Математические вопросы кибернетики и вычислительной техники 19, 1998, 63-76.

Разработка средств поддержки документального формата UNIMARC в информационной системе Isite

В. Г. Саакян, М. Л. Сукиасян, Ю. Г. Шукурян

Институт проблем информатики и автоматизации НАН РА и ЕрГУ

Аннотация

В настоящей статье рассматривается структурная модель Isite – открытой информационной системы, поддерживающей множество механизмов внешнего протокольного доступа, обладающей независимым от типа баз данных прикладным программным интерфейсом (API) и расширяемым поисковым средством, ориентированным на работу с полями. Система Isite разработана в Центре обнаружения и извлечения информации в сетях (CNIDR), финансируемом National Science Foundation. С целью реализации возможности доступа к библиографическим базам данных проводится разработка нового документального типа UNIMARC в рамках данной системы.

Модели доступа к данным и информационная система Isite

Широкий спектр существующих на сегодня моделей сетевого доступа к данным можно классифицировать по двум основным группам: обзорно-ориентированные (браузинговые) и поисково-ориентированные.

Примерами обзорно-ориентированных систем (ООС) являются Gopher и World Wide Web. Поставщики информации, обладая соответствующими инструментальными средствами, могут предоставлять данные для доступа посредством этих систем. ООС основаны на концепции навигации в виртуальном информационом пространстве. Они хорошо приспособлены для извлечения данных в тех случаях, где есть явная взаимосвязь между данными. В таких системах важную роль играет наличие какого-либо принципа организации данных, поскольку успешность обзора зависит от способности пользователя принимать разумные навигационные решения. ООС оказались очень полезным средством локализации баз данных в распределенных структурах глобальной сети Internet, поскольку гипертекстовая модель породила некую рудиментарную организацию среди взаимосвязанных документов.

Информационные системы, основанные на модели поиска (например, по ключевым словам) очень желательно использовать тогда, когда публикуемые данные по тем или иным причинам трудно организовать. В таких случаях автоматизированный поиск по ключевым словам может быть намного экономичнее, чем модель обзора (браузинга), или по крайней мере может содействовать ООС на определенных стадиях.

На практике часто используется комбинация обеих моделей доступа. Например, многие системы поискового доступа на самом верхнем уровне интеракции снабжены обзорно-ориентированным пользовательским интерфейсом. Помимо этого, очень важным требованием современных поисковых систем является их совместимость с протокольными стандартами Internet, что позволяет легко интегрировать их с другими поисковыми и обзорно-ориентированными системами.

Подбор поискового средства, которое соответствует узким специфическим запросам пользователей, является довольно сложной задачей. Среди большого разнообразия поисковых программных пакетов в течение последних десяти лет достаточно широкое распространение получила поисковая модель WAIS (Wide Area Information Server), ставшая одной из ранних наиболее популярных поисковых систем в Internet. WAIS - информационная система, комбинировавшая в себе средства поиска и индексирования с интерфейсом извлечения информации, делала возможным осуществление унифицированного поиска информации в любых базах данных по Internet [2,3].

Однако, одним из недостатков наиболее широко известной реализации технологии WAIS – пакета freeWAIS было то, что не существовало разделения поискового средства и протокола извлечения информации. Кроме этого, процесс внесения изменений в исходный код этого пакета был крайне затруднен. Протоколом, использованным в freeWAIS была ранняя версия ANSI/NISO Z39.50 [5]. Со временем и протокол, лежащий в основе программы, и поисковое средство устарели.

В связи с этим с 1992 г. в центре CNIDR (Center for Networked Information Discovery and Retrieval) начались работы по поддержанию стандарта WAIS (сначала путем доработки пакета freeWAIS) и разработке новой технологии построения распределенных информационных систем. По ходу адаптации freeWAIS для новых условий и обновления исходного протокольного кода, в центре начались эксперименты с различными альтернативными алгоритмами поисковых средств, которые привели к созданию модели системы Isite [7]. Был полностью переписан весь протокольный блок Z39.50, а также отдельно от него реализовано поисковое средство на C++, после чего оба эти компонента были объединены в пакет, названный Isite. Модуль Isearch, ставший поисковым компонентом Isite, был разработан как строительный блок для построения новой информационной системы, а также адаптации к уже функционирующим серверам.

Архитектура системы Isite

Isite использует комбинированную модель представления данных, одновременно обеспечивая различные механизмы доступа через стандартные протоколы. Поисковый интерфейс Isite (SAPI) позволяет также одновременно обслуживать множество баз данных различных форматов. Основным преимуществом системы является то, что в ней используется обобщенная иерархическая модель документального типа, позволяющая описывать частные форматы документов (записей) базы данных с помощью определенных общих функций, необходимых для манипулирования каждым из них (например, функции разборки по записям, по полям и по ключевым словам, а также представления документов в различных форматах). Благодаря гибкому механизму обработки документальных типов вновь инсталлированные базы данных могут стать сразу же доступными через коммуникационные протоколы извлечения данных, поддерживаемые Isite.

В настоящее время Isite поддерживает четыре механизма доступа, основанные на стандартных протоколах Internet: Z39.50, World-Wide Web (HTTP), Electronic Mail и Gopher [4]. Протокол Z39.50, реализованный в Isite в соответствии с версией 2 стандарта ANSI/NISO Z39.50, является основным механизмом доступа системы. Остальные протоколы поддерживаются через программные протокольные шлюзы. Так, в пакете реализован HTTP/Z39.50 шлюз, конвертирующий поисковые запросы, получаемые HTTP сервером в соответствующие запросы протокола Z39.50. Общая архитектура информационной системы представлена на Рис. 1.

