САНКТ - ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Хранилище мультимедиа с группировкой данных по категориям

Дипломная работа студента 461 группы

Тверьянович Марии Андреевны

"Допустить к защите"
Заведующий кафедрой профессор ..... / Терехов А.Н./

Научный руководитель
Старший преподаватель ..... / Луцив Д.В./

Рецензент
..... /Бешко М.Ю.>/

Санкт-Петербург

2010 г.

СОДЕРЖАНИЕ

Введение

В современных файловых системах человек работает с информацией с помощью таких абстракций, как файл и папка [5]. Если рассмотреть совокупность всех путей до всех файлов файловой системы, составленных из вложенных папок, то ее можно представить в виде дерева. Поэтому, в целом такую структуру организации файлов можно назвать древовидной или иерархической. Она является удобной для компьютера, поскольку является единообразной и хорошо масштабируемой. Однако человек мыслит иначе, чем компьютер, человек мыслит ассоциациями. Один и тот же объект соотносится в сознании человека с набором определений, названий, схожих понятий и категорий, к которым этот объект можно отнести. Собственно, человек и определяет сам объект с помощью этого множества ассоциаций. Таким образом, будучи построены по другому принципу, традиционные файловые системы сильно ограничивают возможность описания файла.

Ограничение в описании данных неизменно влечет сложности при их поиске: ведь компьютер сохраняет только малую долю информации о файле, а человеческая память не безгранична. В настоящее время речь идет уже не о нескольких десятках мегабайт информации, хранящихся у каждого человека. Современные пользователи имеют в своем распоряжении уже сотни гигабайт и тысячи уникальных файлов, как например, видео, фотографии и документы. Люди, которые редко используют некоторую часть своей информации, быстро забывают, где и как она хранилась. При этом, когда им необходимо найти ее, они располагают лишь неточным и обрывочным ее описанием. И такое описание составляет определенный образ содержимого необходимого файла или необходимых файлов, по которому пользователь может их искать. Однако единственные легко задаваемые параметры, по которым ведется поиск в современных файловых системах, – это названия всех папок в пути до файла и само название файла. А такие параметры не всегда могут полностью отвечать содержимому файла или, во всяком случае, не всегда удобны для полноценного описания файла. Итак, тот образ нужных пользователю файлов, по которому пользователь может их искать, может далеко не полностью соответствовать параметрам, по которым можно осуществлять их поиск в файловых системах. Вследствие этого важно увеличить эффективность организации информации, чтобы полнее описывать файлы, и соответственно эффективность поиска необходимых файлов. Именно существование большего числа ключевых слов, у хранимого файла может способствовать тому, что человек быстрее и легче найдет необходимую ему информацию в большом массиве данных. Поскольку ключевые слова являются как бы краткими описаниями

содержимого файла и позволяют группировать файлы по категориям соответствующим наличию этих ключевых слов у файла. Таким образом, возникли идеи создания систем, в которых файл будет определяться не только его местонахождением и названием, но и описанием его содержимого. Примерами таких систем являются Windows Media Player [12], Adobe Photoshop Lightroom [8], различные системы версионного контроля и другие системы. Однако все они предназначены лишь для работы с определенными форматами данных, что оправдано для людей, которые большую часть времени работают в конкретной предметной области. Тем не менее, не для всех пользователей это приемлемо. Например, при поиске совокупности файлов совершенно разных форматов придется использовать несколько инструментов и соответственно потратить время на «ручное» сопоставление их результатов. Для конечного пользователя было бы полезнее иметь систему, которая помогала бы ему хранить файлы с ключевыми словами вне зависимости от их формата и отдельной узкоспециализированной предметной области.

Задачей данной дипломной работы является проектирование и разработка хранилища мультимедиа, где наравне с самими файлами должна храниться и дополнительная информация об их содержимом в виде ключевых слов. С помощью этой системы должен производиться поиск файлов, как по названию файла и метаданным, так и по описанию их содержимого. Сама система должна состоять из хранилища и ядра, и, как дополнение в будущем, виртуальной файловой системы. Наиболее подходящей основой для хранения ключевых слов является база данных [4], в которой без особых трудностей можно реализовать отношение «многие ко многим», то есть многие файлы можно описать многими соответствующими ключевыми словами. Таким образом, хранилище должно включать в себя определенную часть традиционной файловой системы, отведенную под хранение самих файлов, и базу данных, в которой должна храниться вся дополнительная информация. Ядро, как основная компонента системы, должно отвечать за управление и редактирование файлов, а также за взаимодействие с базой данных, сохранение целостности связей и актуальности информации. Виртуальная файловая система должна эмулировать традиционную файловую систему с древовидной организацией файлов. Это необходимо для того, чтобы приложения, работающие только с традиционными файловыми системами, могли работать с данными из хранилища. При этом ключевые слова будут отображаться в каталоги виртуальной файловой системы.

Постановка задачи

В рамках данной работы были поставлены следующие задачи:

1. Обзор и анализ существующих решений в области систем по хранению файлов с группировкой данных по категориям или похожих систем, позволяющих хранить описание данных, добавленное пользователем.
2. Проектирование и разработка системы для хранения файлов, реализующей следующие функции:
    a) сохранение файлов в хранилище и удаление файлов из хранилища;
    b) добавление, изменение, удаление ключевых слов, связанных с файлом;
    c) открытие файлов для редактирования в соответствующих приложениях;
    d) поиск файлов по отдельным параметрам;
    e) поиск файлов по поисковому запросу.
3. Разработка алгоритма поиска файлов по поисковым запросам.
4. Создание пользовательского интерфейса для работы с файловым хранилищем.

Литобзор

Существующие подходы

Краткий исторический обзор

Идея предоставить пользователю более дружественную, чем файловая система, среду для хранения данных имеет те же источники что и идея "канцелярских" метафор, таких как "рабочий стол", "папка", "корзина". Источником данных "канцелярских" метафор была компания Xerox, применившая их в системах Xerox Alto и Xerox Star. Именно эти системы стали прототипами систем компании Apple, Inc., о которой пойдёт речь дальше.

Закладывая и укрепляя традиции метафоры "офисного рабочего пространства", соблюдаемые и до сих пор, система Apple Lisa отображала данные файловой системы в виде, привычных человеку, канцелярских объектов. Каталоги показывались в виде папок, содержимое которых можно было просмотреть в окне. Файлы, хранящие документы различной природы (тексты, таблицы, изображения) отображались при помощи разных иконок. На том же рабочем пространстве находились и метафоры инструментов: текстовых процессоров, календарей, часов.

Некоторые метафоры могли показаться по современным меркам слишком буквальными: так, например, был специальный объект "пачка бумаги", из которого можно было извлечь листы для написания нового документа. Обрабатываемые в данный момент документы помещались на рабочий стол. Для того, чтобы сохранить и закрыть документ, необходимо было перетащить его иконку обратно в папку, из которой документ был взят или в другую.[3], [7].

