УДК 004+005+519.7

О необходимости онтологии для классификации и навигации по универсуму компьютерных языков

Шилов Н.В. (Автономная некоммерческая организация высшего образования "Университет Иннополис")

В 2008-2013 гг. в Институте систем информатики им. А.П. Ершова развивался сначала проект по исследованию и классификации парадигм языков программирования, а затем — по разработке классификации компьютерных языков, виртуальный мир которых включает тысячи языков программирования, спецификаций, моделирования и множество других языков, которые можно отнести к (пользуясь терминологией, сложившейся в биологии) разным царствам, классам, семействам и так далее. В настоящей статье представлена сумма основных теоретических результатов исследований 2008-2013 гг. по классификации компьютерных языков (которые, по нашему мнению, остаются актуальными и на сегодняшний день) и обсуждаются новые подходы к задаче разработки портала знаний о компьютерных языках (вернее — его наполнении) с использованием научной периодики и современных лингвистических технологий (вместо размеченных источников в Интернете). Актуальность продолжения исследований по классификации компьютерных языков обусловлена познавательным значением такой классификации, а разработки портала знаний о классификации компьютерных языков — потребностями практики объективного выбора компьютерных языков по спецификации программных проектов.

Ключевые слова: языки программирования, языки разметки, языки запросов, компьютерные языки в целом, классификация, атрибуты, парадигма, портал знаний, навигация по классификации, логика Белнапа, дискипционная логика, темпоральная логика, логика со временем, верификация моделей

1. Введение

Классификация — необходимый шаг/результат при систематическом исследовании какой-либо области природы, техники, искусства, классификация состоит в группировке объектов исследования или наблюдения в соответствии с их общими признаками. При рассмотрении компьютерных языков можно выделить множество характеристик таких как парадигма (например, императивная, декларативная, объектно-ориентированная), дисциплина обработки (например, последовательная, недетерминированная, распределённая),

и виртуальная машина (например, машина Тьюринга, машин продукций, машина логического вывода) и так далее.

Поэтому в 2008-2010 и 2012-2013 гг. в Институте систем информатики им. А.П. Ершова СО РАН развивался сначала проект по исследованию и классификации парадигм языков программирования, а затем — по разработке концепции и автоматизации классификации компьютерных языков [1, 9, 15–17] как с целью изучения мира (универсума) компьютерных языков, так и с целью автоматизации классификации и повышения интеллектуального уровня выбора языков для научных и индустриальных проектов по описанию проекта на естественном языке посредством навигации по классификации при помощи формализованных запросов.

Исследования по парадиматизации (то есть по исследованию и классификации парадигм) языков программирования получили дальнейшее развитие в 2010-2018 гг. в работах [5, 6], причём, особое внимание уделено функциональной и параллельной парадигмам программирования, разработке "лексикона программирования" [8], основанного на выделении парадигм программирования.

По-другому сложилась ситуация с исследованиями задачи автоматизации классификации и навигации по классификации компьютерных языков. Для её решения в 2008-2013 гг. развивался подход, основанный на выделении "определяемых" (динамических) парадигм компьютерных языков по совокупности общих признаков (атрибутам), присущих языкам, (не ограничиваясь "парадигмой" в узком смысле историческом смысле), создании "нетаксономической" (не древовидной) онтологии компьютерных языков. Особое значение придавалось компьютерной поддержке разрабатываемой классификации в форме (прототипа) портала знаний по классификации компьютерных языков [15, 16]. Этот портал предполагалось наполнить сведениями о компьютерных языках из открытых размеченных и хорошо структурированных интернет-ресурсов (например, Wikipedia), организовав навигацию и доступа к собранной информации по логическим запросам, сформулированным на комбинированном языке дискрипционной логики [13, 18], темпоральной логики [10] со временем, построенном над четырехзначной логикой Бэлнапа [19], а в качестве алгоритма решения запросов предлагалось использовать метод верификации конечных моделей (model checking) [10]. Прототип такого портала был создан в 2012 г., но поддерживал только "статические" запросы (и не поддерживал работу со временем и темпоральными конструкциями), выраженными на языке дискрипционной логики над логикой Бэлнапа

[9, 15], но затем был оставлен и в настоящее время не существует.

Начало второго десятилетия XXI-го века совпало с началом “эпохи” целенаправленной массового создания проблемно-ориентированных языков программирования (Domain Specific Languages) [26, 33]. Появлением технологий быстрой разработки таких языков и сред программирования [7, 24, 32, 49], привело к бурному росту числа компьютерных языков, многие из которых не представлены в ни в Wikipedia, ни в других хорошо структурированных и размеченных интернет-ресурсах. (Заметим, что ранее существовала педагогическая методика “создай свой язык программирования”, используемая до сих пор в обучения программированию; активным пропагандистом такого подхода был математик, программист, психолог и педагог, один из основоположников теории искусственного интеллекта, создатель языка Logo — Сеймур Пейперт.) Необходимость осмыслить эту новую реальность в мире компьютерных языков привела участников исследований к решению отложить на время проект реализации портала классификации компьютерных языков.

Ситуация изменилась к настоящему времени — к концу второго десятилетия XXI-го века. Во-первых, появились работы по извлечению “утверждений” или “мнений” (хотя обычно ошибочно говорят об извлечении “знаний”) из плохо структурированных (и только частично размеченных средствами HTML) и — в перспективе — неструктурированных (и неразмеченных) источниках в Интернете [2, 14]. В частности, в цитируемых работах предлагается подход к автоматизации сбора информации о предметной области, использующие методы метапоиска и извлечения информации, базирующиеся на онтологиях и тезаурусах моделируемой области знаний. Во-вторых, получил развитие онтологический подход для формализации знаний в разных научных предметных областях, причём, подход онтологический оказался настолько перспективным, что появилась возможность реализации паттернов содержания при разработке онтологий разных научных областей [3, 50]. С помощью онтологии можно не только представить все необходимые понятия моделируемой области, но и обеспечить их единообразное и согласованное описание. “Заметим, что под научной предметной областью (НПО) здесь понимается предметная область, охватывающая некоторую научную дисциплину или область научных знаний во всех её аспектах, включая характерные для неё объекты и предметы исследования, применяемые в ней методы исследования, выполняемую в ней научную деятельность и полученные в рамках её исследований научные результаты.”[1]

[1]Цитируется по [3].

И, наконец, задача создания и реализации онтологии компьютерных языков стала актуальной в рамках проекта РФФИ №17-07-01600 “Методы извлечения формальных спецификаций программных систем из текстов технических заданий и их верификация” [4, 27]. Этот “проект посвящён проблеме обеспечения качества программных систем с помощью формальных методов. В рамках проекта планируется разработать комплексный подход к извлечению формальных моделей и свойств распределенных программных систем из текстов технической документации с последующей их верификацией.”[2] Так как “правильный” выбор компьютерных языков для реализации программных проектов — одно из (возможно, производных) нефункциональных требований, предъявляемых техническим заданием (одной из разновидностей технической документации) к программному проекту, то автоматизированная система классификации компьютерных языков может стать инструментом верификации формализованных требований к компьютерным языкам, использованным при реализации проекта.

Настоящая статья имеет следующую структуру. В следующей части 2 мы напомним и обсудим основные идеи и понятия, выработанные ранее для создания классификации компьютерных языков. А в части 3 мы обсудим технические решения для реализации принятого подхода, и актуальные задачи, стоящие на пути компьютерной реализации разрабатываемой онтологии. В конце статьи в кратком заключении мы намечаем некоторые ближайшие задачи по развитию проекта.

Работа выполнена при поддержке гранта РФФИ №17-07-01600 “Методы извлечения формальных спецификаций программных систем из текстов технических заданий и их верификация”.

2. Концепция классификации компьютерных языков

Начнём с того, что несколько уточним основные определения, данное в работе [9]. Под компьютерным языком мы понимаем любой искусственный язык, разработанный для форматированного представления, автоматического преобразования, извлечения и управления данными, информацией и процессами. Под классификацией какого-либо универсума (компьютерных языков в частности) мы понимаем подход и инструментарий для выделения сущностей этого универсума (объектов, классов и ролей по отношению друг к другу), а также для навигации между этими сущностями по запросам. Среди всех компьютер-

[2] Цитируется по [4].

ных языков прежде всего выделяются языки программирования — компьютерные языки, предназначенные для организации преобразования данных.

Существует несколько известных (и не очень) де-факто попыток создания классификаций компьютерных языков. Это, прежде всего, популярный плакат [36], охватывающий историю возникновения и взаимного влияния 50 языков программирования, появившихся в период с 1954 г. по настоящее время. На этом плакате классификация проведена по времени возникновения, исторической преемственности между версиями (без определения понятия “версия”) и взаимному влиянию (без конкретизации в чём состояло “влияние”), а навигация сводится к следованию линиям развития языков во времени (вперед или назад) и стрелкам влияния.

На сайте [34] представлен глоссарий 19 основных категорий, относящихся к компьютерным языкам (например, “императивный язык”, “язык четвёртого поколения”), определены понятия “диалект”, “версия” и “реализация”, приведена хронология языков, оказавших наибольшее влияние на развитие других компьютерных языков, а также алфавитно-организованное описание 2500 языков (включая версии,диалекты и реализации) в терминах принятых категорий. Однако, [34] — это, скорее, интернет-справочник по истории и классификации компьютерных языков, а не классификация компьютерных языков и не портал знаний о классификации и компьютерных языках в силу следующих причин:

- единственное средство запросов и навигации между языками — это поиск языка по алфавиту, и переход по внутренним гиперссылкам на связанные языки;
- отсутствуют внешние гиперссылки на источники информации о категоризации языков и об языках (хотя есть ссылки на публикации по некоторым языкам).

В подтверждение сформулированного выше мнения, приведём следующую цитату с данного интернет-ресурса:

> It’s been suggested that the languages in this list should be arranged into categories, but to do so would be extremely difficult. For every classification scheme there will be a large proportion of languages that do not fit. The languages are therefore listed alphabetically, and in fact we think that this is the most useful organization. You’ll find that certain categories have been referred to in the list, but we must emphasize that most languages are not purely one or the other, and we are really categorizing language features.

Классификацию компьютерных языков можно пытаться строить в виде древовидной

таксономии аналогично естественными науками, например “по Линнею”: царство — тип (у растений — отдел) — класс — отряд (у растений порядок) — семейство — род — вид. Так, например, устроена энциклопедия языков программирования HOPL: an interactive Roster of Programming Languages [39], включающая сведения о более чем 8500 языках (однако, следует заметить, что эта таксономия выделяет очень странные классы компьютерных языков). Эта энциклопедия поддерживает навигацию между разнородной информацией, представленной в энциклопедии, по гиперссылкам с разными ролями (как-то, год создания, авторство, взаимное влияние и тому подобное) и поддерживает гиперссылки на внешние источники. Однако, этот уникальный ресурс не имеет языка запросов (кроме как поиск по ключевым словам как-то по названию, автору и так далее) и, кроме того, сама идея таксономии как основы классификации компьютерных языков представляется довольно-таки сомнительной (см. следующие два абзаца).

Во-первых, есть существенная разница между предметной областью в естественных науках и миром компьютерных языков: если в биологии, химии, физике “универсум” сравнительно статичен, то универсум компьютерных языков изменяется очень быстро. Так, например, только за последние два десятилетия мы наблюдали быстрый рост, как существовавших классов компьютерных языков, так и образование новых классов (например, языков представления знаний, языков кластерного/многоядерного программирования, проблемно-ориентированных языков программирования). Некоторые из этих новых компьютерных языков имеют свой специфический синтаксис (например графический), свою семантику (то есть модель обработки информации или виртуальную машину), свою прагматику (то есть сферу применения и распространения). Некоторые из давно существующих классов компьютерных языков сравнительно мало населены (например, языки проектирования СБИС), некоторые — густонаселенны (например, языки спецификаций), а некоторые классы пережили или переживают период роста. В тоже время разработка новых компьютерных языков (а следовательно и образование новых классов компьютерных языков) будет продолжаться по мере компьютеризации новых отраслей человеческой деятельности.

Во-вторых, часто компьютерный язык трудно отнести к одному определённому классу. Например, функциональные языки программирования ML и Рефал появились ещё в 1960-ые годы как языки спецификаций вывода (логического и продукционного), но ещё в 1970-ые годы они стали языками программирования для задач искусственного интеллекта;

в то же время эти языки (в силу их декларативности) по-прежнему могут служить языками спецификаций для вычислительных программ (как, например, в проекте VeriFun [48] функциональный язык используется в качестве спецификации императивной программы, а система доказывает функциональную эквивалентность спецификации и программы). Кроме того, некоторые языки прямо по замыслу их разработчиков не могут быть отнесены к одному конкретному классу: например, активные пропагандисты языка Ruby позиционируют его как смесь (коктейль) из нескольких языков Perl, Smalltalk, Eiffel, Ada и Lisp с хорошим балансом между функциональными и императивными возможностями [44]. (Здесь, однако, нам кажется, происходить определённая путаница между классом языка программирования и стилем программирования [11]: программировать в функциональном стиле можно даже на Algol-60, а в императивном стиле — на языках Рефал и Lisp.)

Исходя из изложенной выше критики известных подходов, при разработке концепции и автоматизации классификации компьютерных языков в ИСИ СО РАН в 2012-2013 гг. [9, 15, 16] предлагалось основывать классификацию универсума компьютерных языков на гибком понимании *компьютерных парадигм*.

Вообще в современной методологии науки парадигмы — это разные подходы к постановке и решению задачи или проблемы. Современным пониманием этого термина мы обязаны широко известной диссертации Томаса Куна [35], впервые опубликованной в 1970 г. Роберт Флойд стал первым, кто употребил это понятие ввёл понятие в информатике и заговорил о *парадигмах программирования* в своей тьюринговской лекции *Paradigms of Programming* в 1978 г. [25], но, к сожалению, он не дал явного определения этого понятия. В 2009 г. Питер ванн Рой опубликовал в Интернете таксономию *The principal programming paradigms* [42], в которой выделил 27 различных парадигм, и предпринял попытку обосновать её в статье [43]. Само понятие парадигмы программирования определено в [43] следующим образом:

> A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Each paradigm supports a set of concepts that makes it the best for a certain kind of problem.

Это определение послужило основой для следующего определения (см. [9, 15, 16]):

- Компьютерные парадигмы — это альтернативные подходы (образцы) к формализации постановок, представления, инструментов и средств решения задач и проблем информатики.

- Компьютерные парадигмы обычно поддержаны строгими математическими теориями/моделями и аккумулированы в соответствующих компьютерных языках.
- Компьютерные парадигмы соответствуют классам компьютерных языков и наоборот:
  - каждый естественный класс компьютерных языков представляет компьютерную парадигму, которую языки этого класса реализуют;
  - каждая компьютерная парадигма естественно характеризуется классом компьютерных языков, реализующих эту парадигму.
- Онтологическое (для универсума компьютерных языков) значение компьютерной парадигмы характеризуется классом задач, для решения которой она подходит более всего или для решения которого она была разработана.
- Образовательное значение компьютерных парадигм состоит в том, что парадигмы учат мыслить по-разному, видеть под разным углом зрения, как конкретные задачи информатики, так и мировоззренческие проблемы информатики.

Синтаксиса, *семантики* и *прагматики* — это категории, которые используются для описания как естественных, так и искусственных языков (компьютерных языков в том числе):

- синтаксис — это "правописание" языка;
- семантика — это правила придания "смысла" правильно написанным фразам языка;
- прагматика — это практика использования корректных фраз языка.

Разумно предположить, что все эти три категории должны использоваться для выделения классов компьютерных языков.

Использование синтаксиса для классификации компьютерных языков должно отражать особенности формального синтаксиса и человеко-ориентированный аспекты: для разработки синтаксического анализатора важно знать, имеет ли язык контекстно-свободный синтаксис, является ли контекстно-свободным и тому подобное; следовательно, эти и другие формально-синтаксические свойства компьютерного языка должны быть атрибутами при классификации компьютерных языков. Однако неформальные (или прагматические) характеристики синтаксиса компьютерного языка также должны учитываться при классификации компьютерных языков: примерами могут служить неформализуемые атрибуты *гибкость* или *краткость*, а также формализуемые атрибут такие как понятие *стиль*, которое можно определить посредством библиотеки размеченных примеров "правильно-

го” и “неправильного”стиля (как, например, функциональные программы написанные в императивном стиле).

Несколько по-другому обстоит дело с использованием неформальной и формальной семантики компьютерных языков. Неформальная семантика мало пригодна для использования для классификации в силу отсутствия общеупотребимых и “понятных” характеристик семантики, а использование формальной семантики для классификации компьютерных языков наталкивается на следующие препятствия [9]:

- слабое знание формальной семантики среди пользователей компьютерных языков (разработчиков, руководителей и менеджеров проектов);
- распространённостью предубеждения, что формальная семантика малополезна и неэффективна на практике;
- избытком индивидуальных нотаций для формальной семантики с разным уровнем абстракции и “вхождения”.

Может быть, все эти препятствия можно преодолеть путём создания унифицированного формализма для описания семантик, развития компьютерного инстументария для поддержки этого формализма, продвижение его преподавания в университетах и популяризация среди разработчиков? Но по нашему мнению [9], этот путь требуют “всемирной” централизации принятия решений о стандартизации формальных семантик, больших капиталовложений и, кроме того, этот путь наверняка будет препятствовать появлению и внедрению новых вариантов формальных семантик.

Например, начиная с 1990-ых годов в Microsoft Research под руководством Юрия Гуревича активно разрабатывался такой унифицированный формализм для описания формальной семантики (прежде всего) языков программирования, сначала под названием эволюционирующие алгебры (evolving algebras), а позже — машины абсрактных состояний (abstract state machines) [23, 28, 29], в рамках которого была описана формальная семантика нескольких языков (например, языка программирования Java [45]). Однако, ни авторитет известного ученого, ни исследовательского подразделения компании Microsoft, ни примеры успешного использования не сделали данный популярный формализм доминирующим среди других формализмов описания семантики. Более того, за время, прошедшее после возникновения машин абстрактных состояний, в информатике появились новые идеи и подходы, которые могут быть использованы для описания формальной семантики, как, например, в работах [20, 21] развивается подход к описанию формальной

семантики, основанный на концептуальных системах переходов, предполагающий построение онтологии синтаксических и семантических понятий формализуемого компьютерного языка.

На наш взгляд [9] эти препятствия могут быть преодолены на пути *многомерной стратификации* семантики наиболее типических компьютерных языков:

- в качестве измерений могут выступать, например, *образовательная* семантика, *реализационная*, *трансформационная* и так далее,
- а в каждом измерении можно выделить несколько *уровней* или *слоёв*, которые условно можно назвать *элементарным*, *основным* (или *базовым*) и *экспертным* (или *мастерским*).

Реализационная семантика компьютерного языка может делить язык на 2-3 или более слоя. Обычно можно условно выделить *ядро*, несколько *промежуточных* слоёв, и *полный* язык. Ядерный слой имеет эффективную реализационную семантику для класса *платформ* и достаточные средства для реализации *методом раскрутки*, *интерпретации* или *трансформаций* конструкций промежуточных слоёв. Полный язык может иметь только частичную трансформационную семантику в более низкие (промежуточные или ядерный) слои, а часть конструкций полного языка может иметь свою индивидуальную реализационную семантику для каждой конкретной платформы. В качестве примера слоёв языка можно сослаться на “уровни” языка C♯ [12].