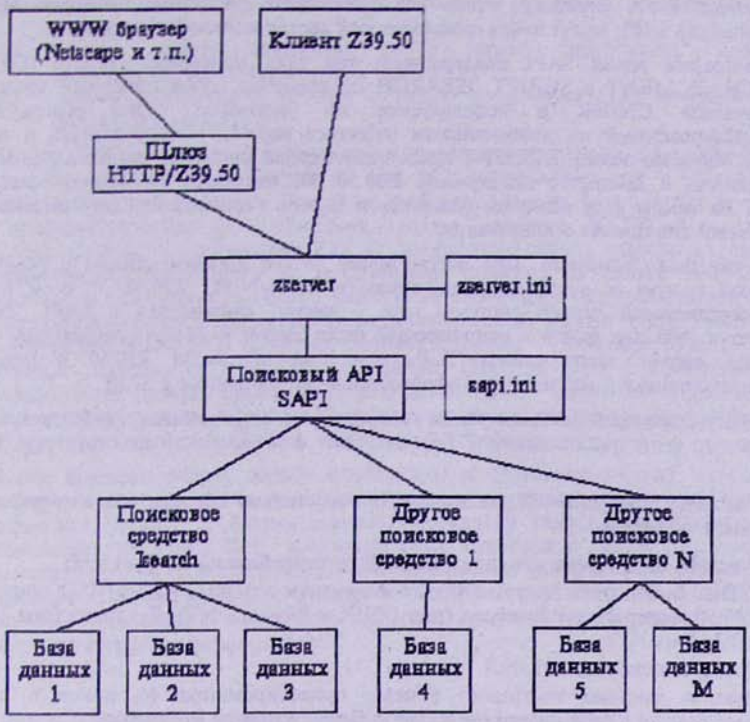

Рис. 1. Архитектура информационной системы Isite.

Многие прикладные программы требуют доступа к услугам текстового поиска и различным СУБД. В ответ на эту общую потребность в CNIDR был разработан поисковый API (SAPI), представляющий собой попытку обобщить доступ к произвольным СУБД через общий API. Поэтому любое приложение, связываемое с этим API, наследует функциональность СУБД, стоящей позади него. Поисковый интерфейс (SAPI) по сути представляет собой некий уровень абстракции между сервером Z39.50 Isite (в дальнейшем Z-сервер) и СУБД, содержащей обслуживаемые данные. Таким образом, для Z-сервера обеспечивается надежный коммуникационный подуровень для доступа к данным, хранимым в различных базах. Поддерживаемые типы СУБД могут варьироваться от сложных систем текстового поиска и реляционных СУБД до простых утилит, таких как "grep". Основной особенностью является то, что SAPI нормализует поведение СУБД для обеспечения интерфейса с Z-сервером. Благодаря этому промежуточному уровню все методы доступа, обеспечиваемые Isite, реализуют себя в базах данных, поддерживаемых SAPI.

С точки же зрения системного администратора функционирование SAPI требует ведения одного или более системных скриптовых файлов, содержащих информацию о том, какие базы данных доступны на текущий момент, их местоположение в файловой системе, тип поискового средства, используемого для извлечения

информации и т.п. Благодаря этому все прикладные программы, расчитанные на использование SAPI, могут иметь динамический доступ к этим базам данных.

В настоящее время SAPI поддерживает три типа поисковых средств (СУБД): ISEARCH, ZCLIENT и SCRIPT. ISEARCH представляет собой поисковое средство, разработанное CNIDR и используемое по умолчанию. Оно обеспечивает классифицированный по релевантности (relevance ranked), полнотекстный, а также булевый поиск по полям. ZCLIENT представляет собой механизм поиска в удаленных базах данных в Internet с поддержкой Z39.50. И, наконец, "поисковое средство" SCRIPT на самом деле является механизмом вызова скриптов или других внешних приложений для поиска и извлечения.

Как уже было отмечено, Isite обеспечивает доступ к базам данных с помощью протокола поиска и извлечения информации ANSI/NISO Z39.50. С этой целью коммуникационный сервер системы Isite - zserver, связывается с SAPI. Zserver включает в себя два файла - исполняемый файл zserver и конфигурационный файл zserver.ini. zserver воспринимает TCP-соединения протокола Z39.50 и передает запросы для поиска и извлечения информации в БД, связанные с SAPI.

Конфигурационный файл zserver.ini содержит всю информацию, необходимую для нормального функционирования Z-сервера. Этот файл имеет ту же структуру, что и sapi.ini.

Что касается программной реализации [3], то система Isite состоит из нескольких программных пакетов:

(1) libcnidr, библиотека исходного кода общеупотребительных функций;
(2) ZDist, библиотека программного обеспечения версии 2 ANSI/NISO протокола Z39.50, содержащая Z-сервер (под UNIX и Windows NT), Z-клиент (под UNIX и Windows NT);
(3) SAPI, поисковый API
(4) Isearch, система текстового поиска, ориентированная на поиск в полях, содержащая в себе интерфейс классов C++ и утилиты Iindex/Isearch.

