовать). В секции *C* указано, что элемент *X* представляет собой имя, а *Y* является числом и состоит из четырех символов. В секции *A*

задано действие-извлечение: в поле *year* заносится строковая запись распознанного года.

Заметим, что по сравнению с *Jape* шаблоны языка *DSTL* (как и шаблоны *LSPL*) менее громоздки. Подобно *LSPL*, правила *DSTL* задают КС-грамматику, расширенную условиями и действиями по извлечению фрагментов конструкции, причем все проверяемые условия сгруппированы в одной секции – это позволяет сделать более наглядной структуру распознаваемой конструкции. Явным достоинством *DSTL* является также широкий набор встроенных функций для раздела критериев.

2.5 Томита-парсер

Система *Томита-парсер* [Томита-парсер] изначально создавалась для извлечения фактов из текстов на русском языке. Для задания распознаваемых в тексте конструкций используются правила, записанные на языке расширенных КС-грамматик. Аналогично языкам *LSPL* и *DSTL*, в правилах парсера кроме условий задаются также действия по извлечению (**интерпретации**) распознанных атрибутов факта. Разбор по грамматикам осуществляется с помощью эффективного *GLR*-алгоритма [Tomita, 1984]. Последовательность этапов обработки текста фиксирована: токенизация, морфологический анализ, лексический анализ (распознавание ключевых слов и словосочетаний по словарю), синтаксический разбор по заданной грамматике, интерпретация.

Правила Томита-парсера имеют вид S –> $S_1...S_n$ {Q};. В левой части (до знака –>) указывается один нетерминальный символ S, в правой – список описаний терминальных и нетерминальных символов $S_1...S_n$, за которым в общем случае записываются условия Q, применяемые ко всему правилу в целом. В свою очередь, описания S_i состоят из трех частей: $N<P_1,...,P_n>$ $interp(I_1;...;I_n)$, где N – имя терминала или нетерминала, P_i – пометы-ограничения на свойства N, а I_i – имена **полей** (атрибутов) фактов, куда при интерпретации записывается фрагмент выявленной конструкции. Пометы могут задавать как ограничения на регистр букв, так и грамматические ограничения (*gram*). К примеру, правило *Born –> "родиться"<gram="praet,sg">* описывает возможные словоформы глагола «родиться» в прошедшем времени (*praet*) и единственном числе (*sg*).

Приведем пример грамматики для распознавания в тексте фактов вида «Иван родился в Самаре» и извлечения информации о городе рождения:

Person –> AnyWord <gram="имя">;
Born –> "родиться" <gram="praet, sg">;
City –> Noun<kwtype=city>;
S –> Person<gn–agr[1]> interp(BornFact.Person) Born<gn–agr[1]>
"в" City interp(BornFact.City);

Первое правило служит для распознавания имени (любая словоформа текста с признаком "*имя*"), второе – для выявления глагола, указывающего

на факт рождения, а третье – для распознавания названия города (существительное *Noun*, тип которого равен *city)*. Последнее правило грамматики описывает всю распознаваемую конструкцию-факт: помета *<gn–agr[1]>* указывает на необходимость согласования *Person* и *Born* в роде и числе, а *interp (BornFact.Person)* и *interp (BornFact.City)* задают извлечение распознанных *Person* и *City* как атрибутов (полей) факта (по умолчанию они нормализуются).

Как и язык *LSPL*, грамматика Томита-парсера является достаточно гибким средством описания лексических и грамматических свойств распознаваемых конструкций русского языка, в то же время превосходя *LSPL* по мощности выразительных средств. Томита-парсер предоставляет богатый набор помет-ограничений, но этот набор не расширяем. От других инструментальных систем парсер отличается наличием встроенных средств описания структуры (полей) извлекаемых фактов – это делает его одним из самых удобных инструментов для построения *IE*-приложений для извлечения именно фактов и событий. Для выявления более простых конструкций средства Томита-парсера явно избыточны.

3 Сравнение возможностей инструментальных систем

В Таблице 1 представлены основные возможности рассмотренных инструментальных систем, существенные с точки зрения мощности языковых средств для описания конструкций русского языка, а также эффективности и удобства разработки *IE*-приложений (знак «+» означает, что возможность присутствует, а знак «–» означает ее отсутствие).

Видно, что рассмотренные языки шаблонов и правил сопоставимы по возможностям. С точки зрения встроенных языковых средств несколько уступает система *GATE*, однако она допускает естественное расширение – добавление любых новых характеристик слов и конструкций (типов и атрибутов аннотаций), а также программных модулей и словарей.

