УДК: 519.688

Название: Система создания электронных архивов газет с поиском по ключевым словам

Авторы: Марчук А.Г. (Институт систем информатики им. А.П. Ершова СО РАН),
Лештаев С.В. (Институт систем информатики им. А.П. Ершова СО РАН).

Аннотация: В данной статье рассматривается вопрос сохранения архивов газет в цифровой форме. Предлагается технология, охватывающая цикл сканирование-подготовка-публикация, причем в качестве ключевых задач представлены: отображение на сайте материала выпусков газет и поиск статей по заданию текстового поискового образа (ключевых слов).

Ключевые слова: газета, сканирование, публикация, Silverlight, Deep Zoom.

1. Введение. Тенденция создания электронных библиотек на основе цифровых технологий охватывает всё большие предметные области. Данная тенденция актуальна в отношении архивов газет. Важно сохранить архивы газетных подшивок так, чтобы пользователи могли получить доступ к их содержимому современными средствами. Это положение касается не только газетного материала известных изданий, но и малотиражных, ведомственных, заводских и вузовских газет, настенных публикаций. Современные технологии позволяют сохранить образ газетного выпуска и предоставить удобные средства просмотра и поиска по множеству выпусков. Архивы газет, как элементы общей архивной системы, могут быть обработаны для упорядочивания и поиска содержащейся в них информации.

В Институте систем информатики осуществляется проект по формированию современно устроенного электронного архива выпускавшегося с 1961 года еженедельника Сибирского отделения РАН «Наука в Сибири» (до 1983 г. «За науку в Сибири»). С конца 1997 года газета выкладывается на веб-сайте 1. В рамках проекта предыдущие выпуски были оцифрованы, сформирован электронный массив, размещенный на платформе Электронного фотоархива СО РАН http://soran1957.ru .

В статье рассматриваются созданные и апробированные основные технические решения по созданию электронных архивов газет. Достаточно подробное изложение ряда программистских решений приведено в магистерской диссертации [1].

1.а. Научная новизна проекта. В рамках междисциплинарного взаимодействия – информатики и источниковедения осуществляется проект подготовки электронной версии архива еженедельника СО РАН «Наука в Сибири». Проект обеспечит доступность контента в Сети широкому кругу исследователей-историков науки, журналистики, всем

интересующимся историей СО РАН. Для этого используется технология отображения изображений высокого разрешения Deep Zoom.

2. Обзор некоторых известных решений. Корпорация Google разработала своё специализированное решение по публикации выпусков газет – Google Newspapers [5]. В сервисе Google News опубликованы некоторые архивы газет с 1738г. по 2009г [10]. Пользователю предоставляется просмотр сканов выпусков с помощью технологии Google, специально разработанной для этих целей на JavaScript в HTML. За основу взята технология для просмотра карт: изображение разделено на клетки 256x256 разного масштаба, область обзора перемещается с помощью курсора. Добавлена «мини-карта», отображающая положение и размер области обзора относительно изображения выпуска, составленного из горизонтального ряда страниц. Сделана попытка определения позиции и размера заголовков, чтобы при клике на заголовок область обзора перемещалась и масштабировалась на начало статьи.

Сайт issuu.com [12] позволяет публиковать выпуски журналов с помощью технологии, использующей для просмотра Adobe Flash приложение Issuu Reader. Недостатки этого подхода заключаются в следующем: 1) Adobe Flash-приложение при работе нагружает процессор; 2) приложение нельзя модифицировать. Несмотря на это, многие сайты публикуют журналы и комиксы в этой технологии, поскольку она обладает рядом преимуществ. Текст с изображений страниц распознается вместе с позициями слов, что позволяет подсвечивать релевантные слова на изображении при поиске. Источником данных в этой технологии является коллекция страниц выпуска в PDF файле. Интерфейс приложения Issuu Reader позволяет просматривать страницы: коллекцией в виде ровного вертикального ряда, подобно приложениям для просмотра PDF; и по одной, с приближением и анимацией перелистывания страницы.

3. Ввод газетных выпусков и первичная обработка. В созданной в ИСИ технологии обработке подвергается как множество отдельных выпусков газет, так и каждый выпуск в отдельности. Выпуск (номер) газеты в системе структуризации рассматривается как отдельный документ, состоящий из упорядоченного множества страниц. Страницы газеты сканируются и их электронные образы (далее сканы) размещаются в хранилище для последующей визуализации. Кроме того, формируется база данных выпусков газет и их страниц. Выпуск, рассматриваемый как документ, содержит: дату выпуска, название и некоторые другие характеристики, экстрагируемые или заполняемые при сканировании. Обычно название документа совпадает с порядковым номером выпуска в газете. В свою очередь, выпуск рассматривается как документ, состоящий из многих частей, каждая часть

соответствует сканированному участку. Задача упорядочивания частей для получения нумерованной последовательности страниц, возложена на оператора сканирования.

Множество страниц сканируются, изображения подвергаются графической обработке и сохраняются. Преобразование производится в несколько естественных стадий:

1) Листы сканируются с качеством, сохраняющим существенные детали. Выбор параметров сканирования устанавливается заранее. Существенными являются: цветность, разрешающая способность, формат сохранения.

2) Каждый скан листа (разворота) разделяется на две страницы, если сканировался разворот. Иногда такое разделение нецелесообразно, тогда на скане могут остаться две страницы. Страницы обрезаются по краям, упорядочиваются по порядку номеров страниц, при необходимости поворачиваются.

3) Полученные изображения страниц сохраняется в репозитории документов с соблюдением иерархии «газета - выпуск – страница».

Полученные директории со сканами газет превращаются в электронный архив документов и хранятся как исходный материал. Создание и ведение архива выпусков газет, организовано в виде серверного приложения.

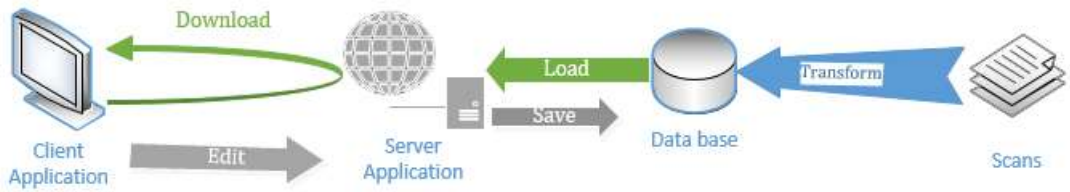

Рисунок 1. Схема web-публикации сканов газет.

На Рисунок 1 приведена общая схема клиент-серверной публикации, в которой электронный архив сканов предварительно преобразуется (Transform) в данные базы данных (Data base). Серверное приложение (Server Application) на запросы от приложений пользователей (Client Application): «получить для просмотра указанный выпуск газеты», загрузит (Load) его из базы данных и отправит в ответ.

Первая стадия схемы публикации – это преобразование электронного архива сканированных страниц в формат базы данных. В качестве базы данных и базы документов (репозитория), используется специальное решение, названное «кассеты» [2]. Такой подход дал возможность распространить способы, технологии и интерфейсы, общие для архивной платформы [3].

Следующие преобразования сканов связаны с проблемами, возникающими на стадии скачивания (Download) приложениями пользователей с серверного приложения. Сканированное изображение в хорошем качестве занимает от 40мб в TIFF формате.

Скачивание пользователем даже одной страницы в таком формате, может занять значительное время. Кроме того, просмотр изображения высокого разрешения в файле больших размеров требует достаточной вычислительной мощности. Нельзя полагаться на то, что конечный компьютер пользователя будет обладать достаточной производительностью и скоростью интернета. Для решения подобной проблемы файл изображения преобразуют с потерей качества в форматы, доступные для демонстрации из браузера: PDF, JPG, PNG. В случае публикации газеты с неудачно подобранными параметрами сжатия, потеря качества изображения для страницы приводит к тому, что шрифт становится сложно читаемым.

Сохранить качество и минимизировать объем передаваемой и обрабатываемой на стороне клиента информации позволяет технология Deep Zoom [11] – решение для web-публикации изображений высокого разрешения от Microsoft. Изображение предварительно преобразуется в формат DZI (*Deep Zoom Image)*, при этом оно разделяется сеткой на кусочки размером 256x256 и каждый сохраняется в формате для быстрого скачивания через интернет JPG или PNG. Имеется несколько вариантов использования технологии для браузеров. Мы использовали технологию Silverlight [17], которая реализована для всех популярных браузеров, не требует от пользовательского компьютера больших вычислительных ресурсов и памяти. Silverlight-приложение в браузере пользователя позволяет обозревать изображение в целом, приблизить некоторую область, при этом приложение скачивает с сервера только попавшие в область фрагменты, по качеству соответствующие масштабу.

Deep Zoom позволяет так же просматривать коллекцию изображений. Для этого используется формат DZC (*Deep Zoom Collection)* – множество расположенных на одной плоскости изображений формата DZI различного масштаба.

Создание DZI и DZC возможно бесплатным пользовательским приложением *Deep Zoom Composer*, кроме того из коллекции изображений пользователь может указать относительные размеры и положения изображений, и получить:

(1) DZC;

*(2)*DZI каждого изображения в коллекции;

(3) DZI всей коллекции, как будто это одно изображение.

При преобразовании архива газет обработка каждого выпуска - коллекции страниц пользователем при создании DZC занимает много времени. Необходим автоматический метод. Deep Zoom Composer предоставляет в SDK .Net библиотеку, с помощью которой в эксперименте было разработано приложение автоматически создающее deepZoomImage и deepZoomCollection. Изображения в коллекции, созданной таким образом, располагаются в горизонтальный ряд без масштабирования.

Изображение в формате DZI можно показывать в Silverlight в компоненте MultiScaleImage или в ASP.NET с помощью компонента *SeaDragon* [16] в наборе AJAX Control Toolkit. При демонстрации deepZoomCollection в Silverlight, можно динамически менять положение и размер каждого изображения в коллекции. Так для демонстрации выпуска страницы выстраиваются в ровный горизонтальный ряд, каждая страница масштабируется до одинаковой константной высоты.

При обработке газеты «Наука в Сибири» выяснилось, что количество страниц архива получается довольно большое, при этом, каждая из страниц, разбивается на множество клеток размера 256x256. При прямом хранении имиджей в файлах, их количество приводит к техническим трудностям хранения и перемещения из-за большого количества файлов (в нашем случае – миллионы). Для решения указанных проблем множество составляющих клеток архивируется в один файл, когда приоритетным фактором является скорость разархивирования. Выбран формат архива без сжатия так, чтобы получить максимальную скорость выборки конкретного изображения. Архивирование произведено по выпускам, соответственно, когда пользователь запрашивает какой-то выпуск, все файлы данного выпуска экстрагируются из архива и сохраняются в кэше.

