УДК 004.912

МОДЕЛЬ ГРАФЕМАТИЧЕСКОГО АНАЛИЗА В СИСТЕМЕ ОБРАБОТКИ ЕСТЕСТВЕННОГО ЯЗЫКА

А. А. Седунов

Воронежский государственный университет

Системы обработки текста на естественном языке в настоящее время представляют собой одно из наиболее перспективных направлений информационных технологий. Обработка текста в таких системах представляет собой комплексный многоэтапный процесс. В настоящей работе рассматривается задача графематического анализа и предлагается вариант ее программного решения в рамках конкретной ЕЯ-системы. Для решения данной задачи разработан формальный подход к графематическому анализу текста. В рамках этого подхода построена информационная модель анализа и выполнена ее программная реализация.

1.ВВЕДЕНИЕ

Системы обработки текста на естественном языке [1, 2, 3] в настоящее время представляют собой одно из наиболее перспективных направлений информационных технологий. Эти системы находят широкое применение в области информационно-поисковых систем, машинного перевода, классификация документов и разработка естественно-языковых пользовательских интерфейсов. Обработка текста в таких системах представляет собой комплексный многоэтапный процесс. В настоящей работе рассматривается задача графематического анализа и предлагается вариант ее программного решения в рамках конкретной ЕЯ-системы.

Графематический анализ представляет собой начальный этап обработки текста, в ходе которого определяются элементы грамматической структуры (слова, знаки пунктуации, числа, сокращения и т. д.). Можно выделить следующие основные функции графематического анализа [1]:

- разбиение текста на графемы;
- определение границ предложений;
- различение слов и служебных графем (например, знаков пунктуации)
- определение регистра слов
- распознавание собственных имен
- распознавание сокращений

Для решения этих задач мы, опираясь на методы теории языков, разработали формальный подход к графематическому анализу текста. В рамках этого подхода была построена информационная модель анализа и выполнена ее программная реализация. Рассмотрению этих вопросов посвящено содержание предлагаемой статьи.

2. ЗАДАЧА ГРАФЕМАТИЧЕСКОГО АНАЛИЗА

2.1. ПОСТАНОВКА ЗАДАЧИ ГРАФЕМАТИЧЕСКОГО АНАЛИЗА

Перейдем теперь к рассмотрению задач, возникающих при анализе текста. Прежде всего, дадим определение самому понятию графематического анализа с точки зрения теории формальных языков [4, 5]. При формулировке требований к языку (в частности, в следующем определении графематического разбиения) мы будем, главным образом, исходить из практических соображений об эффективности процедуры анализа.

Пусть $\Sigma = \{\sigma_i\}_{i=1}^{|\Sigma|}$ — *алфавит* (конечное упорядоченное множество *символов*) и $L \subseteq \Sigma^* = \left\{ s = \langle s_i \rangle_{i=1}^{|s|} \middle| s_i \in \Sigma, |\, s \,| \geq 0 \right\}$ — некоторый *язык*, заданный над этим алфавитом. Введем для описания структуры строк языка L ряд объектов:

- Конечное *множество классов графем* $T = \{T_i\}_{i=0}^{|T|-1}, |\, T \,| > 1$ с выделенным элементом $T_0 \in T$, который будем называть *классом нераспознанных графем*.
- *Функцию классификации* $\tau : \Sigma^* \to T$, которая ставит в соответствие произвольной строке класс графемы. Если строка s не является графемой с точки зрения правил языка, то будем считать по определению, что $\tau(s) = T_0$. Кроме того, $\tau(\varepsilon) = T_0$. Функция классификации долж-

на быть сюръективной, так как в противном случае некоторые классы графем оказываются избыточными.

• *Множество допустимых графем* $L_T = \{s \in \Sigma^* \mid \tau(s) \neq T_0\}$

• Конечную совокупность *множеств элементарных атрибутов* $A = \{\overset{|A|}{\underset{j=1}{A_j}}\}, \mid A \mid \geq 1, A_j \neq \varnothing$;

• *Множество атрибутных функций* $\alpha = \{\overset{|A|}{\underset{j=1}{\alpha_j}}\}$, где $\alpha_j : Dom(\alpha_j) \to A_j, j = \overline{1, \mid A \mid}$, причем $Dom(\alpha_j) \subseteq L_T$.

Содержательный смысл перечисленных объектов зависит от особенностей интерпретации L как естественного языка. Ниже мы увидим, каким образом можно уточнить структуру этих множеств и функций в случае анализа русского языка.

Будем называть разбиением строки $s \in L \setminus \{\varepsilon\}$ кортеж $p(s) = \overset{|p(s)|}{\underset{i=1}{\langle p_i \rangle}}$, где $s = \prod_{i=1}^{|p(s)|} p_i$, $\mid p(s) \mid \leq \mid s \mid$ и $\mid p_i \mid > 0$.