Пакет Zdist представляет собой ядро коммуникационного доступа системы Isite. Доступ к поддерживаемым SAPI базам данных полностью контролируется через протокол Z39.50. Последняя версия Zdist поддерживает службы Initialize, Search, Present и Close протокола Z39.50.

Библиотека написана на языке C++ на основе общедоступных утилит BER, разработанных OCLC (Online Library Computer Center) [6]. Сервер и клиент/шлюз используют эту библиотеку для кодирования и декодирования протокольных блоков данных (PDU) Z39.50. Исходный код прикладного коммуникационного уровня реализован отдельно от уровня кодирования/декодирования и физически располагается в библиотеке libcnidr.

Серверское приложение Z39.50 легко конфигурируется путем изменения текстового файла настроек. Опции кофигурации содержат широкий набор возможностей, включающий в себя указание максимального числа одновременно поддерживаемых соединений, обслуживаемые БД, режим работы сервера и различные установки Z39.50 вплоть до уровня PDU. Рабочий цикл серверской части пакета состоит из следующих основных фаз:

(1) ожидание клиентского запроса на соединение на общеизвестный порт (как правило - 210)
(2) прием запроса на соединение и инициализация соединения с клиентом
(3) прием поисковых запросов

(4) взаимодействие с SAPI с целью обработки запроса и возврата результатов клиенту, после чего
(5) клиент может запросить, чтобы сервер "представил" содержание результатов, в результате чего
(6) сервер контактирует с SAPI с целью извлечения этих результатов и их пересылки клиенту

Клиент Z39.50 разработан в основном с целью использования в качестве базы для шлюзовых приложений. Поэтому в нем нет интерактивности и дружелюбного пользовательского интерфейса. Для того, чтобы реализовать шлюз между протоколом не ориентированным на соединение (каким является HTTP) и протоколом, ориентированным на соединение (Z39.50), необходимо скрыть требования, связанные с организацией соединения от вызывающей стороны. В пакете Isite Z-клиент реализован в виде однопроходного клиента реализующего инициализацию, поиск и представление в непрерывающейся последовательности. Клиент также приспособлен для взаимодействия с CGI (Common Gateway Interface) – общеизвестным стандартом шлюзовых соединений с HTTP серверами. Процесс CGI представляет собой промежуточный уровень между HTTP и шлюзом Z39.50. Также как и Z39.50 сервер, клиент конфигурируем, позволяя пользователю специфицировать всю информацию PDU уровня с целью обеспечения максимальной гибкости.

После краткого обзора общей архитектуры и функциональности системы Isite рассмотрим как реализовано взаимодействие средства поиска и извлечения информации (Isearch) с базами данных. Поскольку Isearch поддерживает любые определенные на языке C++ документальные форматы в своих БД, необходимо рассмотреть, что из себя представляет документальный тип в данном контексте и как происходит интеракция Isearch-DocumentType. На основе этих сведений возможно построение (описание) нового библиографического формата для реализации доступа к библиографическим базам данных.

Isearch и обобщенная модель документального типа

Isearch имеет модульную структуру, в основе которой лежат две группы классов C++:

- те, что относятся к поисковому средству Isearch. Эти классы инкапсулируют в себе функции индексирования по полям, поиска и представления записей.
- иерархия классов документальных типов, определяющая поведение поискового средства Isearch для различных типов документов.

Рассмотрим как осуществляется манипулирование различными документальными типами в Isearch.

Каждый из документальных типов (ДТ) имеет свои особенности, связанные с интерпретацией полей записей и представлением документов. Поэтому в Isearch каждый ДТ реализован в виде отдельного класса, определяющего частную структуру полей и характеристики представления данного типа. Базовый класс DOCTYPE определяет поведение по умолчанию и вызывается при обработке документов, для которых не определен ДТ. Все ДТ должны наследовать в себе класс DOCTYPE, поэтому сначала было бы целесообразным рассмотреть реализацию базового класса и его взаимодействие с Isearch.

В классе DOCTYPE имеется 4 основных метода: AddFieldDefs(), ParseFields(), ParseRecords() и Present(). Первые три из них вызываются во время индексирования, четвертый - во время поиска.

Метод DOCTYPE::AddFieldDefs() передает средству Isearch информацию о том, какие поля следует ожидать в фазе разборки записи.

DOCTYPE::ParseFields() вызывается во время индексирования каждой записи документа и, разбирая ее, вставляет информацию о структуре полей в объект RECORD.

DOCTYPE::ParseRecords() определяет структуру записей в файлах, в которых содержится множество записей данного документального типа. Эта функция полезна в тех случаях, когда структура записи зависит от других аспектов структуры документа и может быть определена только во время работы программы, как например, в случае с MARC записями, которые имеют переменную длину.

Наконец, метод DOCTYPE::Present() определяет то, каким образом выводятся документы в ответ на запросы о различных наборах элементов и синтаксисах представления, что позволяет формировать представление записи после извлечения содержимого ее полей.

Средство Isearch вызывает каждый из этих методов в соответствующие моменты при индексировании и в процессе поиска. В классе DOCTYPE эти методы представлены в минимальной функциональной конфигурации, что позволяет перекрывать их в классах-наследниках. Например, DOCTYPE::AddFieldDefs() и DOCTYPE::ParseFields() не содержат исходного кода и, следовательно, по умолчанию функции индексирования рассматривают документы как не имеющие полей. Isearch заранее не известно структуры полей записи, однако исходный код для поиска по полям присутствует в определениях документальных типов и может быть легко задействован путем определения структуры полей в данном ДТ.