4. Колонка. Структурно газета рассматривается не только как множество страниц, но и как множество статей. Статья может размещаться на страницах частями: в конце каждой части, не являющейся последней, может быть указана ссылка на продолжение, например, «продолжение в следующем номере»; в каждой части, не являющейся первой, может быть ссылка на предыдущую часть. Следовательно, статья — это упорядоченное множество частей, каждая часть статьи содержит множество колонок, множество изображений с подписями. Кроме того, статья как объект данных имеет идентификатор, как документ имеет заголовок, авторов и дату выпуска первой части. Каждая часть напечатана в некотором выпуске на некоторой странице в некоторой позиции, может начинаться на одной странице, а заканчиваться на другой.

Статья структурно определяется как упорядоченное множество частей статьи в выпусках, причем для каждой части фиксируется позиция и размер охватывающей прямоугольной области.

В интерфейсе Silverlight-приложения прямоугольные области выделения расположены поверх MultiScaleImage в прозрачном контейнере в соответствующих позициях. При перемещении и масштабировании области обзора коллекции изображений в MultiScaleImage положения и размеры областей отражения перемещаются и масштабируются соответственно.

Статьи, персоны, события и организации, их отражения, вносят в данные, указывают положение и размер на страницах газет и редактируют пользователи с правами редактора специальными интерфейсами в приложении клиента.

5. Поиск. Задача поиска информации в тексте газет разделена на подзадачи:

1. Распознавание текста, т.е. отображение из множества сканов во множество текстовых файлов.

2. Индексирование текста поисковым движком с полнотекстовым индексом.

3. Полнотекстовый поиск.

5.1. Распознавание текста. В распоряжении программистов имеется множество программ для распознавания текста (OCR) по изображениям. Анализировались на предмет возможности использования OCR-приложения, удовлетворяющие условиям:

1) Осуществлять распознавание текстов на русском языке;
2) Предлагать бесплатное SDK – комплекта средств разработки инструментов и библиотек. При этом SDK может быть предоставлено сторонним приложением.
3) Предоставлять возможность распознавать директорию со всеми поддиректориями и файлами изображений TIFF.

Были протестированы существующие отдельно от приложений бесплатные движки распознавания текста: Cunei [7] и Tesseract [13]. Имеется также множество приложений и сервисов, которые используют готовые движки. С одинаковыми движками приложения и сервисы распознают текст одинаково.

В специальной литературе имеется ряд публикаций, в которых сравниваются OCR-приложения [18]. На практике качество распознавания зависит от качества печати и сканирования, шрифта и цвета изображения. Поэтому следует с осторожностью полагаться на результаты сравнения в данных статьях, если не известны исходные условия сканирования. Необходимо проверить тестовый образец изображения на множестве подходящих к условиям приложениях. В проведённых испытаниях с использованием приложения Cunei Forms было осуществлено распознавание на тестовом образце изображения большего процента слов, чем с помощью других. По этому показателю движок Cunei был выбран для использования в описываемой системе.

Для программируемого использования Cunei движка в C# имеется библиотека Puma.Net. В числе особенностей Puma.Net, которые оказались полезны:

- позволяет запускать только один процесс распознавания одного изображения, но можно перезапускать приложение для каждой страницы;
- нельзя сохранить результат в DOC формате, но можно в RTF.

- ~1% изображений страниц приводят к ошибкам, из-за которых приостанавливается автоматический процесс распознавания. Решение проблемы – ограничение по времени (до 10 минут) распознавания одного изображения.

Для сохранения распознанного текста файла был выбран формат RTF, поскольку:

- он содержит информацию об относительном расположении текста;
- Существуют .Net библиотеки для извлечения текста из RTF фалов;
- RTF фалы с распознанным текстом имеют приемлемые размеры.

При распознавании текста распознаются фотографии и рисунки на изображении, но для задачи поиска их наличие в выходном файле не требуется.

5.2. Индексирование. *Полнотекстовый индекс (Full Text Index)* – обработка массива текстов таким образом, чтобы, в дальнейшем, эффективно решалась задача лингвистического поиска слова в тексте или поиска по набору ключевых слов. Технически, индексируется текст в столбце таблицы в базе данных. Для наших целей требовались уже готовые базы данных, предоставляющие возможность формирования полнотекстового индекса, поддерживающие русский язык и предоставляющие полнотекстовый поиск. Например, Solr [6], MS SQL 2012 Server Express, и другие [9]. Последняя была выбрана для эксперимента.

Текст располагается в одном столбце блоками: по блоку в строке таблицы. Полнотекстовый индекс применяется к текстовому столбцу, размечая все слова, кроме стоп-слов, которые не несут смысловой нагрузки. Текст предварительно обрабатывается: слова приводятся к нормальной форме.

Для решения задачи поиска в тексте газет устанавливается соответствие слова странице, что позволяет найти множество страниц, содержащих указанное слово.

5.3. Полнотекстовый поиск. К индексированному текстовому столбцу применяются функции *полнотекстового поиска* [8] – по множеству слов определяются строки с ненулевым количеством использований слов в тексте колонки строки. Функции полнотекстового поиска встроены в MS SQL Server 2012 express.

6. Экспериментальное апробирование метода. В рамках проекта «Электронный фотоархив СО РАН» был проведён следующий эксперимент. Были сформированы сканы выпусков(~1700) газеты «За науку в Сибири» с 1965 по 1997. В TIFF формате сканы занимают объем более половины терабайта. Эти же имиджи, переведенные в Deep Zoom формат, составили более 70 гигабайт в JPG формате. На машине с Intel® Pentium®4 2,02ггц. 1,5гб. ОЗУ распознавание заняло несколько недель в результате приостановок из-за ошибок, на индексирование было затрачено несколько часов. Была сформирована база данных, выстроен полнотекстовый индекс.

Запрос осуществляется через задание набора ключевых слов, которые являются поисковым образом. Поиск релевантных запросу страниц занимает несколько секунд. Для данных была использована СУБД, где в RDF формате они сохраняются в XML файлах.

За НАУКУ в СИБИРИ

Орган парткома, комитета ВЛКСМ, объединенного комитета профсоюза, Президиума Сибирского отделения АН СССР.

№ 1 (1). | 4 июля 1961 г., вторник. | Цена 2 коп.

ГАЗЕТЕ «ЗА НАУКУ В СИБИРИ»

Президиум Сибирского отделения Академии наук СССР, партийный комитет, комитет комсомола и объединенный комитет профсоюза горячо приветствуют рождение нашей газеты «За науку в Сибири».

Издание многотиражной газеты — большое событие в жизни Сибирского отделения. Газета нужна всем и от нее многого ждут.

Она должна шире освещать творчество ученых, их упорную работу по координации научных исследований и внедрению достижений науки в производство. В газете необходимо особое внимание уделить выполнению обязательств, взятых учеными в честь XXII съезда КПСС, пропаганде опыта передовых людей науки. Нужно смело и принципиально критиковать недостатки в организации научной работы.

Сибирское отделение не только молодое, но и молодежное. Подавляющая часть наших ученых — молодежь. Воспитывать молодые кадры, освещать их смелые искания, стремление к новым достижениям, к новым открытиям — почетная задача газеты.

Пусть газета борется за улучшение быта, за интересный отдых ученых.

Пусть наша газета верно хранит славные ленинские традиции печати!

Президиум Сибирского отделения АН СССР.

Партийный комитет СО АН СССР.
Комитет ВЛКСМ.
Объединенный комитет профсоюза.

Счастливого пути к читателям

Редакция «Вечернего Новосибирска» сердечно поздравляет многотиражную газету «За науку в Сибири» с выходом в свет ее первого номера.

Восьмитысячный отряд ученых, лаборантов, рабочих и слушащих Сибирского отделения Академии наук получил замечательный подарок — свою газе-

вседневная связь с массами позволяет сделать газету «За науку в Сибири» содержательной, живой, злободневной и острой, подлинным помощником партийного комитета.

Пусть редколлегию нашей газеты никогда не покидает состояние творческого беспокойства, чувство

Президент АН СССР Келдыш Мстислав Всеволодович

Келдыш Мстислав Всеволодович

Рисунок 2. Изображение первой страницы первого выпуска газеты “За науку в Сибири” 1961 г. Выделена фотография президента АН СССР Келдыш Мстислава Всеволодовича. Под выделенной фотографией размещена ссылка на страницу с его информационным портретом.

ГАЗЕТЕ «ЗА НАУКУ В СИБИРИ»

Президиум Сибирского отделения Академии наук СССР, партийный комитет, комитет комсомола и объединенный комитет профсоюза горячо приветствуют рождение нашей газеты «За науку в Сибири».

Издание многотиражной газеты — большое событие в жизни Сибирского отделения. Газета нужна всем и от нее многого ждут.

Она должна шире освещать творчество ученых, их упорную работу по координации научных исследований и внедрению достижений науки в производство. В газете необходимо особое внимание уделить выполнению обязательств, взятых учеными в честь XXII съезда КПСС, пропаганде опыта передовых людей науки. Нужно смело и принципиально критиковать недостатки в организации научной работы.

Сибирское отделение не только молодое, но и молодежное. Подавляющая часть наших ученых — молодежь. Воспитывать молодые кадры, освещать их смелые искания, стремление к новым достижениям, к новым открытиям — почетная задача газеты.

Пусть газета борется за улучшение быта, за интересный отдых ученых.

Пусть наша газета верно хранит славные ленинские традиции печати!

Президиум Сибирского отделения АН СССР.

Партийный комитет СО АН СССР.
Комитет ВЛКСМ.
Объединенный комитет профсоюза.

Рисунок 3. Изображение области Рисунок 2 при приближении на неё с помощью Deep Zoom. Заметно улучшение качества картинки: шрифт менее размыт, его можно прочитать. Приведённое приближение не предельное.

По ключевым словам поиск находит только множество страниц. Пользователю необходимо самостоятельно искать внутри каждой найденной страницы ключевые слова и их контекст. Исследование продолжается в направлении определения и указания их позиций на странице.