Разбиение P_s назовем *правым графематическим разбиением* (далее — графематическое разбиение), если каждая его подстрока p_i удовлетворяет следующим условиям:

• p_i является допустимой графемой, т. е. $p_i \in L_T$.

• Пусть $s = q_i p_i r_i$, тогда при любом префиксе v подстроки r_i, где $\mid v \mid > 0$, строка $p_i v$ не является допустимой графемой.

Отметим, что если строка s имеет графематическое разбиение, то оно является единственным. Действительно, предположим противное: пусть существуют два различных разбиения $s = \prod_{i=1}^{n} p_i$ и $s = \prod_{i=1}^{m} q_i$, где все подстроки p_i и q_i являются допустимыми графемами. Тогда можно указать такой номер $k \in \overline{1, m}$, что $p_k \neq q_k$ и $\forall (i = \overline{1, k-1})[p_i = q_i]$ (в противном случае разбиения совпадают). Обозначим $w = \prod_{i=1}^{k-1} p_i = \prod_{i=1}^{k-1} q_i$, тогда $\begin{cases} s = w p_k u \\ s = w q_k v \end{cases}$ (возможен случай $w = \varepsilon$). Так как $p_k \neq q_k$ и обе строки wp_k и wq_k являются префиксами s, то $\mid p_k \mid \neq \mid q_k \mid$. Не ограничивая общности, можно считать, что $\mid p_k \mid < \mid q_k \mid$. Тогда снова в силу того, что строки wp_k и wq_k являются префиксами s, получаем: $q_k = p_k r$, откуда $s = w p_k r v$. В то же время $s = w p_k u$ и, следовательно, $u = rv$, т. е. r — префикс u. Мы получили, что строки p_i и $p_i r = q_i$ одновременно являются допустимыми графемами, что противоречит определению графематического разбиения. Таким образом, если графематическое разбиение существует, то оно единственно.

Строку s назовем *допустимой*, если для нее существует графематическое разбиение. Будем также называть допустимым язык, если каждая непустая строка, принадлежащая ему, допустима.

Теперь мы можем сформулировать основную задачу графематического анализа следующим образом.

Пусть задан кортеж $X = \langle \Sigma, L, T, L_T, A, \tau, \alpha \rangle$. Основной *задачей графематического анализа в условиях* X будем называть задачу *построения для произвольной строки* $s \in L \setminus \{\varepsilon\}$ *ее графематического разбиения* P_s или *выяснения невозможности такого построения*.

2.2. ПРИМЕР ЗАДАЧИ ГРАФЕМАТИЧЕСКОГО АНАЛИЗА

Поясним сказанное выше на примере простого формального языка L_1, заданного регулярным выражением $a^*[0-9]^* b^*$ над алфавитом $\Sigma = \{a, b, 0, 1, \ldots 9\}$. Рассмотрим, например, множество $T = \{T_0, T_1, T_2\}$ из трех классов:

• T_0 — «класс нераспознанных графем»;

• T_1 — «класс буквенных последовательностей»;

• T_2 — «класс цифровых последовательностей».

Функцию классификации τ определим следующим образом (для сокращения записи в качестве аргумента функции указано регулярное выражение, соответствующее некоторому множеству строк):

• $\tau((a \mid b)^*) = T_1$;

• $\tau([0-9]^*) = T_2$;

• $\tau(s) = T_0$ в остальных случаях.

При этом мы получаем множество допустимых графем L_{1T} в виде $L_{1T} = (a \mid b)^* \mid [0-9]^*$.

В качестве множеств атрибутов возьмем $A = \{A_1, A_2\}$, где

• $A_1 = (a \mid b)^*$ — множество буквенных строк над алфавитом Σ;

• $A_2 = \mathbf{N} \cup \{0\}$ — множество неотрицательных целых чисел.

Атрибуты из множества A_1 соотнесем с классом графем T_1 и будем считать значением атрибута саму строку:

1. $Dom(\alpha_1) = \{s \in \Sigma^* \mid \tau(s) = T_1\}$;

2. $\alpha_1(s) = s$.

Аналогично множество A_2 соотнесем с классом T_2, причем значением атрибута будет целое число, десятичная запись которого находится в строке s:

1. $Dom(\alpha_2) = \{s \in \Sigma^* \mid \tau(s) = T_2\}$

2. $\alpha_2(s) = \sum_{i=1}^{|s|} d(s_i) \cdot 10^{|s|-i}$, где функция $d : \{'0', \ldots, '9'\} \to \{0, \ldots, 9\}$ преобразует символ цифры в соответствующее целое число.

Общее число разбиений строки s, как нетрудно видеть, составляет $2^{|s|-1}$. Например, для исходной строки $s = 'aa13b'$ имеется 16 различных вариантов разбиения, в том числе и графематическое: $\langle 'aa', '13', 'b' \rangle$. Следовательно, данная строка является допустимой. Можно показать, что аналогичное утверждение справедливо для любой непустой строки в языке L_1, т. е. допустимым с точки зрения графематического анализа является и сам язык L_1.