Одна из интересных и важных в контексте данной работы особенностей метафоры рабочего пространства - возможность задания одинаковых имён для разных документов Apple Lisa [10]. Уже в начале 1980-х годов создатели системы осознавали, что оператору компьютера имя документа, само по себе, может говорить мало, и позволили для отображения на рабочем столе использовать т.н. виртуальные имена, которые не были уникальны. Так, например, созданная копия документа называлась, так же как и оригинал, хотя при редактировании её и оригинала свойства начинали отличаться. В дальнейшем пользователь различал документы по их свойствам (дате, размеру) и содержимому. При сохранении на диск данные документов записывались в файлы, физические имена которых были уникальны и соответствовали накладываемым файловой системой

ограничениям. Разумеется, пользователь мог изменить и логическое имя документ, если считал нужным. В современных ОС семейств Windows и в UNIX-подобных логические имена файлов не применяются.

Microsoft уделила должное внимание дружественному именованию документов в офисных и домашних версиях своих систем (расширив возможности по именованию файлов файловой системы FAT при помощи надстройки VFAT) более чем на 10 лет позже, в 1995 году при создании Windows 95 [6].

В Windows NT, ориентированной изначально на корпоративный сектор, возможности именования файлов сразу были достаточно богатыми. Часть из них файловой системой NTFS была унаследована от файловой системы HPFS ОС OS/2.

Таким образом, уже в середине 1990-х гг. Microsoft пошла по традиционному пути, предоставив пользователю традиционную файловую систему с достаточно широкими возможностями по сравнению с семейством DOS, например. Эта тенденция сохранялась ещё более 10 лет вплоть до выпуска Windows Vista, в которой была впервые принята некоторая функциональность, относившаяся до этого только к исследовательскому проекту WinFS.

Современные файловые системы

Насущная потребность в новых принципах систематизации данных привела к появлению в последнее время разработок и исследований по добавлению к файлам дополнительной информации об содержимом файла. Определенные разработки на данную тему появились и в последних версиях операционных систем, таких как Windows и Mac OS [11]. Что говорит о действительной значимости данного вопроса.

В них были введены возможности для поиска файлов не только по пути, названию, метаданным, но и по тегам (комментариям в Spotlight Mac OS [11]), добавленными пользователями к файлам.

Рассмотрим подробнее Mac OS X. С появлением в OS X новой функции Spotlight, поиск на Mac стал как никогда быстрым и удобным. Spotlight упростил поиск файлов по тегам (или меткам), не требуя от пользователя знания точного названия файлов или их содержимого.

Тем не менее, присвоить метку к документу для быстрого поиска не так-то просто. Ключевые слова можно включать в название документа или открывать свойства файла и прописывать теги в поле “Комментарии Spotlight”. Для более простого добавления меток в Mac, разработаны дополнительные приложения, такие как Tags и Punakea.

В Windows же начиная с XP и до появления 7й версии, хотя и существовала возможность добавления меток, использовать ее было не только неудобно, но и мало кто

был в курсе данной возможности. Так как чтобы добраться до этого, нужно было зайти в «Свойства», а затем на вкладку «Сводка» и так уже выставить дополнительную информацию. Если представить что данные манипуляции нужно проделать с каждым из тысячи файлов. Однако Microsoft осознало недостатки данной системы и необходимость и удобство для пользователя добавлять дополнительную информацию в виде ключевых слов к файлам для будущего поиска. Таким образом, уже в недавно вышедшей Windows 7 появились удобный интерфейс для добавления ключевых слов без захода в свойства файла, а также модифицированные возможности по каталогизации и категоризации файлов в библиотеках по этим ключевым словам. Под библиотекой в Windows 7 понимается определенный пользователем набор папок специфического содержания, представляющий данные вне зависимости от иерархии папок.

Узкоспециализированные приложения - каталоги видео и фотографий

На темы, аналогичные освещенной в работе, многими небольшими компаниями были созданы узкоспециализированные приложения – хранилища, которые являются сугубо коммерческими проектами и при этом закрытыми для посторонних глаз. Поэтому сложно рассказать, как они решали технические вопросы, связанные с созданием этих приложений. Кроме того, данные хранилища ориентированы на определенные файловые системы и работают только с определенными типами файлов, такими как видео или изображения, иногда тексты. В них есть возможность добавления тегов и поиска по ним.

Например Adobe Photoshop Lightroom использует каталог для отслеживания местоположения файлов и запоминает о них информацию. Каталог, как база данных, содержит записи фотографий пользователя. Эта запись хранится в каталоге и содержит такие данные, как ссылки, указывающие, где находятся фотографии на компьютере, метаданные, описывающие фотографии При оценивании фотографии можно добавить метаданных и ключевые слова, организовывать фотографии в коллекции или удалить фотографии из каталога, даже если исходные файлы в автономном режиме фото-настройки хранятся в каталоге.

Помимо работы с музыкой, проигрыватель Windows Media Player 11 предоставляет новые огромные возможности хранения и использования цифрового мультимедиа. Он значительно упрощает доступ ко всем видеозаписям, изображениям и записанным телепередачам на компьютере. Не только музыка, но другие виды мультимедиа (видео, изображения и записанные телепередачи) выделены теперь в отдельную категорию на панели «Библиотека». В библиотеке можно отсортировать музыку по ключевым словам.

WinFS - система хранения и управления данными на основе реляционной базы данных

Единственным далеко продвинувшимся проектом по разработке более универсальной файловой системы на основе категорий, к которой также было приложено много сил, является проект Microsoft [14]. Он заключался в создании новой файловой системы, сокращенно называемой WinFS. Предполагалось, что она будет представлять собой файловою систему будущей операционной системы. К сожалению, данный проект был закрыт, по причине усложненности задачи, желания охватить все возможные ее аспекты и качественно их разработать. В нем предусматривались не только сложная функциональность, но и унификация всех файлов под единую структуру [13]. Однако все же наработки по этому проекту все же войдут в следующие операционные системы, а также в Microsoft SQL Server и ADO.NET [14].

Используемые технологии

Microsoft .NET

В качестве основной платформы в данной работе используется платформа Microsoft .NET. Поскольку существует необходимость тесного взаимодействия программной части и базы данных, нужно было выбрать наиболее удобные технологии по созданию базы и по созданию программного обеспечения работающего с этой базой.

Языком программирования для реализации хранилища мультимедиа был выбран C#. Таким образом, код реализации можно будет скомпилировать с помощью Mono, а Mono может исполнять модули на многих других операционных системах кроме Windows, например, Lunix. Тем самым обеспечивается возможность в будущем использования разработанного прототипа хранилища мультимедиа кроссплатформенно. А для разработки базы данных используется Microsoft SQL Server, включающий в себя .NET.

Проект Irony

Проект Irony представляет собой средство разработки и реализации языков на .NET. Он содержит платформу для реализации языков, LALR и NLALR парсер. С помощью Irony можно писать свои собственные грамматики, по средством которых можно распарсить строку ввода в дерево с помощью парсера, который также описан внутри отдельной библиотеки. Кроме того данный проект написан полностью на C#. Он

использует гибкость и мощь языка C # и. NET Framework 3.5 для осуществления совершенно новой и модернизированной технологии построения компиляторов.