Прагматика компьютерных языков (в отличие от синтаксиса и формальной семантики) — это плохо формализованные, плохо структурированные и плохо размеченные человеческие “знания” о компьютерных языках, извлекаемые из научных и учебных статей о конкретных языках, из форумов, каналов и порталов, посвященных практике использования языков в образовании, в разработке, в информационных технологиях и так далее. Здесь, однако, уместно напомнить, что мы употребляем слово “знания”, но на самом деле надо говорить только о представлениях или мнениях (beliefs), так как обычно в статьях, в постах и на страницах публикуют авторские мнения, а не знания в строгом научном смысле слова: согласно традиции, восходящей к Платону, знания — это представления о реальности доказательно соответствующие действительности (а, следовательно, непротиворечивые), а мнения людей, участвующих в жизненном цикле компьютерных языков, могут быть просто противоречивыми (даже ку одного человека), или могут не соответствовать действительности.

Такое понимание прагматики компьютерных языков естественно приводит [9, 15, 16] к идее использования *онтологии* для представления прагматики компьютерных языков [30]. Согласно [38]

> Ontology is the theory of objects and their ties. It provides criteria for distinguishing different types of objects (concrete and abstract, existent and nonexistent, real and ideal, independent and dependent) and their ties (relations, dependencies and predication).

Поэтому под онтологией предметной области (прагматики компьютерных языков в частности) мы будем понимать онтологию, реализованную в некотором формализме для представления мнения "экспертов" об объектах этой области, их классах и отношениях между объектами и объектами, объектами и классами, классами и классами, и так далее. Будем говорить, что онтология задана явно, если она явно представлена в виде графа, вершины которого — объекты онтологии, а отношения между объектами представлены дугами (или рёбрами); в противном случае будем говорить о неявной онтологии. Wikipedia (`www.wikipedia.org`) — это хорошо известный пример неявной онтологии. А популярный инструмент создания явных онтологий — это платформа Protégé [47].

Следуя [9], подчеркнём, что онтологическое представление прагматики компьютерных языков должна быть *публичной*, *открытой*, *эволюционирующей*, *темпоральной* и *со временем* онтологией. Под публичностью понимается доступ для получения данных и внесения изменений без ограничений и привилегий пользователей: любой участник жизненного цикла компьютерного языка может обратиться к онтологии (на правах анонимности или добровольной самоидентификации) с любым допустимым запросом для получения данных, удовлетворяющих запросу, и может добавить любые данные, удовлетворяющие поддерживаемым форматам. Открытость означает, что в онтологии принята модель открытого мира, означающая, что не все существующие объекты и отношения уже представлены в онтологии. Под эволюционностью понимается не только развитие онтологии во времени, но и поддержку историй правок, хронику её развития так, что в любой момент можно получить справку о её состоянии в любой момент в прошлом. И, наконец, темпоральность и использование времени означает наличие "штампов времени" у правок, возможность формулировки запросов с привязкой ко времени (неформальный пример: "что думали в 1970 году лауреаты премии Тьюринга о перспективах развития языков программирования, созданных в 1960-ые годы, к 1980-ому году?") и использованием темпоральных конструкций

(например: *до тех пор пока* и так далее). Wikipedia может служить хорошим примером публичной открытой эволюционирующей онтологии, однако, язык запросов к этой неявной онтологии — это язык позитивных булевских запросов без темпоральных конструкций.

3. На пути к реализации онтологической классификации

В работах [9, 15, 16] идея использовать аппарат формальных онтологий для представления "знаний" о прагматике компьютерных языков была распространена на онтологию "знаний" о компьютерных языках вцелом. Объектами этой онтологии могут быть все компьютерные языки, причём, слои и уровни следует считать отдельными компьютерными языками, "диалектами" друг друга. Отношения между языками в этой онтологии — это отношения, типичные для области компьютерных языков, например: *быть диалектом*, *быть версией*, *наследует синтаксис* и другие "связи" между языками (исторические, реализационные, гносеологические, и так далее). Атрибутами объектов и отношений могут выступать (формальные и неформальные) характеристики синтаксиса и семантики, (неформальные) характеристики прагматики компьютерных языков, временны́е теги (например, год первой публикации), ссылки на проеты, выполненные с использованием этих языков, и так далее.

Заметим, что постер Computer Languages History [36] и справочник The Language List [34] уже реализуют онтологический подход к классификации (хотя явно это нигде не сказано). Ещё в большей степени понятию онтологии соответствует HOPL: an interactive Roster of Programming Languages [39]: в этой онтологии представлено более 8500 объектов (языков программирования в данном случае), почти 1800 библиографических ссылок (в качестве атрибутов языков) и почти 5500 отношений (точнее — связей) между языками (влияние, диалекты, версии и реализации, связи по авторам, по году и месту рождения и так далее), поддержана (уже упоминавшейся нами довольно-таки спорной) таксономия классов языков программирования. Средства навигации по этой онтологии представлены связями языков и таксономией классов. Недостатком этой онтологии является её непубличность (закрытость для редактирования) и фиксированное состояние (на 2006 г.).

Ещё один существующий проект классификации — это двуязычная (имеет русскую и английскую версии) открытая эволюционирующая wiki-подобная неявная онтология языков программирования Progopedia [41]. Она отслеживает три вида связей между языками (диалект, версия, реализация), предлагает две простых таксономии языков (из 31 парадиг-

мы программирования и 13 вариантов типизации в языках программирования), но менее населена объектами чем The Language List [34] и тем более чем HOPL [39]: в Progopedia представлены сведения о 171 языках программирования, 83 их диалектах, 349 реализациях и 734 версиях (то есть, в нашей терминологии, о 1337-ми объектах). Средства навигации между объектами — алфавитный порядок, диалекты, версии и реализации, таксономии по парадигмам и вариантам типизации.

Однако общим недостатком классификаций [34, 36, 39, 41] компьютерных языков (языков программирования в частности) мы считаем отсутствие явного понимания, что они имеют дело с онтологией соотвествующего универсума и, следовательно, неиспользование современных технологий Semantic Web для онтологий предметных областей.

Поэтому при разработке публичной открытой эволюционирующей темпоральной онтологии для классификации компютерных языков в 2012-2013 гг. в работах [9, 15, 16] в качестве основы была выбрана логика описаний понятий (известная так же как дискрипционная логика — Description Logic, DL) [13, 18, 30] с использованием технологий языка OWL (Web Ontology Language) [30, 31]. Поэтому ниже мы в описании подхода [9, 15, 16] будет использована терминология DL и OWL (что будет обозначаться через слеш — термин DL / термин OWL).

Объектами разрабатываемой онтологии являются компьютерные языки, включая слои и уровни как отдельные объекты. Например, Pascal, LISP, PROLOG, SDL, а также OWL-Lite, OWL-DL и OWL-full — все эти языки могут быть объектами. Связи между объектами задаются ролями/свойствами, которые могут быть специфицированы ролевыми термами DL, построенных из явно заданных элементарных ролей/свойств. Например, *наследовать синтаксис* или *быть диалектом* могут быть элементарными ролями/свойствами, которые задаются явным указанием пар объектов (компьютерных языков), связанных соответствующим отношением. Понятиями/классами являются множества объектов, которые могут быть специфицированы (выделены) посредством понятийных термов DL, построенных из явно заданных элементарных понятий. Примерами элементарных понятий могут быть *является функциональным языком* или *является языком спецификаций*, их содержание должно быть задано явным указанием объектов (компьютерных языков), входящих в соответствующий класс.

В работах [13, 18, 30] компьютерные парадигмы выделяются посредством понятийных термов DL: каждый понятийный терм определяет компьютерную парадигму посредством

выделения понятия/класса компьютерных языков. (В такой трактовке парадигмы компьютерных языков перестают быть древовидной таксономией, а становятся подрешёткой решётки множеств компьютерных языков.) Объекты (компьютерные языки) могут быть аннотированы разнообразными формальными атрибутами (например, характеристиками формального синтаксиса) и неформальными атрибутами (например, библиотеками примеров хорошего стиля для языка); список возможных атрибутов не фиксирован, но некоторые атрибуты обязательны (то есть автоматически связываться с любым объектом при его инициализации), например:

- дата рождения языка,
- авторство языка,
- рекомендуемая библиография,
- ссылка на доступную пробную версию.

(Заметим, что значения этих атрибутов могут быть неопределенны.)

Так же как и некоторые атрибуты, некоторые элементарные понятия/классы должны автоматически присутствовать в нашей онтологии (поскольку являются классикой предметной области), например: функциональные языки, языки спецификаций, исполняемые языки, языки со статическими типами и так далее (см., например, списки парадигм на [34, 39, 41, 42]). Кроме того, по-видимому обязательным является элементарный класс *типических* или *парадигмальных* языков, который должен содержать хотя бы по одному языку (но немного) из каждого другого элементарного понятия/класса (который автоматически добавляется при создании нового элементарного понятия/класса).

Наполнение элементарных понятий/классов должно происходить на основе явно указанных атрибутов объектов (компьютерных языков), например, язык попадает в языки спецификаций, если у этого языка значение соответствующего атрибута установлено явно. Таким образом, семантика элементарных понятий/классов имеет вполне определённый экстенсиональный характер: с одной стороны, содержание каждого элементарного понятия/класса в каждый конкретный момент времени задаётся путём явного перечисления, но, стороны, у объекта могут быть как недоопределённные так и переопределённые атрибуты. Например, у языка C♯ может быть не указано мнение, что язык является функциональным, а может быть указано два противоречивых мнения, что этот язык является функциональным, и что не является. А вот неэлементарные понятия/классы определяются чисто экстенсионально через элементарные понятия/классы посредством понятий-

ных термов DL, их содержание может быть не полностью определено (даже в каждый конкретный момент времени). Например, *исполнимые языки спецификации* — это пересечение понятий/классов *языков спецификаций* и *исполнимых языков*, недоопределенность содержания этого понятия/класса может произойти только из-за недоопределенности содержания каждого из онятий/классов *языков спецификаций* и *исполнимых языков*; а вот *неисполнимые языки спецификации* — это пересечение понятий/классов *языков спецификаций* и дополнения класса *исполнимых языков*, недоопределенность содержания этого понятия/класса может произойти не только из-за недоопределенности содержания каждого из онятий/классов *языков спецификаций* и *исполнимых языков*, но из-за неопределённости операции дополнения в модели открытого мира (см. следующий абзац).

Обсудим подробнее вариант определения отрицания в модели открытого мира (то есть всегда неполной информации) для универсума компьютерных языков. В [9] предлагался подход, в котором одновременно с введением какого-либо нового *позитивного атрибута* будет автоматически порождаться его *негативный* напарник. (Например, при создании позитивного атрибута *является исполняемым языком* будет автоматически порождён и его негативный атрибут-напарник *не является исполняемым языком*.) Введение негативных атрибутов (которые также могут иметь неопределённые значения) позволяет частично решить проблему дополнения для понятий/классов: в дополнение до какого-либо элементарного класса, определённого позитивным атрибутом, попадают только те языки, у которых явно проставлено значение соответствующего негативного атрибута. Например, язык программирования C не является языком спецификаций в онтологии, если только явно проставлено значение его соответствующего негативного атрибута (то есть такое мнение представлено в онтологии).

Элементарные роли/свойства — это естественные отношения между компьютерными языками: *быть диалектом*, *наследовать синтаксис* и так далее. Например: “C-light является промежуточным слоем C”, “OWL наследует синтаксис XML” и так далее. Для построения сложных ролей/свойств из элементарных можно использовать стандартный набор монотонных операций алгебры бинарных отношений: объединение, пересечение, инверсию (взятие обратного отношения), транзитивное замыкание. Например, роль/свойство *использует синтаксис диалекта* — это просто композиция элементарных ролей/свойств *использует синтаксис* и *является диалектом*. (Однако с операцией дополнения ролей/свойств возникают те же трудности, что и с операцией дополнения понятий/классов, но для ча-

стичного преодоления этих затруднений использовать подход, основанный на создании негативного напарника для каждой позитивной элементарной роли/свойства.)

В мире компьютерных языков есть три роли/свойства, специфические для этой предметной области: *быть диалектом*, *быть версией*, *быть реализацией*. (Эти роли/свойства используются в [36, 41].) Разумеется, эти роли/свойства должны быть элементарными в этой онтологии и задаваться явным перечислением пар языков. Но для пользователя онтологией надо предусмотреть правило, которыми следует руководствоваться при причислении пар языков к одному из этих отношений. К сожалению, нет консенсуса по определению этих отношений между языками. Так, например, Progopedia [41] считает, что реализация имеет несколько версий, а The Language List [36] считает наоборот, что версия может иметь несколько реализаций. Поэтому в [9] было предложено закрепить следующие определения:

- Диалекты — это языки у которых общий элементарный уровень. Например, Common LISP и Home LISP — это диалекты друг друга.
- Версии (или варианты) — это языки с общим основным уровнем. Например, Visual C и Borland C — это, наверное, версии друг друга.
- Реализация — это платформа-специфическая версия языка. Например, Visual C — это версия языка C Карнигана и Риччи для платформы Windows.

В 2012 г. альфа-версия онтологии компьютерных языков (ограниченной функциональности) была реализована А.А. Акининым в виде портала знаний по классификации компьютерных языкаов и доступна в течении нескольких месяцев для ознакомления в сети Интернет (по адресу `http://complang.somee.com/Default.aspx`). Этот портал был реализован как веб-приложение, интерфейс позволял пользователю просматривать и редактировать основные элементы онтологии — это компьютерные языки (объекты), элементарные свойства/классы объектов, элементарные роли/отношения между объектами, атрибуты (аннотации объектов) и базу знаний (совокупность предложений DL, которые претендуют на то, что бы быть законами универсума компьютерных языков). Для внутреннего представления данных использовался RDF-репозиторий. Были реализованы два основных сервиса: это решатель запросов как основное средство выделения классов, навигации по онтологии и проверки совместности, и визуализация онтологии в виде графа. Решатель — это верификатор моделей с явным представлением состояний (explicit state model checker), но для варианта DL, построенного не над булевской двузначной логикой,

а над четырехзначной логикой Бэлнапа [19], обогащённого алгебраическими конструкциями из анализа формальных понятий (Formal Concept Analysis — FCA) [13, 30]. Первоначальное наполнение портала было произведено путём импорта данных о компьютерных языках из открытых хорошо структурированных и размеченных источниках в Интернете — Progopedia [41] и DBpedia (проект, направленный на извлечение структурированной информации из данных, созданных в рамках проекта Википедия); фиксированная структура и разметка данных в Progopedia и DBpedia позволяли осуществить поиск данных простым парсингом страниц Progopedia и DBpedia.

Однако, (как уже было сказано в части 1) к 2013 г. проект создания классификации компьютерных языков и соответсвующего компьютерного инструментария столкнулся с рядом вызовов, на которые в 2013 г. участники не смогли в то время ответить. Это, во-первых, наступление эры проблемно-ориентированных компьютерных языков, о большинстве которых никогда не появится информация ни в Progopedia, ни в Wikipedia, ни в DBpedia и даже в научной периодике по компьютерным языкам, но только в литературе по конкретной предметной области, для которой создавался тот или иной компьютерный язык.

Во-вторых, для компьютерных языков имеет значение аспект *популярности* языка, которую можно определить по-разному. Например, популярность можно оценить при помощи так называемого индекса TIOBE (TIOBE programming community index) [46]. Этот индекс оценивает популярность языков программирования на основе подсчёта результатов поисковых запросов, содержащих название языка, на порталах Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo!, Bing, Amazon. (Можно предположить, что существует связь между количеством найденных страниц и количеством программных проектов, разработчиков и вакансий, связанных с тем или иным языком.) Однако, такое определение популярности не учитывает развитость социальной сетей (сообществ) профессионалов, студентов и любителей, вовлечённых в жизненный цикл конретных языков, сервиса (поддержки) ответов на вопросы со стороны сообществ и компаний.

В результате проект созданий классификации компьютерных языков и соотвествующего компьютерного инструментария был на время отложен (причём, сожалению, когда ещё не было опубликовано ни формальное описание использованной логики, ни алгоритма решателя логических запросов.)

Однако с 2013 г. прошло достаточно времени, и сейчас в 2018 г. мы надеемся, что сумеем

дать ответ на вызовы, ранее остановившие проект классификации компьютерных языков и создания портала знаний о классификации компьютерных языков. Во-первых, для автоматического поиска и извлечения информации по компьютерным языкам можно попытаться применять методы извлечения информации из текстов [22] (используя для первоначального наполнение портала импорт данных из Progopedia и DBpedia). И, во-вторых, для определения популярности компьютерных языков — методы анализа социальных сетей [37] (на ряду с индексом TIOBE).

4. Заключение

И так, подведём итог данной статьи. Во-первых, настоящая статья относится к жанру position paper: "Position papers in academia enable discussion on emerging topics without the experimentation and original research normally present in an academic paper. Commonly, such a document will substantiate the opinions or positions put forward with evidence from an extensive objective discussion of the topic."[3] Однако, особенность настоящей статьи является то, что, с одной стороны, она написана по материалам предыдущих исследований, выполненных в период 2008-2013 гг., а, с другой стороны, — после значительного перерыва в пять лет перед возобновлением приостановленных исследований, но уже с использованием новых подходов.

Сумма основных результатов исследований в период 2008-2013 гг.:

- Была выработана концепция онтологического подхода к классификации компьютерных языков, основанного на гибком понимании парадигм языков, которые определяются всей совокупностью характеристик языка и его связей с другими языкам.
- Была осознано значение компьютерной поддержки онтологии компьютерных языков и эффективного языка запросов для навигации по этой онтологии, причём, язык запросов должен поддерживать дискрипционные темпоральные временны́е запросы в модели открытого мира (всегда неполной информации).
- Был реализован прототип портала реализующий статический фрагмент онтологии компьютерных языков (содержавший сведения по более чем 1200 языкам программирования, извлечёнными из структурированных и размеченных DBpedia и Progopedia) и языка запросов на основе дискрипционной логики над четырёхзначной логикой Белнапа.

[3] Цитируется по [40].

Задачи, которые надо решить в ближайшее время для продолжение работ по проекту классификации и навигации по миру компьютерных языков:

- Разработать инструменты (основанные на методах анализа текстов на естественном языке и машинного обучения) для извлечения сведения о компьютерных языках из текстовых интернет-ресурсов и социальных сетей (используя в качестве начального "знания" (обучающей выборки) "знания" из структурированных и размеченных DBpedia и Progopedia).
- Разработать схему публичной, открытой, эволюционирующей, темпоральной и со временем онтологии и архитектуру портала знаний о классификации компьютерных языков, а затем — приступить к прототипированию этого портала (используя в качестве начального наполнения сведения о языках программирования из DBpedia и Progopedia) и интеграции его с инструментами извлечения сведений о компьютерных языках из интернет-ресурсов.
- Формально описать дискрипционный темпоральный временно́й логический язык запросов над четырёхзначной логикой Белнапа, исследовать алгоритмы верификации запросов на этом языке в конечных открытых моделях, прототипировать решатель таких запросов и интегрировать его с порталом знаний о классификации компьютерных языков.