Процесс ***построения определений полей*** и структурных таблиц несколько усложнен вследствие наличия различных уровней вложенности и инкапсуляции данных.

Экземпляр класса DFDT (Data Field Definitions Table - таблица определений полей данных) хранится в поисковом средстве Isearch, причем методы DOCTYPE могут добавлять новые определения к этой таблице. Таблица DFDT является списком объектов DFD (определение поля данных). Каждый DFD, в свою очередь, состоит из имени поля и списка атрибутов (класс ATTRLIST), содержащего объекты типа ATTR, которые можно использовать для хранения дополнительной информации о каждом определении поля. Информация об определении какого-либо поля может быть добавлена во внутренний экземпляр DFDT поискового средства в любом месте в методах DOCTYPE, за исключением случая когда производится попытка генерации информации о структуре поля для записей, ссылающихся на поля, которые еще не были определены. Есть две веские причины, по которым стоит генерировать объекты DFD внутри DOCTYPE::AddFieldDefs():

- Это удобное место для сбора всей информации об определениях полей
- Это позволяет оптимизировать работу средства Isearch.

Примером, в котором возникает необходимость генерации определений полей в DOCTYPE::ParseFields() является то, когда информация о полях недоступна заранее, как, например, в тех случаях, где имя поля извлекается на основании содержимого обрабатываемого документа. Наглядным случаем иллюстрирующим последнее является документальный тип SGMLTAG, при обработке которого текст документа сканируется по SGML-тегам и имя тега воспринимается как имя поля, а разграничиваемый текст - как его содержимое. Для обеспечения максимального разнообразия структуры полей и вместе с этим избежания необязательных "расходов" по определению полей, не наличествующих ни в одном из документов, генерация определений полей в AddFieldDefs() требует знания информации о полях a posteriori. Проблема разрешается путем добавления информации о каждом поле по мере ее обнаружения в фазе разборки записей документа.

Построение информации о структуре полей похоже на построение DFDT. Структура поля инкапсулируется в объекте DFT (Data Field Table - таблица полей данных), который является членом класса RECORD. Таким образом, на каждую индексируемую запись документа приходится по одному DFT. DFT, в свою очередь является списком объектов DF (Data Field - поле данных), каждый из которых состоит из имени поля и FCT (Field Coordinate Table - таблица координат поля). FCT представляет собой пары координат FC (Field Coordinates - координаты поля), разграничивающих данное поле в пределах записи документа. Вставка множества объектов FC в один и тот же FCT позволяет реализовать в Isearch такую важную особенность, как повторяющиеся поля, которые в данном контексте определяются как последовательность многократно повторяющегося одного и того же поля в пределах записи документа.

Во время поиска эти определения полей и их структуры могут быть извлечены и затем использованы для представления текста документа. Поисковое средство Isearch обеспечивает метод локализации содержимого определенного поля внутри конкретной записи документа. Однако, архитектура документальных типов создает дополнительный уровень абстракции. Один из методов поискового средства Isearch, называемый IDB::Present(), доступен для основного приложения с целью высокоуровневого представления текста документа. Это происходит путем передачи управления методу Present() конкретного документального типа, ассоциируемого с записью документа, к которой ораганизовывается доступ. Поведение по умолчанию, определяемое в DOCTYPE::Present() включает в себя извлечение и показ содержимого полей набора элементов. Вдобавок к этому, определяются наборы элементов "B" и "F", требующие представление "короткой" и "полной" записей соответственно. Целью такой архитектуры является обеспечение возможности переопределения DOCTYPE::Present() в классах потомках для реализации различных форм представления. Например, набор элементов может быть синтезирован из нескольких полей, извлеченные текстовые данные могут быть отформатированы для подходящего вывода и т.д. DOCTYPE::Present() значительно обогащает процедуру представления записей, состоящих из полей, таким образом, позволяя адаптацию к конкретному типу документа, к которому осуществляется доступ.

Рассмотрим механизм формирования и использования индексных файлов баз данных в системе Isite.

При создании базы данных индексных файлов с помощью утилиты Iindex, производится последовательное считывание всех файлов основной информационной базы данных и на основе содержащейся в них информации строится очередь (файлов БД) IQ1, являющаяся последовательностью объектов RECORD с установленными путем, именем файла БД и документальным типом. Основным классом, с помощью которого производятся все операции, служит IDB – класс индексной базы данных. Затем управление передается методу IDB::Index(), который, в свою очередь, на основе IQ1 строит очередь IQ2, представляющую собой последовательность объектов типа RECORD для каждой из записей в основной базе данных. С этой целью для разборки файлов данных на отдельные записи и локализации их координат используется метод ParseRecords() соответствующего документального типа. В результате, очередь IQ2 представляет собой список информации о месте хранения (путь, имя файла и смещение в файле) и документальном типе каждой отдельной записи.