GNU libextractor

GNU libextractor - это библиотека, используется в данном проекте для извлечения метаданных из файлов произвольного типа. Она создана с использованием вспомогательных библиотек, производящих извлечение данных, и легко расширяется внешними модулями, модулями, поддерживающими другие типы файлов. libextractor это пакет GNU. GNU открытая Unix подобная операционная система.

Она имеет определенные недостатки, такие как проблема с кодировками не поддержка UTF-8. Кроме того в версия для Windows более старая, чем текущие и не содержат определенных исправлений багов системы. Однако данные недостатки можно обойти и в большинстве случаев. Использование этой библиотеки позволяет удобно получить метаданные по файлам совершенно разных форматов, таких как HTML, PDF, PS, OLE2 (DOC, XLS, PPT), OpenOffice (sxw), StarOffice (sdw), DVI, MAN, FLAC, MP3 (ID3v1 and ID3v2), NSF(E) (NES music), SID (C64 music), OGG, WAV, EXIV2, JPEG, GIF, PNG, TIFF, DEB, RPM, TAR(.GZ), ZIP, ELF, S3M (Scream Tracker 3), XM (eXtended Module), IT (Impulse Tracker), FLV, REAL, RIFF (AVI), MPEG, QT и ASF.

Архитектура

Компонентая структура системы

Главной частью системы является ядро или, так называемая, управляющая система, которая будет отвечать как за введение данных в хранилище, так и за обработку запросов поиска к хранилищу, то есть отвечать за всю работу с данными.

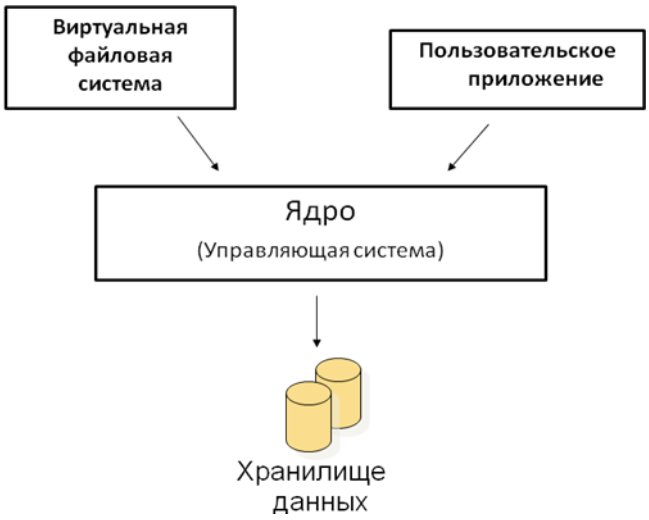


Схема 1. Общая структура компонент системы.

Определено само хранилище, позволяющее хранить не только сами данные, но и всю дополнительную информацию, добавленную пользователем, при этом имеющее адекватную не древовидную структуру. Структура должна поддерживать отношение многие ко многим. Так как каждый файл может иметь много тэгов, а тег может быть у многих файлов.

Между ядром и пользователем или приложениями, так сказать, внешним миром, создаются специальные библиотеки. Эти библиотеки являются интерфейсами, скрывающими и структурирующими функциональность ядра.

Вся система должна быть удобной и простой. Удобство, потому что, не имеет смысла создавать новую файловую систему, более неудобной, чем существующая файловая система. Простота, так как любое чрезмерное усложнение в постановке задачи ведет к усложнению в ее реализации и не гарантирует выигрыш в эффективности.

Обобщенную модель хранения файлов в файловой системе на основе категорий можно представить в виде простой модели Сущность-Связь [2]. Сущностями являются файлы, информация о которых должна сохраняться в хранилище, а также теги. Между файлами и соответствующими тегами создается связи, причем эти связи типа многие ко многим, вследствие того, что файл может быть связан со многими тегами, а тег со

многими файлами. Атрибутами в данном случае могут являться, например, название файла, или некоторые метаданные, такие как дата создания и дата модификации файла. Поиск ведется как по атрибутам, так и по связанным тегам. Поэтому основополагающий признак этой модели состоит в том, чем больше критериев мы вводим в поиск, тем уже выборку из подходящих результатов мы получаем.

При условии разработки абстрактного ядра, хранилище данных может быть по сути любым. Однако была выбрана реляционная база данных, как наиболее простая, удобная модель хранения данных и представления их, как для человека, так и для машины. С помощью табличной структуры можно показать всю многогранность определений сущности файла. А база данных наиболее эффективна для хранения любой информации. При этом существует возможность варьировать структуру в зависимости от необходимости.

Кроме того, современные технологии хорошо разработаны в рамках создания самих РСУБД [3]. Конечно, не исключено, что специализированное под нашу модель хранилища данных может оказаться лучше. Но на данном этапе его разработка не предусмотрена.

Необходима разработка фронт-эндов для обеспечения возможности для существующих приложений работать с новой файловой системой. При этом для приложения не должны требоваться существенные изменения. Так как никто не станет делать новые версии приложений под какую-либо новую файловую систему. Что в полнее логично.

Для приложений, которые будут поддерживать новую файловую систему, будут разработаны библиотеки. Эти библиотеки будут позволять работать приложениям с моим хранилищем через ядро вместо обычной файловой системы. Также для существующих приложений, поддерживающих плагины, можно создать плагин для работы с библиотекой.

Не все приложения смогут читать данные из новой файловой системы посредством библиотек. Так, например, легаси приложения. Они могут взаимодействовать только с файловой системой древовидной структуры. Для таких приложений нужно разработать виртуальную файловую систему по нашей файловой системе, которая представляет собой древовидную структуру, через которую легаси приложения смогут иметь доступ к данным.

Для обоих фронт-эндов можно рассматривать путь к файлу как запрос к базе. Так как оба фронт-энда работают с ядром от нашей системы, а она находит файл по запросу к базе.

Представленную выше модель можно расширить, с помощью импликации меток, а также добавления синонимов, и предикатов - отношений между метками или данными, например отношения следования. Это конечно осложнит функциональность, но может дать новые возможности.

В хранилище можно хранить не только сами файлы и метки, связанные с файлами, но также и их метаданные. Кстати это может быть довольно полезно для пользователей. Так как им интересно не только, есть ли в хранилище такой объект с таким именем и метками, но и какие у данного объекта есть параметры, характеристики, как бы его “физические” данные. Для этого необходимо извлекать метаданные и добавлять их в базу.

Вся система состоит из трех частей:

1. База данных содержащая схему базы и хранимые функции, написанные на T-SQL, на MSServer.
2. Приложения, реализованные на языке C#, состоящие из:
    a. Class Library – Ядро
    b. Оконное приложение – GUI, предполагающее использование Ядра через API
3. Модуль синтаксического анализа для строки запроса клиента (см. раздел Грамматика)
    a. Отдельная библиотека Irony[5].
    b. Библиотека, содержащая грамматику.