Список литературы

1. Андреева Т. А., Ануреев И. С., Бодин Е. В., Городняя Л. В., Марчук А. Г., Мурзин Ф. А., Шилов Н. В. Образовательное значение классификации компьютерных языков // Прикладная информатика. 2009. № 6(24). С.18-28.
2. Ахмадеева И.Р., Боровикова О.И., Загорулько Ю.А., Сидорова Е.А. Сбор онтологической информации для интеллектуальных научных Интернет-ресурсов // Системная информатика. 2014. № 3. С. 13–23.
3. Боровикова О.И., Загорулько Ю.А. Подход к реализации паттернов содержания при разработке онтологий научных предметных областей // Системная информатика. 2018. № 12. С. 27–40.
4. Гаранина Н.О., Зюбин В.Е., Лях Т.В. Онтологический подход к организации шаблонов требований в рамках системы поддержки формальной верификации распределенных программных систем // Системная информатика. 2017. № 9. С. 111–132.
5. Городняя Л.В. Парадигмы программирования: анализ и сравнение. Новосибирск: Издательство СО РАН, 2017. 232 с.
6. Городняя Л.В. Парадигмальный подход к факторизации определений языков и систем про-

граммирования // Системная информатика. 2018. №12. С. 1-26.

7. Дмитриев С. Языково-ориентированное программирование // RSDN Magazine. 2005, №5. [Электронный ресурс]. Систем. требования: любой интернет-браузер. http://rsdn.org/article/philosophy/LOP.xml (дата обращения: 15.12.2018).
8. Ершов А.П. Предварительные соображения о лексиконе программирования // Избранные труды. Новосибирск. 1994. С. 395–406.
9. Идрисов Р.И., Одинцов С.П., Шилов Н.В. Онтологический подход к проблеме классификации компьютерных языков: состояние и перспективы // Системная информатика. 2013. № 1. С. 63-78.
10. Карпов Ю.Г. Model Checking. Верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2010. 552 с.
11. Непейвода Н.Н. Стили и методы программирования. Курс лекций. Учебное пособие. М.: Бином. Лаборатория знаний / Интернет-Университет Информационных Технологий (ИНТУИТ), 2005. 320 с.
12. Непомнящий В.А., Ануреев И.С., Дубрановский И.В., Промский А.В. На пути к верификации C♯-программ: трехуровневый подход // Программирование. 2006. № 4. С. 4–20.
13. Шилов Н.В. Формализмы и средства создания и поддержания онтологий. В "Модели и методы построения информационных систем, основанных на формальных, логических и лингвистических подход" коллективная монография под ред. Марчука А.Г.. Новосибирск: из-во СО РАН. 2009. С.10-48.
14. Akhmadeeva I.R., Zagorulko Y.A., Mouromtsev D.I. Ontology-based information extraction for populating the intelligent scientific internet resources // Communications in Computer and Information Science. 2016. Vol. 649. P. 119–128
15. Shilov N.V., Akinin A.A., Zubkov A.V., and Idrisov R.I. Development of the Computer Language Classification Portal // Springer Lecture Notes in Computer Science. 2012. Vol. 7162. P. 340-348.
16. Akinin A.A., Zubkov A.V., Shilov N.V. New Developments of the Computer Language Classification Knowledge Portal // Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012), May 30-31, Perm, Russia. Moscow: Institute of System Programming. 2012. P. 54-58.
17. Anureev I., Bodin E., Gorodnyaya L., Marchuk A., Murzin F., Shilov N. On the problem of computer language classification // Computer Science. 2008. №28 P. 31–42.
18. Baader F., Calvanese D., McGuinness D.L., Nardi D., Patel-Schneider P.F. The Description Logic Handbook: Theory, Implementation, Applications. Cambridge: Cambridge University Press, 2003. 624 p.
19. Belnap N.D. How a computer should think // Contemporary Aspects of Philosophy: Proceedings of the Oxford International Symposium. Oriel Press. 1977. P.30–56. (Есть русский перевод: Бэлнап Н. Как нужно рассуждать компьютеру // Белнап Н., Стил Т. Логика вопросов и ответов. М.: Прогресс. 1981. [Электронный ресурс]. Систем. требования: любой интернет-браузер. URL: http://scicenter.online/logika-scicenter/kak-nujno-rassujdat-62531.html (дата обращения:

15.12.2018).)

20. Anureev I.S., Promsky A.V. Conceptual transition systems and their application to development of conceptual models of programming languages // System Informatics. 2017. N 9. P. 133–154.
21. Anureev I.S. Operational conceptual transition systems and their application to development of conceptual operational semantics of programming languages // System Informatics. 2017. N 9. P. 155–200.
22. Baeza-Yates R., Ribeiro-Neto B. Modern Information Retrieval: The Concepts and Technology behind Search (second edition). Addison-Wesley, 2011. 944 p.
23. Börger E., Stärk R. Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag, 2003. 438 p.
24. Efftinge S., Köhnlein J., Friese P. Build your own textual DSL with Tools from the Eclipse Modeling Project. 2008. [Электронный ресурс]. Систем. требования: любой интернет-браузер. http://www.eclipse.org/articles/article.php?file=Article-BuildYourOwnDSL/index.html (дата обращения: 15.12.2018).
25. Floyd R.W. The paradigms of Programming // Communications of ACM. 1979. Vol. 22. P. 455–460. (Есть русский перевод: Флойд Р. О парадигмах программирования // Лекции лауреатов премии Тьюринга за первые двадцать лет. 1966–1985. М.: Мир, 1993. 560 с.)
26. Fowler M. Domain-Specific Languages. Addison-Wesley, 2011. 640 p. (Есть русский перевод: Фаулер М. Предметно-ориентированные языки программирования. Вильямс, 2017. 576 с.)
27. Garanina N.O., Zubin V., Lyakh T., Gorlatch S. An Ontology of Specification Patterns for Verification of Concurrent Systems // In: New Trends in Intelligent Software Methodologies, Tools and Techniques. Proceedings of the 17th International Conference SoMeT-18, Granada, Spain, 26-28 September 2018. H. Fujita and E. HerreraViedma (Eds.). Frontiers in Artificial Intelligence and Applications. Vol. 303. Amsterdam: IOS Press. 2018. P. 515–528.
28. Gurevich Yu. Evolving Algebras: An Attempt to Discover Semantics // Current Trends in Theoretical Computer Science / Eds. G. Rozenberg and A. Salomaa. World Scientific, 1993. P. 266–292.
29. Gurevich Yu. Sequential Abstract State Machines capture Sequential Algorithms // ACM Transactions on Computational Logic. 2000. Vol. 1. №1. P. 77–111. (Есть русский перевод: Гуревич Ю. Последовательные машины абстрактных состояний охватывают последовательные алгоритмы // Системная информатика. Вып. 9 / Пер. П.Ф. Емельянова. Новосибирск: Изд-во СО РАН, 2004. С. 7–50.)
30. Handbook on Ontologies. International Handbooks on Information Systems / Editors Staab S., Studer R. Springer. 2009. 2nd edition. 660 p.
31. Hitzler P., Krötzsch M., Rudolph S. Foundations of Semantic Web Technologies. Chapman & Hall/CRC, 2009. 456 p.
32. JetBrains MPS. 2006-2017. [Электронный ресурс]. Систем. требования: любой интернет-браузер. http://www.jetbrains.com/mps/ (дата обращения: 15.12.2018).
33. Internationa Conference on Software Languages Engineering. 2018. [Электронный ресурс]. Си-

стем. требования: любой интернет-браузер. URL: http://www.sleconf.org (дата обращения: 15.12.2018).

34. Kinnersley B. The Language List. [Электронный ресурс]. Систем. требования: любой интернет-браузер. https://web.archive.org/web/20160506170543/http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm (дата обращения: 15.12.2018).
35. Kuhn T.S. The structure of Scientific Revolutions. Univ. of Chicago Press, 3rd Ed., 1996. (Есть русский перевод: Кун Т. Структура научных революций. М.: Издательство АСТ, 2003. 605 с.)
36. Levenez E. Computer Languages History. [Электронный ресурс]. Систем. требования: любой интернет-браузер. https://www.levenez.com/lang/ (дата обращения: 15.12.2018).
37. McCulloh I., Armstrong H., Johnson A. Social Network Analysis with Applications. Wiley, 2013. 320 p.
38. Ontology: Theory and History. [Электронный ресурс]. Систем. требования: любой интернет-браузер. https://www.ontology.co. (дата обращения: 15.12.2018).
39. Pigott D. HOPL: an interactive Roster of Programming Languages. 1995-2006. [Электронный ресурс]. Систем. требования: любой интернет-браузер. https://web.archive.org/web/20080511193953/http://hopl.murdoch.edu.au/ (дата обращения: 15.12.2018).
40. Position paper. From Wikipedia, the free encyclopedia. [Электронный ресурс]. Систем. требования: любой интернет-браузер. https://en.wikipedia.org/wiki/Position_paper (дата обращения: 15.12.2018).
41. Progopedia. [Электронный ресурс]. Систем. требования: любой интернет-браузер. http://progopedia.ru/ (дата обращения: 15.12.2018).
42. van Roy P. Classification of the principal programming paradigms. 2009. [Электронный ресурс]. Систем. требования: любой интернет-браузер. http://www.info.ucl.ac.be/∼pvr/paradigms.html (дата обращения: 15.12.2018).
43. van Roy P. Programming Paradigms for Dummies: What Every Programmer Should Know // New Computational Paradigms for Computer Music. IRCAM/Delatour, France. 2009. P. 9–38.
44. Ruby. A Programmer's best friend. [Электронный ресурс]. Систем. требования: любой интернет-браузер. URL:http://www.ruby-lang.org/en/about/ (дата обращения: 15.12.2018).
45. Stärk R., Schmid J., Börger E. Java and the Java Virtual Machine. Springer, 2001. 381 p.
46. TIOBE Index | TIOBE — The Software Quality Company [Электронный ресурс]. Систем. требования: любой интернет-браузер. https://www.tiobe.com/tiobe-index/ (дата обращения: 15.12.2018).
47. Tudorache T., Nyulas C. I., Musen M. A., Noy N. F. WebProtégé: A Collaborative Ontology Editor and Knowledge Acquisition Tool for the Web // Semantic Web Journal. 2011. Vol. 11. № 165. P. 1–11.
48. Walther Ch., Schweitzer St. About VeriFun // Springer Lecture Notes in Computer Science. 2003. Vol. 2741. P. 322–327 .
49. Xtext 2006-2017. [Электронный ресурс]. Систем. требования: любой интернет-браузер.

http://www.eclipse.org/Xtext/ (дата обращения: 15.12.2018).

50. Zagorulko Y., Borovikova O., Zagorulko G. Pattern-Based Methodology for Building the Ontologies of Scientific Subject Domains. In: New Trends in Intelligent Software Methodologies, Tools and Techniques. Proceedings of the 17th International Conference SoMeT-18, Granada, Spain, 26-28 September 2018. H. Fujita and E. HerreraViedma (Eds.). Frontiers in Artificial Intelligence and Applications. Vol. 303. Amsterdam: IOS Press. 2018. P. 529–542.

УДК 004.6

Архитектура и основные особенности библиотеки PolarDB работы со структурированными данными

Марчук А.Г. (Институт систем информатики СО РАН, Новосибирский государственный университет)

В статье представлен анализ созданной в ИСИ СО РАН библиотеки PolarDB работы со структурированными данными. Это – библиотека классов, реализованная на платформе .NET Core. Она предназначена для создания систем структурирования, хранения и обработки данных, в том числе большого объема. Библиотека построена на ранее разработанной системе рекурсивной типизации и охватывает ряд существенных задач, таких как структурирование данных, сериализация, отображение данных на байтовые потоки, индексные построения, блочная реализация динамических байтовых потоков, распределение данных и обработки данных, резервное копирование и восстановление. Применение библиотеки PolarDB позволяет создавать эффективные решения для специализированных баз данных в разных парадигмах: последовательности, реляционные таблицы, key-value хранилища, графовые структуры.

Ключевые слова: *структурированные данные, типизация данных, базы данных, PolarDB, Big Data.*

1. Введение

При построении информационных систем, практически единственным решением относительно базы данных, является применение той или иной универсальной СУБД. В Институте систем информатики СО РАН был предложен и реализован подход, связанный с программированием требуемых решений средствами библиотеки. Подход был сформирован на основе системы типизации, имеющейся в языке программирования Поляр [1]. Начальная часть библиотеки формировалась в рамках проектов по исторической фактографии [2]. Первый рабочий вариант собственно системы PolarDB был представлен в докладе [3]. Подход и библиотека были опробованы при реализации некоторых системных и прикладных проектов [4, 5]. Библиотека используется в учебном курсе «Методы и технологии обработки больших данных», созданном по заказу АО «ВымпелКом» при поддержке Министерства науки и образования Новосибирской области.

2. Концептуальный базис

Структурное значение (structured value, поляровское структурное значение, p-value) - древовидное построение (значение), интерпретируемое в соответствии с заданным типом.

Тип (Type, поляровский тип, p-type) – древовидное построение, задающее интерпретацию для структурных значений. Может быть выражено как структурное значение. Тип может быть примитивным (атомарным) или конструируемым (составным).

Объект базы данных – сформированное в некотором рабочем поле или в хранилище структурное значение.

Объектное представление структурного значения – внутренняя конструкция в ОЗУ, вместе с типовым значением задающая структурное значение. Объектное представление реализуется средствами какой-то (напр. .NET) системы программирования, доступно через язык программирования и служит связующим средством между программными объектами и объектами базы данных по следующей схеме: объект базы данных полностью или частично, отдельными полями, может быть реализован значениями объектного представления. Соответственно, возможно "чтение" объектов базы данных или их полей в объектное представление. И наоборот, возможна "запись" значений объектного представления в объекты базы данных или их поля.

Текстовая сериализация (текстовое представление) структурного значения – точные правила представления любого структурного значения последовательностью символов. Текстовое представление, совместно с явно или неявно определенным типом этого представления, изображает структурное значение

Бинарная (байтовая) сериализация структурного значения – правила представления любого структурного значения в виде потока байтов. Бинарная сериализация осуществляется только в контексте типа сериализуемого или десериализуемого структурного значения.

Примитивные типы:

boolean, character, integer, longinteger, real – логическое, символьное, целое (32 разряда), длинное целое (64 разряда), число с плавающей точкой (64 разряда), byte – байт – 8-разрядный код, none – множество значений, не содержащее ни одного элемента, string – обычные строки объектно-ориентированного программирования (C#).

Конструируемые типы:

Запись (record) – фиксированный набор типизированных полей. Есть поля 0, 1, ... n-1, каждое из которых представляет значение заданного в определении записи типа. Количество

полей – фиксировано, типы полей – фиксированы. Иногда полям сопоставляют имена, являющиеся идентификаторами.

Последовательность (sequence) – упорядоченный набор, состоящий из неопределенного (ноль или более), но конкретного, числа однотипных элементов.

Объединение (union) – значение, состоящее из тега и подзначения. Тег (динамически) определяет вариант типа для подзначения. Теги нумеруются, начиная с 0.

Объектное представление. Любое структурное значение может быть представлено в виде объекта по следующей схеме: примитивные типы реализуются значениями соответствующих системных типов: логического, целого, длинного целого, с плавающей точкой (double), символьного (char), байта, строки. Значения типа none представляются значениями null. Конструируемые типы представляются массивами object[], элементами которых будут подзначения структур. Запись – массив объектных значений своих элементов, последовательность – массив объектных значений элементов. Значение объединенного типа представляет собой массив из двух элементов. Первый элемент – целое значение тега. Второй элемент – объектное представление подзначения соответствующего тегу типа.

Примеры. Объектные значения (object)22, (object)"demo string", (object)true могут быть проинтерпретированы только как целое, строка и логическое. Объект new object[] {2, 33} может быть проинтерпретирован в зависимости от типа или как запись двух целых или как последовательность целых или как объединение с вариантом 2 и подзначением 33. Для последовательности записей, состоящих из строкового и целого, значением в объектном представлении может быть: new object { new object[] {"str1", 111}, new object[] {"str2", 222} }

Текстовая сериализация. Атомарные значения изображаются в виде текста: типа none – пустой строкой, типа boolean, character, integer, longinteger, real – традиционно, как напр. в C#. @byte изображается 16-ричным кодом, строка – как обычно (в C#). Запись изображается заключенным в фигурные скобки перечислением через запятую значений всех полей записи, начиная с нулевого и далее по порядку. Возможно использование имен полей (как в Паскале). Последовательность, изображается перечислением элементов через запятую, все перечисление (нуль или более элементов) помещается в квадратные скобки. Объединение изображается тегом – числом в диапазоне 0-255, следующим за ним символом ^ и далее идет изображение значения того типа, который динамически задан тегом. Значение может быть помещено в круглые скобки, это нужно для определенности разбора. По структурному значению и его типу однозначно, с точности до несущественных синтаксических элементов (пробелы, перевод строки, tab) и "лишних" скобок, определяется текстовая развертка. По корректной текстовой развертке и типу, однозначно определяется структурное значение.

Примеры. Для структурного значения типа последовательности записей строкового и целого полей, заданного в объектном виде как new object { new object[] {"str1", 111}, new object[] {"str2", 222} }, текстовая сериализация будет: [{"str1", 111}, {"str2", 222}].

Бинарная сериализация. Бинарная сериализация выполняется следующим образом: атомарные значения отображаются на последовательность байтов так, как определяется системой программирования .NET (C#) через метод System.IO.BinaryWriter и двойственный ему System.IO.BinaryReader. Байт отображается в байт, целое в 4 байта, длинное целое - в 8 байтов, число с плавающей точкой в 8 байтов, значение типа none - в 0 байтов. Строка отображается сложнее. Сначала идет целое (1 или более байтов) число, равное количеству символов в строке. Далее, если количество символов больше 0, то идут символы в виде потока байтов, сформированных по правилам UTF-8: byte[] info = new UTF8Encoding(true).GetBytes(str). Запись сериализуется поставленными "встык" сериализованными значениями полей. Последовательность вначале имеет 64-разрядное (8 байтов) целое, фиксирующее количество элементов последовательности (0+), а за ним подряд идут соответствующее количество значений элементов последовательности. Объединение реализуется постановкой 1 байта, который фиксирует код тега (0-255), за этим байтов непосредственно следует подзначение соответствующего тегу типа.