По окончании формирования IQ2 управление передается методу AddRecordList класса INDEX, являющегося членом IDB, который осуществляет разборку записей по полям и формирует индексную базу данных, состоящую из файлов *.inx, *.dbi, *.dfd, *.mdt, *.mdk, *.mdg и *.NNN. Алгоритм INDEX::AddRecordList() приведен на Рис. 2. Файл *.inx содержит глобальные указатели на отсортированный список ключевых слов, *.dbi – информацию о базе данных, *.dfd – определения полей, причем каждому

полю ставится в соответствие трехзначное число, соответствующее индексному файлу *.NNN. Файлы *.mdt, *.mdk и *.mdg представляют собой базу таблицы MDT, используемой для трансляции глобальных указателей в реальные координаты записи. Объект MDT содержит в себе указатель на таблицу экземпляров класса MDTREC, представляющего собой полную информацию о каждой из записей основной БД.

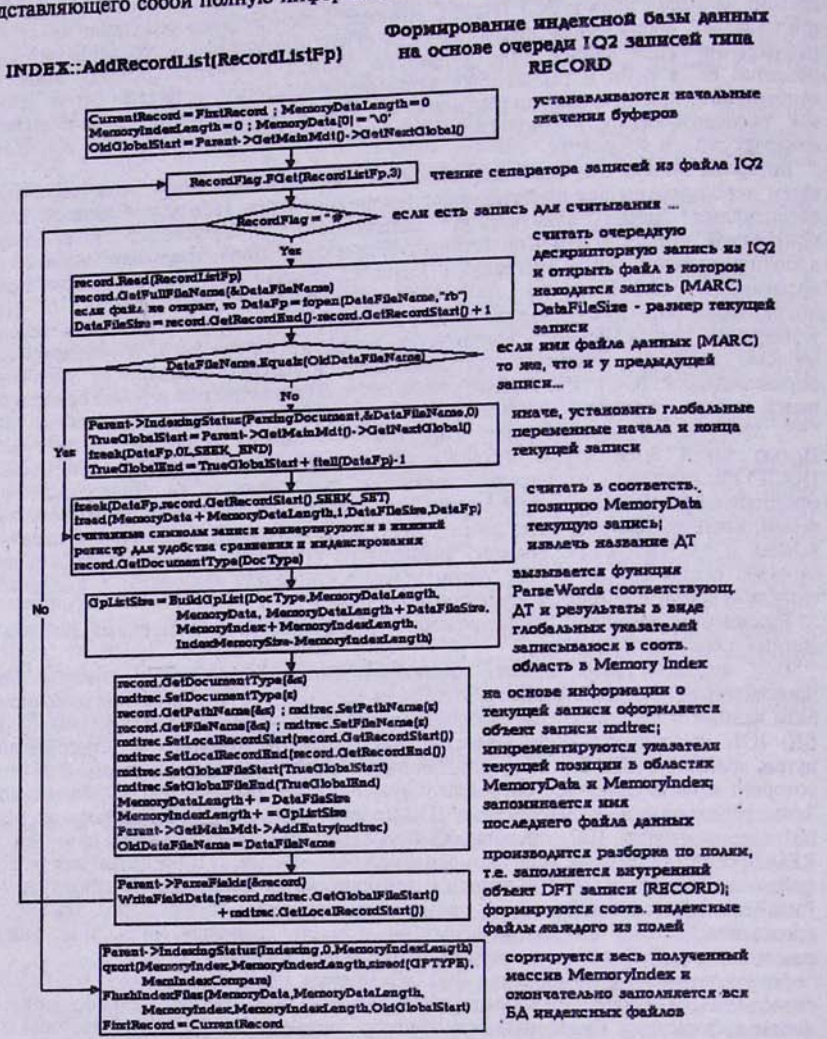

Рис. 2. Алгоритм метода INDEX::AddRecordList()

Несмотря на некоторые сложности, преимущества использования модели архитектуры документального типа перевешивают сравнительно небольшой объем усилий, требуемых для ее интеграции в уже существующие системы. Поскольку модель скорее представляет собой схему поведения документального типа, чем физическое описание документов, то диапазон представления не имеет каких-либо ограничений. Документальный тип может быть как настолько общим, чтобы обрабатывать все SGML документы, так и настолько узконаправленным, чтобы быть привязанным только к частному формату документа.

Модель документального типа во многом обязана своим успехом преимуществам объектно-ориентированного программирования, таким как расширяемость, модульность, поддержка кода, скрытие данных и способность строить на основе уже созданных классов при помощи наследования. Документы, описанные посредством документального типа "знают" как себя представлять, что минимизирует необходимость введения рискованных и утомительных внутренних модификаций в поисковое средство Isearch. В дополнение к этому, сам DOCTYPE может быть расширен с целью обеспечения дополнительных услуг доступа в поисковом средстве, в то время как классы-потомки могут одновременно с этим добавить специфические функции, связанные с каждым частным документальным типом. Поскольку поисковое средство Isearch поддерживает записи различных ДТ в одной и той же базе данных, представляется возможной обработка разнотипных текстовых данных посредством нормализации функциональности в методах классов ДТ. Например, база данных международных патентов, вследствие наличия разных национальных стандартов описания, может состоять из различных форматов данных. Вместо того, чтобы реализовывать громоздкую конвертацию данных, модель ДТ позволяет нормализацию форматов во время выполнения операций (run-time) посредством поддержки функции разборки разнотипных полей и уровня форматированного представления. Разнотипные данные могут храниться в своих "родных" форматах в одной и той же базе данных, а операции Search и Present могут абстрагироваться на основе структурных различий.