В то же время легко привести пример языка, который не будет допустимым. Пусть язык задан регулярным выражением $(ab)^*$ и функция классификации определена следующим образом:

- $\tau(a) = T_1$
- $\tau(aba) = T_3$
- $\tau(bab) = T_3$

Тогда строка $s = 'abab'$ не имеет графематического разбиения. Действительно, в данном случае существует единственное разбиение, состоящее только из графем: $\langle 'a', 'bab' \rangle$. Однако это разбиение не удовлетворяет второму условию определения.

Далее мы установим связь между задачей графематического анализа и распознаванием строк формального языка.

2.3. СВЯЗЬ ГРАФЕМАТИЧЕСКОГО АНАЛИЗА С РАСПОЗНАВАНИЕМ ЯЗЫКА

Как мы уже отметили выше, функция классификации τ отображает произвольную строку над алфавитом Σ на соответствующий ей класс графемы. Пусть теперь $t \in T$ — некоторый класс графем. Рассмотрим полный прообраз t при отображении τ: $\Lambda_t = \tau^{-1}(t) = \{s \mid \tau(s) = t\}$. Очевидно, Λ_t является языком над алфавитом Σ и включает все строки, которые характеризуются данным классом графемы. Поскольку в допустимом разбиении строки s все подстроки являются допустимыми графемами, т. е. относятся к классам, отличным от класса T_0, то каждая из них описывается языком типа Λ_t, где $t \neq T_0$. Из сюръективности функции τ следует, что $\bigcup_{i=0}^{|T|-1} \Lambda_i = \Sigma^*$ (здесь введено обозначение $\Lambda_i = \Lambda_{T_i}$), причем языки Λ_i в силу однозначности τ попарно не пересекаются, т. е. $\Lambda_i \cap \Lambda_j = \varnothing$ при $i \neq j$. Отсюда получаем, что $\Lambda_0 = \Sigma^* \setminus \bigcup_{i=0}^{|T|-1} \Lambda_i$.

Таким образом, задание функции τ эквивалентно описанию множества языков $\Lambda = \{\Lambda_i\}_{i=1}^{|T|-1}$.

Теперь мы можем определить соответствие строки $s \in \Sigma^*$ некоторому классу T_i как принадлежность ее множеству Λ_i. В результате задача графематического анализа сводится к известной из теории формальных языков *задаче распознавания*, и мы можем построить алгоритм графематического анализа, опираясь на распознаватели языков Λ_i. Для этого нам понадо $r_i(s)$ бится понятие максимального префикса.

Пусть $s \in L$. Обозначим через $\lambda_i(s)$ *максимальный префикс строки* s, *распознаваемый языком* Λ_i, т. е. такую строку $\lambda_i(s)$, что

- $\lambda_i(s) \in \Lambda_i$;
- $s = \lambda_i(s) r_i(s)$;
- Если u — произвольный префикс $r_i(s)$, то $\lambda_i(s)u \notin \Lambda_i$

Здесь — остаток строки s после префикса $\lambda_i(s)$.

Очевидно, что если максимальный префикс существует, то он единственен. Действительно, если бы существовали два различных максимальных префикса λ_i и λ'_i, то они имели бы разные длины. Для определенности будем считать, что $|\lambda_i| < |\lambda'_i|$. Отсюда, в силу того, что λ_i и λ'_i — префиксы одной строки, следует соотношение $\lambda'_i = \lambda_i u_i$. Так как $\begin{cases} s = \lambda_i r_i \\ s = \lambda'_i r'_i = \lambda_i u_i r'_i \end{cases}$, то $r_i = u_i r'_i$, т. е. u — префикс r_i и, согласно определению максимального префикса, $\lambda'_i = \lambda_i u_i \notin \Lambda_i$. Таким образом, мы получили противоречие с исходным предположением и заключаем, что максимальный префикс действительно единственен (при условии, что он вообще существует).

Введем множество $\lambda(s)$ всех максимальных префиксов, которые существуют для данной строки (при различном выборе языка Λ_i) $\lambda(s) = \{t \mid \exists i[t = \lambda_i(s)]\}$.

Рассмотрим разбиение $p(s)$ строки $s = s_1$, в котором:

$$\begin{cases} p_i = \max \lambda(s_i) \\ s_i = p_i s_{i+1} \end{cases}$$

где $i = \overline{1,|\ p(s)\ |}$ и $s_{|p(s)|+1} = \varepsilon$

Здесь наибольшее значение $\max \lambda(s_i)$ понимается в смысле длины строки, т. е. $p_i = \max \lambda(s_i) \Leftrightarrow \forall t \in \lambda(s_i)[|\ p_i\ | \geq |\ t\ |]$.