Язык определения типов. Поляровские типы задаются некоторыми формулами, задающими типовые (ударение на первый слог) значения. Типовые значения обычно создаются специальными конструкторами классов PType, PTypeRecord, PTypeSequense, PTypeUnion. Кроме того, эти значения можно рассматривать как обычные структурные значения и существуют процедуры перевода из поляровского объектного представления в типовое и обратно. Также существует (как и в других языках) специальный язык определения типов. Синтаксис языка в БНФ:

```
типовое_определение: имя_типа = типовая_формула ;
типовая_формула: none
     | bool | byte | char | int | long | float
     | { (имя поля : )? типовая_формула,.. }
     | [ типовая_формула ]
     | имя_тега ^ типовая формула
```

3. Общая часть библиотеки программ

Есть базовая часть рассматриваемых решений – пространство имен Polar.DB. В сборке размещен ряд ключевых компонентов, применяемых в других пакетах библиотек. Наиболее принципиальные компоненты связаны с типами. Как уже отмечалось, поляровское структурное значение осмыслено в контексте его типа. А тип, это также структурное

значение, формируемое или статически, как константа в коде, или динамически, с помощью подходящих средств. Отметим рекурсивность ситуации с типами: значение осмысленно только в контексте другого значения. Соответственно, у типового значения также есть тип, в библиотеке он задается статическим значением TType.

Объекты поляровского типа имеет класс PType, конструктор этого класса позволяет конструировать стандартные простые типы: булевское, целое, вещественное, строка и др. Составные типы создаются на базе классов PTypeRecord, PTypeSequence, PTypeUnion, формирующих типы записи, последовательности и объединения соответственно. Кроме того, есть возможность превращать типовые значения в объекты и наоборот – объекты трансформировать в типовые значения. Такая гибкость дает средства комбинирования типов и значений. Например, можно записать в файл сначала типовое значение, а потом значение этого типа. Соответственно, при чтении, корректной будет последовательность чтения объекта типа (на базе TType) с переводом его в тип, напр. T, с последующим чтением объекта (на базе T).

В целом, тип может быть создан, у него есть признаки (Properties) и атрибуты, но главное – он может быть использован при работе со структурированными данными. Как уже указывалось ранее, поляровские структурированные данные имеют объектное представление (в оперативной памяти) и два вида разверток – текстовую и байтовую. Текстовая развертка – это текстовое представление структурного значения. Оно несколько похоже на JSON [6], однако не является самодостаточным. Это означает, что для правильной интерпретации текстового представления поляровского значения все равно требуется явно указанный тип, хотя в простых случаях, текст легок для понимания. Есть ряд базовых процедур, затрагивающих «тройку» тип, объект, среда развертки в процедурах сериализации/десериализации:

```
public static void TextFlow.Serialize(TextWriter tw, object v, PType tp);
public static object TextFlow.Deserialize(TextReader tr, PType tp);
public static void ByteFlow.Serialize(BinaryWriter bw, object v, PType tp);
public static object ByteFlow.Deserialize(BinaryReader br, PType tp);
```

Эти процедуры или формируют поток символов/байтов из объектного представления значения (сериализация) или создают объект значения по имеющемуся потоку (десериализация).

В общей части библиотеки PolarDB также имеется поддержка для основной структуры для многих построений – последовательности. В частности, есть поддержка последовательностей с нефиксированным размером элементов и с фиксированным размером, есть добавление элементов, «очищение» последовательности, выстраивание индексов, использование шкалы.

Эта часть библиотеки предназначена для программирования самых быстрых решений по обработке, в которых особенности структуризации, хранения и доступа, добавляются программно, а не за счет универсализма элементов и действий.

4. Поляровские ячейки и индексы

Некоторая фиксация ряда решений, связанных с отображением структур на потоки байтов (Streams), делается в пакетах Polar.Cells и Polar.CellIndexes. Ячейки класса PaCell – это специально организованные байтовые потоки, как правило – файлы, для хранения структурированных данных и манипулирования этими данными. Естественно, каждая ячейка предназначена для хранения данных только одного типа. Ячейку можно создать или, если уже существует, подсоединиться. В ячейку можно положить значение предписанного типа или прочитать значение. Можно сказать, что ячейка – «внешняя» память для структурированных значений. Эта память может быть уже занятой значением и тогда другое значение не запишется, но есть явный метод очищения (Clear()) ячейки, тогда можно снова записывать.

К динамически изменяемым базам данных ячейка имеет то отношение, что последовательность верхнего уровня может не только записываться целиком, но и расти через последовательное добавление элементов. Для считывания из ячейки не всего значения, а части, существует механизм выделения части с последующим чтением. Это делается через некоторый аналог указателей – структуру класса PaEntry. Технология здесь следующая. Значение класса PaEntry указывает на часть общего значения – на подзначение. Это подзначение обладает каким-то типом и этот тип непосредственно динамически зафиксирован в указателе. А далее, есть набор методов «углубления» в подзначение – выделение поля для записи, выделение элемента в последовательности и выделение варианта в объединении. А в конце углублений значение можно прочитать. Или даже записать вместо имеющегося в поле содержимого, новое значение. Но это допустимо только для полей фиксированного размера, таких как числа, байты и записи, состоящие из подзначений фиксированного размера. Фиксированный размер определен для типа или он зависит от значения. Например, числа являются значениями фиксированного размера, а строки – не являются.

В набор методов работы с полями также входят сканирования элементов последовательности, когда вырабатывается поток значений имеющихся значений или для каждого элемента выполняется заданное обработчиком действие.

В пространстве имен Polar.CellIndexes библиотеки расположено конкретное индексное решение, точнее некоторое семейство решений. Суть его заключается в том, что индексное построение для последовательности (в частном случае – таблицы) выстраивается «вокруг» класса TableView, который представляет собой последовательность элементов, заданного в определении поляровского типа, снабженных дополнительным логическим признаком deleted. Если этот признак истинен, то элемент считается уничтоженным. Эта конструкция позволяет выполнять полный набор редактирующих операций: добавление элемента, уничтожение элемента и изменение элемента (через уничтожение и добавление нового). Причем уничтожение элемента выполняется не «физическим изъятием», а отметкой в соответствующем поле.

Индексные построения выполняются по специальной схеме, когда формируется «статическая» часть индекса в виде сортируемой последовательности, и она расширяется динамической частью индекса, выполненной в виде структуры в оперативной памяти (применяется хеш-таблица Dictionary). Индексы создаются относительно независимо и регистрируются в опорной таблице. Соответственно, далее операции, требующие отработки в индексной структуре, автоматически доносятся до структуры через регистрацию. При создании индекса, задается функция вычисления ключа на значениях элементах последовательности. Также указывается таблица, к которой относится индекс и, возможно, шкала. Шкала – это отображение значений числового ключа или полуключа в диапазон отсортированного индексного массива в котором элементы с такими значениями могут находиться. Такое отображение делается через равномерное разделение множества значений ключа с фиксацией в отдельном массиве (шкале) получившихся диапазонов.

Достоинством такай схемы индексного построения, является высокая эффективность работы и использования оперативной памяти. Недостатком является необходимость периодического перестроения индекса, в том числе – в динамике обслуживания запросов. Недостаток не проявляется для типичных ситуаций применения динамических индексов. Статическая часть индекса (индексный массив) различается в зависимости от того, производится ли индексация по ключу фиксированного размера (IndexKeyImmutable), произвольному ключу (IndexViewImmutable) или полуключу, когда используется хеш-функция ключа (IndexHalfkeyImmutable). Причем для первого и третьего вариантов возможно использовать шкалу, что заметно ускоряет доступ к хранимым элементам.

В пространстве имен Polar.CellIndexes имеется также 2 сформированных решения для типичных случаев применения библиотеки. Первое – универсальная таблица имен (NameTableUniversal), второе – простая таблица (TableSimple). Суть таблицы имен

заключается в порождении биекции между множеством строковых значений и множеством числовых кодов. Причем ее организация оптимизирует по времени преобразование строка –> код и обратно код –> строка. Подобные построения востребованы в современных информационных технологиях, в том числе, связанных с большими данными, их место, в основном в реализации key-value хранилищ данных.

Простая таблица также формируется на основе универсальной таблицы и универсальных индексов раздела. Но для упрощения и задания таблицы, все делается в «стандартном» для парадигмы реляционных таблиц представлении, когда задаются столбцы и на столбцы «навешиваются» индексы. Соответственно, задание таблицы довольно лаконично и интуитивно понятно. Например, конструктор класса выглядит как:

```
    public TableSimple(PType tp_elem, int[] indexcolumnnoms, Func<Stream>
getstream);
```

В нем задается тип элементов, предполагается, что это запись (набор колонок), второй аргумент определяет номера индексированных колонок, функция задает генератор байтовых потоков (Streams), нужных для выполнения построения. Генератор потоков должен быть таким, чтобы при запуске конструктора простой таблицы, потоки бы создавались, а при уже созданных потоках, происходило бы подключение к ним.

5. Странично организованные потоки байтов

Пространство имен Polar.PagedStreams предназначено для реализации программными средствами произвольного набора потоков байтов (Streams). В качестве «строительного материала» для этого используются подходящие массивы байтов (блоки, страницы) одинакового размера. Данный раздел библиотеки позволяет решить множество задач, связанных с хранением данных и с доступом к данным. К таким задачам относятся: упаковка базы данных или СУБД в один файл или предписанное количество файлов, экономная работа с большим и очень большим (миллионы и более) числом потоков, использование специальных (своих) схем кеширования доступа к файловым объектам, построение универсальных и специализированных решений по обеспечению надежного хранения через задаваемую избыточность и схемы контроля и восстановления.

Общая схема логического устройства блочной или страничной памяти задается абстрактным классом SetOfBlocks. Пока в библиотеке имеется единственная реализация, расширяющая этот класс FileOfBlocks, использующая файл как источник блоков через простое «нарезание» потока байтов на логически независимые блоки. Соответственно, файловый набор блоков через методы предоставляет пользователям произвольное

количество объектов класса PagedStream, имеющий интерфейс Stream, будем говорить о FileofBlocks как о генераторе потоков.

Организация страничной памяти байтовых потоков похожа на классические решения в ОС Unix [7], в частности, дескриптор потока имеет вид:

```
struct {
      long stream_length, n_blocks;
      struct { long D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12;}
sub_blocks;
      long beginblock_length;
}
```

При этом, поток начинается как продолжение дескриптора, потом 10 длинных целых указывают на начала следующих страниц, D10 указывает на следующие страницы с одинарной косвенностью, D11, D12 – с двойной и тройной соответственно.

Существенным отличием примененного подхода является то, что нет встроенной системы директорий, директорных и файловых атрибутов. Считается, что эти свойства могут быть привнесены в конкретной программе и для этого, библиотечных средств достаточно. Например, простое key-value файловое хранилище можно организовать из трех потоков. В одном потоке хранится последовательность записей, состоящих из ключа, метаинформации (включая имя файла), начала и длины хранимого файла. Во втором потоке – байты подряд идущих файлов, в третьем потоке – индекс по ключу первой последовательности.

Создание простого кеша страниц для хранилища потоков, еще одно свойство реализации FileOfBlocks. В принципе, это свойство необязательно и может быть отключено, но если используется, то это может заметно повысить скорость доступа к данным.

6. Реализация, применения, эффективность

Библиотека реализована в системе .NET Core на языке программирования C# в виде открытого кода, расположенного в репозитории https://github.com/agmarchuk/PolarDB. Также она доступна через Nuget-пакеты с именами Polar.DB, Polar.Cells, Polar.CellIndexes, Polar.PagedStreams.

Поскольку .NET Core – многоплатформенное решение, библиотека может быть применена для программ под Windows, Linux, iOS. Первые два варианта успешно были проверены.

Библиотека применяется в проектах по исторической фактографии, ведущихся в ИСИ СО РАН, НГУ. Она используется в проекте «Открытый архив СО РАН» - открытой платформы построения и интеграции электронных архивов.

Данная библиотека активно используется в обучении, в частности в учебном курсе «Алгоритмы и технологии работы с большими данными» https://github.com/agmarchuk/BDLearningCourse.

Библиотека рассчитана на простые формы работы с базой данных, когда всю работу можно вести через один модуль или сервис, когда синхронизация запросов может выполняться в рамках последовательного исполнения и когда не требуется организовывать сложные транзакции. В таких условиях, библиотека показывает хорошие свойства и высокие характеристики, к которым относятся: компактность, способность работать на устройствах с малыми ресурсами, высокая скорость загрузки данных, высокая скорость доступа к данным, отсутствие потребности в сервере.

Список литературы

1. Марчук А.Г., Лельчук Т.И. Язык программирования Поляр: описание, использование, реализация. Новосибирск, 1986. 96 с. [Marchuk A.G., Lel’chuk T.I. Yazyk programmirovaniya Polyar: opisanie, ispol’zovanie, realizatsiya, Novosibirsk, 1986, 96 s. (in Russian)].
2. А.Г. Марчук, П.А. Марчук Платформа реализации электронных архивов данных и документов // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды XIV Всероссийской научной конференции RCDL'2012. Переславль-Залесский, Россия, 15-18 октября 2012 г. - г. Переславль-Залесский: изд-во "Университет города Переславля", 2012, С. 332-338.
3. Марчук А.Г. PolarDB - система создания специализированных NoSQL баз данных и СУБД // Моделирование и анализ информационных систем. Т. 21, № 6 (2014), с.169-175.
4. А.Г.Марчук, С.В.Лештаев Экспериментальная реализация Sparql-1.1 и RDF Triple Store // Аналитика и управление данными в областях с интенсивным использованием данных, XVII Международная конференция DAMDID/RCDL’2015, Обнинск, 13-16 октября 2015 года, Труды конференции, сс. 83-87.
5. А.Г.Марчук, С.В.Лештаев Электронный архив газет: Web-публикация, ассоциация информации с базой данных, создание полнотекстового поиска // Аналитика и управление данными в областях с интенсивным использованием данных, XVIII Международная конференция DAMDID/RCDL’2016, Ершово, Московская обл., Россия, 11-14 октября 2016 года, Труды конференции. Торус пресс, Москва, 2016. Сс. 155-160.
6. Введение в JSON // https://www.json.org/json-ru.html
7. Карпов В.Е., Коньков К.А. Основы операционных систем. Курс лекций // Учебное пособие / Под редакцией В.П. Иванникова. — Электронное издание. — М.: ИНТУИТ. РУ "Интернет университет информационных технологий", 2005. — 536 с. — ISBN: 5-9556-0044-2

УДК 004.82:004.912

Лексико-семантические шаблоны как инструмент декларативного описания языковых конструкций и лингвистического анализа текста

Тимофеев П.С. (Институт систем информатики СО РАН),
Сидорова Е.А. (Институт систем информатики СО РАН)

Статья посвящена проблемам извлечения языковых конструкций, в том числе числовых и символьных данных, значимых для заданной предметной области. Предложен подход к описанию естественно-языковых конструкций с помощью лексико-семантических шаблонов и рассмотрен язык описания шаблонов на основе языка YAML. Лексико-семантический шаблон – это структурный образец целевой языковой конструкции с указанным составом и лексико-семантическими свойствами. В случае успешного сопоставления шаблона с фрагментом текста формируется лексический объект, которому приписываются формальные (позиционные) и семантические (класс и свойства) характеристики. В статье представлена архитектура веб-редактора для разработки и тестирования лексико-семантических шаблонов и описан эксперимент по созданию двух специализированных словарей: 1) словарь наименований институтов, должностей, званий и их сокращений и 2) словарь числовых/временных конструкций. Создаваемая технология поддерживает лексико-семантический анализ текста на основе шаблонов и может быть использована как независимо при решении задачи извлечения информации из небольших текстов, так и в составе других систем извлечения информации. Предлагаемый метод эффективен для распознавания параметрических конструкций, содержащих оценку значений параметров объектов (сущностей или событий) предметной области.

Ключевые слова: *лексико-семантический шаблон, извлечение языковых конструкций, язык описания шаблонов, предметно-ориентированный словарь.*

1. Введение

Необходимость извлечения информации из огромного количества существующих и постоянно появляющихся текстов на естественном языке делает востребованным развитие методов интеллектуального анализа текста. При анализе текста большой пласт информации

представляется языковыми конструкциями, не обрабатываемыми инструментами, основанными на морфологических словарях. В первую очередь это:

- числовые данные заданные в явной форме (*год, вес, расстояние*);
- символьные данные (сокращения, аббревиатуры, номера телефонов и т. д.);
- параметрические конструкции, т.е. языковые конструкции содержащие оценку значения параметров объектов выраженных не только в явной форме (лексически или числом), но и в неявной форме (например, в сравнении с другими параметрами).

Данные языковые конструкции распространены в различного рода документах - медицинских протоколах, научно-технической литературе, производственной документации и др. Они, как правило, являются значимыми с точки зрения информационного содержания документа и часто интерпретируются в зависимости от предметной области и решаемой задачи. Для извлечения такого рода конструкций используются либо регулярные выражения, либо специализированные языки шаблонов, которые ориентированы на специалистов филологов и экспертов в различных областях знаний.

В литературе, в зависимости от типа учитываемой в конструкции языковой информации, шаблоны подразделяются на грамматические [7], лексико-грамматические [5] и лексико-синтаксические [1,6]. К грамматическим относятся шаблоны, в которых указываются грамматические характеристики слов, входящих в состав языкового выражения. В лексико-грамматических шаблонах также могут указываться конкретные лексемы, а лексико-синтаксические обеспечивают проверку согласования входящих лексем по грамматическим характеристикам. Обычно, все эти виды шаблонов используются для извлечения терминов (в том числе многословных) в задачах построения словников. Следует отметить язык Jape [8], который позволяет записывать правила распознавания языковых конструкций и формировать на их основе атрибутно-объектную модель текста для последующей работы на языке Java. На основе данного языка была разработана система выделения специфических объектов из текста RCO Pattern Extractor [2]. Данная система использует собственные язык и редактор правил, но является коммерческой и закрытой, что затрудняет ее исследование и использование в научных проектах.

Разрабатываемое в нашем коллективе решение позволяет анализировать произвольные символьные конструкции и формировать на их основе множество объектов предметной области. В рамках данного решения формируются предметно-ориентированные словари лексико-семантических шаблонов и разрабатывается инструментарий, обеспечивающий разработку, тестирование и применение словарей в лингвистическом анализе текста. В предыдущих работах [3, 4] для разработки шаблонов использовались специализированные

редакторы, которые обеспечивали только графическое представление шаблонов. Данные средства, несмотря на удобство для конечного пользователя, не поддерживают совместную разработку ресурсов, сложны для использование сторонними пограммными продуктами, а также трудоемки в поддержке и масштабировании решений. Таким образом, назрела потребность в разработке языка для записи шаблонов на основе стандартных форматов представления данных и создание веб-ресурса, обеспечивающего пользователя универсальной средой для разработки специализированных словарей .

2. Язык описания лексико-семантических шаблонов

Лексико-семантический шаблон – это структурный образец целевой языковой конструкции с указанным составом и лексико-семантическими свойствами. В случае успешного сопоставления шаблона с фрагментом текста формируется лексический объект, которому приписываются формальные (позиционные) и семантические (класс и свойства) характеристики. Разрабатываемый язык описания лексико-семантических шаблонов Diglex поддерживает набор логических конструкций: альтернатива, ссылка на шаблон, повторитель, опциональность, условие на контекст, дистантный контекст и др. В рамках данной работы предложен новый синтаксис языка на основе языка YAML (http://yaml.org/spec/1.2/spec.html), который обладает следующими преимуществами.