Silverlight поддерживается многими браузерами, но только в Microsoft Windows. Например, распространены устройства с touch screen (что подходит для навигации в Deep Zoom) и с операционной системой Android. Для других операционных систем заявлена только альтернатива: Moonlight [14] (от разработчиков Mono [15]). Но текущая версия Moonlight последняя, работа над ней остановлена [4]. Вероятно, и Silverlight тоже последняя версия, поскольку активно предлагается поддержка браузерами стандарта HTML 5, обеспечивающего браузерам большую часть функциональности Silverlight.

Для операционных систем и браузеров, не поддерживаемых для публикации газет, необходимо найти альтернативы Silverlight. Решений может быть несколько: либо «подстроить» систему под Sea Dragon, либо подождать, пока разработчики не начнут поддерживать Deep Zoom Collection. Еще один путь – самостоятельно разработать Deep Zoom альтернативу на HTML 5, который бы поддерживался новыми версиями браузеров.

Заключение. Таким образом, при создании электронного архива газеты «Наука в Сибири» была апробирована технология отображения изображений высокого разрешения Deep Zoom. Использованное решение может быть заменено на Seadragon. Функция поиска по

ключевым словам может быть дополнено определением и отображением позиции на странице, альтернативными программами распознавания текста и индексирования.

Авторы выражают благодарность за помощь в проведении данного исследования сотрудникам Института систем информатики им. А.П. Ершова СО РАН: Фурсенко Алексею Александровичу, Павловской Ирине Юрьевне, Крайневой Ирине Александровне, Филиппову Владимиру Эдуардовичу.

Список Литературы

1. **Лештаев С.В.** Архитектура и программное обеспечение архивных фактографических систем: работа с многостраничными растровыми изображениями: дис… маг. 5.13.11. — Новосибирск, 2012.

2. **Марчук А.Г., Марчук П.А.** Особенности построения цифровых библиотек со связанным контентом // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Сб.трудов / XII Всеросс. научн. Конф. RCDL'2010, Казань, Россия 13–17 октября 2010 г. — Казань: Казан. ун-т, 2010. — С. 19–23.

3. **Марчук А.Г., Марчук П.А.** 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.

4. Мигель де Иказа: “Мы прекратили работу над Moonlight” [В Интернете]. - http://www.linux.org.ru/forum/talks/7810987

5. About Google News Archive Search [Online]. - HYPERLINK "http://support.google.com/news/bin/answer.py?hl=en&answer=1638638&topic=9312&ctx=topic" http://support.google.com/news/bin/answer.py?hl=en&answer=1638638&topic=9312&ctx=topic

6. Apache Solr [Online]. - HYPERLINK "http://lucene.apache.org/solr/" http://lucene.apache.org/solr/ .

7. CuneiForm [Online]. - HYPERLINK "http://en.wikipedia.org/wiki/CuneiForm_(software)" http://en.wikipedia.org/wiki/CuneiForm_(software)

8. Fulltext search [Online] // Wikipedia. - HYPERLINK "http://en.wikipedia.org/wiki/Full_text_search" http://en.wikipedia.org/wiki/Full_text_search .

9. Fulltext search engines [Online] // - HYPERLINK "http://www.mediawiki.org/wiki/Fulltext_search_engines" http://www.mediawiki.org/wiki/Fulltext_search_engines .

10. Google Newspapers [Online]. - HYPERLINK "http://news.google.com/newspapers" http://news.google.com/newspapers .

11. Inside Deep Zoom Part II: Mathematical Analysis [Online] / auth. Gasienica Daniel. - HYPERLINK "http://www.gasi.ch/blog/inside-deep-zoom-2/" http://www.gasi.ch/blog/inside-deep-zoom-2/ .

12. Issuu [Online]. - HYPERLINK "http://issuu.com/" http://issuu.com/ .

13. tesseract-ocr [Online]. - HYPERLINK "http://code.google.com/p/tesseract-ocr/" http://code.google.com/p/tesseract-ocr/.

14. Moonlight [Online]. - HYPERLINK "http://en.wikipedia.org/wiki/Moonlight_(runtime)" http://en.wikipedia.org/wiki/Moonlight_(runtime).

15. Mono [Online]. - HYPERLINK "http://www.mono-project.com/About" http://www.mono-project.com/About.

16. SeaDragon [Online]. // Wikipedia - HYPERLINK "http://en.wikipedia.org/wiki/Seadragon_Software" http://en.wikipedia.org/wiki/Seadragon_Software.

17. Silverlight [Online]. - HYPERLINK "http://www.microsoft.com/rus/silverlight/overview/default.aspx" http://www.microsoft.com/rus/silverlight/overview/default.aspx.

18. Кривошей А. Системы оптического распознавания текста в Linux - обзор и сравнительное тестирование. 2011 [Online] - HYPERLINK "http://rus-linux.net/nlib.php?name=/MyLDP/office/OCR/OCR_review.html" http://rus-linux.net/nlib.php?name=/MyLDP/office/OCR/OCR_review.html

УДК: 519.688

Title: System for creating digital archives of newspapers with keyword search

Authors: Alexander G. Marchuk (A.P. Ershov Institute of Informatics Systems)
Sergey V. Leshtaev (A.P. Ershov Institute of Informatics Systems).

Abstract: This article examines the issue of storage of digital archives of newspapers. Technology is proposed covering scanning-preparation-publication cycle, and as the key challenges presented: presenting online of editions of newspapers and search for articles by keywords.

Keywords: newspaper, scan, publish, Silverlight, Deep Zoom.

УДК 002.53:004.89

Сбор онтологической информации для интеллектуальных научных Интернет-ресурсов

Загорулько Ю.А. (Институт систем информатики СО РАН),

Боровикова О.И. (Институт систем информатики СО РАН),

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

Ахмадеева И.Р. (Новосибирский государственный университет)

В работе рассматриваются проблемы сбора информации для тематических интеллектуальных научных интернет-ресурсов, обеспечивающих систематизацию и интеграцию научных знаний, информационных ресурсов, относящихся к определенной области знаний, и средств их интеллектуальной обработки, а также содержательный доступ к ним. Предлагается подход к автоматизации сбора информации, объединяющий методы метапоиска и извлечения информации, базирующиеся на онтологиях и тезаурусах моделируемой области знаний.

Ключевые слова: *научный интернет-ресурс, метапоиск, извлечение информации, онтология, тезаурус.*

1. Введение

В мире накоплено огромное количество информации по различным областям знаний, причем значительная ее часть представлена непосредственно в сети Интернет, но, несмотря на это, проблема эффективного обеспечения научного сообщества информацией по интересующим его тематикам остается на повестке дня. Нерешенной остается и проблема удобного доступа к средствам обработки данных, собранных по этим тематикам. Даже уже представленные в Интернет в виде веб-сервисов реализации методов их обработки остаются недоступными широкому кругу пользователей из-за отсутствия содержательной информации о них.

Для решения этих проблем, в частности для информационной и аналитической поддержки научной и производственной деятельности в определенных областях знаний создаются тематические интеллектуальные научные интернет-ресурсы (ИНИР) [4].

Эффективность использования ИНИР будет тем выше, чем более полно в нем будет представлена информация по его тематике. Однако сбор и накопление такой информации – довольно трудоемкая задача, решить которую можно только за счет автоматизации сбора релевантной информации по тематике ИНИР из сети Интернет. Описанию подхода, поддерживающего такую автоматизацию, и посвящена данная работа.

Работа выполнена при финансовой поддержке РФФИ (проект № 13-07-00422) и Президиума РАН (интеграционный проект СО РАН № 15/10).

2. Информационная модель ИНИР

Тематический ИНИР представляет собой информационную систему, обеспечивающую систематизацию и интеграцию научных знаний и информационных ресурсов определенной области знаний, содержательный эффективный доступ к ним (поиск и навигацию) и средствам их интеллектуальной обработки.

Ядро информационной модели ИНИР составляет онтология, которая, вводя формальные описания понятий некоторой области знаний, типов информационных ресурсов и методов их интеллектуальной обработки в виде классов объектов и отношений между ними, одновременно задает структуры для представления информации о реальных объектах моделируемой области знаний, интегрируемых информационных ресурсах и методах и средствах их обработки. Данная информация хранится в контенте ИНИР в виде семантической сети, типы информационных объектов и отношений которой определяются классами объектов и отношений онтологии ИНИР.

На основе онтологии организуется удобная навигация по научным знаниям и информационным ресурсам, интегрированным в ИНИР, а также содержательный поиск данных и средств их интеллектуальной обработки.

В систему знаний ИНИР включен также тезаурус, содержащий термины моделируемой области знаний, т.е. слова и словосочетания, с помощью которых понятия онтологии представляются в текстах и пользовательских запросах. Тезаурус задает смысл понятий посредством соотнесения одних понятий с другими с помощью семантических отношений. Благодаря этому он может применяться при поиске и аннотировании информационных ресурсов, интегрируемых в ИНИР.

Поскольку предлагаемый подход к сбору информации из сети Интернет существенно базируется на онтологии ИНИР, рассмотрим ее подробнее.

Онтология ИНИР состоит из трех взаимосвязанных онтологий: онтологии области знаний ИНИР, онтологии научных информационных ресурсов и онтологии задач и методов.

Онтология области знаний ИНИР строится на основе двух базовых онтологий – онтологии научной деятельности и онтологии научного знания. Первая из этих онтологий включает классы понятий, относящиеся к организации научной и исследовательской деятельности, такие как *Персона, Организация, Событие, Конференция, Проект, Публикация* и др. Вторая базовая онтология содержит понятия, необходимые для представления научных дисциплин. В частности, она содержит такие классы, как *Раздел науки, Метод исследования, Объект исследования, Научный результат* и др.

Онтология задач и методов предназначена для описания задач, на решение которых нацелен ИНИР, и методов их решения. Кроме этих методов, в онтологии представлены методы интеллектуальной обработки данных, содержащихся в интегрируемых в ИНИР информационных ресурса, а также описания web-сервисов, реализующих эти методы.

Основным классом онтологии научных информационных ресурсов является класс *Информационный ресурс*, служащий для описания информационных ресурсов. Набор атрибутов и связей этого класса основан на стандарте Dublin core [8]. Он имеет следующие атрибуты: *название ресурса, язык ресурса, тематика ресурса, тип доступа к ресурсу* и т.п. Объекты этого класса связываются семантическими отношениями с другими информационными объектами, представляющими в контенте ИНИР организации, персоны, публикации, проекты, разделы науки и т.п.