Отдельная оценка языков шаблонов и правил требуется с точки зрения разработки шаблонов. Обычно шаблоны и правила подготавливаются лингвистом или специалистом по предметной области анализируемых текстов, без участия программистов. В этом аспекте проигрывают язык *Jape* и ему подобные – они в большей степени похожи на языки программирования, их использование требует определенной квалификации и опыта, а записанные на их основе шаблоны языковых конструкций громоздки и плохо читабельны. Лингвистические шаблоны и правила языков *LSPL*, *DSTL* и Томита-парсера более наглядны. Сравним, к примеру, описание согласованной именной группы из прилагательного и существительного для Томита-парсера и *LSPL*:

Томита-парсер: *NP –> Adj<gnc—agr[1]> Noun<gnc—agr[1]>;*
Язык *LSPL*: *NP –> A N <A=N>*

Последний шаблон лаконичнее (язык *LSPL* особенно удобен для описания различного вида именных групп), да и порог вхождения в этот язык существенно ниже.

LSPL – единственный язык, в который встроена возможность синтеза нового шаблона. Однако в нем затруднено и до сих пор не применяется подключение сторонних словарей, вся словарная информация задается в шаблонах, что порой неудобно. В тоже время возможно подключение новых модулей в открытую программную библиотеку *LSPL* (для сравнения: в *RCO Pattern Extractor* подключение модулей реализуется только через программный интерфейс системы).

Таблица 1. Средства инструментальных систем для разработки *IE*-приложений

	GATE	*RCO*	*LSPL*	*DSTL*	*Томита*
Встроенные средства языков шаблонов и правил					
Морфологические характеристики	–	+	+	+	+
Грамматическое согласование	–	–	+	+	+
Неграмматические характеристики слов	–	+	–	+	+
Введение новых характеристик	+	+	–	–	–
Извлечение распознанных конструкций	–	–	+	+	+
Синтез новых шаблонов	–	–	+	–	–
Поддержка разработки приложений					
Подключение словарей	+	+	–	–	+
Включение новых модулей	+	+	+	–	–
Визуальный интерфейс пользователя-разработчика	+	+	+	–	–

Построение *IE*-приложений предполагает отладку лингвистических шаблонов, поэтому важна визуальная среда анализа текстов по шаблонам, которая поддерживает процесс составления шаблонов и правил, позволяя просматривать и анализировать результаты распознавания и извлечения конструкций. Такая среда пользователя создана в большинстве инструментальных систем, однако в Томита-парсере отсутствует полноценный графический пользовательский интерфейс, хотя просмотр результатов извлечения возможен в текстовом или HTML-файле.

4 Направления развития языка *LSPL*

Успешное применение языка *LSPL* для разработки ряда ЕЯ-приложений, наиболее крупное из которых – комплекс процедур для автоматического терминологического анализа научно-технических текстов [Ефремова, 2010] позволило выявить его "проблемные точки", что вкупе с проведенным сравнительным анализом инструментальных средств других систем дало возможность наметить пути дальнейшего развития языка с целью расширить его новыми средствами, упрощающими построение разных приложений.

Безусловно необходимым является доработка языковых средств *LSPL* для работы с подключаемыми словарями. Полезным расширением языка будет введение логических операций отрицания (*!*) и дизъюнкции, применяемых к условиям согласования и конкретизации морфологических характеристик – это позволит записать, к примеру, шаблон *V<!(t=past)>*, соответствующий глаголу в любом времени, кроме прошедшего.

Более существенным расширением является введение операции-связки ~, обозначающей произвольный порядок вхождения элементов шаблона в описываемую конструкцию – это даст возможность компактно описать такие конструкции русского языка (в котором относительно свободный порядок слов), как глагол и его дополнение, которое может стоять как до, так и после глагола. Так, запись $N \sim V$ обозначает существительное и глагол, идущие в любом порядке друг за другом.

Еще одно перспективное направление развития языка связано с многоуровневостью ЕЯ-текста и процесса его анализа. К его основным уровням относятся уровни графематики, морфологии, синтаксиса и семантики. Встроенные средства языка *LSPL* в основном относятся к морфологии, уровни графематики и синтаксиса представлены далеко не полно. Добавление новых выразительных средств, относящихся к этим уровням, в частности, графематических признаков (регистр букв и др.), шаблонов знаков препинания, шаблонов разрывных синтаксических групп и др. безусловно повысит мощность языка, расширив сферу его применимости для построения ЕЯ-приложений.