Покажем, что это разбиение является графематическим. Предположим, что это не так. Тогда существует номер $k = \overline{1,|\ p(s)\ |}$, такой, что подстрока p_k не удовлетворяет требованиям графематического разбиения. Так как $p_k \in \lambda(s_k) \subset L_T$ (p_k является допустимой графемой), то отсюда следует, что существует такой префикс $v \neq \varepsilon$ строки s_{k+1}, что $p_k v \in L_T$. Мы получили, что строка s_k имеет префикс $p_k v$, который по длине превышает префикс p_k. В то же время (в силу построения разбиения $p(s)$) p_k — это максимально возможный префикс строки s_k, который является допустимой графемой. Следовательно, $p_k v \notin L_T$. Мы получили противоречие, которое доказывает исходное утверждение: разбиение $p(s)$ является графематическим. Мы будем обозначать такое разбиение $p_{\max}(s)$.

В силу единственности графематического разбиения, мы можем утверждать, что графематическое разбиение P_s строки s существует тогда и только тогда, когда существует разбиение $p_{\max}(s)$, причем $P_s = p_{\max}(s)$. Иными словами, построение графематического разбиения сводится к построению $p_{\max}(s)$.

Заметим, что время $T_{\max \lambda(s)}$, которое необходимо затратить на поиск максимального префикса p_i, зависит от свойств языков Λ_i. Так как

$$T_{\max \lambda(s)}(s) = \sum_{\lambda \in \lambda(s)} T_{\lambda_i(s)}(s) \leq |\ \lambda(s)\ | \max_i(T_{\lambda_i(s)}(s)) \leq$$
$$\leq (|\ T\ | - 1) \max_i(T_{\lambda_i(s)}(s)),$$

то $T_{\max \lambda(s)}(s) = O(\max_i(T_{\lambda_i(s)}(s))$. Если все языки Λ_i являются детерминированными, то $T_{\lambda_i(s)}(s) = O(|\ s\ |)$ и, следовательно, $T_{\max \lambda(s)}(s) = O(|\ s\ |)$. В этом случае время $T_{p_{\max}}$, необходимое на построение, разбиения $p_{\max}(s)$, соответствует $\sum_{j=1}^{m} O(|\ s_j\ |)$, где $\sum_{j=1}^{m} |\ s_j\ | = |\ s\ |$. В худшем случае данное выражение приводит к квадратичной оценке: $T_{p_{\max}} = O(|\ s\ |^2)$. Тем не менее, в случае анализа ЕЯ-текстов данный результат может быть улучшен благодаря некоторым особенностям естественных языков:

- наличие во многих языках символов-разделителей (например, пробелов), которые не являются частью допустимых графем (исключением могут быть сокращения, например, «и т. п.»);
- различные графемы часто имеют различные начальные символов (при этом множества $\lambda(s_i)$ содержат не более одного элемента)

При таких ограничениях графематическое разбиение можно построить в результате однократного прохода по исходной строке и, следовательно, время, необходимое для анализа составляет $O(|\ s\ |)$. Модель графематики, принятая в CALP, соответствует указанным условиям с некоторыми исключениями (например, в отношении анализа сокращений).

Далее мы кратко обсудим особенности применения формального подхода к описанию графематической структуры русского языка.

2.4. ГРАФЕМАТИЧЕСКАЯ МОДЕЛЬ РУССКОГО ЯЗЫКА

В принятой нами графематической модели русского языка выделяются следующие классы графем (для краткости мы опустим определения в терминах введенного нами формализма и воспользуемся простым словесным описанием):

- *Собственное слово* — произвольная последовательность букв русского алфавита; рассматривается как основной смысловой элемент текста;
- *Иностранное слово* — произвольная последовательность Unicode-букв, которые не входят в русский алфавит; рассматривается как дополнительный смысловой элемент;
- Объединение двух предыдущих классов образует *множество слов*. Это множество само по себе не является классом графем, а используется в целях классификации;
- *Буквенная последовательность* — произвольная последовательность Unicode-букв, которая не является словом; в данной реализуется не интерпретируется;
- *Сокращение* — последовательность букв, пробелов и точек, которая квалифицируется с помощью специального справочника сокращений
- *Целое число* — произвольная последовательность цифр
- *Дробное число* — две произвольные последовательности цифр, разделенные без пробелов запятой

• *Буквенно-цифровая последовательность* — произвольная последовательность Unicode-букв и цифр; в данной реализуется не интерпретируется;

• *Разделитель предложений* — произвольная последовательность символов точек, восклицательных или вопросительных знаков;

• *Знак пунктуации* — один из следующих символов:

o одинарная или двойная кавычка

o открывающая или закрывающая скобка

o двоеточие

o запятая

• *Нераспознанная графема*

В качестве алфавита используется 2-байтовый набор символов Unicode.