3. Модель сбора информации

Прежде чем представить предлагаемую модель, заметим, что проблемой сбора информации из Интернет занимаются многие исследователи и разработчики. Однако, как показывает обзор [7], большая часть таких исследований направлена на извлечение информации, необходимой для решения задач электронной коммерции или анализа новостного потока и социальных сетей, и лишь их незначительная часть – на извлечение информации для нужд научной деятельности [5; 6].

Сложность задачи сбора информации для ИНИР определяется большим разнообразием видов извлекаемой информации и способов ее представления в Интернет. В частности, необходимо собирать информацию об организациях, проектах, публикациях, интернет-ресурсах, веб-сервисах и других сущностях, описываемых онтологией научной деятельности. Эта информация может быть представлена как в виде интернет-страниц, имеющих различную структуру, так и в виде текстовых документов в различных форматах. В связи с этим мы посчитали нецелесообразным использовать популярные в настоящее время методы извлечения информации, основанные на обучении на примерах (см. например, [10]), а

применили подход, базирующийся на онтологии. В соответствии с этим было решено для каждого типа сущностей (класса онтологии научной деятельности) разработать свой метод сбора и обработки информации, настраиваемый на предметную область и типы интернет-ресурсов и документов.

Заметим, что предлагаемый подход развивает методы сбора онтологической информации об интернет-ресурсах [2], разработанные в рамках технологии построения порталов научных знаний. В то же время он близок к подходу, представленному в [6], который базируется на концептуальной модели предметной области и предлагает использовать для каждого вида сущности свои шаблоны и обработчики.

Сбор информации для ИНИР включает следующие этапы:

1. Поиск релевантных области знаний ИНИР интернет-ресурсов и документов.
2. Извлечение информации из найденных интернет-ресурсов и документов.
3. Занесение полученной информации в контент ИНИР.

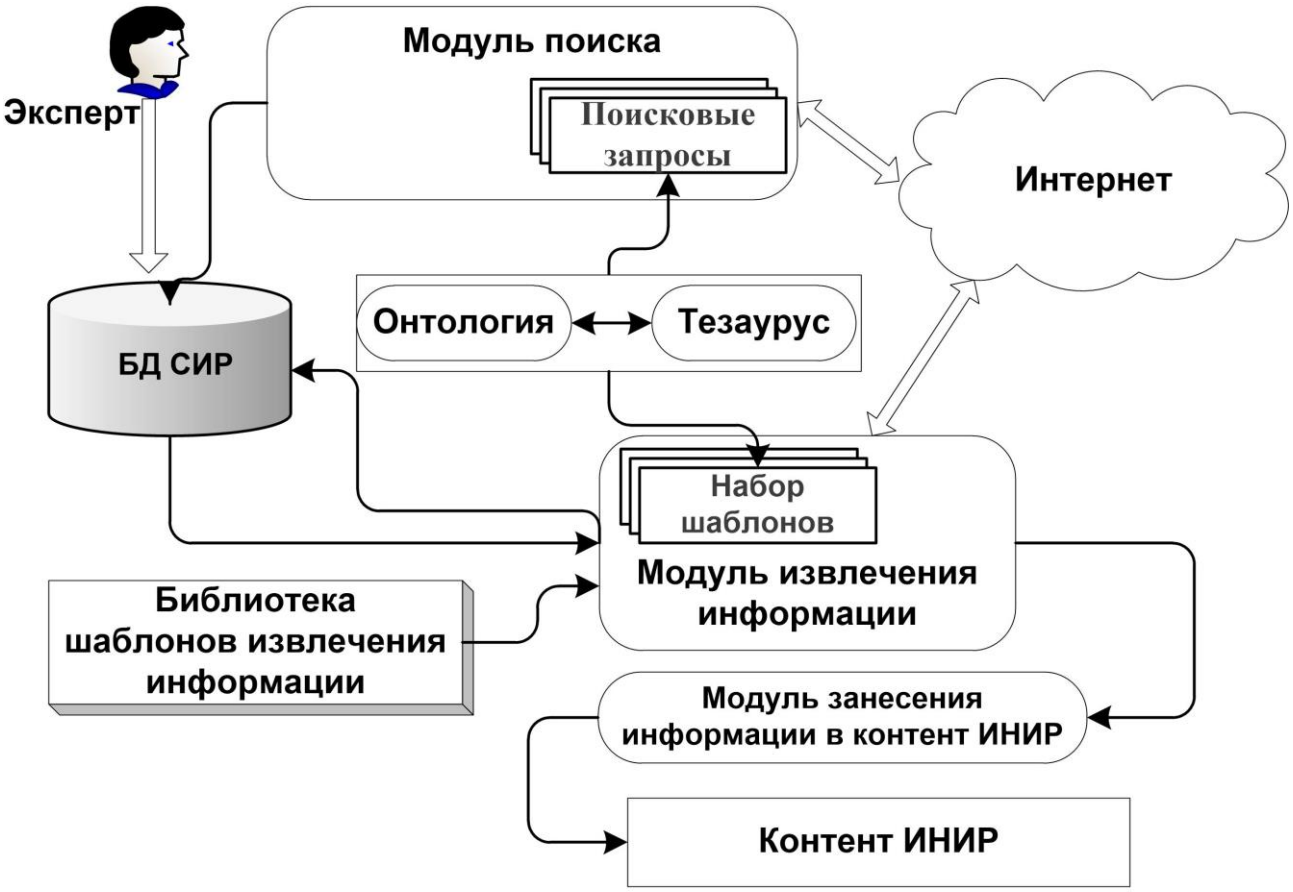

Рис.1. Подсистема сбора информации из Интернет

В соответствии с предложенной схемой подсистема сбора информации из сети Интернет включает следующие компоненты (см. Рис.1): модуль поиска релевантных интернет-

ресурсов, модуль извлечения информации из интернет-ресурсов, модуль занесения найденной информации в контент ИНИР, а также базу данных ссылок на интернет-ресурсы (БД СИР).

3.1. Сбор релевантных интернет-ресурсов

При настройке ИНИР на область знаний выполняется заполнение БД СИР ссылками на релевантные, по мнению экспертов, интернет-ресурсы. При этом для каждой ссылки указывается класс (классы) онтологии, объект (объекты) которого описывает соответствующий ей ресурс. С каждой ссылкой также связывается следующая метаинформация: дата загрузки, частота обновления, периодичность повторной закачки, дата последней проверки, статус обработки. Первые четыре параметра вводятся для отслеживания актуальности ресурса, последний – для указания статуса ссылки (релевантная, нерелевантная, необработанная).

Список ссылок пополняется не только вручную, но и автоматически – модулем поиска интернет-ресурсов, который выполняет сбор ссылок на релевантные интернет-ресурсы по поисковым запросам, сформированным на основе названий классов онтологии и терминов тезауруса, представляющих понятия моделируемой области знаний. Он запускается с заданной при настройке ИНИР периодичностью. При этом модуль поиска обращается к поисковым системам Google, Яндекс и Bing через их программные интерфейсы, т.е. использует механизм метапоиска с последующей фильтрацией дубликатов и нерелевантных ссылок [1].

БД СИР может также пополняться ссылками, обнаруженными при извлечении информации из обрабатываемых интернет-ресурсов. Эти ссылки в дальнейшем анализируются экспертами, которые принимают решение об их релевантности.

3.2. Методы извлечения информации

Для заполнения контента ИНИР собирается информация из таких источников, как порталы знаний, электронные библиотеки и журналы, сайты организаций, ассоциаций, проектов и конференций, новостные ленты, социальные научные сети, вики-ресурсы, реестры (каталоги) веб-сервисов и др. Как было сказано выше, из этих источников извлекается информация о проектах, организациях, персонах, конференциях и публикациях, т.е. обо всех объектах базовых классов онтологии научной деятельности, а также информация о самих источниках, которая представляется в контенте ИНИР в виде объектов классов онтологии научных информационных ресурсов.

Для каждого из этих классов создается свой метод извлечения информации, включающий набор шаблонов и связанных с ним обработчиков. Шаблоны генерируются на основе онтологии. Для повышения полноты извлечения информации вариативность этих шаблонов увеличивается за счет использования альтернативных терминов из тезауруса (синонимов и гипонимов). В шаблонах для каждого типа извлекаемой информации указываются обработчики, реализующие алгоритмы обхода и анализа соответствующих фрагментов интернет-страниц или документов.

Модуль извлечения информации осуществляет анализ интернет-ресурсов, которые он скачивает по ссылкам, заданным в БД СИР.

Для облегчения анализа HTML-страница ресурса представляется в виде DOM-дерева в соответствии со стандартом DOM (Document Object Model), регламентирующим способ представления содержимого документа (в частности, HTML-страницы) в виде набора объектов [9]. Анализ DOM-дерева каждой страницы выполняется на основе соответствующего шаблона, при этом определяется релевантность загруженной страницы тематике ИНИР и извлечение описанной этим шаблоном информации.

Например, интернет-ресурс, на котором размещена информация о проекте, может быть представлен сайтом проекта, разделом сайта организации или персоны или публикацией, описывающей проект. Для каждого из этих способов представления на основе класса онтологии *Проект* строится свой шаблон.