- Широкое распространение и известность.
- Поддержка всеми основными редакторами исходного кода (продуктами JetBrains, Sublime, Atom, Eclipse, VS Code и другими).
- Богатый синтаксис, состоящий из известных типов данных, таких как: ассоциативный массив, список, строка, число и т. д.
- Компактность записи.

Рассмотрим примеры простых шаблонов Diglex, обеспечивающих извлечение из текста объектов класса “должность”.

```
---
# символом “#” обозначаются комментарии
class:
  # наименование класса
  name: должность
  # наименование базового класса (не обязательно)
  parent: базовый-класс
  # произвольный набор имен и значений свойств класса
```

```
  properties:
    Одушевленность: true
    Тематика: работа и профессия
  # для каждого класса описывается список шаблонов
  templates:
    - name: младший научный сотрудник
      properties:
        Квалификационный-уровень: 1
      # Список возможных образцов для сопоставления шаблона
      cases:
        - младш{...} научн{...} сотрудник{...}
        - мнс
    - name: ведущий научный сотрудник
      properties:
        Квалификационный-уровень: 3
      cases:
        - ведущ{...} научн{...} сотрудник{...}
        - внс
```

Класс описывается именем (поле “name”), указанием родительского класса (поле “parent”) ассоциативным массивом произвольным набором свойств (поле “properties”), а также набором шаблонов (поле “templates”). В данном примере шаблоны записываются вместе с определением класса. Также допустим синтаксис, в котором шаблон описывается отдельно от класса с указанием его имени.

```
---
template:
  - name: старший научный сотрудник
    class: должность
    properties:
      Квалификационный-уровень: 2
    cases:
      - старш{...} научн{...} сотрудник{...}
      - снс
```

Имя класса служит уникальным идентификатором для организации связи с шаблонами и другими классами при наследовании. Набор свойств класса (поле “properties”) пользователь задает в зависимости от решаемой задачи и предметной области. Данные свойства – это

семантические атрибуты (приписываемые найденным по шаблонам объектам), которые можно описать на уровне класса с возможностью переопределить значения на уровне шаблона. В Diglex каждое свойство имеет свой тип определяемый в зависимости от типа YAML-значения по умолчанию. В данном примере поле “Одушевленность” имеет логический тип данных, “Тематика” – строковый, а “Квалификационный-уровень” – числовой. Отсутствие требования всегда явно указывать тип данных приводит к более компактной записи.

Каждый шаблон содержит список образцов (поле “cases”), состоящий хотя бы из одного образца для сопоставления с текстом на естественном языке. Сопоставление с шаблоном считается успешным при успешном сопоставлении с любым элементом из списка “cases”. Для записи таких образцов предложен формальный язык подобный регулярным выражениям. Например, сопоставление с образцом ‘снс’ будет успешным только при точном вхождении в текст (без учета регистра). Язык поддерживает использование следующих возможностей.

1. Хвост. Позволяет указать неизменяемую часть слова.
   Пример: *старш{...}*
   Такой образец будет соответствовать словам старший, старшая и т. д. Кроме того, имеется возможность указать длину хвоста интервалом значений: *старш{...<2,3>}*
2. Регистр. По умолчанию регистр слов игнорируется, т. е. образец “внс” будет соответствовать как слову “ВНС” так и “внс”, “Внс” и т. д. Для указания того, что регистр учитывается, необходимо писать: *внс<cs>*.
3. Ссылка на другой шаблон. Текст образца может иметь ссылку на другой шаблон, для этого необходимо указать имя шаблона в квадратных скобках. Пример: *”институт{...} математики [СО РАН]”*.
4. Повторитель. Используется для описания последовательности повторяющихся элементов. Имеется возможность указать точную длину последовательности либо интервал с количеством возможных повторений. Пример образца для извлечения номера телефона: *[цифра]<2,3>-[цифра]<2>-[цифра]<2>*.
5. Опция. Используется для обозначения необязательных элементов. Сопоставление с образцом произойдет в случае наличия или отсутствия элемента.
   Пример: *н{.}<?>{ }<?>с{.}<?>*
   Данный шаблон соответствует таким строкам: “н.с.”, “н. с. ”, “нс”, и т. д.

6. Дистантный контекст. Позволяет выделить фрагмент текста неизвестной заранее длины по заданным начальным и конечным элементам.
   Пример: *Уважаемый<-.->!*
   Элемент <-.-> будет соответствовать любому фрагменту текста, начинающемуся со слова “Уважаемый” и заканчивающемуся на восклицательный знак.
7. Исключающее условие. Предназначено для накладывания ограничений на контекст образца.
   Пример: *научный<not> сотрудник{...}*
   Такой шаблон будет соответствовать строке “Наш сотрудник”, но не “научный сотрудник”.
8. Группировка. Для применения модификатора к группе элементов применяется следующий синтаксис:
   {научный сотрудник}<not> опубликовал статью
   В данном примере модификатор <not> применяется к строке “научный сотрудник”.

3. Веб-редактор шаблонов

Благодаря синтаксису YAML для создания шаблонов Diglex пользователь может воспользоваться практически любым современным редактором без каких-либо специальных плагинов. Это удобно для быстрого старта и для пользователей, которые уже имеют привычку работы в определенных редакторах. Однако, в таком случае возникают следующие проблемы.

- Так как имена классов и шаблонов это просто строки в YAML - отсутствует их автоматическое дополнение и возможность “перейти к определению”.
- Инструменты поиска, сортировки и быстрого доступа к шаблонам по имени и другим характеристикам.
- Образцы для сопоставления в рамках YAML представлены обычными строками. Соответственно для них отсутствует не только автоматическое дополнение, но и элементарная подсветка синтаксиса.
- Отсутствие встроенной возможности проверять работу шаблонов во время их разработки, например путем применения к небольшому тексту.
- Отсутствие проверки корректности системы шаблонов, например, отсутствие циклов в ссылках, отсутствие использование неопределенных шаблонов и т.п.
- Необходимость использовать дополнительные инструменты для коллективной работы над шаблонами. При этом, обычной практикой является использование систем

контроля версий таких как Git, что является достаточно сложной задачей для неподготовленного пользователя.

Для решения этих проблем есть два пути: разработать плагин для одного или нескольких редакторов либо разработать специализированный редактор базирующийся на существующих решениях с открытым/свободным исходным кодом. Подход с созданием плагина требует выбора определенного редактора, что сильно ограничивает дальнейшее развитие его возможностями. Более перспективным представляется вариант разработки специализированного редактора, причем работающего в режиме онлайн, что может позволить предоставить простые возможности коллективной разработки. Также немаловажным аргументом в пользу веб-редактора является отсутствие требования установки дополнительного ПО на компьютер пользователя. Это удобно с точки зрения пользователя, а также снимает с разработчика необходимость поддерживать сборку редактора и ядра Diglex для различных операционных систем.

Рассмотрим требования к веб-редактору Diglex.

- Подсветка синтаксиса.
- Возможность навигации по иерархии классов и шаблонов.
- Возможность перейти к определению класса или именованного шаблона.
- Автоматическое дополнение. Если пользователь начинает вводить имя класса (шаблона) в месте предусмотренном для этого синтаксисом языка, то он должен увидеть выпадающий список со всеми возможными именами классов (шаблонов).
- Возможность проверить корректность работы шаблона в режиме онлайн путем его сопоставления с текстом, который также доступен для редактирования.
- Экспорт и импорт словаря шаблонов, текстов и результатов их обработки как на локальную машину пользователя, так и на серверную часть.
- Возможность управления правами доступа пользователей для редактирования словарей Diglex.

На данный момент реализован прототип редактора (diglex.forkode.ru), удовлетворяющий минимальным требованиям и позволяющий опробовать предложенный язык лексико-семантических шаблонов на практических примерах. Архитектура редактора (см. Рис.1) заложена такой, чтобы делать редактор простым в реализации и одновременно не препятствовать его дальнейшему развитию.

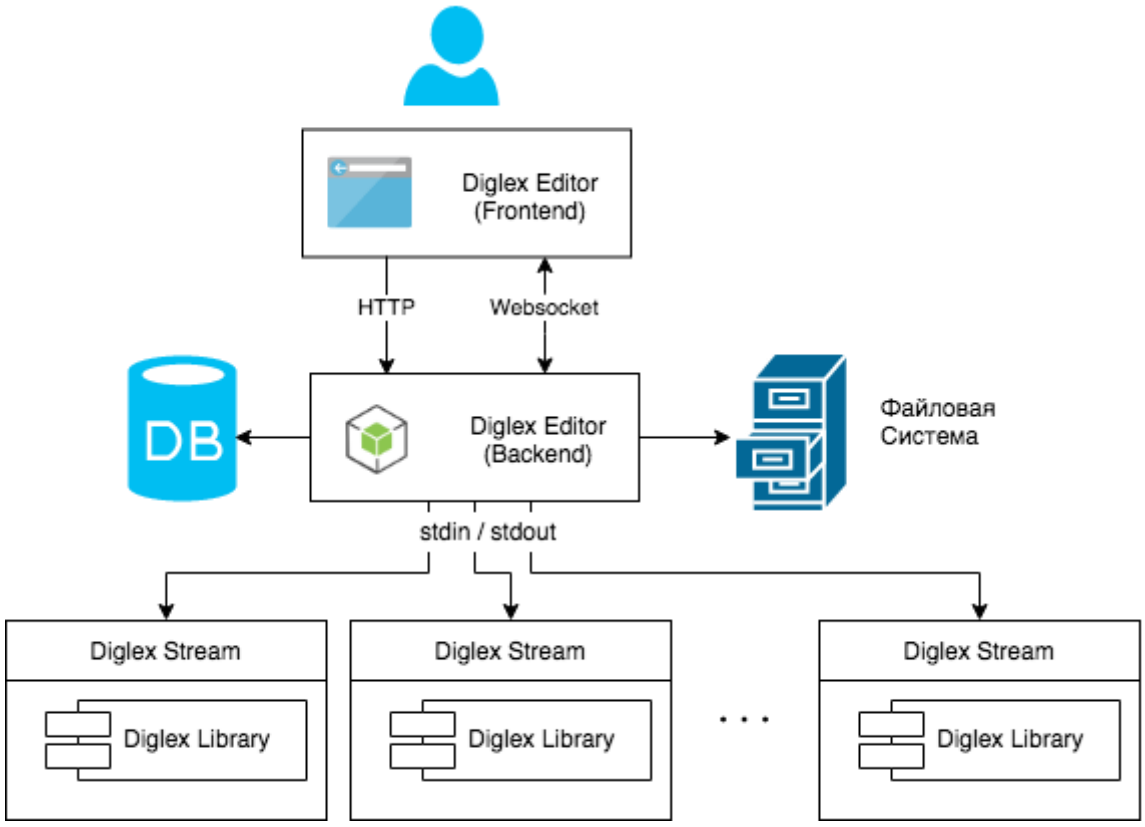

Рис.1. Архитектура веб-редактора Diglex.

В результате были сформулированы следующие архитектурные принципы на основе клиент-серверного подхода:

- Клиентская часть (Frontend) отвечает за взаимодействие с пользователем. Построена по принципу “Single Page Application” (SPA), т. е. используется единственный HTML-документ который подгружает все необходимые для работы скрипты и стили динамически. Это позволяет сделать богатый и отзывчивый интерфейс по качеству сравнимый с нативными приложениями для операционных систем. Для Frontend части был выбран JavaScript и React фреймворк.
- Серверная часть (Backend) отвечает за логику работы редактора такую как: работа с файловой системой, управление пользователями, постановка задач на вычисление с помощью ядра Diglex и предоставляет полученные результаты для Frontend. Backend контролирует работу ядра Diglex на предмет наличия ошибок и обеспечивает его перезапуск в случае необходимости. Для реализации был выбрана платформа Node.js.
- Серверная часть не линкуется напрямую с ядром Diglex, вместо этого создано отдельное приложение Diglex Stream (язык реализации C++), которое взаимодействует с Backend посредством стандартных потоков ввода-вывода (stdin, stdout).

Предложенная архитектура делает модули хорошо изолированными и простыми в программировании, сохраняя возможность легкого масштабирования. Масштабирование может производиться по принципу один процесс Diglex Stream - на некоторое количество пользователей, либо на каждый словарь - некоторое количество процессов Diglex Stream.

4. Экспериментальное исследование

С использованием разработанного инструментария были созданы два словаря: 1) словарь наименований институтов, должностей, званий и их сокращений и 2) словарь числовых/временных конструкций. Первый словарь был разработан на основе номенклатурных списков РАН. Всего в словарь входит 646 шаблонов и 36 классов. Словарь позволяет находить в тексте:

- Наименования организаций РАН. В частности: наименования академий, институтов, университетов, колледжей, заводов, библиотек, училищ, музеев и научных центров.
- Наименование должностей, степеней и званий РАН. Например: *ведущий специалист, младший научный сотрудник, доктор биологических наук, доцент, академик-секретарь* и т. д.

Второй словарь разработан с целью представить шаблоны с семантикой “время”. В него входят шаблоны позволяющие искать в текстах следующие временные конструкции.

- Часовое время. Отвечают на вопросы: “сколько времени?”, “который час?”, “когда?”, “в котором часу” и т.д. Например: *в час дня, три часа дня, пятнадцать минут первого*.
- Обозначение даты, времени действия. Такие конструкции отвечают на вопросы: “какое сегодня число?”, “Когда?”, “Какого числа?” и т. д., “В какой день?”. Например: *сегодня пятое декабря, этой зимой, на той неделе*.
- Обозначение продолжительности действия. Такие конструкции отвечают на вопросы: “сколько времени?” -Например: *одну секунду, в течение часа, на протяжении* и т. д.
- Обозначение момента начала и конца действия. Такие конструкции отвечают на вопросы: “с какого времени”, “до какого времени?”, “с которого часа и до которого часа?” и т. д. Например: *через два часа, с утра до вечера, с апреля по ноябрь включительно*.
- Обозначение времени, по прошествии которого произойдет, закончится или начнется действие. Такие конструкции отвечают на вопросы: “когда?”, “через сколько времени?”. Например: через пятнадцать минут, спустя годы.

- Обозначение срока действия и времени необходимого для достижения результата действия. Такие конструкции отвечают на вопросы: “за сколько времени?”, “за какое время?”. Например: *в одно мгновение, за одну минуту*.
- Обозначение срока, в течение которого сохраняется результат действия. Такие конструкции отвечают на вопросы: “на сколько времени?”, “на какой срок?”, “на какое время?”. Например: *на пять дней, на две недели*.
- Обозначение времени повторяющегося действия. Такие конструкции отвечают на вопросы: “когда?”, “как часто?”. Примеры: *ежедневно, время от времени, раз в два дня*.
- Обозначение одновременности действий. Такие конструкции отвечают на вопросы: “когда?”, “в какое время?”. Примеры: *в ходе, в то же самое время*.
- Выражение последовательности действий. Такие конструкции отвечают на вопросы: “когда?”, “в какое время?”. Например: *до того как, прежде чем*.

Созданные словари были опробованы на двух коллекциях текстов: “Хроники СО РАН” (http://www.nsc.ru/HBC/events/chronicle.html) и “Постановления Президиума СО РАН”. коллекция “Хроники СО РАН” содержит краткие описания 1218 событий, связанных с СО РАН, происходивших с 1957 г. по 1992 г. Каждый текст – это отрывок из документов различных жанров: официальных постановлений, деловых писем, газетных статей и т.п. В коллекцию “Постановления Президиума СО РАН” входит 159 официальных постановлений, положений и приложений к ним. Статистика результатов применения словаря на данных коллекциях текстов представлена в Таблице 1.

Отсутствие размеченного корпуса текста не позволяет применить классические способы оценки полноты и точности анализа. Общим свойством разрабатываемых вручную шаблонов является обеспечение высокой точности. Мы предложили эвристическую методику оценки точности и полноты на основе косвенных признаков, поддающихся автоматическому вычислению. В качестве критерия точности мы рассматриваем наличие вариативности извлеченных объектов в одной позиции. Например, из текста “*если во время приема граждан решение поставленных вопросов невозможно*” извлекается конструкция “*во время*” как объект обозначающий продолжительность действия и, одновременно, как объект обозначающий одновременные действия. Верный вариант может быть только один, но в данном случае система не в состоянии его определить и формирует оба. В качестве критерия полноты рассматриваем наличие частично собранных шаблонов, таких как инициалы с невыделенной фамилией (как показатель ненайденной фамилии персоны), аббревиатуры, используемые в составе названий институтов (СО, АН, СССР) и др. Наличие таких частей

означает, как правило, что объект не смог собраться полностью из-за отсутствия необходимых описания, что ухудшает показатели полноты.

Таблица 1. Результаты применения лексико-семантических шаблонов на текстовых коллекциях.

	Коллекция “Хроники СО РАН”		**Коллекция “Постановления Президиума СО РАН”**	
	Словарь организаций	Словарь временных конструкций	Словарь организаций	Словарь временных конструкций
Размер коллекции (число слов)	57 521		153 331	
Применено шаблонов	120	18	134	18
Найдено классов	19	12	19	14
Извлечено объектов	7 400	2 642	27 296	5 248
Точность	90.64 %	99.47 %	84.81 %	99.40 %
Полнота	94.03 %	90.82 %	90.04 %	74,14%

Результаты показали, что полученные словари извлекают достаточно большое количество объектов и достаточно хорошо покрывают требуемые конструкции. Низкие показатели точности относительно полноты для словаря наименований связаны со сменой аббревиатуры в названиях институтах в 90-х годах, что дало вариативность при определенных сокращениях. Во втором словаре показатель точности ожидаемо высок. Также получены хорошие показатели полноты для словарей наименований, что связано в первую очередь с жанром источников, в которых принято использовать номенклатурные наименования. Для словаря временных конструкций получены низкие показатели полноты, что связано с необходимостью совершенствовать систему временных шаблонов, в частности для извлечения дат.

В ходе разработки шаблонов при необходимости записать в шаблоне множество словоформ одной лексемы наблюдалась нехватка возможности указания их грамматических

признаков. Частично эту проблему решает возможность указания “хвоста”, но для ряда случаев он охватывает словоформы других лексем. Характерный пример: все словоформы лексемы “май” записываются как “ма{...}”, что соответствует как слову “май”, так и слову “март”. Несмотря на обнаруженные недостатки, система в целом продемонстрировала свою пригодность для выполнения практических задач.

5. Заключение

В работе предложен подход к описанию естественно-языковых конструкций с помощью лексико-семантических шаблонов. Представлен язык описания шаблонов Diglex на основе синтаксиса языка YAML, рассмотрены его основные конструкции и возможности. Для удобства разработки шаблонов Diglex спроектирован веб-редактор и реализован его прототип. Полученный инструментарий опробован при создании двух словарей.

Язык шаблонов Diglex отличается простотой и декларативностью. С его помощью удобно создавать шаблоны для извлечения параметрических конструкций, извлечения числовых значений и нестандартных символьных обозначений. Разрабатываемый веб-редактор снижает порог вхождения для новых пользователей Diglex и предоставляет им возможности для коллективной работы. Система в целом расширяет возможности традиционных лексикографических систем и упрощает их использование.