Отметим также основные атрибутные функции графем:

• *Регистр* — определена на множестве слов и принимает одно из следующих значений:

o нижний — все символы слова находятся в нижнем регистре

o заголовочный — первый символ слова находится в верхнем регистре, остальные — в нижнем

o верхний — все символы слова находятся в верхнем регистре

o смешанный — любая другая комбинация регистров

• Ряд логических атрибутов:

o Признак начала предложения — присваивается первой графеме предложения (определен для всего множества графем).

o Признак собственного имени — присваивается графеме, если она квалифицирована как собственное имя (определен для множества слов). Слово считается именем собственным, если оно содержится в специальном справочнике, либо находится в заголовочном регистре и не является первой графемой предложения

o Наличие предшествующего пробела — присваивается графеме, если в исходном тексте перед ней находился символ пробела (или цепочка таких символов); данный атрибут используется для разрешения ряда неоднозначностей, в частности, связанных с использованием точки (наличие пробела может влиять на интерпретацию точки как разделитель предложений или часть другой графемы).

3. РЕАЛИЗАЦИЯ ГРАФЕМАТИЧЕСКОГО АНАЛИЗА

3.1. АРХИТЕКТУРА ЕЯ-СИСТЕМЫ CALP

В целях исследования способов построения моделей естественного языка на различных уровнях, а также путей эффективной реализации этих моделей в программных средствах автором была предпринята разработка универсальной системы естественно-языковой обработки текстов CALP (Computer-Aided Language Processing). В качестве технологической платформы для данной системы используется платформа Java™.

К настоящему моменту в рамках CALP разработаны, реализованы и протестированы ЕЯ-модели, находящиеся на уровне предсинтаксического анализа текста (графематика и морфология). Дальнейшее развитие проекта предполагает реализацию синтаксических моделей на основе одного из грамматических формализмов.

На рис. 1 представлена общая диаграмма компонентов, образующих ядро CALP.

Система CALP включает 3 уровня:

1. На *уровне представления* определяются базовые функции, которые используются при реализации общих ЕЯ-моделей и конкретных модулей. К функциям модуля представления относятся:

a. механизм назначения и хранения атрибутов;

b. механизм представлений;

c. общие ЕЯ-модели (ЕЯ-ядро).

2. Уровень платформы обеспечивает функционирование системы в целом, взаимодействие с модулями расширения, предоставляет доступ к ресурсам, а также обеспечивает общую поддержку средств локализации модулей. *Модуль* CALP представляет собой некоторый программный код (совокупность Java-классов), который расширяет систему, реализуя ЕЯ-обработку для конкретных языков. Модули могут использовать различные вспомогательные данные (например, конфигурационные файлы, справочники, словари и т. д.), которые называются *ресурсами*.

3. На *уровне* конкретного *модуля* реализуется ЕЯ-модели, специфичные для данного языка, а также выполняется интерпретация ресурсов, требующихся для работы модуля. Кроме того, модуль обязательно содержит компонент, обеспечивающий взаимодействие с платформой системы CALP.