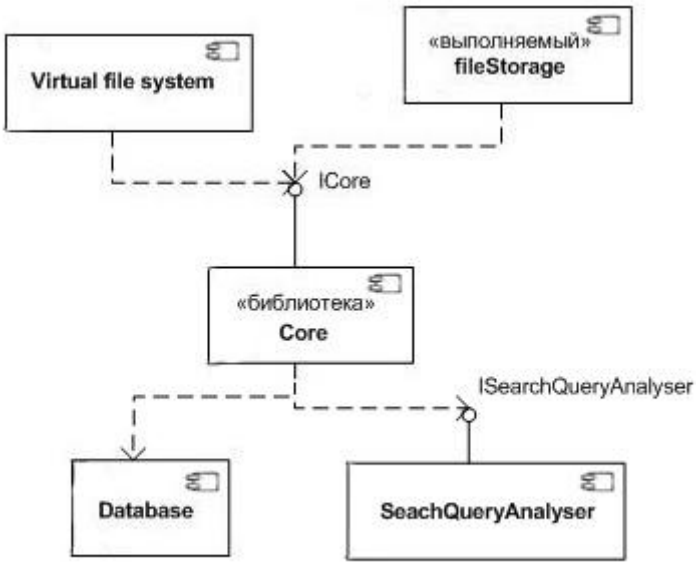


Схема 2. Диаграмма компонент и их взаимосвязь.

База данных как основа хранилища системы

Файл и база данных

Сами файлы хранятся не в базе, а в файловой системе. Это объясняется нецелесообразностью реализации фактического хранения файла в базе, по причине не универсальности типов файлов, так как в базе данных предполагается хранение однотипных элементов в столбцах. Из-за этого пришлось бы не только создавать универсальный тип для всех видов файлов, что само по себе крупная задача, но и модифицировать базу данных под эти нужды.

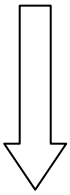
При внесении информации о файле, необходимо учитывать, что следует вносить новую информацию в другие таблицы, например, если какого-то расширения в базе нет, необходимо его добавить.

При удалении файла надо учитывать, что в базе могут остаться соответствующие метки или расширения, которые больше ни с чем не связаны, в таком случае также стоит их удалить.

Поиск по базе данных

Есть набор данных, которые могут быть получены от пользователя для поиска:

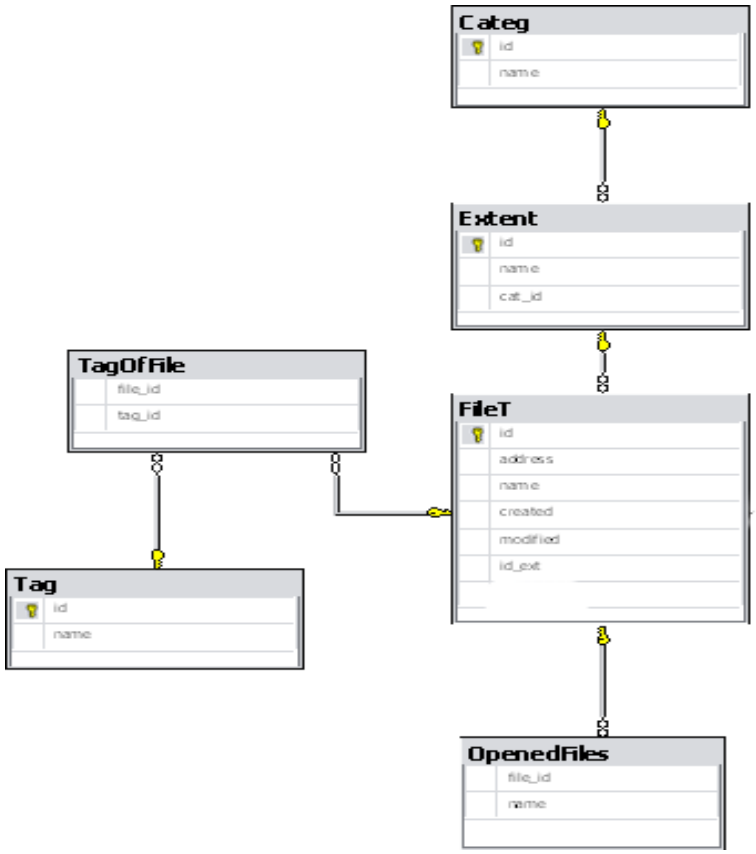
a. Имя
b. Расширение
c. Категория
d. Метаданные
e. Метки



В этом списке они упорядочены по значимости, то есть если у нас есть имя то, скорее всего сначала искать надо по нему и затем искать со следующими данными относительно полученной выборки.

Упрощения

1. Пока поиск по автору не учитывается. Решение как это можно будет сделать см. пункт 4 списка аналогично, только для Таблицы Автор.
2. Если нам дано расширение по категории не ищем
3. Расширение файла (как и категория) в поисковых данных может быть в единственном числе, то есть конкатенация результатов не предусмотрена и два варианта расширения можно считать взаимоисключающими (результат = 0) или просто игнорировать все кроме первого.



4. В будущем, возможно, при поиске предложением/ введенным пользователем, необходимо учитывать, что метки,, выделенные с помощью синтаксического анализатора, могут являться именами. Поэтому искать лучше по объединенной таблице Tag и имен (то есть будет необходимость сделать отдельное представление соответствующее структуре таблицы Tag (идентификатор, имя файла)).

Схема 3 База данных ля хранения данных о файле

Организация поиска

Вопрос состоит в том посредством, чего лучше искать.

Существует следующие варианты способа реализации функциональности базы:

1. Хранимые процедуры с динамическим
2. Статические хранимые процедуры
3. Код на .NET в Ядре, т. е. Реализации вне самой базы
4. CLR функции

В поисковых запросах нам важна скорость работы, как известно методы, написанные на CLR[1], удобны для использования более сложной функциональности, которую нельзя или почти нельзя написать на T-SQL. Но они работают медленнее, чем T-SQL, кроме того, их использование несколько портит кроссплатформенность.

В плане скорости хороши хранимые процедуры и функции, без использования динамического SQL. Однако если писать все статически, то нужно около 20 отдельных функций, с учетом значимости данных и для всех вариантов их сочетания. Хочу заметить, что написание нескольких хранимых функций, без зависимости друг от друга и затем

[1] CLR функции по сути такая же функциональность, как и хранимые функции базы данных написанные на T-SQL, только они создаются на .NET и загружаются в базу данных MSS из сборки.

объединение (пересечение) результатов, изначально считается не оптимальным. Поскольку лучше вести поиск по меньшему множеству, если его уменьшение возможно по другим более конкретным данным.

Существует еще вариант использования динамического SQL в процедурах, что уменьшит общее число методов. С процедурами на T-SQL есть еще одна сложность необходимость использования локальных таблиц, для передачи ей результатов работы предыдущей процедуры (например, результат работы поиска по расширению должен быть передан в запрос процедуры поиска по меткам). Только функции можно непосредственно вставлять в запрос без необходимости передачи ее результата в локальную переменную.