В Isite реализованы различные модульные архитектурные компоненты, основанные, где это возможно, на открытых стандартах. Взаимодействуя вместе, они реализовывают мощную, расширяемую информационную систему, способную оставаться совместимой с постоянно меняющимися парадигмами.

Однако, для реализации доступа к библиографическим данным в информационной системе Isite необходимо обеспечить поддержку наиболее распространенного библиографического документального формата MARC [1,8,9,10]. С этой целью необходимо разработать новый документальный тип UNIMARC, который позволит осуществлять функции индексирования, поиска и извлечения информации в БД записей в формате UNIMARC.

Разработка средств поддержки MARC-формата в системе Isite

Как уже было отмечено ранее, система Isite содержит в себе универсальный механизм манипулирования различными документальными типами в обслуживаемых базах данных. В системе уже имеется поддержка следующих форматов: SIMPLE, FIRSTLINE, COLONDOC, IAFADOC, MAILFOLDER, REFERBIB, IRLIST, LISTDIGEST, MAILDIGEST, MEDLINE, FILMLINE, MEMODOC, SGMLNORM, HTML, ONELINE, PARA, FILENAME, FTP, EMACSINFO, GOPHER, BIBTEX и DIF. Но для реализации доступа к библиографической информации необходимо обеспечить обработку данных в формате MARC. С этой целью требуется разработка

утилит для манипулирования записями в обобщенном формате MARC и создание нового документального типа UNIMARC, в котором и хранятся данные.

Ниже приводится описание разработанного документального типа UNIMARC. Документальный тип UNIMARC реализуется в виде класса C++ следующего вида:

```
class UNIMARC
  : public DOCTYPE  /* наследование от класса DOCTYPE */
{
public:
  UNIMARC(PIDBOBJ DbParent); /* конструктор класса UNIMARC */
  void ParseRecords(const RECORD& FileRecord); /* метод разборки записей для
                                    локализации их координат в файле данных */
  void ParseFields(PRECORD NewRecord); /* разборка записей по полям */
  GPTYPE ParseWords (CHR* DataBuffer, INT DataLength, INT DataOffset, GPTYPE*
                     GpBuffer, INT GpLength); /* пословная разборка для
                     формирования словаря ключевых слов индексных файлов */
  void Present(const RESULT& ResultRecord, const STRING& ElementSet, STRING*
               StringBuffer); /* представление записи в соответствии с указанным
               набором элементов */
  void Present(const RESULT& ResultRecord, const STRING& ElementSet,
               const STRING& RecordSyntax, STRING* StringBuffer);
               /* представление записи в соответствии с указанным набором
               элементов и синтаксисом (для функции Present Z39.50) */
  ~UNIMARC(); /* деструктор класса */
};
```

Методы UNIMARC::ParseRecords(), UNIMARC::ParseFields() и UNIMARC::ParseWords() используются при создании индексных файлов библиографической базы данных, в то время как UNIMARC::Present() необходим при выводе результатов поиска.

Метод UNIMARC::ParseRecords() предназначен для составления карты расположения записей MARC в файле библиографических записей. Это делается путем сканирования файла MARC-записей и анализа первых пяти байтов каждой записи, которые содержат ее длину. Затем производится инкрементация указателя текущей позиции в файле на эту величину и на основе этого анализируется заголовок следующей записи и т.д. Данный метод используется при создании базы данных индексных файлов документов в формате UNIMARC. С этой целью в базовом типе DOCTYPE содержится экземпляр наследуемого поля типа IDB под названием Db. Класс IDB (Index Database) предназначается для манипулирования с индексными файлами, представляющими своеобразную базу данных. Метод ParseRecords() вызывает метод IDB::DoctypeAddRecord() для каждой из сканируемых записей. В результате этого формируется файл очереди описаний записей IQ2 (для каждой из записей туда заносятся полное имя файла и относительные координаты начала и конца), на основе которого и строятся индексные файлы.

Метод UNIMARC::ParseFields() представляет собой функцию разборки записи UNIMARC по полям и занесения этой информации в специальные структуры данных (классы) [см. Рис. 4]. Единственным параметром является указатель на экземпляр класса RECORD с заполненными полями PathName, FileName, RecordStart, RecordEnd и DocumentType. Класс RECORD представляет собой удобную структуру для манипулирования записями. Помимо перечисленных полей в нем содежится экземпляр DFT (Data Field Table), представляющий собой таблицу координат всех полей записи и соответствующие имена полей. Именно эти координаты и вставляются в DFT при выполнении UNIMARC::ParseFields(). Метод UNIMARC::ParseFields() вызывается из метода INDEX::AddRecordList(), представляющего собой основную функцию индексирования базы данных записей.

ParseFields() при этом помогает при создании отдельных индексных файлов для каждого поля записи.