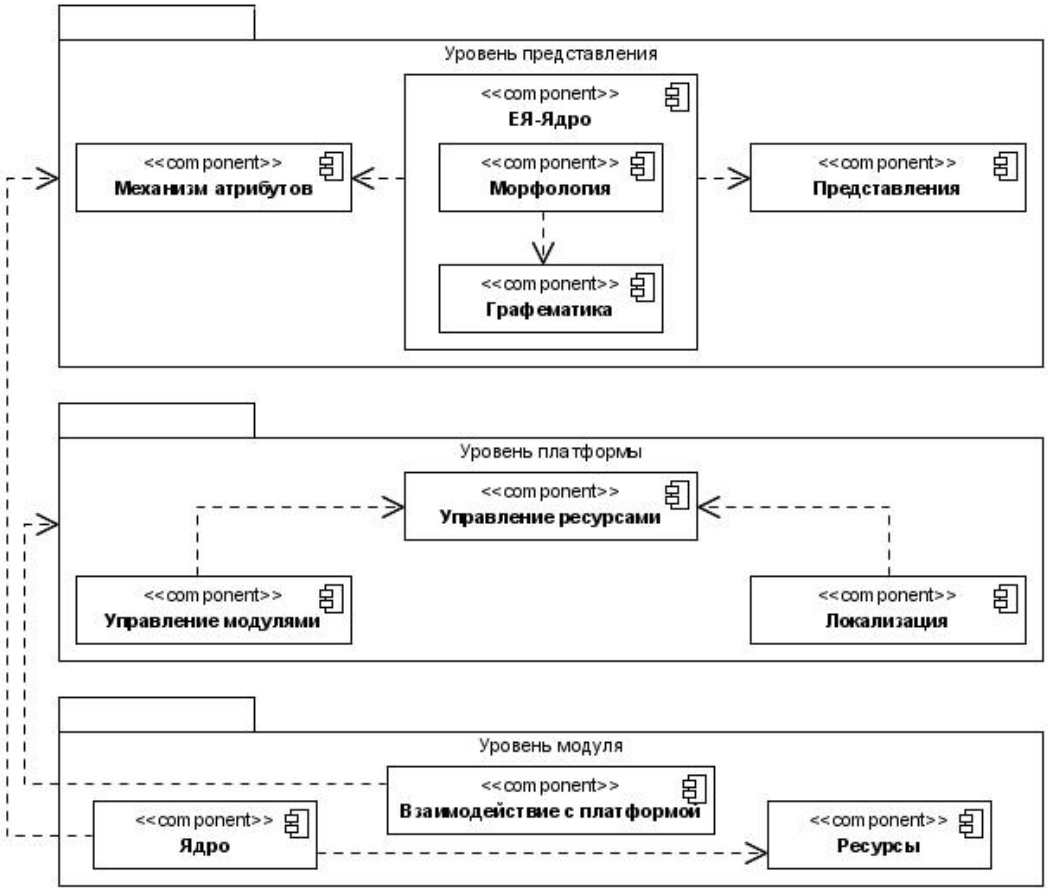

Рис. 1. Общая диаграмма компонентов CALP

Мы подробно остановимся на структуре компонентов, отвечающих за реализацию механизмов атрибутов и представлений, поскольку они имеет непосредственное отношение к построению ЕЯ-моделей и, кроме того, рассмотрим графематический компонент ЕЯ-ядра системы.

3.2. МЕХАНИЗМ ПРЕДСТАВЛЕНИЙ

Механизм представлений является общей моделью результатов анализа. Основу этого механизма составляет объект-представление View, который содержит упорядоченный список элементов ViewElement. На рис. 2 показана соответствующая диаграмма классов:

Основная функциональность представления связана с перечислением входящих в него элементов. Объект View позволяет получить ссылку на первый элемент (метод getFirstElement()), используя которую можно выполнить обход всех элементов с помощью метода getNext() интерфейса ViewElement.

С представлением также может быть связан объект-создатель ViewProducer и исходное представление, на основе которого был создан данный объект View. Представления, таким образом, можно объединять в цепочки. Данная возможность соответствует поэтапному характеру процесса ЕЯ-обработки текста. Для удобства реализации данной возможности в механизм представлений включен класс ViewProducerChain, который представляет собой специальную разновидность объекта-создателя ViewProducer, порождающего новый объект View как результате цепочки последовательных преобразований исходного представления.

3.3. МЕХАНИЗМ АТРИБУТОВ

Механизм атрибутов представляет собой средство общего назначения, предназначенное для вычисления и хранения атрибутов. Под атрибутом в данном случае понимается некоторое значение, приписываемое данному объекту. Как правило, значения атрибутов принадлежат небольшому множеству альтернатив. В сущности механизм атрибутов является удобной и достаточно эффективной объектно-ориентированной оболочкой для традиционного способа описания атрибутов с помощью битовых строк. На рис. 3 изображена диаграмма классов, опи-