Одним из недостатков существующего на данный момент языка лексико-семантических шаблонов является отсутствие средств для указания грамматических признаков распознаваемых лексических единиц, что является направлением дальнейших исследований. Также требует развития веб-редактор. На данный момент разработан только его прототип и планируется разработка полноценного редактора в соответствии с требованиями, обозначенными в настоящей статье.

Работа выполнена при поддержке РФФИ (грант № 17-07-01600).

Список литературы

1. 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: тр. межд. конференции Диалог‘2007. М.: Изд-во РГГУ, 2007. С. 70-75.
2. Ермаков А.Е., Плешко В.В., Митюнин В.А. RCO Pattern Extractor: компонент выделения особых объектов в тексте. //X II Международная научная конференция. Сборник трудов Москва, 2003. С. 312-317.

3. Жигалов В. А. Жигалов Д. В., Жуков А. А., Кононенко И.С., Соколова Е.Г., Толдова С.Ю Система Alex, как средство для многоцелевой автоматизированной обработки текстов // Труды международного семинара Диалог'2002 "Компьютерная лингвистика и интеллектуальные технологии". — Москва : Наука, 2002. — Т. 2. – С.192-208.
4. Ковалев А.И., Сидорова Е.А. Инструмент разработки предметных словарей на основе лексических шаблонов DigLex // Материалы Всероссийской конференции с международным участием «Знания – Онтологии – Теории» (ЗОНТ–2015), 6 - 8 октября 2015 г., Новосибирск. – Новосибирск: Институт математики им. С.Л. Соболева СО РАН, 2015. –Т1. – С. 123-130.
5. Митрофанова О.А., Захаров В.П. Автоматизированный анализ терминологии в русскоязычном корпусе текстов // Компьютерная лингвистика и интеллектуальные технологии: тр. межд. конференции «Диалог–2009». М.: 2009. С. 321-328.
6. Рабчевский Е., Булатова Г., Шарафутдинов И. Формализм записи лексико-синтаксических шаблонов в задаче автоматизации процесса построения онтологий // Труды десятой всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» –RCDL'2008. Дубна: ОИЯИ, 2008. С. 103-106.
7. Сидорова Е. А. Многоцелевая словарная под- система извлечения предметной лексики // Компьютерная лингвистика и интеллектуальные технологии: тр. межд. конф. «Диалог–2008». М.: 2008. Вып. 7 (14). М.: Изд–во РГГУ, 2008. С. 475-481.
8. General Architecture for Text Engineering. http://www.gate.ac.uk/

УДК 004.82:004.912

Проблемы извлечения терминологического ядра предметной области из электронных энциклопедических словарей

Кононенко И.С. (Институт систем информатики СО РАН),
Ахмадеева И.Р. (Институт систем информатики СО РАН),
Сидорова Е.А. (Институт систем информатики СО РАН),
Шестаков В.К. (Институт систем информатики СО РАН)

Статья посвящена проблемам автоматического построения терминологической системы предметной области. Предложен метод извлечения терминов предметной области на базе электронных энциклопедических источников данных. Особенностью предлагаемого подхода является тщательный анализ структуры термина, распознавание ошибок на базе их лингвистической классификации, автоматическая генерация лексико-синтаксических шаблонов, представляющих многокомпонентные термины, и использование набора эвристических методов обработки «особых» терминов. Использование энциклопедических словарей позволяет автоматически сформировать эталонный список наименований понятий и применять его для оценки качества формируемых словарей.

Ключевые слова: *извлечение терминов, многословный термин, терминологическое ядро предметной области, омонимия, предметный словарь.*

1. Введение

В данной работе рассматриваются проблемы автоматического формирования первоначального концептуального состава онтологии научной предметной области (ПО), который определяется совокупностью терминов – слов и словосочетаний, являющихся наименованиями понятий моделируемой ПО. Источником соответствующих знаний могут служить электронные тексты различных жанров: учебники и предметные указатели к ним, научные статьи и монографии, научно-популярная литература, терминологические и энциклопедические словари.

Анализ литературы [4, 5, 8] показывает, что при извлечении терминологии из большого массива текстов используются подходы, объединяющие лингвистические и статистические

методы. Для формирования списков терминов-кандидатов, удовлетворяющих заданным лингвистическим условиям, используется метод шаблонных конструкций, описывающих классы языковых выражений. В зависимости от типа учитываемой в конструкции языковой информации, применяемые в различных работах шаблоны подразделяются на грамматические [12], лексико-грамматические [5, 8] и лексико-синтаксические [1, 9]. Извлечение терминов-кандидатов сопровождается накоплением статистики встречаемости и подсчетом весов для фильтрации и сортировки полученных списков. В результате процедуры в список терминов-кандидатов попадают не только сложившиеся в данной области обозначения основных специальных понятий, но и многочисленные общенаучные, периферийные и авторские термины, которые, как показано в работе [2], характеризуются большой степенью варьирования языковой формы. В этой ситуации необходим этап экспертной оценки, на котором ранжированные списки предъявляются эксперту для отбора истинных терминов.

Задача формирования терминологического ядра онтологии научной ПО требует строгого отбора сложившихся в данной области обозначений специальных понятий. Многие исследования, ориентированные на извлечение и формализацию предметных знаний в виде онтологии, опираются не на корпус разножанровых текстов, а на более доступные структурированные источники, такие как специализированные глоссарии, предметные указатели, терминологические словари и энциклопедии. Эти источники преимущественно используются в задаче извлечения знаний о взаимосвязях понятий, например, родовидовых отношений, отношений эквивалентности и т.п. [6, 7, 10-11, 14-16]. Однако не менее перспективно использование словаря как эксплицитного источника понятийного ядра ПО и стандарта его терминологического представления. Это снимает проблему отбора основного ядра терминов экспертом (соответствующая работа проделана составителями словарей), что не исключает необходимости накопления статистической информации, учитывая наличие не вошедших в словарь периферийных терминов, общенаучных терминов и терминов смежных дисциплин.

Задачей данного исследования является разработка методов извлечения терминологического ядра предметной области из структурированных интернет-источников, содержащих энциклопедические данные научных областей знаний. Используемый метод предполагает наличие в таких источниках эталонного массива надежных терминов, который может быть автоматически извлечен и использован для оценки качества формируемых терминов. Для достижения поставленных целей 1) проведен эксперимент по автоматическому извлечению терминов с помощью базовой технологии, основанной на методе грамматических шаблонов, 2) проанализирован полученный массив терминов-кандидатов и выявлены

проблемные ситуации, 3) предложены дополнительные эвристические методы для решения выявленных проблем и 4) проведен эксперимент по извлечению терминов с помощью расширенной методики и осуществлена сравнительная оценка эффективности двух методик.

2. Электронный словарь как источник терминов

В качестве материала для данного исследования были выбраны два электронных энциклопедических словаря – словарь терминов теории графов (объемом 151 словарных статей) и толковый словарь по искусственному интеллекту (объемом 564 словарных статьи) – в предположении, что данные источники содержат достаточно представительное ядро надежных терминов и их взаимосвязей, характерных для соответствующих ПО. Исследуемые источники являются энциклопедическими словарями, содержащими термины и их дефиниции, что позволяет использовать их не только для извлечения терминов, но и в дальнейшем ставить задачу извлечения отношений между понятиями на основе терминов и их толкований. Общим свойством двух словарей является тесная связь с общенаучной терминологией и терминологией смежных дисциплин, поскольку исследования в этих областях носят междисциплинарный характер.

Контент в обоих словарях имеет HTML-разметку, что облегчает выделение границ текстовых сегментов, а систематическое использование в словарях гиперссылок может служить для выделения соответствующих вхождений терминов в текст дефиниций.

Рис.1 демонстрирует словарные статьи из словаря по ИИ и словаря по теории графов, представленные в виде текста с сохраненной разметкой.

В структуре словарной статьи выделяются две части – левая и правая. В левой части статьи представлено заголовочное слово (заглавный термин или множество заглавных терминов), выделенное заголовочными тегами (словарь по ИИ) или жирным шрифтом (словарь теории графов). Совокупность заглавных терминов представляет словник словаря. Подавляющее большинство заглавных терминов представлено номинативными конструкциями. Правая часть статьи – это зона дефиниций, которая содержит дефиницию-толкование.

Рис. 1. Примеры представления терминов в энциклопедических словарях.

Термины словника, используемые в толковании, выделены гиперссылочными тегами (словарь по ИИ) или курсивом (словарь теории графов). В первом примере приведен многозначный заглавный термин, которому соответствуют две альтернативных дефиниции. В правой части вместо толкования могут быть представлены синонимичные заглавные термины, маркированные тегами-ссылками на соответствующие словарные статьи и лексическими маркерами (“то же, что”, “см.”, “синоним для”). Зоны грамматических и стилистических помет отсутствуют, что характерно для словарей энциклопедического типа.

Оба словаря используют прием ввода вложенных дефиниций в правую часть словарных статей: толкование термина *простая цепь* в статье *цепь* (словарь теории графов), толкование термина *унификатор* в статье *унификация* (словарь по ИИ).

Особенности словаря по ИИ:

(а) “*вложенный*” термин является заглавным, ему соответствует собственная словарная статья, дефиниционная часть которой представляет собой ссылку с маркером “термин объясняется в статье”;

(б) ввод иллюстративного материала в зону дефиниции с помощью лексических маркеров “*примером может служить*”, “*например*”. Так, отношение "*быть супругом*" приводится как иллюстрация термина *отношение симметричное*;

(в) использование в дефиниционной части графических инициальных сокращений, которые терминами не являются, а служат окказиональной заменой термина/компонента термина в пределах его словарной статьи*: каузация – К., диссонанс когнитивный – Д.К.*

Словарь теории графов отличают следующие особенности:

(а) нестандартные способы ввода синонимов – в левой части, путем перечисления после заглавного термина, возможно, в скобках: *Вершина,Узел**;** Вполне несвязный граф(пустой граф,нуль-граф)**;** Независимое множество вершин(известное также каквнутренне устойчивое множество);*

(б) широкое использование в толкованиях числовых записей величин, буквенных обозначений переменных (на базе латиницы), различных символьных последовательностей, характерных для обозначений в языке математики; переменные, как правило, выделяются разметочными тегами.

(в) окказионально используемое (3 случая) нестандартное представление словарной статьи, когда заглавный термин вводится в автонимном контексте с предикатом называния: *Полным графомназывается граф, в котором…*

(г) активное использование инверсной структуры терминов (почти 50 процентов многословных заглавных терминов), т.е. изменение прямого порядка слов в терминологическом словосочетании с выведением слова, несущего максимальную смысловую нагрузку, в позицию ведущего слова: *область предметная, автомат детерминированный*. Этот прием используется в различных номенклатурных списках, предметных указателях и т.п., позволяя сгруппировать родовой и видовые термины в одной зоне словаря.

Основную массу заглавных терминов составляют существительные, а также именные группы с зависимыми от главного существительного согласованными прилагательными или причастиями и существительными в родительном падеже. В обоих словарях заголовочные слова (лексемы и словосочетания) преимущественно представлены в основной (нормальной) форме, которая определяется именительным падежом лексемы или главного существительного.

3. Извлечение терминологии

Научный дискурс и жанр электронного словаря определяют специфику лексико-семантических, морфологических, синтаксических и структурных параметров анализируемых текстов. В данной главе описываются особенности процесса анализа текстового контента

электронных энциклопедических словарей и построения на их основе терминологического ядра предметной области.

Процесс извлечения терминологии включает такие этапы как а) выделение основного текстового контента и очистка от незначимых элементов, б) сегментация, направленная на выделение структурных сегментов текста, определяющих границы дальнейшего анализа, в) графематический анализ, обеспечивающий токенизацию и выделение нетекстовых элементов (формул, гиперссылок, числовых данных, обозначений и пр.) и иноязычных вкраплений, г) лексико-морфологический анализ (лемматизация, определение лексико-грамматических признаков, представление парадигмы, нормализация), д) выделение терминоподобных словосочетаний (идентификация на основе предопределенных грамматических моделей и нормализация), е) применение различных эвристик для снятия омонимии и улучшения качества извлечения терминов. Рассмотрим подробнее особенности данного процесса.

3.1. Сегментация

Процесс сегментации начинается с очистки исходного HTML документа от лишней разметки. В тексте оставляются только значимые теги: <b>, <a>, <i>, которые затем используются при анализе структуры документа.

Затем в очищенном тексте выделяются сегменты (фрагменты текста): «Определение», «Левая часть», «Правая часть». Выделение таких сегментов осуществляется на основе их декларативного описания (множество шаблонов сегментов), которое строится экспертом в зависимости от структурной организации словарной статьи электронного словаря (Рис.2).

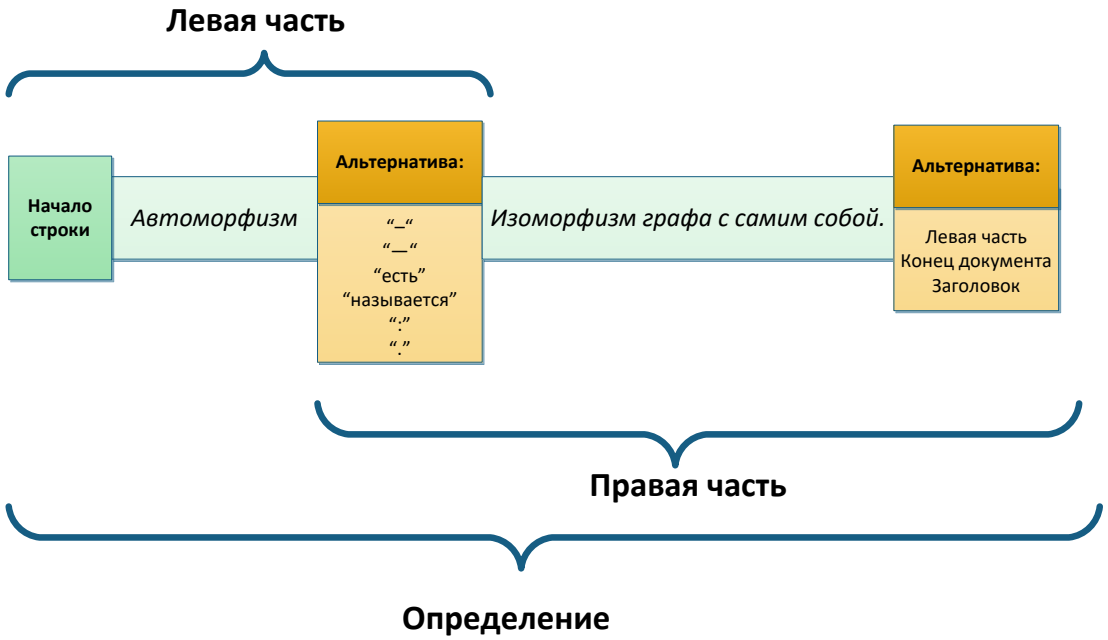

Рис. 2. Структура словарной статьи.

Для описания шаблона сегмента нужно указать тип сегмента, ограничения, которым должен удовлетворять сегмент (в зависимости от типа сегмента), а также уникальное имя, позволяющее использовать этот сегмент как часть в определении более сложных шаблонов. Пример декларативного описания сегментов, построенного для сегментации словаря терминов по теории графов, представлен на Рис. 3. Заметим, что представленные шаблоны позволяют выделять сегменты в электронных словарях схожей структуры, например, таких как глоссарии Википедии.

```
{
        "type":"or",
        "name":"Разделитель_определения",
        "segments": ["s/–", "s/—", "s/есть", "s/называется", "s/:", "s/."]
},
{
        "type":"classic",
        "name":"Левая_часть",
        "begin":"Начало_строки",
        "end":"Разделитель_определения"
},
{
        "type":"classic",
        "name":"Правая_часть",
        "begin":"Левая_часть",
        "end": ["Левая_часть", "__end__", "t/h2"]
},
{
        "type":"sequence",
        "name":"Определение",
        "segments":["Левая_часть", "Правая_часть"]
}
```

Рис. 3. Декларативное представление сегментов.

Элементарными маркерами сегментов могут быть конкретные последовательности символов или регулярное выражение, а также слова и фразы подключаемого предметного словаря. Поддерживается несколько типов шаблонов и ограничений, которым должен удовлетворять искомый сегмент.

a) Сегмент можно задать с помощью ограничений на его начало и конец, которыми могут быть другие сегменты (так задаются сегменты «Левая часть» и «Правая часть»).
b) Сегмент может быть задан последовательностью других сегментов, например, «Определение» является последовательностью сегментов «Левая часть» и «Правая часть».

c) Сегмент можно определить как альтернативу других сегментов: «Разделителем определения» может быть один из нескольких вариантов.

Таким образом, на вход сегментатору подаются шаблоны сегментов, представленные в JSON формате, и очищенный текст, а на выходе для каждого шаблона формируется множество найденных в тексте сегментов, удовлетворяющих условиям шаблона.

В дальнейшем анализе участвует только текст, соответствующий сегментам типа «Определение». При первом проходе в левой части определений выделяются заглавные термины (с помощью технологии Клан, рассматриваемой ниже). Заметим, что если в тексте присутствуют теги, то в первую очередь в качестве заглавного термина алгоритм пытается взять фрагмент, выделенный тегами. Так, на Рис. 4 внутри сегмента «Левая часть» красным отмечены вхождения сегментов «Выделение_тегами». Если выделенных тегами фрагментов несколько, то первый из них рассматривается как главный, остальные – синонимы. Так, в рассматриваемом примере термин «Двудольный граф» (выделен красным) считается главным термином, а термины «биграф» и «четный граф» его синонимами (выделены зеленым).

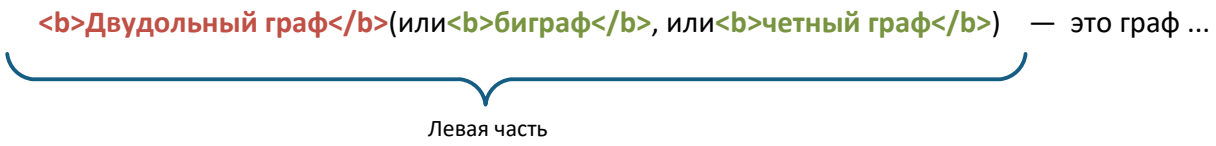

Рис. 4. Пример разбора левой части определения.

При втором проходе анализируются правые части определений и отмечаются все вхождения дескрипторов (которые мы нашли на предыдущем шаге) в определения, а также указываются их морфологические характеристики, также с помощью технологии Клан.

3.2. Технология Клан

Лексико-морфологический и поверхностно-синтаксический анализ и поиск терминов осуществляются системой Клан [12]. Данная система позволяет создавать предметно-ориентированные словари и использовать их при автоматическом анализе текстов.