Наконец, в принципе, можно конструировать строку запроса в Ядре, затем выполнить этот запрос в базе, и вернуть результат в Ядро.

Получаем четыре варианта написания кода

a. Написать часть в виде процедур, где динамически подставляются данные в запрос

b. Написать отдельные статические функции (так называемые Inline stored functions) для каждого случая на T-SQL

c. Написать функции на CLR, что, скорее всего, будет медленнее работать, чем вариант с T-SQL.

d. Сделать строку запроса в Ядре и выполнить ее из него же.

Пока в качестве основы был выбран вариант использования всего, кроме CLR (он правда тоже нужен, если допустим, будет нужно сделать агрегирующую функцию для конкатенации строк, например). С учетом того что скорость важней, лучше всего использовать Хранимые процедуры. Однако увлекаться не стоит. Считаю, достаточно будет использовать их для случаев, когда есть только данные одного рода или из двух типов - имя и еще что-либо.

Кроме того такие данные, как метаданные, в запросе участвуют скорее всего в неравенствах (например ‘`data_created < 04.10.09`’) или равенствах; трудно предугадать какой знак будет использоваться, поэтому оптимально для поиска с метаданными передавать метаданные как готовую строку, вставляющуюся в запрос, а лучше как вызов функции с параметрами. То есть для поиска с метаданными можно использовать следующие варианты: хранимые процедуры с динамическим SQL или код на .NET в Ядре.

Остальное не сложно написать, собирая строку запроса в Ядре.

Получаем:

- 4 хранимые функции от одного параметра данных, а также еще 3 функции от двух.
- 1 динамическая процедура для поиска с метаданными, если нам не даны метки. Чтобы не было необходимости в локальной таблице для извлечения результата из этой процедуры и использования его при поиске по меткам.
- 4 функции возвращающие строки (скорее всего лежащие в ядре) для собирания строки запроса в Ядре.

Функциональность

Функции и процедуры, которые написаны на T-SQL и находятся в базе.

Хранимые функции из базы:

Возвращают

1. table
    a. `SearchByName(@name nvarchar(50))`
    b. `SearchByExt(@ext nvarchar(4))`
    c. `SearchByCat(@cat nvarchar(20)) – категория`
    d. `SearchByTags(@tags nvarchar(50))`
    e. `SearchByExtwName(@ext nvarchar(4),@name nvarchar(50))`
    f. `SearchByCatwName(@cat nvarchar(20),@name nvarchar(50)) – категория`
    g. `SearchByTagswName(@tags nvarchar(50),@name nvarchar(50))`
    h. `GetTagsOfFile(@fid uniqueidentifier)`
    i. `GetCategs()`
    j. `GetExtOfCat(@cat nvarchar(20))`
    k. `GetExt()`
    l. `GetTagsIdOfFile(@fid uniqueidentifier)`
2. uniqueidentifier
    a. `GetIdOfFile(@fadr nvarchar(50))`
    b. `GetExtId(@ext nvarchar(4))`
    c. `CheckExExt(@ext nvarchar(4))`
    d. `CheckExName(@ext nvarchar(4),@name nvarchar(50))`
    e. `CheckExTags(@list ntext, fid uniqueidentifier)`

Хранимые Процедуры:

1. `SearchByMeta(@metadata nvarchar(50),@X nvarchar(50))`- Процедура, в которой запрос выполняется по динамически из составленной строке, где параметр X – строка, содержащая таблицу, из которой берутся данные. То есть множество, откуда мы выбираем данные, может быть уже более узким, в зависимости от наличия других данных.
2. `UpdF(@fadr varchar(50), @name varchar(50), @idext uniqueidentifier)`
3. `checkExistExt(@extern nvarchar(4), @res uniqueidentifier OUTPUT)`
4. `checkExistTags(@list ntext,@fileid uniqueidentifier)`
5. `InsrtF(@fadr nvarchar(50), @name nvarchar(50), @ext nvarchar(4), @fileid uniqueidentifier OUTPUT)`
6. `deleteRedundantExt(@extid uniqueidentifier)`
7. `deleteRedundantTags(@fileid uniqueidentifier)`
8. `DeleteF(@fileid uniqueidentifier)`
9. `UpdTags(@fileid uniqueidentifier, @list ntext)`

Ядро системы

Ядро - основная часть системы контролирует обработку файлов, посылает запросы к базе для добавления, модификации данных о файлах.

Алгоритм работы

Общий алгоритм работы ядра представлен на схеме. В зависимости от действия, которое нужно произвести, либо идет поиск по базе данных с выводом результата поиска, либо производится управление файлами.

Управление файлами состоит из двух независимых этапов, составляющие одну транзакцию. Они проводятся последовательно, так чтобы в случае возникновения ошибки, система раньше отменила проделанные изменения. Физически файлы обрабатываются на втором этапе, поскольку тогда сложнее отменить действия, произведенные в файловой системе, и так как при взаимодействии с базой данных вероятнее возникновение неполадок.

При возникновении ошибки, работа прекращается, и система возвращает код ошибки. При этом, если в хранилище были произведены изменения, они отменяются.

Подробнее алгоритм поиск будет рассмотрен в разделе «Поиск файла».