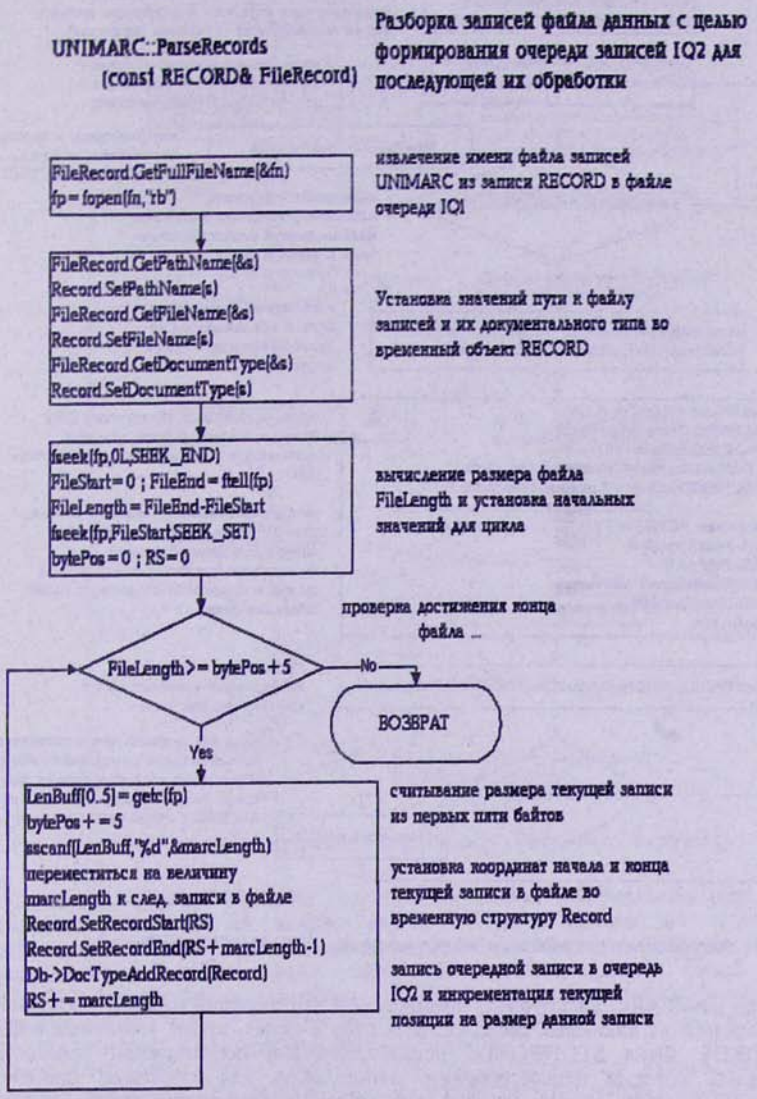

Рис. 3. Алгоритм метода разборки записей UNIMARC::ParseRecords()

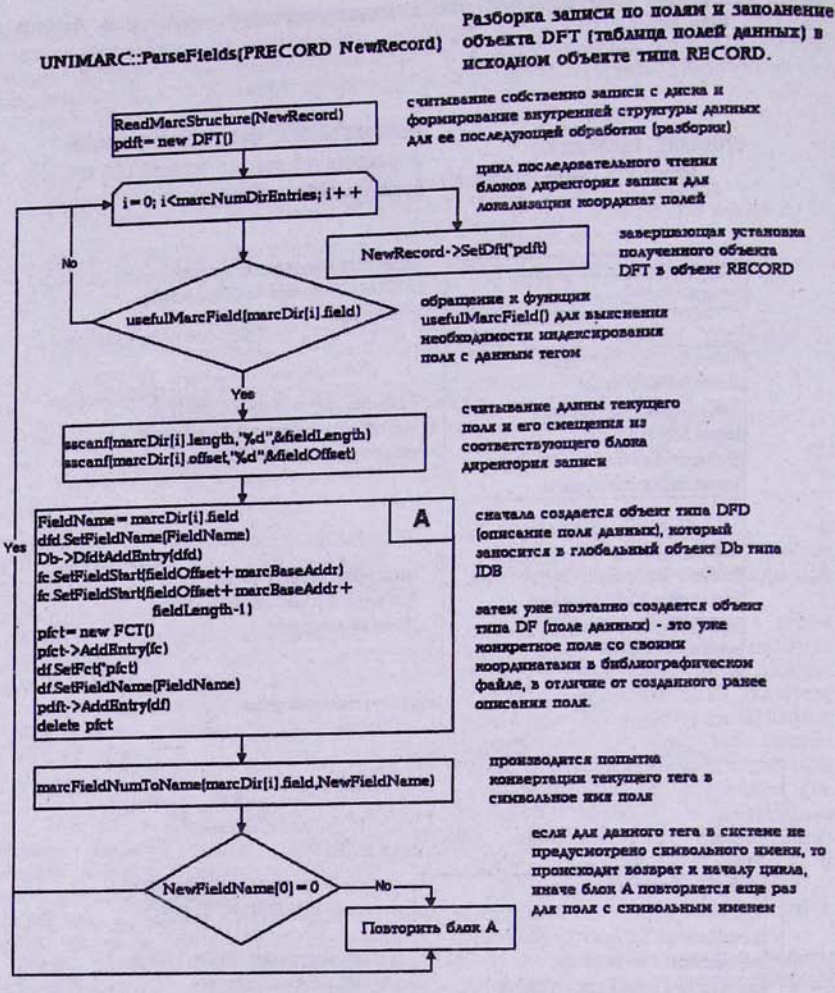

Рис. 4. Алгоритм метода разборки записи по полям UNIMARC::ParseFields()

Метод UNIMARC::ParseWords() предназначен для создания массива указателей на начало каждого из ключевых слов, т.е. всех слов в полях, кроме указанных в файле STOPWORDS. Файл STOPWORDS используется для перечисления предлогов и других слов, которые нецелесообразно использовать как ключевые. Данные из полученного массива служат основой для создания основного индексного файла, служащего для полнотекстного поиска и поиска по полям в базе данных.