Рассмотрим один из вариантов организации такого шаблона, но сначала дадим описание свойств класса *Проект* онтологии научной деятельности (см. Рис.2).

```
class Проект (Название: string; Аббревиатура: string; Описание: string;
      Дата начала: date; Дата окончания: date; Номер: string; URL: string;
      Стадия: Этап_проекта; Ключевые слова: set_of_string)
relation Проект_Включает < Проект, Проект >
relation Проект_Поддерживается < Проект, Организация >
relation Задача_Проекта < Проект, Задача >
relation Участник_Проекта < Проект, Персона > (Роль: set_of_Роль)
relation Участник_Проекта_Орг < Проект, Организация >
relation Научное_направление < Проект, Раздел_науки >
relation Результат_Деятельности < Проект, Результат >
relation Исследует_Объект < Проект, Объект_исследования >
relation Использует_Метод < Проект, Метод_исследования >
relation Публикация_о_Проекте < Проект, Публикация >
relation Интернет_Ресурс_Проекта <Проект,Интернет_Ресурс>
```

Рис.2. Описание класса *Проек*т

На рис.3 представлен шаблон для извлечения информации на основе класса *Проект*.

```
<Class Name= "Проект" engine = " FragmentSearch " >
  <Marker Term = "О проекте" PType= "Menu/Head" FragType= "Page/Block " />
  <Marker Term = "Проект" PType= " Head " FragType= " Block " />
          <Attr Name= "Название" type= "string" engine = "NameEntity" >
             <Marker Term = "Проект" Ptype = "link" FragType= "LinkText" />
             <Marker Term = "Проект" Ptype = "sentence" FragType= "QuoteText" />
             <Marker Term = "Проект" Ptype = "Head" FragType = "Head" />
          </Attr>
          <Attr Name = "Аннотация" type= "text" >
             <Marker Term = "Аннотация/Содержание проекта/Описание
             проекта/ О проекте " PType = "Head" FragType= "Block/Page" />
           </Attr>
<Relation Name = "Публикация_о_Проекте" >
  <Marker Term = "Публикации" PType= "Menu/Head" FragType="Page/Block" />
  <Marker Term = "Список публикаций" PType="Menu" FragType="Page" />
  <Marker Term = "Литература" PType= "Menu/Head" FragType="Page/Block" />
  <Marker Term = "Библиография" PType= "Menu/Head" FragType= "Page/Block" />
<Object Name = "Публикация"engine ="PublicationsList" />
</Relation>
<Relation Name = "Участник проекта" >
  <Marker Term = "Об участниках" PType= "Menu" FragType="Page" />
  <Marker Term = "Список участников" PType= "Head" FragType="Block"/>
  <Marker Term = "Исполнители" PType= "Head" FragType="Block"/>
  <Marker Term = "Участники" PType= "Head" FragType="Block" />
<Object Name = "Персона" engine = "PersonList" />
</Relation>
</Class>
```

Рис.3. Шаблон класса *Проект*

Шаблон, предназначенный для извлечения объектов заданного класса, описывается блоком **Class** и содержит блоки атрибутов (**Attr**), отношений (**Relation**) и аргументов отношений (**Object**). Каждый из этих блоков может описываться одним или группой альтернативных маркеров (**Marker**), задающих свойства фрагмента текста, содержащего извлекаемую информацию. Маркер, приписанный непосредственно блоку **Class**, выделяет текстовый фрагмент, описывающий объект и определяющий область дальнейшего поиска маркеров.

К параметрам маркера относятся: (1) Term – термин тезауруса, представленный множеством альтернативных написаний термина, (2) PType – тип фрагмента, в тексте которого должен располагаться термин маркера, (3) FragType – тип фрагмента, который должен извлекаться, (4) engine – имя обработчика, который будет извлекать требуемую информацию в найденном по маркеру фрагменте.

Анализ входной страницы, представленной после предварительной обработки в структурированном виде (DOM-структура), осуществляется обработчиком верхнего уровня, который решает следующие задачи:

- поиск шаблона, подходящего для данной страницы или ее фрагмента, на основе маркеров блока **Class**;
- поиск маркерных терминов и извлечение текстовых фрагментов в соответствии с параметрами маркера;
- вызов специализированных обработчиков, формирование входных данных и обработка результата их работы;
- формирование объекта заданного онтологического класса и его связей.

Рис.4. Сайт проекта «Национальный корпус русского языка»

Например, на сайте проекта «Национальный корпус русского языка» (http://www.ruscorpora.ru) в разделе меню «о проекте» можно найти краткое описание

проекта, в разделе «участники проекта» – информацию о персонах и организациях, участвующих в проекте, в разделе «публикации» – информацию о публикациях по теме проекта и т.д. (Рис.4)

Шаблон, построенный на основе класса *Проект*, позволит извлечь эту информацию со страниц данного сайта. При этом для извлечения информации, составляющей контекст проекта и, как правило, определяемой отношениями класса *Проект*, например, данных о публикациях по теме проекта, персонах и организациях, участвующих в проекте, используются обработчики и шаблоны, специально построенные для извлечения информации такого типа и многократно используемые в других шаблонах, соответствующих таким базовым понятиям онтологии, как *Публикация*, *Персона*, *Организация* и др.

5. Заключение

Тематический интеллектуальный научный интернет-ресурс позволяет исследователям значительно сократить время, требуемое для обеспечения доступа к необходимой информации и ее анализа, за счет аккумуляции в своем контенте описаний релевантных интернет-ресурсов и методов их обработки. При этом использование ИНИР будет тем эффективнее, чем более полно в нем будет представлена информация по его тематике. Добиться такой полноты можно только за счет автоматизации сбора информации из сети Интернет.

В настоящее время реализован ряд компонентов подсистемы сбора информации из сети Интернет, а именно: модуль поиска релевантных интернет-ресурсов, модуль извлечения информации, база данных ссылок на интернет-ресурсы. На данный момент разработаны методы извлечения информации о проектах, организациях и событиях, включая сопутствующие шаблоны и обработчики, реализующие извлечение информации о персонах и публикациях. Заметим, что для анализа списков публикаций и персон используются ранее разработанные нами средства генерации формальных описаний научных статей [3].

Список литературы

1. Ахмадеева И.Р., Загорулько Ю.А., Саломатина Н.В., Серый А.С., Сидорова Е.А., Шестаков В.К. Подход к формированию тематических коллекций текстов на основе интернет-ресурсов // Вестник НГУ. Серия: Информационные технологии. 2013. Том.11, выпуск 4. С. 5-15.
2. Загорулько Ю.А. Автоматизация сбора онтологической информации об интернет-ресурсах для портала научных знаний // Известия Томского политехнического университета. Т. 312. № 5. Управление, вычислительная техника и информатика. 2008. С. 114–119.

3. Загорулько Ю.А., Дяченко О.О. Автоматическое наполнение информационных систем библиографическими сведениями о научных публикациях // Труды XIII Всероссийской научной конференции RCDL'2011 «Электронные библиотеки: перспективные методы и технологии, электронные коллекции». Воронеж, 19-22 октября 2011 г. Воронеж: Издательско-полиграфический центр Воронежского государственного университета, 2011 С.347-353.
4. Загорулько Ю.А., Загорулько Г. Б., Шестаков В.К., Кононенко И.С. Концепция и архитектура тематического интеллектуального научного интернет-ресурса // Труды XV Всероссийской научной конференции RCDL'2013. 14-17 октября 2013 г. Ярославль: ЯрГУ, 2013. С.57–62.
5. Ланин В.В., Мальцев П.А., Лядова Л.Н. Технологии сбора и анализа информации для исследовательского портала // Материалы Четвертой международной научно-технической конференции «Инфокоммуникационные технологии в науке, производстве и образовании» (Инфоком 4): Часть I, 2010. С. 218–222.
6. DeRose P., Shen W., Chen F., Doan AH, Ramakrishnan R. Building Structured Web Community Portals: A Top-Down, Compositional, and Incremental Approach // VLDB '07, September 23-28, 2007, Vienna, Austria. P. 399-410.
7. Ferrara E., De Meo P., Fiumara G., Baumgartner R.. Web Data Extraction, Applications and Techniques: A Survey // Preprint submitted to Knowledge-based systems. June 5, 2014. 41p.
8. Hillmann D. Using Dublin Core, 2005. [Электронный ресурс]. URL: http://dublincore.org/documents/usageguide/ (дата обращения: 17.11.2014).
9. Stenback J., Le Hégaret P., Le Hors A. Document Object Model (DOM) Level 2 HTML Specification // W3C Recommendation, 2003. [Электронный ресурс]. URL: http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/ (дата обращения: 17.11.2014).
10. Zhai Y., Liu B. Extracting Web Data Using Instance-Based Learning // Proceedings of 6th International Conference on Web Information Systems Engineering (WISE-05), 2005. P. 318–331.

UDK 002.53:004.89

An ontological information collection for intelligent scientific internet resources

Yury A. Zagorulko (A.P. Ershov Institute of Informatics Systems),

Olesya I. Borovikova (A.P. Ershov Institute of Informatics Systems),

Elena A. Sidorova (A.P. Ershov Institute of Informatics Systems),

Irina R. Ahmadeeva (Novosibirsk State University)

The paper considers the problems of information collection for thematic intelligent scientific internet resources providing the systematization and integration of scientific knowledge, information resources, related to certain area of knowledge, and methods of intelligent processing of data contained in them as well as the content-based access to them. The approach to automatization of information collection combining metasearch and knowledge extraction methods based on using ontology and thesaurus of the modeled area of knowledge is proposed.

Keywords: *Scientific Internet resources, metasearch, information extraction, ontology, thesaurus.*

УДК 004.43

Списки и строки в предикатном программировании

Шелехов В.И. (Институт систем информатики СО РАН, Новосибирский государственный университет)

Определяются языковые средства и методы реализации строковых объектов. Строки реализуются как массивы литер в стиле языков C и C++. Для строковых объектов доступен весь аппарат работы со списками. В связи с этим в данной работе пересматривается язык списков. В работе представлена часть библиотеки для строковых объектов.

Ключевые слова: *Функциональное программирование, предикатное программирование, автоматное программирование, список, строки в языке C.*

1. Введение

Предикатная программа состоит из набора программ на языке P [2] (определений предикатов) следующего вида:

```
<имя программы>(<описания аргументов>: <описания результатов>)
    pre <предусловие>
    { <оператор> }
    post <постусловие>
```

В предикатном программировании запрещены такие языковые конструкции, как циклы и указатели, серьезно усложняющие программу. Вместо циклов используются рекурсивные функции, а вместо массивов и указателей – списки и деревья. Предикатная программа проще в сравнении с императивной программой, реализующей тот же алгоритм.

Эффективность предикатных программ достигается применением при трансляции следующих оптимизирующих трансформаций:

- замена хвостовой рекурсии циклом;
- подстановка тела программы на место ее вызова;
- склеивание переменных: замена всех вхождений одной переменной на другую переменную;
- кодирование алгебраических типов (списков, деревьев, …) с помощью массивов и указателей.

В результате получается программа на императивном расширении языка P, которая далее конвертируется на язык C++. Отметим, что язык императивного расширения недоступен пользователю.

В данной работе трансформации операций со списками и строками рассматриваются лишь частично. Цель работы – определить в рамках языка P язык списков и строк, обеспечивающий надежность и эффективность программы. Тем не менее, особенности кодирования списков и строк в императивном расширении языка P в существенной степени предопределяются предлагаемыми языковыми возможностями.

В языках программирования наблюдается пестрая картина в реализации строковых объектов. Во многих функциональных языках, например в Haskell, строковый тип string определен как список литер; в коде строковый объект реализуется односвязным списком. В большинстве императивных языков тип string не определен – возможности работы со строковыми объектами обычно представлены в виде библиотеки. В языках C и C++ строковые объекты представляются указателями на массивы литер, завершающиеся нулем. Кроме того, имеются библиотеки на языке C++ с другим способом представления строк.