Морфологический анализ осуществляется на базе модуля Диалинг (www.aot.ru), который содержит универсальный словарь русского языка и обеспечивает поиск слова в словаре, определение его грамматических признаков и нормальной формы. Также поддерживается функция предсказания [13], которая по незнакомому слову формирует гипотезы (как правило, около 3 вариантов) о его части речи, нормальной форме и других признаках. Так, например, для слова *когнитивный* были сформированы следующие предсказания:

когнитивная *Сущ, жр, но /a (пустая парадигма)

когнитивный *Сущ, жр, но /a (пустая парадигма)

когнитивный *Прил /S (ен, на, ная, нее, ней, но, ного, ное, ной, ном, ному …)

когнитивна *Сущ, жр, од, имя /b (, а, ам, ами, ах, е, ой, ою, у, ы,)

когнитивные *Сущ, жр, но /a (пустая парадигма)

когнитивных *Сущ, жр, но /a (пустая парадигма)

Особенностью морфологического представления в рамках системы Клан, является формирование морфологического класса термина и обеспечение его уникальности в рамках своего класса. Это означает, что одинаковые по написанию термины (в нормальной форме) не могут принадлежать одному морфологическому классу, а возможная лексико-семантическая неоднозначность поддерживается функционалом формирования альтернативных групп семантических характеристик.

Морфологический класс определяется частью речи, набором лексических признаков слова (например, *одушевленность* или *род* у существительных) и типом парадигмы, определяющим изменяемые морфологические признаки, которыми дополнительно снабжаются разные формы слова. Набор морфологических классов может быть изменен пользователем, в зависимости от решаемых задач, однако потребность в этом возникает достаточно редко. Исключением являются случаи, когда используются дополнительные специализированные словари терминов, например, словарь имен или географических названий, или необходимость включать в словарь слова другого языка.

Таким образом, однословный термин словаря Клан обладает уникальным идентификатором <*норма, морф_класс*>, что позволяет однозначно распознавать термин по набору его признаков и делает возможным применение методов автоматического пополнения словаря на основе корпусов текстов.

Другой важной особенностью системы является поддержка многословных терминов. Многословный термин в системе Клан – это словосочетание, сформированное по одному из правил, реализующему поверхностно-синтаксический анализ (для русского языка). Большинство многословных терминов включают от 2 до 4 слов и формируются с помощью правил вида П+С (*планарный граф*), П+П+С (*новая информационная технология*), С+Срд (*сеть петри*), П+С+Срд (*локальная степень вершины*), С+Прд+Срд (*обработка естественного языка*), С+Срд+Срд (*компонента связности графа*). Имеются также термины более сложной структуры, например, с зависимыми предложными группами С+Предл+С (*путь в орграфе, рассуждение по умолчанию*), С+Предл+С+С (*поиск в пространстве состояний*), С+Предл+П+С (*автомат с переменной структурой*) и т.д. В системе реализован

собственный компонент сборки словосочетаний русского языка, который по заданному набору слов и их грамматическим характеристикам проверяет согласование в соответствии с одной из синтаксических моделей и синтезирует нормальную форму многословного термина. Многословный термин словаря Клан однозначно идентифицируется набором *<норма, правило, <лексический_состав>>*. У такого термина можно, при необходимости, взять синтаксическую вершину – однословный термин – и грамматические признаки, которые формируются на основе грамматических признаков вершины.

Семантический компонент словаря обеспечивается иерархией семантических классов, набором семантических атрибутов, фиксацией значения (дескриптора) и механизмом формирования групп признаков у термина. Такой механизм позволяет задавать как альтернативные, так и синкретически-выраженные значения терминов. В рамках данной работы с помощью такого рода признаков фиксируются формально-текстовые признаки, отражающие, в какой части определения встретился термин, есть ли у него омонимы и т.п.

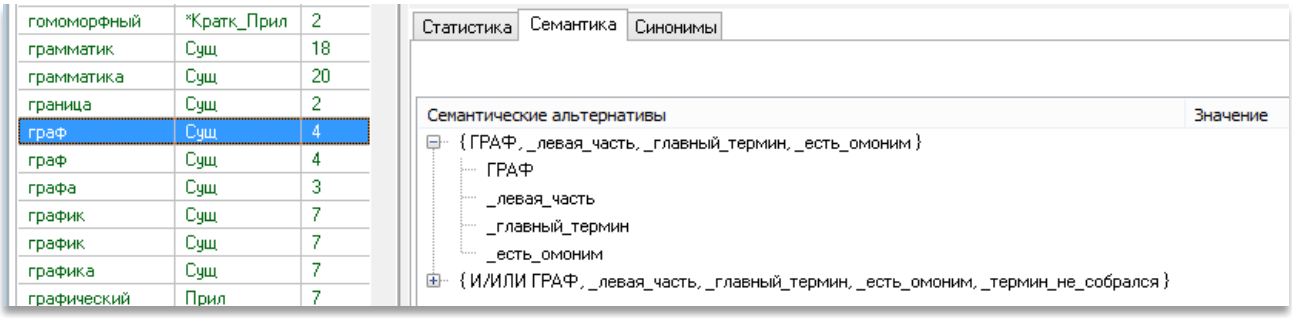

Рис. 5. Формальные характеристики термина в базовом словаре.

Для терминов также может накапливаться статистика встречаемости в тексте(текстах).

Таким образом, с помощью системы Клан реализуется терминологический анализ определений, найденных в электронном энциклопедическом ресурсе. Результатом обработки текста с помощью данной системы являются два базовых словаря лексических единиц:

– однословные кандидаты в простые термины и компоненты составных терминов и

– многословные кандидаты в составные термины.

3.3. Типизация проблем выделения терминов

При анализе списков терминов-кандидатов, полученных с помощью рассмотренных выше средств, был выявлен ряд особенностей и проблем, связанных с некорректностью и неполнотой извлечения простых и составных терминов.

3.3.1. Особенности выделения простых терминов и компонентов терминологических словосочетаний

В данном разделе приводится типизированный перечень проблем, характеризующих извлечение однословных терминов и компонентов составных терминов[1]. Корректность извлеченного из текста термина-кандидата определяется по следующим параметрам: нормальная форма (лемма), часть речи, лексические признаки, класс словоизменения, парадигма. Возможные ситуации: термину соответствует группа извлеченных омонимичных вариантов, лишь один из которых корректен; термину соответствуют только некорректные варианты; термину не сопоставлено ни одного кандидата. Учитывая это, все ошибки можно подразделить на три типа: омонимия, некорректность и неполнота.

Омонимия. Формируемый в процессе анализа текста словник содержит все леммы, имеющие омонимичные словоформы, т.е. группы лексических и лексико-грамматических омонимов из универсального словаря и группы предсказанных в сходных позициях альтернатив, различающихся по подмножеству/совокупности параметров.

(1) Наличие в универсальном словаре лексических или лексико-грамматических омонимов, лишь один из которых является термином и/или компонентом термина словаря. Частотным примером является совпадение ряда форм одноосновных существительных, различающихся по роду и/или одушевленности: *граф* (С,мр,но) – *граф* (С,мр,од) – *графа* (С,жр,но), *логика – логик*, *графика – график, метод – метода*, *клика – клик, домна – домен, мир – миро*. Более редкие примеры: *машина* (С,жр,но) – **машин* (С,од,фам).

(2) Предсказанные слова, образованные от словарных слов наращиванием стандартного префикса (*анти, мульти, полу, пере, псевдо, мета, гипер, спец, ко, су, макси-, мини-*), наследуют грамматические свойства производящих слов: **мультиграф* (С,мр,но) – *мультиграф* (С,мр,од) – **мультиграфа* (С,жр,но); *макси-код* (С,мр,но) – **макси-кода* (С, жр,но)[2].

(3) Источником омонимии в рамках одной части речи являются паронимы (*временной – временный, языковой – языковый, образный – образной*) и стилистические варианты (*знание – знанье, рассуждение – рассужденье, отношение – отношенье*), контраст которых нейтрализуется в определенных синтаксических позициях. В результате формируются неверные дубликаты терминов или словосочетаний.

(4) Предсказанная частеречная омонимия “существительное vs. прилагательное” характерна для прилагательных-компонентов составных терминов: *абдуктивный, деонтический, когнитивный, секвенциальный, кликовый*, *цикломатический*. Менее частотны частеречная

[1] Искажения, связанные с ошибками в текстах исходных словарей, не рассматриваются как ситуации некорректности.

[2] Здесь и далее звездочкой помечены некорректные гипотезы.

омонимия “прилагательное vs. причастие” (*ориентированный – ориентировать*), омонимия “существительное vs. глагол” (*фрактал – *фрактать, лок – *лочь*). Имеются и редкие варианты “существительное vs. прилагательное” (*остов – *остовый*), “существительное vs. прилагательное vs. глагол” (*решатель – *решателый – *решатеть*).

(5) Большинство иностранных фамилий, используемых в составе терминов (*универсум Эрбрана, сеть Петри, решетка Келли репертуарная*), являются незнакомыми словами и регулярно предсказываются многовариантно, например, фамилии *Эрбран* соответствуют гипотезы **эрбрана* (С,жр,но), **эрбраный* (Кр_П, кач) и **эрбрать* (Кр_Прич).

(6) Предсказание незнакомых дефисных сложных прилагательных является еще одним источником регулярной омонимии (*контекстно-связанный – *контекстный-связанный*).

Некорректность термина-кандидата определяется по подмножеству / совокупности значений приведенных выше параметров. Учитывая их взаимосвязь, можно подразделить данный тип ошибок на два подтипа.

А. Лексическая некорректность – недостатки определения словарной формы термина (неполная или искаженная форма). Так, некорректность леммы в ситуациях (7)-(8) связана с использованием графических терминоэлементов (символов, знаков, цифр, некириллических букв, вариантов графем), которые при лексико-морфологическом анализе пропускаются. Есть случаи некорректности нормальной формы в связи с тем, что общепринятая нормализация не всегда терминологична (9)-(10).

(7) Ряд терминов имеют дефисноoформленные буквенные префиксы с использованием букв латинского или греческого алфавита: *n-факторизация, k-связный*.

(8) Кириллические префиксы (*ИИ-программирование, мимд-архитектура)*, как правило, проблем не вызывают, за исключением случая нетрадиционного использования кавычек: *“лямбда”-исчисление.*

(9) В результате нормализации существительные, выступающие в качестве одиночных терминов, приводятся к виду С,им,ед. Однако стандартная нормализация по числу не всегда адекватна с терминологической точки зрения: некоторые термины-существительные (и формируемые на их основе словосочетания) представляют множественные понятия, что маркировано в толковании и отражается в выборе множественного числа существительного в качестве заглавного термина. Пример: *знания* (совокупность сведений, образующих целостное описание…).

(10) Нормализация превосходной степени качественных прилагательных, употребляемых в составе математических терминов, таких как *унификатор наибольший общий* (*наибольший*

=> *большой*), приводит к искажению смысла термина, в состав которого входит данная форма.

Б. Лексико-грамматическая некорректность диагностируется по подмножеству/совокупности значений параметров, определенных механизмом предсказания для отсутствующих в универсальном словаре слов. В (11)-(12) приведены типовые примеры ситуаций, в которых термину соответствуют только некорректные варианты.

(11) Механизм предсказаний является источником ошибок при формировании леммы и парадигмы дефисных сложных прилагательных при корректности других параметров (**аппаратный-программный* – **аппаратный-программного*), а также парадигмы существительных с изменяемой первой частью (*граф-звезда* – **граф-звезды, фрейм-прототип* – **фрейм-прототипа*).

(12) Большинство иностранных фамилий являются незнакомыми словами и предсказываются неверно. Как правило, генерируется одна или несколько ложных гипотез, например, изменяемое **крипка* (С,жр,но) от *Крипке*; неизменяемое **петри* (С,жр,но) и **петрить* (Глаг) от *Петри.* Во всех случаях отсутствует лексический признак “фам”.

Неполнота определяется в ситуации, когда термин или компонент термина в списке не представлен ни одним кандидатом.

(13) Большинство акронимов (терминологических аббревиатур, используемых в качестве эквивалентов сложных и громоздких терминов) отсутствуют в списке: из пяти терминов этого типа, представленных в словаре по ИИ, в универсальном словаре имеется только термин *СУБД.* Термин *АСУ* предсказывается неверно, остальные не извлекаются.

(14) Леммы притяжательных прилагательных на *-ово, -ево* от иностранных фамилий, регулярно употребляемые в составе терминов (*ламанов граф*), в словнике отсутствуют.

(15) Отсутствие предсказаний для некоторых фамилий, например, *Черч*.

3.3.2. Особенности выделения терминологических словосочетаний

Значительную часть общего числа терминов, включенных в словники исследуемых источников, составляют терминологические словосочетания (56,3% в словаре по теории графов и 78% в словаре о искусственному интеллекту). Их извлечение основано на синтаксических моделях, предопределенных в качестве правил формирования (извлечения и нормализации) СК в технологии Клан. Ниже приводится типизированный перечень ошибок, выявленных в результате эксперимента по извлечению многословных терминов. Все ошибки можно подразделить на три группы: неполнота, избыточность и некорректность.

Неполнота сборки термина имеет целый ряд причин, таких как наличие нелексических компонентов в составе термина, особенности репрезентации термина в тексте, неполнота стандартных моделей, отсутствие моделей для словосочетаний большой длины и инверсных словосочетаний.

А. Нелексические компоненты в составе термина.

(16) Использование символов или цифровых обозначений числа в позиции приложения *регулярный граф степени 0*.

Б. Разделители в составе термина.

(17) Использование кавычек, запятых (при осложнении постпозитивным причастным оборотом) и других разделителей: *поиск типа “сперва вглубь”*; *система, основанная на знаниях*; *область предметная, плохо структурированная*; *и/или граф*.

(18) Использование скобок (уточнение заглавного термина) *глубинная структура (предложения)*; *индукция полная (математическая)*.

(19) Разрыв термина разметочными тегами (если не рассматривать эту ситуацию как ввод релятора для различения омонимичных терминов):

- *Длинамаршрута— количество рёбер в маршруте (с повторениями)*;
- *Длинапути— число дуг пути (или сумма длин его дуг, если последние заданы)*.

В. Неполнота термина в тексте, связанная с особенностями его текстовой репрезентации, в частности, редукцией части термина (эллипсис) и использованием разрывающих термин элементов разметки.

(20) Редукция многословного термина в левой части словарной статьи:

Полным двудольнымназываетсядвудольный граф, в котором ….

(21) Редукция многословного термина в правой части словарной статьи:

- *ориентированный<a>граф*,
- *отношение называется<a>нетранзитивным, а если не выполняется ни для какой, тройки элементов, то -<a>антитранзитивным*.

Г. Неполнота стандартных моделей.

(22) Наличие примыкающих неизменяемых зависимых (наречия, частицы): *вполне несвязный граф, правила де моргана*.

(23) Наличие управляемых зависимых, отличных от генитивных: *система управления производством*.

(24) Возможность употребления страдательного причастия в адъективных позициях, не учтенная в стандартных моделях с прилагательным, кроме П+С, например, *закон исключенного третьего* по модели С+Прд+Срд.

(25) Возможность субстантивации адъективных компонентов (прилагательных и причастий), не учтенная в стандартных моделях с существительным: *дерево составляющих.*

Д. Нестандартная структура термина (длинные и инверсные термины).

(26) Именные группы большой длины при наличии вложенных именных групп в составе сложной *закон снятия двойного отрицания, компонента сильной связности графа*. В этой ситуации существуют полные покрытия стандартными моделями с пересечением и без пересечения: *закон снятия, двойное отрицание, снятие двойного отрицания*;

(27) Именные группы с инверсией позиции согласованного прилагательного. Стандартная инверсная модель С+П покрывает подавляющее большинство инвертированных терминов. Однако ряд случаев инверсии остается неучтенным. Их можно рассматривать либо как инверсный вариант исходной модели П+С+Срд (*система управления автоматизированная*) или П+П+С (*сеть семантическая интенсиональная. унификатор наибольший общий,*), либо как комбинацию исходных моделей, П+С и С+Срд+Срд (*язык представления знаний логический*), а также П+С и С+Прд+Срд (*система пятого поколения вычислительная*).

Избыточность в списке многословных терминов.

(28) Омонимия словоформ в текстовых позициях, удовлетворяющих условиям сборки по стандартным моделям; в результате наряду с верным термином формируются его “ложные” дубликаты:

- *база знаний – *баз знаний* (4 варианта сборки в позиции *<h3>база знаний</h3>* на основе омонимии словоформы *база* по роду и словоформы *знаний*, реализующей два стилистических варианта);
- *гомеоморфный граф – *гомеоморфная графа* (3 варианта сборки в позиции *гомеоморфные графы* на базе омонимии словоформы *графы* по роду и одушевленности);
- *тип данных, ациклический граф, диаметр графа, обработка естественного языка* (омонимия по одушевленности);
- *логика здравого смысла – *логик здравого смысла* (омонимия по роду);

- *машина параллельного вывода – *машин параллельного вывода, ориентированный граф – *ориентировать граф* (частеречная омонимия);
- *автомат линейно-ограниченный – *автомат линейный-ограниченный* (вариативность предсказаний леммы дефисного прилагательного).

Некорректность сборки термина по стандартной модели распадается на два подтипа.

А. Некорректная нормализация.

(29) Неверная нормализация по модели П+С, где в позиции П – страдательное причастие: **пометить граф* (вместо *помеченный граф*), **расширить сеть* (вместо *расширенная сеть*), **породить подграф* (вместо *порожденный подграф*).

(30) Нетерминологичность стандартной нормализации по числу для множественных понятий – ср. (9): *кратные ребра* (несколькоребер,инцидентныходной и той же паре вершин); *гомеоморфные графы* (графы, получаемые из одного графа с помощью последовательности подразбиений ребер).

(31) Случаи рассогласования: **метод прямого волны* vs. *метод обратной волны*.

(32) Случаи неверного выбора падежной формы: **рассуждение по аналогия* vs. *рассуждение по умолчанию*.

Б. Некорректный термин определяется в ситуации, когда термин построен по стандартной модели на базе неверно (по совокупности параметров) предсказанных компонентов. В отличие от предыдущего типа, корректный вариант в списке словосочетаний отсутствует.

(33) Некорректная лемма дефисного прилагательного – **аппаратный-программное средство*.

(34) При наличии грамматически правильного предсказания компонентов-прилагательных, таких как *абдуктивный, деонтический, когнитивный*, словосочетания *вывод абдуктивный, логика деонтическая, диссонанс когнитивный* строятся по модели С+Срд на базе неверного предсказания части речи и класса словоизменения (неизменяемое существительное). Словосочетания *структура/графика/психология/модель когнитивная* строятся по модели С+Срд на базе компонентов-прилагательных, неверно предсказанных как неизменяемые существительные с леммой *когнитивная*. Заметим, что для терминов с препозицией прилагательного *когнитивная наука/психология/процесс/структура* используется верное предсказание и корректная сборка по модели А+С, что позволяет предположить ошибки перебора вариантов при наличии альтернативных предсказаний с частеречной омонимией.

(35) В отсутствие грамматически правильного предсказания компонента термина (например, притяжательных прилагательных от фамилий), словосочетания собираются по

неверным моделям С+Сим (**эйлеров цепь*), С+Срд (**гамильтон граф,*ламан граф),* либо вообще не собираются (отсутствуют *эйлеров цикл, эйлеров путь, гамильтонов цикл, гамильтонов путь*).