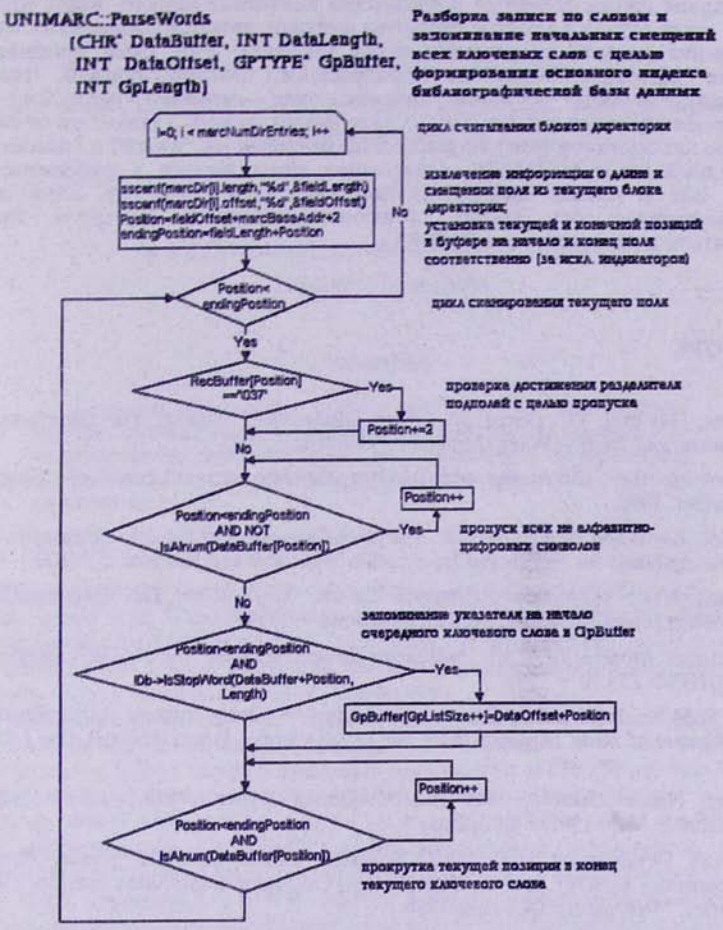

Рис. 5. Алгоритм метода локализации ключевых слов UNIMARC::ParseWords()

Структура данных GpBuffer представляет собой последовательный список глобальных указателей на каждое ключевое слово, причем для локализации конкретного смещения слова в конкретном файле на основе глобального указателя, используется индексный файл MDT и соответствующий класс. Метод UNIMARC::ParseWords() вызывается из INDEX::AddRecordList() в фазе формирования индексных файлов *.inx. В качестве исходных данных передается указатель на буфер, в котором содержится разбираемая запись UNIMARC.

Метод UNIMARC::Present() используется для представления данных результатов поисковых запросов к базе данных. Поэтому в качестве параметров предусмотрены те, что необходимы для разрешения запроса Present протокола Z39.50. К ним относятся

спецификация набора элементов и синтаксиса выводимых записей. Класс RESULT, который также служит в качестве параметра функции, является специальным классом для хранения записей из результирующих множеств. Этот класс заполняется в результате разрешения поискового запроса и, помимо прочего, содержит информацию о типе документа, полном пути, смещении начала и конца результирующей записи в файле данных. Для вывода записей в указанном синтаксисе и формате используются функции разработанных библиотек "marc.c" и "marclib.c".

Описанный класс UNIMARC в настоящее время внедрен в информационную систему Isite и служит для организации доступа к опытному Z39.50-серверу библиографических баз данных, функционирующему в Институте Проблем Информатики и Автоматизации НАН РА.

Литература

1. Cooper, Michael D. *Design of Library Automation Systems; File Structures, Data Structures, and Tools.* - Wiley Computer Publishing – 1995.
2. Denenberg, Ray. *Structuring and Indexing the Internet.* – Library of Congress. – December, 1996.
3. Gamiel, Kevin and Nassar, Nassib. *Structural Components of the Isite Information System.* – Clearingshouse for Networked Information Discovery and Retrieval (CNIDR). – 1997.
4. Gamiel, Kevin. *The Isite Information System. Ver. 2.0.* – The Clearingshouse for Networked Information Discovery and Retrieval – 1998.
5. *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification* – ANSI/NISO Z39.50 – 1995.
6. ISO 8824 – Information Processing Systems – Open Systems Interconnection – *Specification of Basic Encoding Rules (BER) for Abstract Syntax Notation One (ASN.1)* – 1990.
7. Nassar, Nassib. *Searching with Isearch. Moving Beyond WAIS.* – Web Techniques Magazine – May, 1997. Vol. 2, Issue 5.
8. Network Development and MARC Standards Office. *USMARC Concise Formats for Bibliographic, Authority, and Holdings Data.* - Cataloging Distribution Service, Library of Congress, Washington, DC, June 1988.
9. *UNIMARC Concise Bibliographic Format* – International Federation for Library Automation – May, 1997.
10. *UNIMARC. Вводный курс IFLA UBCIM.* – под ред. Я.Л. Шрайберга. – М.:1995, ГПНТБ.