Строковый тип string – стандартный тип в языке предикатного программирования P [2], определяемый как list(**char**). Применимы все операции, определенные для списков. Строковые объекты представляются как в языках C и C++. Нуль как ограничитель значения строкового типа не входит в само значение, однако используется в алгоритмах со строками. Фактически вместо проверки на nil реализуется проверка на нуль для очередной литеры. По этой причине библиотеки для списков неприменимы для строковых объектов.

В языке P основным представлением списка является массив. С учетом того, что все действия по заказу и освобождению памяти под списки полностью скрыты от программиста, эффективная реализация операций со списками является проблематичной. Для достижения эффективности язык списков расширен дополнительными возможностями. В частности, допускается задание максимального числа элементов списка.

Описание нового языка списков и методов его реализации дано во втором разделе настоящей работы. Операции с памятью для списков и другие особенности описываются во третьем разделе. Четвертый раздел определяет язык строк на базе определенного ранее аппарата списков. Описываются особенности, определяемые способом кодирования строк с использованием завершающего нуля. В пятом разделе представлена часть программ из библиотеки для строковых объектов. Эта библиотека, в отличие от библиотеки на C++, не является классом. В соответствии с объектно-ориентированной технологией, класс скрывает

от пользователя детали реализации, в частности механизм отведения и освобождения памяти для строковых объектов. В предикатном программировании предполагается другой стиль: программист полностью контролирует эффективность своей программы. В шестом разделе приведен пример программы по обработке текстов.

Работа выполнена при поддержке РФФИ, грант № 12-01-00686.

2. Списки

Тип «список» – встроенный алгебраический тип со следующим определением:

```
type list (type T) = union (
    nil,
    cons(T car, list(T) cdr)
);
```

Здесь T – тип элемента списка, nil и cons – конструкторы. Канонический способ работы со списками определяется оператором выбора:

```
switch (s) {
    case nil: <оператор1>
    case cons(c, y): <оператор2>
};
```

Вхождения переменных c и y являются определяющими, причем c – начальный элемент списка s, а y – «хвост» списка s. Оператор выбора эквивалентен следующему оператору:

```
if (nil?(s)) <оператор1> else {{ T c = s.car || list(T) y = s.cdr}; <оператор2>};
```

Вместо распознавателя nil?(s), истинного при пустом списке s, привычнее использовать условие s == nil. Отметим, однако, что отношение равенства не определено для двух произвольных списков.

Определим набор языковых конструкций для работы со списками.

<выражение типа список> ::=
    <терм типа список> | <конкатенация списков> |
    <список-агрегат> | <конструктор списка> | <терм типа элемента списка>

Список-агрегат – это агрегат, определяющий список перечислением его элементов. Например: [[a, b, c]]. Терм типа элемента списка в позиции выражения типа список интерпретируется как одноэлементный агрегат.

<терм типа список> ::=
    <переменная типа список> | <терм типа список> . cdr |
    prec (<выражение типа список>) | <вырезка списка>

Пусть s – список. Тогда s.cdr определяет хвост списка, т.е. список без начального элемента; prec(s) – список без последнего элемента.

<конкатенация списков> ::=
    <выражение типа список> + <терм типа список> |
    <выражение типа список> + <терм типа элемента списка>
<вырезка списка> ::=
    <терм типа список> [<выражение> .. <выражение>] |
    <терм типа список> [<выражение> ..]

Значением вырезки вида s[m..n] является список, начинающийся элементом с номером m и заканчивающийся элементом с номером n. Элементы списка нумеруются с нуля. Значением вырезки будет пустой список (значение nil), если m больше номера последнего элемента, либо если n < m. Значением вырезки вида s[m..] является список, начинающийся элементом с номером m и включающий все элементы до конца списка.

<конструктор списка> ::=
    nil |
    cons (<терм типа элемента списка> , <выражение типа список>) |
    <специальный конструктор списка>

Здесь nil и cons(c, y), где c – элемент списка и y – список, являются стандартными конструкторами в соответствии с определением типа list. Специальный конструктор списка определен в следующем разделе.

<терм типа элемента списка> ::=
    <переменная типа элемента списка> | <терм типа список> . car |
    last (<выражение типа список>) | <терм типа список> [<индекс>]

Пусть s – список. Тогда s.car – начальный элемент списка; last(s) – список без последнего элемента; s[m] – элемент списка s с номером m. Напомним, что элементы в списке нумеруются с нуля.

Имеются другие конструкции со списками. Пусть s – выражение типа список. Тогда len(s) – длина списка s, т.е. число элементов в списке; nil?(s) – распознаватель, истинный для пустого списка s; cons?(s) – распознаватель, истинный для непустого списка s. Вместо данных распознавателей могут использоваться отношения s == nil и s != nil.

Отметим, что все конструкции, за исключением вырезки вида s[m..], в точности соответствуют описанию языка P [2]. Новые по отношению к [2] конструкции определены в следующем разделе.

3. Реализация списков

Основным способом представлением списка является массив. В качестве возможных альтернатив рассматриваются другие способы в виде: односвязного списка,

двунаправленного списка, кольцевого списка. Представление в виде кольцевого буфера следует считать модификацией представления в виде массива.

Далее рассматривается лишь представление списка в виде массива.

В соответствии с формальной семантикой языка P вычисление нового значения списка как результата некоторой операции сопровождается выделением памяти для этого значения. Буквальная реализация этого положения оказалась бы весьма расточительной. Например, при исполнении оператора s = x + y + z предполагается отведение памяти для результатов выражений x + y и (x + y) + z, а также для нового значения переменной s. Если заранее подсчитать длину списка x + y + z и отвести достаточную память для переменной s, то исходный оператор заменяется последовательностью "s = x; s = s + y; s = s + z", исполнение которой не требует дополнительной памяти.

Для любых видов строковых выражений, определенных в предыдущем разделе, возможен такой способ реализации, при котором удается отложить отведения памяти до момента присваивания списковой переменной, находящейся в левой части оператора присваивания. Поэтому отведение памяти далее рассматривается по отношению к списковым переменным.

Оптимальной реализацией является использование одного экземпляра памяти для всех присваиваний одной списковой переменной. Такое возможно, если известно верхнее ограничение L числа элементов списка.

Для изображения типа списка допускается использование следующих типовых термов:

list(T)

list(T, L)

Здесь T – тип элемента списка, L – максимальная длина списка. Размер памяти, отводимой для переменной типа list(T, L), будет достаточным для размещения L элементов списка.

Допустим, отведенная для списковой переменной память есть массив A с индексами в диапазоне 0..N. Тогда значение списковой переменной можно представить вырезкой A[m..n], где $0 \le m \le n \le N$, однако в случае пустого списка $m > n$. В большинстве случаев $m = 0$ и свободное место в памяти остается слева в диапазоне m+1..N. Однако бывают случаи, когда свободную часть памяти надо оставить справа для того, чтобы реализовать присваивание вида s = y + s, как, например, в работе [4], где используется присваивание buf = stf + buf. Для формирования нестандартного размещения значения списка используется специальный конструктор.

<специальный конструктор списка> ::=
    consLeft (<выражение типа список> , <индекс>) |
    consRight (<выражение типа список>) |
    consRight (<выражение типа список> , <максимальная длина>)

Конструктор вида consL(y, m), где m – номер элемента, формирует представление списка y в массиве, сдвинутое на m элементов относительно начала массива. Конструктор вида consRight(y) формирует представление списка y в массиве прижатым вправо. Конструктор вида consRight(y, L) формирует представление списка y в массиве длины L прижатым вправо. При исполнении оператора s = consRight(y, L) отводится новая память для строковой переменной s; если до присваивания переменная s уже имела некоторое значение, то транслятор с языка P должен обеспечить возврат старой памяти переменной s.

В языке P нет операторов отведения и освобождение памяти для списковых переменных. Вставка в код соответствующих действий реализуется транслятором. Отведение памяти реализуется при первом присваивании переменной. Размер памяти определяется по значению правой части оператора присваивания, если он явно не указан описанием типа.

Для значения списковой переменной не допускается выход значения за границы памяти, отведенной для переменной. Соответствующий контроль возлагается на программиста. Для этой цели предусмотрены следующие конструкции.

max_len(s)
store(s)
left_store(s)
resize(s, n)

Здесь s – переменная типа список. Значением функции max_len(s) является максимальное число элементов списка, которое можно разместить в массиве для переменной s. Функция store(s) определяет число элементов, которое можно разместить справа от значения s в свободной части памяти. Функция left_store(s) определяет число элементов, которое можно разместить слева от значения s в памяти. Оператор resize(s, n) отводит новую память размера n элементов, переписывает туда значение переменной s, освобождая старую память.

В дополнение к основному режиму, в котором программист полностью контролирует распределение памяти для списковых переменных, следует также предусмотреть режим, задаваемый прагмой, в котором контроль выхода за границы и заказ памяти большего размера реализуется автоматически.

Будем различать следующие виды присваиваний списковым переменным: создание (копирование), модификацию и сканирование. Данная классификация распространяется

также на аргументы вызова типа список, поскольку они рассматриваются как операторы присваивания соответствующим формальным параметрам.

Далее будем использовать следующие обозначения: s и y – переменные типа список, e и d – выражения типа список, m и n – переменные типа **nat**.

Присваивание вида копирования для оператора s = e реализуется копированием списка, вычисленного выражением e, в память для значения переменной s. Для оператора s = e + d в массив для хранения переменной s копируется значение e и вслед за ним копируется значение d.

Присваивание вида модификации[1] реализует изменение значения, размещаемого памяти для переменной s. Итоговое значение помещается в тот же участок памяти. Оператор s = s + e копирует список (значение выражения e) вслед за значением списка s в памяти. Оператор s = e + s копирует список (значение выражения e) перед значением списка s в памяти. Оператор s = s.car реализует отсечение хвоста списка. Операторы s = s[m..n] и s = s[m..] реализуют сужение значения списка к вырезке исходного значения s. Вместо сдвига значения s в начало массива проводится соответствующая корректировка позиции значения списка внутри памяти для списковой переменной. Операторы s = s.cdr и s = prec(s) также реализуются коррекцией позиции в памяти.