3.4. Эвристические методы

Рассмотренные выше проблемы выявили необходимость разработки специальных методов, позволяющих разрешать неоднозначность, формировать «недостающие» термины, улучшать качество синтеза нормальной формы терминов и т.п.

3.4.1. Сравнение с «эталоном»

Особенностью рассматриваемого жанра электронного ресурса является структура словарных статей, в которых выделяются левые части, с определяемыми терминами, и правые части – с толкованиями. Практически всегда, за исключением случаев использования предиката *называется*, в левой части термин представлен в «эталонном» виде. Под эталонном понимается нормализованная форма термина, принятая в данной предметной области, которая, как правило, представляет собой именную группу, главным термином которой является существительное в именительном падеже. Указанная жанровая особенность позволяет автоматически выделить границы терминов и построить эталонный словарь терминов, представляющих собой эталонные строковые выражения.

Анализ и корректировка базового словаря (однословных и многословных терминов, полученных системой Клан) опирается на информацию о границах эталонных терминов и сопоставление эталонного выражения с найденными в данных границах однословными и многословными терминами базового словаря.

Сравнение найденных терминов с эталонным образцом позволяет разрешить следующие проблемы путем выбора правильного варианта, совпадающего по норме с эталоном (в скобках будем указывать номера из классификатора проблем выделения терминов, представленного в п. 3.3):

a) однословная омонимия (1-4),
b) вариативность многословного термина (28), причиной которой чаще всего является вариативность компонентов (3-6).

В остальных случаях, когда правильный вариант в списке терминов отсутствует, необходимо проанализировать компонентный состав эталонного термина и сопоставленных ему вариантов.

При сравнении границ термина и его нормальной формы с эталонными можно выделить следующие виды несоответствия.

a) Неполнота определения термина в случаях, когда длина термина меньше эталонного выражения (7,8,16-19).
b) Неполнота определения термина в случаях, когда не нашлось подходящей модели сборки. Такая ситуация характеризуется покрытием эталонного выражения совокупностью терминов (22-27).
c) Формирование (синтез) неправильной нормальной формы термина (8-12,29-35) в границах эталонного термина.
    - Ошибки выбора формы зависимого компонента при стандартной нормализации (29,31,32).
    - Нетерминологичность стандартной нормализации (9,10,30).
    - Ошибки нормализации первой части дефисных терминов, как правило, прилагательных (8,11).
    - Ошибки предсказания компонента термина (12) – при отсутствии правильного варианта (35) либо при выборе неправильного варианта (34).
d) Отсутствие термина или компонента термина, определяемого в границах эталонного термина, в базовом словаре (13-15).

Предлагаемый метод решения заключается в создании корректных составных терминов на основе шаблонных описаний и опирается на разработанную типизацию ошибок формирования терминов (п.3.3.).

3.4.2. Лексико-синтаксические шаблоны

Для решения задач описания сложных составных терминов, а также для проверки различных эвристик был предложен язык лексико-синтаксических шаблонов.

В работе [8] лексико-синтаксический шаблон определяется как модель (структурный образец) языковой конструкции, в котором указываются существенные грамматические характеристики множества лексем, которые входят в языковые выражения, принадлежащие данному классу, и синтаксические условия употребления языкового выражения, построенного в соответствии с шаблоном (например, правила согласования морфологических признаков лексем). В [1] предусмотрена возможность описания сложного шаблона с использованием именованных подшаблонов, таких как грамматически согласованная именная группа. Т.о. шаблоны представляют собой формальные описания лингвистических свойств языковых выражений, формулируются экспертами (лингвистами или специалистами по ПО) и

используются для автоматического выделения конструкций в тексте (извлечение именованных сущностей, терминов, их связей и т.п.). Так, в недавней работе [3] шаблоны применяются в задаче извлечения и отбора терминов для предметных указателей.

Предлагаемый язык шаблонов характеризуется следующими отличительными особенностями:

1) поддержка логического комбинирования лексических элементов – элементами конструкции могут выступать однословные или многословные термины (формируемые технологией Клан), элементы разметки или произвольные символьные последовательности;
2) поддержка альтернативных вариантов при описании элементов языковой конструкции;
3) именование шаблонов, позволяющее определять все более сложные шаблоны на основе уже известных шаблонов и тем самым постепенно наращивать их мощность.

Шаблон определяет структурный образец целевой языковой конструкции, ее лексический состав и поверхностно-синтаксические свойства. Объекту, найденному по шаблону, приписываются соответствующие признаки – имя (совпадающее с именем шаблона), грамматические характеристики, вычисляемые на основе пересечения множеств характеристик составляющих элементов (если они присутствуют), и формальные (позиционные) признаки. Синтаксис языка лексико-синтаксических шаблонов, как и язык описания сегментов, основан на языке JSON, что позволяет применять стандартные парсеры и редакторы для просмотра и отладки шаблонов.

Особенностью нашей задачи является необходимость автоматической генерации шаблонов, представляющих многокомпонентные термины предметной области.

Общая структура шаблона, представляющего термин, выглядит следующим образом:

```
{
   "name": "ЭТАЛОННОЕ_ИМЯ_ТЕРМИНА",
   "segments": [  // множество разнотипных элементов
            // список возможных типов элементов
            "ph/МНОГОСЛОВНЫЙ_ТЕРМИН",
            "w/ОДНОСЛОВНЫЙ_ТЕРМИН/индекс_морф_класса",
            "s/СТРОКА",
            "ИМЯ_ШАБЛОНА"
    ],
   "type": "sequence"/ "or" // последовательность или альтернативы
}
```

Данная схема демонстрирует поля json-формата для внесения данных и приводит возможные значения, используемые для описания шаблонов.

Генерация шаблонов осуществляется за счет наличия эталонного термина, формирующего имя шаблона, границ эталонного термина в левой части определения (которые обычно выделяются с помощью тегов) и покрытия данного фрагмента текста терминами словаря Клан, из которых формируется состав шаблона (тип *sequence*). Для представления альтернатив формируется шаблон с типом *or*.

Реализуются следующие варианты формирования шаблона.

A. Шаблон формируется из вложенных терминов (22-27).

```
{
    "name": "закон снятия двойного отрицания",
    "segments": [
            "w/закон/e",
            "s/ ",
            "w/снятие/k",
            "s/ ",
            "ph/двойное отрицание",
     ],
     "type": "sequence"
}
```

B. Шаблон формируется из комбинации строк и/или вложенных терминов (7, 8, 13, 15-17).

```
{
    "name": "k-дольный граф",
    "segments": [
            "s/k-",
            "ph/дольный граф"
    ],
    "type": "sequence"
}
```

Частным случаем такого шаблона является разбиение существующих терминов на компоненты (11, 33, 35).

```
{
    "name":"граф-звезда"
     "segments": [
            "s/граф-",
             "w/звезда/c"
     ],
     "type": "sequence"
}
```

C. Шаблон формируется из полного термина, но с другим эталонным именем (9, 10, 30).

```
{
    "name": "кратные ребра",
    "segments": [
            "ph/ кратное ребро"
    ],
    "type": "sequence"
}
```

D. Шаблон формирует список альтернатив (синонимов):

```
{
    "name": "автоматизированная система управления",
    "segments": [
            "ph/автоматизированная система управления",
            "s/АСУ"
    ],
    "type": "or"
}
```

или альтернативу с уточнением (18):

```
{
    "name": "глубинная структура (предложения)",
    "segments": [
            "ph/глубинная структура",
            "ph/глубинная структура предложения"
    ],
    "type": "or"
}
```

E. Шаблон формируется с использованием вложенных шаблонов.

```
{
    "name":"автоматизированная система управления предприятием"
     "segments": [
            "автоматизированная система управления",
            "s/ ",
             "w/предприятие/k"
     ],
     "type": "sequence"
}
```

В общем случае для формирования шаблона выбирается вариант A, при наличии полного покрытия терминами эталонного выражения, и вариант B – при неполном покрытии. Вариант C выбирается при отсутствии «правильных» терминов (т.е. терминов или их компонентов, совпавших с эталоном) и, как правило, комбинируется с A путем создания альтернативы термина с разбиением. Альтернативы создаются с помощью варианта D. Если термин, для которого уже создан шаблон, входит в состав другого термина, то используется шаблон E.

Термины специального вида требуют применения специализированных методов их обработки.

3.4.3. Методы обработки «особых» терминов

Метод обработки определения, использующего предикат называния. Данный метод позволит сформировать термин с помощью лексико-синтаксического шаблона на основе структурного критерия. Основная идея заключается в формировании термина как шаблона, включающего элементы, расположенные слева и справа от слова *называется (обозначается)*. При этом случай, когда в качестве компонентов шаблона выступают термины базового словаря, не вызывает трудности (вариант формирования шаблона А). Проблемными являются ситуации омонимии (здесь уже не получится использовать эталон) и наличия повторов в левой и правой частях.

Метод обработки инверсных терминов. Данный метод применяется для энциклопедических источников, в которых заглавные термины имеют инвертированный порядок слов.

В рамках метода осуществляется сборка групп прилагательных, стоящих после главного существительного, и их перестановка в начало термина перед главным существительным. Итоговый термин формируется как шаблон, включающий множество альтернатив – инверсный термин и варианты с перестановкой (если прилагательных несколько, то вариантов тоже может быть несколько).

```
{
    "name": "ЗНАНИЯ ПРАГМАТИЧЕСКИЕ",
    "segments": [
            "ph/знание прагматическое",
            "ph/прагматическое знание"
    ],
    "type": "or"
}
```

Метод обработки дефисных предсказаний. Следует отметить, что выявить наличие ошибки в предсказании дефисных терминов удается не всегда. Существуют случаи, когда нормальная форма термина предсказывается корректно (т.е. совпадает с эталонной), но парадигма неправильная (например, *граф-звезда* из случая (11)), что в дальнейшем приведет к проблемам при поиске вхождений данного термина в текстах. Поэтому для всех предсказанных терминов с дефисом осуществляется общая схема обработки по следующему принципу.

Если для термина выполняются следующие условия:

- термин – *существительное*,
- первая часть дефисного слова – *существительное*,
- у первой части нет признака *неизменяемый*

⇒ Формируется шаблон с разбивкой существующих терминов (вариант B формирования шаблонов).

В остальных случаях применяется стандартная процедура сравнения с эталоном и либо выбор совпадающего с эталоном варианта, либо формирование шаблона из частей, если совпадения с эталоном нет.

Метод синтеза сокращения. Как правило, сокращения в левых частях определений указываются в скобках. Однако, помимо сокращений, в скобках может быть указано уточнение термина (18), либо набор его синонимов. Возникает необходимость проверки, является ли конструкция в скобках акронимом или графическим сокращением (13). Такая проверка осуществляется путем сопоставления букв предполагаемого сокращения с начальными буквами слов термина. Если конструкция оказывается сокращением, то формируется шаблон с альтернативами (вариант формирования D).

Метод обработки терминов в скобках. Данный метод применяется для анализа терминов, представленных в скобках в левых частях словарных статей.

Выражение в скобках:

- множество терминов через запятую – считаем, что это синонимы;
- конструкция, включающая точки, – сокращение;
- однословный термин, содержащий заглавные буквы, – термин проверяется на сокращение методом синтеза;
- термин, вершина которого является существительным в именительном падеже, – синоним;
- другое – уточнение термина.

Если термин в скобках – уточнение или сокращение, то формируется шаблон с альтернативой (вариант формирования D). Имя шаблона будет содержать значения в скобках в случае уточнения.

Выявление синонимов требует специальной обработки, рассмотрение этой проблемы не входит в задачи данной работы.

3.4.4. Дополнительные методы формирования терминов

Методы, указанные в данном подразделе, пока не получили достаточной экспериментальной оценки. Мы приводим их исключительно для представления более полной картины покрытия выделенных видов проблем и методов их решения.

А. Метод разрешения омонимии (вариативности) на основе статистики. Идея данного метода связана с тем, что у омонимичных вариантов термина могут быть разные парадигмы,

соответственно, анализ текста всей энциклопедии (или текстов данной ПО в целом) может дать статистический критерий – правильный вариант будет встречаться чаще.

B. Дополнительной обработки требует ситуация тегов, идущих подряд. Теги, если они присутствуют в тексте, служат границами терминов. Ошибки в расстановке тегов или внутренние теги, призванные выделить термин внутри термина, приводят к формированию «ложных» границ терминов. Однако явного критерия ошибочности в расстановке тегов пока не обнаружено.

C. Метод приоритетов. В зависимости от предметной области, пользователь предварительно может указать «приоритеты» грамматических признаков, например, исключать термины-глаголы (полностью или при неоднозначности, как в (4)), исключать одушевленные существительные, не рассматривать фамилии и т.п.

4. Результаты экспериментов

Метод автоматической оценки правильности формирования словаря терминов опирается на сравнение полученного словаря с эталонным.

В таблице 1 приведено сравнение основных качественных показателей обработки двух энциклопедических словарей базовыми методами (базовый словарь) и с добавлением эвристик (итоговый словарь), а также произведена оценка качества терминологического ядра формируемой онтологии. Терминологическое ядро – это множество имен понятий, полученных из итогового словаря путем объединения альтернатив, т.е. терминов, найденных в одной позиции в левой части словарных статей источников, с помощью лексико-синтаксических шаблонов.

Таблица 1. Результаты экспериментов.

Источник	Базовый словарь		Итоговый словарь		Терминологическое ядро онтологии		
	Полнота	Точность	Полнота	Точность	Полнота	Точность	F-мера
Теория графов	0,83	0,36	0,90	0,45	0,90	0,78	0,84
Искусственный интеллект	0,86	0,46	0,97	0,51	0,996	0,998	0,997

Как видно из приведенной таблицы, применение эвристических методов дало возможность значительно улучшить качество построения терминологический словарей. Более низкие показатели для словаря по теории графов объясняются тем, что не удалось в полной мере

решить проблему снятия омонимии для существительных, различающихся по одушевленности, например, для термина *граф* и всех образуемых от него однословных (*мультиграф, орграф, подграф, псевдограф* и т.п.) и многословных терминов.

Заключение

В работе предложена методология построения терминологического ядра предметной области на базе электронных энциклопедических источников данных. Особенностью предлагаемого похода является тщательный анализ структуры термина, распознавание ошибок на базе их лингвистической классификации, автоматическая генерация лексико-синтаксических шаблонов, представляющих многокомпонентные термины, и использование набора эвристических методов обработки «особых» терминов. Применение указанного подхода позволило значительно повысить качество создаваемых словарей по сравнению с классическими методами обработки текста (морфологическим и поверхностно-синтаксическим анализом).

Создаваемое терминологическое ядро является основой для дальнейшего построения онтологии предметной области, качество которой в значительной степени зависит от корректности выделения терминов. Следующим шагом исследования является применение созданных словарей для анализа правой части словарных статей источников и автоматическое выявление синонимов и родовидовых отношений между терминами предметной области.

Работа выполнена при поддержке Президиума СО РАН (Блок 36.1. Комплексной программы ФНИ СО РАН II.1).

Список литературы

1. 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: тр. межд. конференции Диалог‘2007. М.: Изд-во РГГУ, 2007. С. 70-75.
2. Большакова Е.И., Васильева Н.Э. Терминологическая вариантность и ее учет при автоматической обработке текстов // Одиннадцатая национальная конференция по искусственному интеллекту с международным участием: труды конференции. М., 2008. Т. 2. С. 174-182.
3. Большакова Е.И., Иванов К.М. Выделение терминов и их связей для предметного указателя научного текста // Сборник трудов XVI Национальной конференции по искусственному интеллекту с международным участием (КИИ-2018). Т1. М.: РКП, 2018. С. 253-261.

4. Добров Б.В., Лукашевич Н.В., Сыромятников С.В. Формирование базы терминологических словосочетаний по текстам предметной области // Труды пятой всероссийской научной конференции «Электронные библиотеки: Перспективные методы и технологии, электронные коллекции». 2003. С. 201-210.
5. Захаров В.П., Хохлова М.В. Автоматическое выявление терминологических словосочетаний // Структурная и прикладная лингвистика. С.-Петербург: Изд-во С-Петерб. гос. ун-та, 2014. Вып.10. С. 182–200.
6. Киселёв Ю.А., Поршнев С.В., Мухин М.Ю. Метод извлечения родо-видовых отношений между существительными из определений толковых словарей // Программная инженерия. 2015. №6. С. 34–40.
7. Лезин Г.В., Клименко Е.Н., Силина Е.Ф. Онтологическая интерпретация дефиниций терминологического словаря // Прикладная лингвистика в науке и образовании: сб. трудов VII межд. конференции. СПб.: «Книжный дом», 2014. С. 50–54.
8. Митрофанова О.А., Захаров В.П. Автоматизированный анализ терминологии в русскоязычном корпусе текстов // Компьютерная лингвистика и интеллектуальные технологии: тр. межд. конференции «Диалог–2009». М.: 2009. С. 321-328.
9. Рабчевский Е., Булатова Г., Шарафутдинов И. Формализм записи лексико-синтаксических шаблонов в задаче автоматизации процесса построения онтологий // Труды десятой всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» –RCDL'2008. Дубна: ОИЯИ, 2008. С. 103-106.
10. Рубашкин В. Ш., Бочаров В. В., Пивоварова Л. М., Чуприн Б. Ю. Опыт автоматизированного пополнения онтологий с использованием машиночитаемых словарей. // Компьютерная лингвистика и интеллектуальные технологии. По материалам ежегодной международной конференции «Диалог». Вып. 9 (16). М.: Изд–во РГГУ, 2010. С. 413–418.
11. Рубашкин В.Ш., Капустин В.А. Использование определений терминов в энциклопедических словарях для автоматизированного пополнения онтологий // XI Всероссийская объединенная конференция «Интернет и современное общество». СПб., 2008. С.32-39.
12. Сидорова Е. А. Многоцелевая словарная подсистема извлечения предметной лексики // Компьютерная лингвистика и интеллектуальные технологии: тр. межд. конф. «Диалог–2008». М.: 2008. Вып. 7 (14). М.: Изд–во РГГУ, 2008. С. 475-481.
13. Сокирко А.В. Морфологические модули на сайте www.aot.ru // Компьютерная лингвистика и интеллектуальные технологии: Тр. межд. конференции Диалог’2004 / Под ред. И.М.Кобозевой, А.С. Нариньяни, В.П. Селегея. М.: Наука, 2004. С. 559-564.
14. Chodorow M.S., Byrd R.J., Heidorn G.E. Extracting semantic hierarchies from a large on-line dictionary// Proc. of the 23rd Annual Conference of the Association for Computational Linguistics, Chicago, 1985. P. 299–304.

15. Dolan W., Vanderwende L., Richardson S. Automatically Deriving Structured Knowledge Bases From on-Line Dictionaries // Proc. of the Pacific Association for Com putational Linguistics. PACL Press, 1993. P. 5–14.
16. Navigli R., Velardi P. From Glossaries to Ontologies: Extracting Semantic Structure from Textual Definitions // Proc. of the conference on Ontology Learning and population: Bridging the Gap between Text and Knowledge. IOS Press Amsterdam. 2008. P. 71–78.