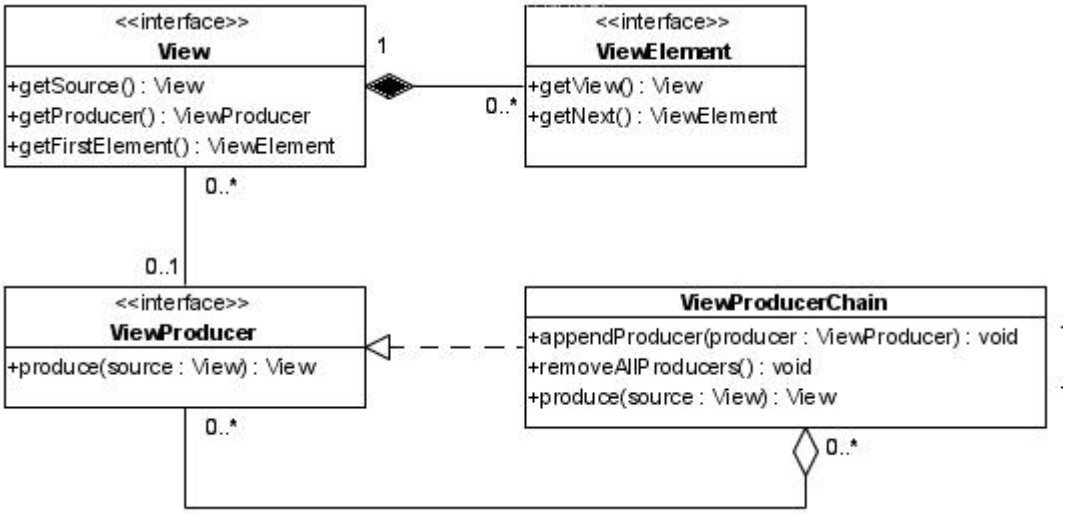

Рис. 2. Диаграмма классов представлений

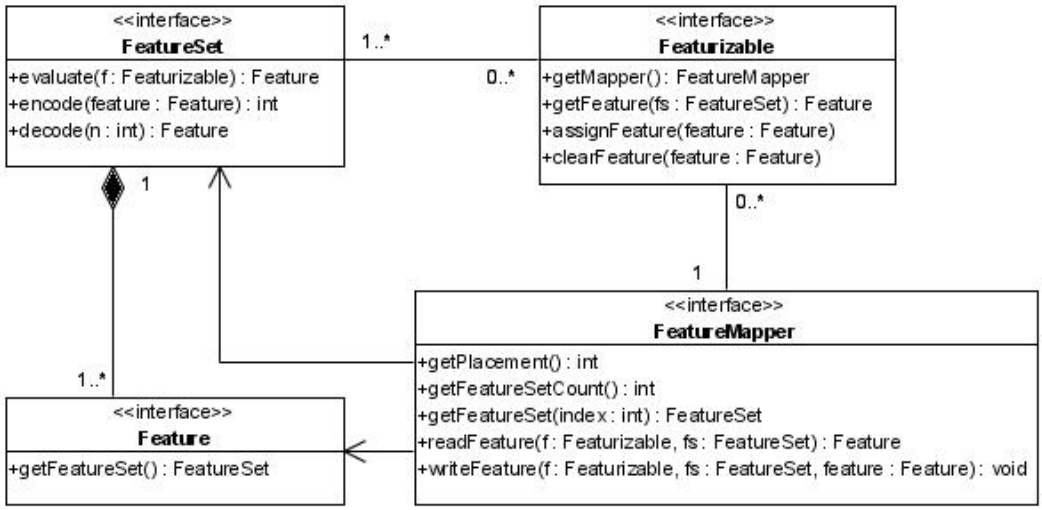

Рис. 3. Диаграмма основных классов механизма атрибутов

сывающая ключевые составляющие механизма:

• Описываемый объект (Featurizable) — объект, которому приписываются атрибуты.

• Атрибут (Feature) — объект, описывающий одно значение атрибута.

• Набор атрибутов (FeatureSet) — объект, реализующий операцию вычисления значения атрибута, а также приписывающий атрибутам уникальные номера, которые используются при чтении или записи атрибута.

• Механизм доступа к атрибутам (FeatureMapper) — объект, реализующий конкретный способ хранения атрибутов.

Ядро CALP содержит две основных реализации механизма FeatureMapper: IntFeatureMapper и LongFeatureMapper. Обе реализации хранят атрибуты внутри битовой строки и отличаются только ее длиной (32 и 64 бита соответственно). Эти классы реализую только операции над содержимым битовой строки, сама же строка хранится в объекте Featurizable. Соответственно определены две реализации этого интерфейса: IntFeaturizable и LongFeaturizable.

Для наборов атрибутов FeatureSet также предусмотрены две основные реализации. Класс BooleanFeatureSet реализует логический набор, который содержит всего два атрибута: истинный (он определяется конкретной реализацией интерфейса) и ложный (общий для всех реализаций). Логический набор можно также интерпретировать как один атрибут, который либо присутствует (если равен своему истинному значению), либо нет. Класс EnumerableFeature-

Set реализует набор с произвольным количеством значений. При этом сами объекты-значения Feature хранятся в массиве.

Взаимодействие с механизмом атрибутов организуется, в основном, через методы интерфейса Featurizable:

• Для чтения значения атрибута вызывается метод getFeature(). Если атрибут еще не был присвоен, он автоматически вычисляется и сохранятся в объекте Featurizable

• Метод assignFeature() позволяет установить значение атрибута вручную. Данная возможность требуется для некоторых атрибутов (см. ниже описание модуля для русского языка), которые вычисляются не соответствующим им набором FeatureSet, а, скажем, программным кодом анализатора

• С помощью метода clearFeature() можно снять установленное значение атрибута

Механизм атрибутов связан с представлениями (View): элемент любого представления реализует интерфейс Featurizable и, следовательно, способен хранить атрибуты.

3.4. МОДЕЛЬ ГРАФЕМАТИЧЕСКОГО АНАЛИЗА

Ядро графематического анализатора не зависит от конкретного языка. Оно построено на основе специализированного представления GraphematicalView, элементами которого являются объекты-графемы (Grapheme). Основные элементы модели графематического анализа представлены на диаграмме рис. 4.

Объект GraphematicalView хранит исходный текст в виде одного символьного массива. При этом устраняется необходимость создания большого количества объектов типа String для хранения отдельных фрагментов.

Объект Grapheme описывает одну графему исходного текста. Он расширяет интерфейс ViewElement и, кроме того, позволяет получать доступ к символам графемы с помощью метод length(), возвращающего длину графемы и метода charAt(), который возвращает символ с заданным номером.