Введение в язык средств, относящихся к разным уровням анализа текста, влечет за собой и изменение принципов его реализации. Логично разделить выразительные средства языка на

- ядро, определяющее абстрактный синтаксис и семантику лингвистических шаблонов (основные операции над шаблонами);
- библиотеку стандартных шаблонов, относящихся к конкретным уровням анализа текста (в частности, туда войдут шаблоны слов конкретных частей речи *A*, *N* и др., шаблоны графематики и т.д.).

Указанный принцип делает язык более универсальным, позволяя расширять его новыми шаблонами путем добавления их в библиотеку (не изменяя ядра), а также давая возможность поддерживать разные реализации библиотечных шаблонов.

5 Заключение

В работе охарактеризованы пять инструментальных систем, используемых для построения приложений по автоматическому извлечению информации из текстов на русском языке. Проведенное сравнение их основных возможностей позволило обозначить направления развития языка *LSPL*, которые призваны улучшить его выразительные средства и расширить область его применения.

6 Литература

[Большакова, 2007] Большакова Е.И., Баева Н.В., Бордаченкова Е.А., Васильева Н.Э., Морозов С.С. Лексико-синтаксические шаблоны в задачах автоматической обработки текстов // Компьютерная лингвистика и интеллектуальные технологии: Труды Межд. конференции Диалог ‘2007. – М.: Издательский центр РГГУ, 2007, с.70-75.

[Большакова, 2010] Большакова Е.И., Носков А.А. Программные средства анализа текста на основе лексико-синтаксических шаблонов языка LSPL // Программные системы и инструменты: Тематический сборник, № 11 / Под ред. Королева Л.Н. – М.: МАКС Пресс, 2010, с. 61-73.

[Ермаков, 2003] Ермаков А.Е. и др. RCO Pattern Extractor: компонент выделения особых объектов в тексте // Информатизация и информационная безопасность правоохранительных органов: XI Межд. научная конференция. Сборник трудов – Москва, 2003. с. 312-317.

[Ефремова, 2010] Ефремова Н.Э., Большакова Е.И., Носков А.А., Антонов В.Ю. Терминологический анализ текста на основе лексико-синтаксических шаблонов // Комп. лингвистика и интеллектуальные технологии: По материалам Междунар. конф. «Диалог» (Бекасово, 26-30 мая 2010 г.) Вып. 9 (16). – М.: Изд-во РГГУ, 2010, с. 124-129.

[Жигалов, 2002] Жигалов В.А. и др. Система Alex как средство для многоцелевой автоматизированной обработки текстов // Труды Междунар. семинара Диалог’2002: "Компьютерная лингвистика и интеллектуальные технологии". – М.: Наука, 2002. Т.2, С.192-208.

[Скатов, 2010] Скатов Д.С., Вдовина Н.А., Ливерко С.В., Окатьев В.В. Язык описания правил в системе лексического анализа ЕЯ-текстов Dictascope Tokenizer // Комп. лингвистика и интеллектуальные технологии: По материалам Междунар. конф. «Диалог» (Бекасово, 26-30 мая 2010 г.) Вып. 9 (16). – М.: Изд-во РГГУ, 2010, с. 442-449.

[Томита-парсер] Томита-парсер. Руководство разработчика. URL: https://tech.yandex.ru/tomita/doc/dg/concept/about-docpage/ (дата обращения: 28.02.2015).

[Хорошевский, 2004] Хорошевский В.Ф. OntosMiner: семейство систем извлечения информации из мультиязычных коллекций документов // Девятая Национальная конф. по искусственному интеллекту КИИ-2004. Т. 2. – М.: Физматлит, 2004, с. 573-581.

[Bontcheva et al., 2003] Bontcheva K., Maynard D., Tablan V., Cunningham H. GATE: A Unicode-based infrastructure supporting multilingual information extraction. In: Proceedings of Workshop on Information Extraction for Slavonic and Other Central and Eastern European Languages (IESL’03), Borovets, 2003.

[Cunningham et al., 2000] Cunningham H., Maynard D., Tablan V. JAPE: a Java Annotation Patterns Engine. (Second Edition). Technical report CS--00--10, University of Sheffield, Department of Computer Science, 2000.

[GATE] General Architecture for Text Engineering – http://www.gate.ac.uk/ (дата обращения: 28.02.2015)

[Grishman , 2003] Grishman R. Information extraction. In: The Oxford Handbook of Computational Linguistics. Mitkov R. (ed.). Oxford University Press, 2003, p. 545-59.

[Tomita, 1984] Tomita M. LR parsers for natural languages. COLING: 10th International Conference on Computational Linguistics, 1984, p. 354–357.