Списковая переменная, все действия с которой реализуют лишь анализ списка с продвижением по нему без его модификации[2] в памяти, называется *переменной сканирования*. Для переменной сканирования s присваивание вида s = y реализуется не копированием значения y, а создание *объекта сканирования*, ассоциированного с переменной y, и присваиванию его переменой s. Операторами сканирования являются s = s.cdr, s = prec(s), s = s[m..n] и s = s[m..]. Их отличие от соответствующих операторов присваивания в режиме модификации в том, что они не модифицируют переменной y. Операторы сканирования являются аналогами итераторов в императивных языках.

4. Строковый тип

Строковый тип string является предопределенным в языке предикатного программирования P [2]. Его определение имеет вид:

```
type string = list(char);
```

[1] Модификация переменных возможна как в исходной предикатной программе, так и в результате склеивания переменных после проведения трансформации склеивания переменных.

[2] Точнее, допускается модификация отдельных элементов списка без изменения их состава.

Набор средств, определенных в разделах 3 и 4 для списков, применим также для строк с некоторыми ограничениями. В дополнении к этому в языке P определены строковые константы.

Основным представлением строкового объекта является массив литер, завершающийся нулем, причем нуль не входит в значение строки. Иначе говоря, для строкового типа фактически действует следующее определение:

```
type string = subtype(list(char) s: s != nil & last(s) == 0[3]);
```

Допускаются также другие представления строковых объектов; например, см. раздел 6. Однако в данном разделе рассматривается только основное представление.

Проверка строки s на пустоту реализуется оператором s.car == 0, а не s == nil. В принципе, возможна реализация, в которой конструктор nil кодируется значением из единственного нулевого элемента, однако это такое решение приведет к потере эффективности. В итоге, типы list и string несовместимы: со строковым объектом нельзя работать как со списком, в частности, нельзя подставлять строковый объект параметром типа list. Как следствие, библиотеки для списков неприменимы для строковых объектов.

Из-за специфики основного представления механизм реализации списков, описанный в разд. 3, лишь частично переносится на реализацию строковых объектов.

Следует сохранить возможность указания размера памяти для строковых объектов. В качестве альтернативного способа изображения строкового типа предлагается использовать string(L): память для значения типа string(L) вмещает ровно L литер. Полезны также конструкции, введенные для списков:

```
max_len(s)
store(s)
resize(s, n)
```

Исключается возможность сдвига вправо значения строки относительно начала памяти, т.е значение строки всегда размещается с начала памяти.

Целесообразно использовать два вида объектов сканирования: один – для сканирования с начала строки, второй – с конца. В первом случае объект сканирования может быть представлен указателем на начальный элемент строки, во втором – дополнительно требуется длина строки, при этом строка, представленная объектом сканирования, нулем не завершается.

---

[3] В операции сравнения предполагается неявное приведение last(s) к типу **int**.

Вводятся дополнительные конструкции для строковых объектов. Для определения числа элементов строки s вместо функции len(s) используется length(s). Конструктор nil, распознаватель nil?(s), а также отношения s == nil и s != nil не используются для строковых объектов. В качестве пустой строки используется конструктор empty, значением которого является строка из единственного нулевого элемента.

5. Некоторые программы из библиотеки для строковых объектов

Для объектов типа string не определено отношение равенства «==». Для лексикографического сравнения строк используется функция Compare с результатом 0 при совпадении строк, -1 – если первая строка меньше второй и 1 – если первая строка больше второй.

```
Compare(string s, t: int ) {
    char c = s.car, d = t.car;
    if (c == d) {
       if (c == 0) return 0 else return Compare(s.cdr, t.cdr)
    } else return c - d
}
```

Программа SubString заменяет строку s ее вырезкой длины n начиная с элемента по номеру p, т.е. вырезкой s[p..p + n]. При этом итоговая вырезка не может выходить за границу исходной строки s.

```
SubString(string s, nat p, n: string s'){
    if (p >= length(s)) s' = empty
    else s' = s[p..min(p + n, length(s) - 1)];
}
```

При p≠0 итоговое значение s′ получается сдвигом исходного значения s в памяти. Не следует использовать данную программу в случае, когда значение переменной s′ далее не модифицируется. Предпочтительней использовать операции вырезки, которая будет реализована через объект сканирования.

Программа Insert вставляет строку t внутрь строки s начиная с позиции p.

```
Insert(string s, t, nat p: string s'){
    if (p > length(s) or t == empty) return;
    nat L = length(s) + length(t);
    if (L > max_len(s)) resize(s, L);
    if (p = 0) { s' = t + s; return };
    s' = s[0..p-1] + t + s[p..]
}
```

Эффективная реализация оператора s′ = s[0..p-1] + t + s[p..] в императивном расширении обеспечивается парой вызовов программ из внутренней библиотеки:

```
right(s, p, length(s), p + length(t));
copy(s, t, p)
```

Вызов right сдвигает вырезку s[p..] вправо (вместе с завершающим нулем) на length(t) позиций. Вызов copy переписывает значение строки t в строку s начиная с позиции p.

Программа Replace заменяет часть строки, соответствующей вырезке s[p..p + n], на строку t. Если p + n выходит за границы строки s, заменяемая вырезка ограничена концом строки.

```
Replace(string s, t, nat p, n: string s′){
    if (p > length(s)) return;
    if (p = length(s)) { s′ = s + t; return };
    nat m = (p + n >= length(s))? length(s) – 1 : p + n;
    nat L = length(s) + length(t) – m + p – 1;
    if (L > max_len(s)) resize(s, L);
    s′ = s[0..p-1] + t + s[m+1..]
}
```

Функция StartsWith проверяет, является ли строка t начальной частью строки s.

```
StartsWith(string s, t: bool ) {
    if (t.car == 0) return true;
    if (s.car != t.car) return false;
    return StartsWith(s.cdr, t.cdr)
} post ∃ string u. s = t + u;
```

Функция EndsWith проверяет, является ли строка t конечной частью строки s.

```
EndsWith(string s, t: bool ) {
    return EndsWith(string s, t, length(s), length(t));
} post ∃ string u. s = u + t;

EndsWith(string s, t, nat js, jt: bool ) {
    if (s[js] != t[jt]) return false;
    if (jt == 0) return true;
    if (js == 0) return false;
    return StartsWith(s, t, js - 1, jt - 1)
};
```

Гиперфункция Index определяет первую позицию элемента c в строке s начиная с позиции p. Результат – позиция q. При отсутствии элемента c реализуется вторая ветвь #noel.

```
Index(string s, char c, nat p: nat q #yes : #noel){
    if (p >= length(s)) #noel;
    Index1(s, c, p: q #yes : #noel)
}

Index1(string s, char c, nat q: nat q' #yes: #noel){
    if (s[q] == c) {q' = q; #yes };
    if (s[q] == 0) #noel;
    Index1(s, c, q+1: q' #yes : #noel)
}
```

Гиперфункция RIndex определяет последнюю позицию элемента c в строке s не превышающую позиции p. Результат – позиция q. При отсутствии элемента c реализуется вторая ветвь гиперфункции #noel.

```
RIndex(string s, char c, nat p: nat q #yes: #noel){
    if (p>= length(s)) p = length(s) – 1;
    RIndex1(s, c, p: q #yes: #noel)
}

RIndex1(string s, char c, nat q: nat q' #yes : #noel){
    if (s[q] == c) {q' = q; #yes };
    if (q == 0) #noel;
    RIndex1(s, c, q – 1: q' #yes : #noel)
}
```

Гиперфункция Index определяет позицию первого вхождения строки t в строке s начиная с позиции p. Результат – позиция q. При отсутствии вхождений, а также в случае пустой строки t реализуется вторая ветвь гиперфункции #nostr.

```
Index(string s, t, nat p: nat q #yes: #nostr){
    if (t == empty) #nostr;
    Index1(s, t, p, length(s) – length(t) : q #yes : #nostr)
}
Index1(string s, t, nat p, Lts: nat q #yes : #nostr){
    if (p > Lts) #nostr;
    Index2(s, t, p, 0 : q #yes : )
    Index1(s, t, p+1, Lts: q #yes : #nostr)
}

Index2(string s, t, nat p, j: nat q #yes : #no){
    if (t[j] == 0) {q = p; #yes};
    if (t[j] != s[p+j]) #no;
    Index2(s, t, p, j+1: q #yes : #no)
}
```

После трансформаций замены рекурсии циклом и подстановки тел программ на место вызовов программа Index приводится к следующему виду:

```
Index(string s, t, nat p: nat q #yes: #nostr){
    if (t == empty) #nostr;
    nat Lts = length(s) – length(t);
    for (; ; p = p+1) {
        if (p > Lts) #nostr;
        for (j=0; ; j = j+1) {
            if (t[j] == 0) {q = p; #yes};
            if (t[j] != s[p+j]) break;
        }
    }
}
```

Гиперфункция RIndex определяет позицию последнего вхождения строки t в строке s не превышающую позиции p. Результат – позиция q. При отсутствии вхождений, а также в случае пустой строки t реализуется вторая ветвь гиперфункции #nostr.

```
RIndex(string s, t, nat p: nat q #yes : #nostr){
    if (t == empty) #nostr;
    if (p>= length(s)) p = length(s) – 1;
    RIndex1(s, t, p, length(t) – 1: q #yes : #nostr)
}

RIndex1(string s, t, nat p, Lt1: nat q #yes : #nostr){
    if (Lt1 > p) #nostr;
    RIndex2(s, t, p, Lt1 : q #yes : )
    RIndex1(s, t, p – 1, Lt1: q #yes: #nostr)
}

RIndex2(string s, t, nat q, j: nat q' #yes : #no){
    if (t[j] != s[q]) #no;
    if (j == 0) {q' = q; #yes};
    RIndex2(s, t, q – 1, j – 1: q' #yes : #no)
}
```

Гиперфункция FirstOf определяет позицию первого вхождения любого элемента строки t в строке s начиная с позиции p. Результат – позиция q. При отсутствии вхождений, а также в случае пустой строки t реализуется вторая ветвь гиперфункции #noel.

```
FirstOf(string s, t, nat p: nat q #yes: #noel) {
    if (t == empty) #noel;
    FirstOf1(s, t, p, length(s) : q #yes : #noel)
}
```

```
FirstOf1(string s, t, nat p, Ls: nat q #yes : #nostr){
    if (p >= Ls) #nostr;
    FirstOf2(s, t, p, 0 : q #yes : )
    FirstOf1(s, t, p+1, Ls: q #yes : #nostr)
}

FirstOf2(string s, t, nat p, j: nat q #yes : #no){
    if (t[j] = s[p]) {q = p; #yes};
    if (t[j] == 0) #no;
    FirstOf2(s, t, p, j+1: q #yes : #no)
}
```

6. Программа выделения первого слова

В данном разделе рассматривается программа обработки текста, представленная на сайте [1] в качестве иллюстративного примера при описании методов автоматного программирования А.А. Шалыто [3].