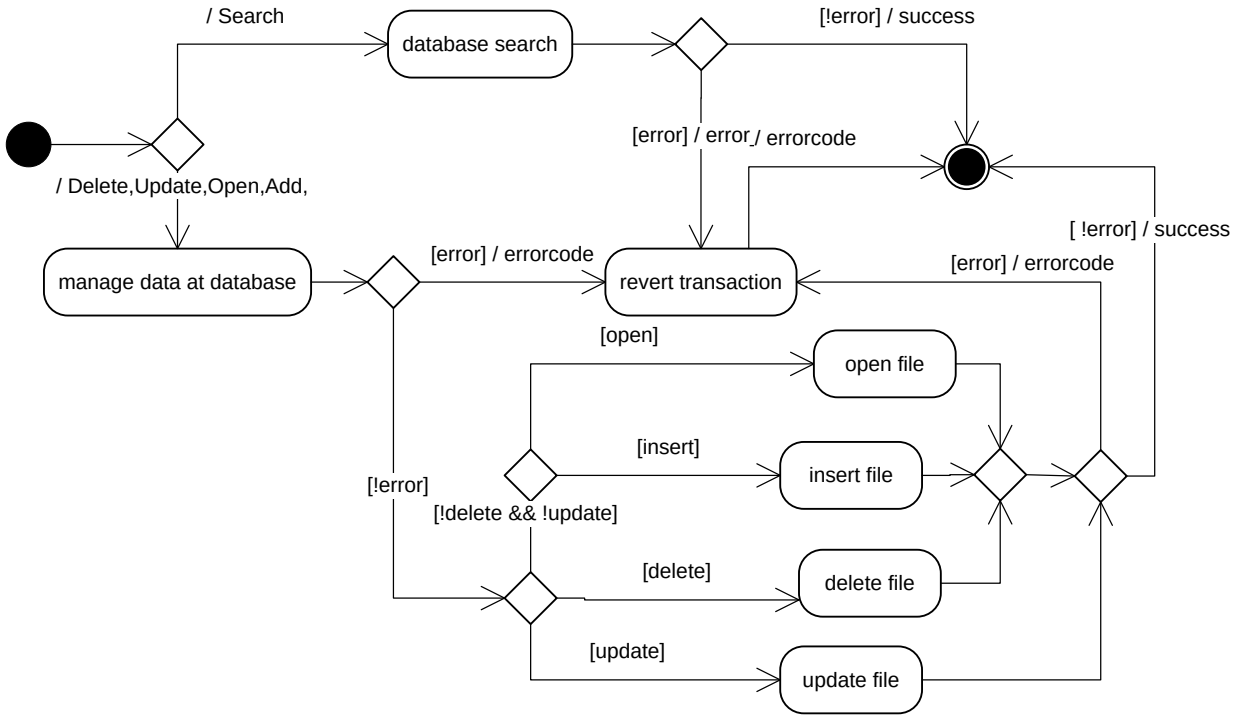


Схема 4 Алгоритм работы ядра. Управление файлами состоит из двух частей: изменение соответствующих данных в базе данных – manage data at database, управление файлом в хранилище физически – open, delete, update, insert file.

Таким образом, работа ядра делится по действию на:

- администрирование файлов:
  - сохранение,
  - удаление,
  - модификация,
  - открытие;
- поиск файлов.

И по обработке данных.

- Обработка информации о файле в базе данных.
- Обработка физического файла.

Модель ядра

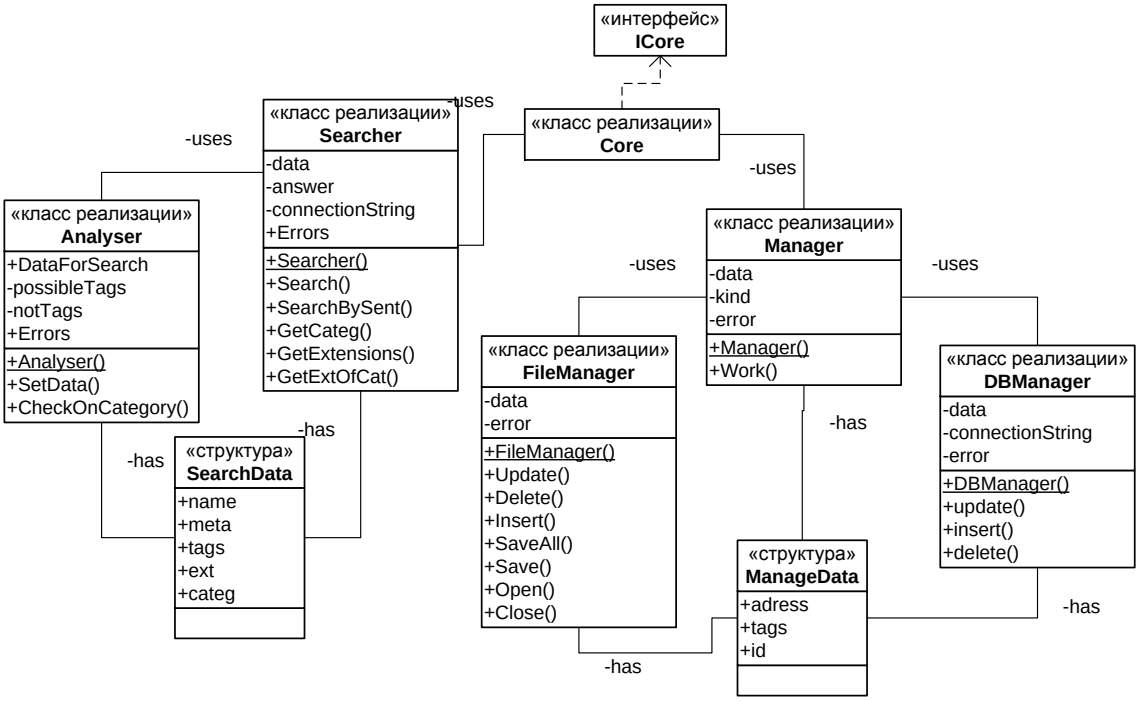


Схема 5. Обобщенная диаграмма классов ядра.

Ядро (Core) – управляющая система. Ядро состоит из поисковика (Searcher), отвечающего за поиск данных по базе данных, менеджера (Manager), отвечающего за управление взаимодействием между менеджера файлов (FileManager) и менеджера базы данных DBManager. В свою очередь менеджер файлов работает с файлами из хранилища на уровне файловой системы, а менеджер базы данных – на уровне данных в базе данных. Для поиска и для управления существуют отдельные структуры, в которые сохраняется информация, с помощью которой осуществляется необходимые действия, соответственно SearchData и ManageData. Также в ядре содержится анализатор (Analyser), который обрабатывает пользовательский запрос для поисковика, так чтобы его можно было сохранить как SearchData, по которому и будет составляться необходимые запросы к базе данных.

API ядра

Для создания удобных интерфейсов, в API предусмотрены основные методы, необходимые для работы с хранилищем. Их реализация скрыта от пользователей.

API включает функции:

- поиск,
- вывод всех файлов в системе,
- добавление файла,
- удаление файла,

- модернизация данных по файлу,
- сохранение изменений в файлах,
- открытие/закрытие файла,
- удаление файла из хранилища (перенос в обычную файловую систему),
- проверка существования файла с таким же именем (включая расширение файла) в хранилище,
- вывод всех расширений файлов находящихся в системе,
- вывод всех расширений файлов, соответствующих конкретной категории,
- вывод всех категорий файлов системы.

Работа с файлом

Сохранение, удаление, модификация файлов состоит из двух независимых этапов: внесение данных в базу данных и обработка файла на уровне файловой системы.

Сами файлы хранятся не в базе, они находятся в отдельной области файловой системы, специально выделенной под хранилища. Так как нет необходимости хранить файл внутри базы. Единственная проблема, которая может возникнуть в данном случае, удаление или перемещение и изменение файла без синхронизации с базой данных. Однако чтобы этого не произошло, файлы будут храниться в отдельном скрытом месте, выделенном для хранения, когда он добавляется в базу.

Причем во избежание проблем с именами, было решено хранить файлы в виде файлов, имена которых соответствуют их идентификаторам в базе, а расширение является собственным расширением хранилища.

Положительные стороны данного подхода можно изложить в следующем. Мы, как было сказано выше, избегаем проблемы с переименованием файла при наличии другого файла с аналогичным именем, но иным содержанием. При этом доступ к нему должен быть ограничен, поскольку файл нельзя изменять без уведомления системы. При подобном подходе извне нельзя будет точно определить “на глаз” какой файл на самом деле находится под тем или иным идентификатором. Также файл однозначно определяется, как хранимый в хранилище, что добавляет возможность проведения ассоциации между данным расширением и приложением для работы с хранилищем.