Основная функциональность сосредоточена в методе load(), который работает в 3 этапа. На первом этапе из входного потока загружается массив с исходным текстом. При этом используется объект-делегат CharBufferLoader, функция которого состоит в чтении массива из потока с выполнением всех необходимых преобразований (включая декодирование текста из байтового потока).

Далее конструируется объект, описывающий первую графему, причем сначала этот объект охватывает все содержимое буфера. После этого происходит обращение к объекту-анализатору Tokenizer. Назначение этого объекта состоит в реализации процедуры графематического анализа для конкретного языка, поэтому реализацию Tokenizer предоставляет модуль расширения. В процессе своей работы анализатор расщепляет текущий объект Grapheme на 2 части:

• в левой части остается очередная графема исходного текста

• в правой части остается необработанный фрагмент строки

Так, при первом расщеплении исходный объект Grapheme образует первую графему и остаток строки, при втором расщеплении остаток дает вторую графему и т. д. Этот алгоритм соответствует рассмотренной выше процедуре построения графематического разбиения исход-

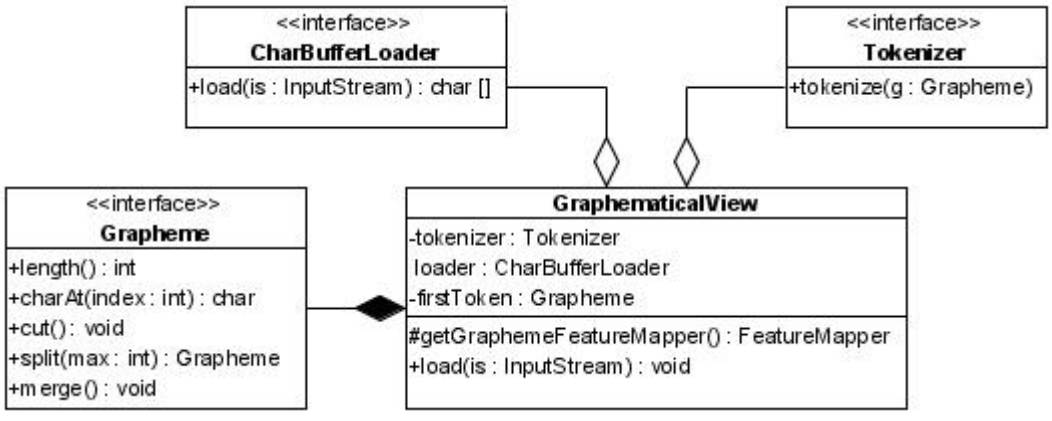

Рис. 4. Диаграмма классов графематического анализа

ной строки. Изменение графематической структуры основано на 3-х методах интерфейса Grapheme:

• split() — это основной метод, который выполняет расщепление графемы на две части. Параметр метода задает размер первой части и вычисляется в процессе анализа.

• Метод cut() удаляет из графемы первый символ. При этом длина графемы уменьшается на 1. Основное применение данного метода связано с удалением из исходной строки пробелов и недопустимых символов в процессе анализа.

• Метод merge() является обратным по отношению к split() и реализует операцию соединения данной графемы с последующей. В процессе анализа метод merge() используется при обработке сокращений.

Таким образом, графематический компонент ядра CALP обеспечивает реализацию общей структуры данных, необходимой для хранения результатов графематического анализа. Собственно процедура анализа, а также конкретные атрибуты графем предоставляются модулем расширения, однако их описание выходит за рамки настоящей статьи.

4. ЗАКЛЮЧЕНИЕ

В заключение отметим основные результаты, полученные в рамках настоящей работы:

• Разработан формальный подход к проблеме графематического анализа и выявлена его связь с задачей распознавания формальных языков.

• На основании данного подхода разработана общая модель графематики естественного языка и частная модель, ориентированная на русский язык

• Разработана и протестирована программная реализация графематических моделей в рамках универсальной расширяемой системы естественно-языковой обработки CALP

ЛИТЕРАТУРА

1. *Peter Jackson, Isabelle Moulinier.* Natural Language Processing for Online Applications. — John Benjamins Publishing, 2002. — 237 p.

2. *Bolshakov I.A., Gelbukh A.* Computational Linguistics. Models, Resources, Applications. — IPN-UNAM-FCE, 2004, 186 pp.

3. *Шемакин Ю.И.* Начала компьютерной лингвистики: Учеб. пособие. М.: Иэд-во МГОУ, А/О "Росвузнаука", 1992.

4. *Гладкий А.В.* Формальные грамматики и языки.: М., 1973. — 368 с.: ил.

5. *Ахо А.В., Хопкрофт Дж., Ульман Дж. Д.* Структуры данных и алгоритмы.: Пер. с англ.: М.: Издательский дом «Вильямс», 2003. — 384 с.: ил. — Пар. тит. англ.

Статья принята к опубликованию 25 октября 2007 г.