Требуется написать программу, читающую из потока стандартного ввода текст, состоящий из строк, и для каждой строки печатающую первое слово этой строки и перевод строки. Слова разделяются пробелами. Знаки препинания отсутствуют. Пробелы могут находиться в начале строки. Конец строки определяется литерой '\n', конец текста – литерой EOF. Литера конца текста обязательно присутствует в тексте.

В работе [4] определены два класса программ: программы-функции и программы-процессы. Программа принадлежит классу *программ-функций*, если она не взаимодействует с внешним окружением. Точнее, если возможно перестроить программу таким образом, чтобы все операторы ввода данных находились в начале программы, а весь вывод собран в конце программы. В противном случае программа принадлежит классу программ-процессов и адекватно представима в виде автоматной программы [4]. Применение автоматного программирования для программ-функций противопоказано. Автоматные программы по своей структуре существенно сложнее программ-функций.

Представленная программа обработки текстов относится к классу программ-функций. Посимвольно вводимый входной текст можно предварительно собрать в строковой переменной in. Единственная особенность – в качестве конца строкового значения используется литера EOF. Выходной текст можно накапливать в строковой переменной out и распечатать по завершению программы.

Сначала воспроизведем эквивалентную предикатную программу для исходной программы на сайте [1].

```
char EOF; // литера конца текста
char SP = ' '; // литера «пробел»
FirstWords(string in : string out) { FirstWords1(in, nil : out); }
```

Программа FirstWords сводится к более общей программе FirstWords1. Второй параметр – начальное значение переменной out.

```
FirstWords1(string in, string out : string out') {
    skipSpaces(in.cdr, in.car : string in1, char c);
    getWord(in1, c, out : string in2, char c2, string out1);
    string out2 = out1 + '\n';
    skipLine(in2, c2 : string in3, char c3);
    if (c3 != EOF) FirstWords1(in3, out2 : out');
}
```

Программа skipSpaces реализует пропуск возможных пробелов в начале очередной строки. Ее результатами являются: остаток входного текста in1 и последний прочитанный символ c, отличный от пробела. Использование переменной c необходимо во избежание повторного чтения литер из входного текста. Программа getWord выделяет начальное слово очередной строки и записывает его в переменную out. Программа skipLine сканирует остаток строки до ее конца. Ниже приведем более привычную для императивных программистов эквивалентную версию программу FirstWords1 с использованием модифицируемых переменных.

```
FirstWords1(string in, string out* : ) {
    skipSpaces(in.cdr, in.car : in, char c);
    getWord(in, c, out : in, c, out);
    string out = out + '\n';
    skipLine(in, c : in, c);
    if (c != EOF) FirstWords1(in, out : out);
}
```

Ниже представлены программы skipSpaces, getWord и skipLine.

```
skipSpaces(string in*, char c* :)
{   if (c == SP) skipSpaces(in.cdr, in.car : in, c); }

getWord(string in*, char c*, string out* :)
{    if (c != SP & c != '\n' & c != EOF)
        getWord(in.cdr, in.car, out + c : n1, c, out);
}

skipLine(string in*, char c* : )
{  if (c != '\n' & c != EOF) skipLine(in.cdr, in.car : in, c); }
```

Для приведенной предикатной программы наряду со штатными трансформациями применяются следующий набор определяемых пользователем трансформаций:

```
/*#  c = in.car; in = in.cdr → input(c)
   forall char x. out = out + x → print(x)
   out = nil →
#*/
```

Данные трансформации заменяют операции со строками in и out операторами input(c), print(c) и print('\n'). В результате трансформаций получим в точности исходную императивную программу на сайте [1]. Данная программа имеет очевидные дефекты в эффективности: проверка очередного символа на совпадение с пробелом и символами конца строки и конца текста реализуется дважды. Наша задача – показать, что в рамках предикатного программирования можно устранить эти и другие дефекты и провести предельную оптимизацию любого алгоритма. Разумеется, в реальном программировании подобные дефекты в большинстве случаев не являются критичными, однако имеются приложения, где любые потери эффективности недопустимы.

Ниже представлена эффективная программа обработки текста, использующая аппарат гиперфункций [4]. В ней изменены программы getWord и skipLine.

```
FirstWords1(string in, string out* : ) {
      skipSpaces(in.cdr, in.car : in, char c);
      getWord(in, c, out : in, out #sp: in, out #lf: in, out #e);
sp:   skipLine(in : in #lf: #e);
e:    out = out + '\n'; return
lf:   FirstWords1(in, out + '\n' : out);
 }
```

Программа getWord – гиперфункция с тремя ветвями, реализуемыми, соответственно, при достижении пробела, конца строки и конца текста. Поскольку текущий символ c после вызова skipSpaces отличен от пробела, проверка на пробел в getWord проводится для следующего символа.

```
getWord(string in*, char c, string out* : #sp:  #lf:  #e)
{   if (c == '\n')  #lf;
    if (c == EOF) #e;
    out = out + c; c = in.car; in =in.cdr;
    if (c == SP)  #sp;
    getWord(in, c, out : in, c, out);
}
```

```
skipLine(string in*: #lf: #e)
{    char c = in.car; in =in.cdr;
     if (c == '\n') #lf;
     if (c == EOF) #e;
     skipLine(in : in #lf: #e)
}
```

Покажем построение эффективной императивной программ методом трансформаций для новой версии предикатной программы. На первом этапе хвостовая рекурсия заменяется циклом.

```
FirstWords1(string in, string out* : ) {
for (;;) {
     skipSpaces(in.cdr, in.car : in, char c);
     getWord(in, c, out : in, out #sp: in, out #lf: in, out #e);
sp:   skipLine(in.cdr, in.car : in #lf: in #e);
e:    out = out + '\n'; return
lf:   out = out + '\n';
}
}

skipSpaces(string in*, char c* :)
{ while (c == SP) { c = in.car; in =in.cdr }}

getWord(string in*, char c, string out* : #sp:  #lf:  #e)
{  for (;;) {
     if (c == '\n')  #lf;
     if (c == EOF) #e;
     out = out + c; c = in.car; in =in.cdr;
     if (c == SP)  #sp;
}
}
skipLine(string in*,: #lf: #e)
{    for (;;) {
        char c = in.car; in =in.cdr;
        if (c == '\n') #lf;
        if (c == EOF) #e;
     }
}
```

На втором этапе трансформаций реализуется постановка тел программ skipSpaces, getWord и skipLine на место их вызовов. Затем тело FirstWords подставляется на место вызова. Получим:

```
  FirstWords(string in : string out) {
out = nil;
for (;;) {
      char c;
      do { c = in.car; in =in.cdr } while (c == SP);
      for (;;) {
          if (c == '\n')  #lf;
          if (c == EOF) #e;
          out = out + c; c = in.car; in =in.cdr;
          if (c == SP)  #sp;
      }
 sp:  for (;;) {
          c = in.car; in =in.cdr;
          if (c == '\n') #lf;
          if (c == EOF) #e;
      }
 e:   out = out + '\n'; return
 lf:  out = out + '\n';
}
  }
```

На третьей стадии трансформаций реализуются замены:

$$c = in.car;\ in = in.cdr \rightarrow input(c)$$

$$out = out + x \rightarrow print(x)$$

В итоге получим окончательную программу:

```
  FirstWords() {
for (;;) {
      char c;
      do input(c) while (c == SP);
      for (;;) {
          if (c == '\n')  #lf;
          if (c == EOF) #e;
          print(c); input(c);
          if (c == SP)  #sp;
      }
 sp:  for (;;) {
          input(c);
          if (c == '\n') #lf;
          if (c == EOF) #e;
      }
 e:   print('\n'); return
 lf:  print('\n');
}
  }
```

Заключение

В данной работе проведена доработка языка предикатного программирования [2] для списков и строк. Новые языковые средства обеспечивают лучший контроль эффективности программы, что соответствует стилю парадигмы предикатного программирования. В частности, обеспечивается аккуратная работа с памятью без явных операторов отведения и освобождения памяти. Это, однако, потребует дополнительной нагрузки на транслятор и используемый в нем потоковый анализ.

Отметим, что в отличие от предыдущих проектов языка списков и строк, не фиксируется представление в коде списков и строк. Возможный способ кодирования приводится лишь для объектов сканирования строк. Однако очевидно, представления списков и строк существенно различаются

В пятом разделе значительная часть библиотеки для строковых объектов (файлы string.h и string.cpp) переписана на язык P с использованием новых возможностей, представленных в разделе 3. Стиль работы с памятью можно проследить на примере программ SubString, Insert и Replace. Если предикатный программист правильно оценивает размеры строковых объектов, то новой памяти не потребуется, тогда как соответствующие программы в string.cpp всегда отводят память с оглядкой на возможность наличия нескольких ссылок на исходную строку. Следует отметить, что программа, использующая string.cpp, будет довольно часто «трясти» (заказывать и освобождать) память. Такой стиль программирования неприемлем для встроенных систем.

Следует отметить, что библиотека (string.h и string.cpp) не штатная, а скорее специальная с хакерским стилем реализации. Штатной видимо является библиотека для языка C, недавно формально верифицированная в работе [5].

Список литературы

1. Автоматное_программирование. 2014 [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Автоматное_программирование (дата обращения: 10.12.2014).
2. Карнаухов Н.С., Першин Д.Ю., Шелехов В.И. Язык предикатного программирования P. Версия 0.12. Новосибирск, 2013. 28с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/plang12.pdf. (дата обращения: 10.12.2014).
3. Поликарпова Н.И., Шалыто А.А. Автоматное программирование / СПб.: Питер. 2009. 176с. [Электронный ресурс]. URL: http://is.ifmo.ru/books/_book.pdf (дата обращения: 10.12.2014).

4. Шелехов В.И. Язык и технология автоматного программирования // Программная инженерия, №4, 2014. С. 3-15. [Электронный ресурс].
URL: http://persons.iis.nsk.su/files/persons/pages/automatProg.pdf (дата обращения: 10.12.2014).
5. Carvalho, Nuno, et al. Formal Verification of kLIBC with the WP Frama-C Plug-in // NASA Formal Methods. Springer International Publishing, 2014. P. 343-358.