Нужно заметить, что при возникновении ошибки на любом этапе все изменения в базе и в хранилище, если они были сделаны, должны быть удалены.

Добавление и удаление файлов

При сохранении данные о файле вводятся в базу данных.

Сам файл переноситься из текущего места нахождения в хранилище, отдельная часть файловой системы. Перед этим совершая проверку на существование файлов с таким же полным именем, имя вместе с расширением. В случае существования аналогичных файлов, следует выводить их в виде списка пользователю, для того чтобы он мог сориентироваться, был ли аналогичный файл уже внесен в хранилище или в нем содержится его старая версия, которую необходимо заменить.

Соответственно при удалении аналогично данные о нем удаляются из базы, а файл физически удаляется из хранилища. Заметим, что в таком случае в будущем необходимо предусмотреть восстановление файла при необходимости, повышая сохранность данных.

Кроме полного удаления файла, существует вывод файла из хранилища, а именно сохранение его по какому либо адресу с его полным именем и последующим удалением его из хранилища.

Открытие файла в сторонних приложениях

Для того чтобы изменить сам файл, его нужно открыть через ядро системы, передав ему, либо адрес файла, либо идентификатор.

Процесс открытия файла в хранилище состоит из двух этапов.

1. Создание копии файла с его настоящим именем во временной папке.
2. Открытие копии в соответствующей программе.

Таким образом, задаются следующие определения: файл из хранилища является *открытым*, если у него существует временная копия, занятая в каком либо процессе, *модифицированным*, если копия имеет более позднюю дату изменения файла. Хочется заметить, что, по сути, модифицированные файлы можно так же считать открытыми, поскольку на самом деле изменения еще не были внесены в файл из хранилища.

В ядре так же предусмотрены настройки, в которых сохраняются данные, о том какие файлы были открыты и где их временная копия находится. В таком случае удобно заменять старую версию файла в хранилище на временный файл, если в него были внесены изменения и пользователь желает их сохранить. Для этого можно послать ядру указание сохранить изменения (все файлы, открытые до этого момента, заменяются своими временными копиями, затем копии удаляются) или сохранить изменения для конкретного файла (конкретный файл заменяется на его временную копию, копия удаляется). При этом определим следующее, *закрыть* файл в хранилище – это следующая комбинация действий: убрать файл из списка открытых, внеся или не внеся изменения в сам файл, удалить копию. Пока мне не хочется делать автоматическую замену открытого файла при освобождении его копии процессами. В таком случае можно, например, с

легкостью закрыть файл без внесения изменений, если вдруг были сделаны не желательные изменения. Такая возможность тоже предусмотрена в ядре.

Ядро может сообщать список открытых файлов, для которых существуют копии. Это облегчит мониторинг открытых файлов.

Отслеживание изменений в файле

Как было сказано выше, необходимо обеспечить слежение за изменениями в файлах. Это можно сделать с помощью двумя вариантами.

1. Предотвращение изменения файла без использования специального пользовательского интерфейса.
2. Отслеживание в реальном времени за изменением файлов.

Для отслеживания в реальном времени необходимо, чтобы система работала все время во время работы компьютера. Кроме того если система была отключена некоторое время, то необходимо обеспечить сканирование всех файлов на предмет несоответствие с данными из базы данных. В данном случае нам нет необходимости отслеживания в реальном времени, так как основная задача хранилища мультимедиа добавления ключевых слов и реализация поиска по ним. Поэтому для реализации прототипа системы был выбран первый вариант.

Поиск файлов

Ядро возвращает набор данных содержащий данные из таблиц соответствующие критериям поиска.

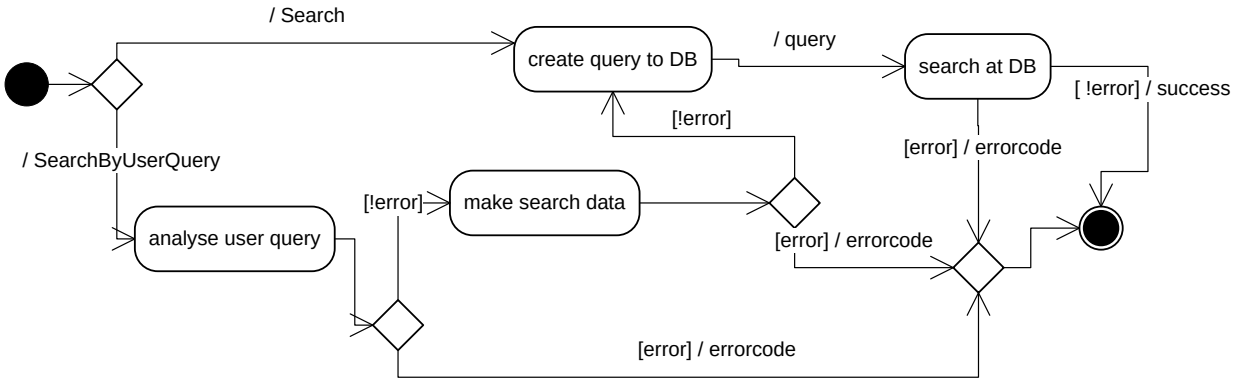


Схема 6. Алгоритм работы поиска.

Общая схема поиска проста, мы получаем данные для поиска, строим по ним запрос к базе данных и посылаем запрос, если все в порядке возвращаем результат.

Было решено обеспечить пользователей возможностью искать файлы с помощью так называемых пользовательских запросов, аналогичных интернет поисковым запросам к различным интернет поисковикам, таким как, например, Google и Yahoo. Т.е.

пользовательский запрос представляет собой строку, в которой перечислены все критерии поиска.

Если же необходимо обработать пользовательский запрос, то алгоритм усложняется. С помощью анализатора производится анализ информации из пользовательского запроса, по нему строятся данные для поиска, а дальше аналогично обычному поиску.

Виртуальная файловая система

Виртуальная файловая система необходима для того что бы файлы можно было открывать без использования специального пользовательского интерфейса над хранилищем а прямо из сторонних приложений. Т.е. она должна эмулировать традиционную файловую систему с древовидной организацией файлов. При этом ключевые слова будут отображаться в каталоги виртуальной файловой системы. А файлы будут соответствовать файлам из хранилища.

Виртуальную файловую систему можно реализовать различными вариантами. Однако создание виртуальной файловой системы с нуля и ее реализация в задачу данной работы не входит. Поэтому, хочется заметить, каким образом ее можно реализовать и соотнести с хранилищем.

Во первых, существует не мало разработок в сфере самих виртуальных файловых систем. Например, Callback File System SDK позволяет создавать виртуальные файловые системы и диски, которые позволяют управлять данными, так как если бы они были файлы на локальном диске. Однако он не предоставляется бесплатно и не является открытым, хотя считается одной из лучших наработок в этой сфере.

Во вторых, в качестве виртуальной файловой системы можно использовать локальный ftp сервер. Данное решение значительно проще в реализации, однако усложняется необходимостью настройки возможности доступа к серверу, которая, например, в Windows особенно жесткая для защиты самого пользователя. Однако поскольку такое решение проще, оно и является оптимальным.

Особенности реализации

Поиск по запросу пользователя

Во время реализации возник вопрос: как организовать поиск по запросу пользователя? Идея запроса пользователя аналогична по использованию пользователем поисковым запросам к интернет поисковикам. Однако, так как хранилище хранит не только ASCII объекты, то строить поиск аналогично по алгоритму не представляется возможным или сколь либо адекватным. Важным свойством использования пользовательского запроса, которое надо реализовать, и их удобство, что можно получить результат по содержимому совершенно разного вида и сочетания критериев в более близкой для человека форме. Например, «фотографии Кати с морем, на природе созданные с 10.06 до 31.08»

Использование грамматики

Поисковой запрос содержит в себе критерии поиска, которые надо разобрать, и максимально эффективно построить по этим критериям запрос к базе данных. Критерии поиска разделяются на виды: метки, название, категория, расширение, дата создания или модификации. Для распознавания информации в строке можно использовать нейронные сети. Однако разработка подходящей нейронной сети и обучение ее же в большинстве случае сложнее и тем самым неоправданно в рамках данной задачи. Задача состоит в том, чтобы адекватно разобрать строку на части относящиеся к одному виду. Ее можно решить и с помощью специально написанной грамматики и парсера. Для реализации этой части прототипа системы была выбрана вспомогательная открытая библиотека Irony.

С помощью Irony была написана специальная грамматика для разбиения пользовательского запроса в дерево. Правда, пока язык пользовательского запроса должен быть на английском. Хотелось создать как бы псевдо язык близкий к человеческому. Но создать псевдо язык на русском с учетом всех склонений очень сложно, английский же не содержит склонений слов.

Пример конструкций псевдо языка следующий: `<something1>` with `<something2>` или просто `<something>` , где `<something>`– список всех данных о файле по которым будет вестись поиск, записанные по правилам грамматики. Кроме with можно также использовать where и about.

Такая строка введенная пользователем передается ядру, где с помощью, отдельно написанной библиотеки для работы с пользовательскими запросами, SearchQueryAnalyser, которая работает с Irony и с грамматикой, она преобразуется в дерево, которое

анализируется и данные из него разбиваются на метки, метаданные, расширение и т.п. Так же теоретически можно считать что перед «with» скорее всего имя или категория. Так образом можно сравнить сразу данные перед with с набором категорий из базы (выбираются они с помощью хранимой функции), для упрощения поиска. Поэтому грамматика написана с учетом этой особенности, так что в первой части перед with нельзя писать метки.

Грамматики состоит из следующих терминалов:

```
Terminal WITH = Symbol("with");
Terminal ABOUT = Symbol("about");
Terminal WHERE = Symbol("where");
Terminal comma = Symbol(",");
Terminal pc = Symbol(";");
Terminal dot = Symbol(".");
Terminal ABOUT = Symbol("about");
Terminal greater = Symbol(">");
Terminal less = Symbol("<");
Terminal equal = Symbol("=");
Terminal BETWEEN = Symbol("between");
Terminal AT = Symbol("at");
```

Правила:

```
F.-> FE | SWOTg + WE;
FE.Rule -> GT + FE | GT | AdvT | AdvT + FE;
GT -> PTg | Extens;
WithT -> WITH | ABOUT | WHERE;
WE -> WithT + PTg| WithT + AdvT| WE + PTg | WE + AdvT;
SWOTg -> GT + SWOTg| GT;
AdvT -> MTgs| ME;
MTgs -> TBSep + TgS;
TgS -> LngTg + TBSep + TgS| LngTg;
Sep -> pc| comma;
LngTg -> LngTg + PTg| Tag| Eof;
TBSep -> Tag + Sep;
Ineq -> greater | less| equal;
ME -> Ineq + Date | BETWEEN + Date + Date | AT + Date;
PTg -> identifier;
```

```
Extens -> ext;
Tag -> identifier;
Date -> day + dot + mnth + dot + year | day + dot + mnth;
```

Заключение

При выполнении данной работы были получены следующие результаты:

1. Проведен обзор и анализ следующих решений:
    a) Windows Media Player;
    b) Adobe Photoshop Lightroom;
    c) семейство операционных систем Windows NT: Windows XP, Windows Vista, Windows 7;
    d) Mac OS X;
    e) WinFS – система хранения и управления данными на основе реляционной базы данных.
2. Спроектирована архитектура прототипа системы и реализованы следующие его компоненты:
    a) хранилище с базой данных (Microsoft SQL Server);
    b) ядро (разработано на языке C#);
    c) виртуальная файловая система.
3. Разработан и реализован алгоритм поиска файлов по поисковым запросам в файловом хранилище.
4. Разработан механизм поддержки и сохранения целостности и актуальности информации в хранилище при модификации файлов из сторонних приложений.
5. Создан пользовательский интерфейс для работы с файловым хранилищем.

Список литературы

[1] *Новиков Б.А., Домбровская Г.Р.* Настройка приложений баз данных. Санкт-Петербург: БХВ-Петербург, 2006, 240с.

[2] *Chen P.* The Entity-Relationship Model-Toward a Unified View of Data. // ACM Transactions on Database Systems (TODS), vol. 1, No. 1, 1976, ACM, New York, USA, p. 9-36.

[3] *Craig D.* Lisa 1 Owner Guide, Apple Computer Inc. 1983. 418 p.

[4] *Date C. J.* An Introduction to Database Systems, 8th edition. Addison-Wesley, 2003. 1024 p.

[5] *Giampaolo D. B.* Practical File System Design with the Be File System. San Francisco, California: Morgan Kaufmann Publishers, 1998. 256 p.

[6] *Stan M.* Inside the Windows 95 file system. O'Reilly, 1997. 360 p.

[7] *Williams G.* Journal Byte, BYTE Publications Inc, feb. 1983, p. 37-50.

[8] Adobe. The library module. About Lightroom catalogs, 2010. http://help.adobe.com/en_US/Lightroom/2.0/WS31C90D9B-2D4C-490f-B72F-EDD9D8DF60B6.html

[9] Codeplex. Irony Project Specification, November, 2009. http://irony.codeplex.com/

[10] LisaFAQ. What is the Apple Lisa? 2006. http://lisafaq.sunder.net/single.html#lisafaq-hs_about_lisa

[11] Macworld. Organize files with Spotlight comments, May, 2007. http://www.macworld.com/article/58012/2007/05/spotcomments.html

[12] Microsoft. Что нового в проигрывателе Windows Media 11 для Windows Vista. Все развлечения в одном месте, 2010. http://www.microsoft.com/windows/windowsmedia/ru/player/windowsvista/features.aspx#AllYourEntertainmentinOnePlace

[13] MSDN. A Developer's Perspective on WinFS: Part 1, March, 2004. http://msdn.microsoft.com/en-us/library/ms996622.aspx

[14] MSDN. Blog of WinFS project, 2006. http://blogs.msdn.com/winfs/

[15] MSDN. WinFS 101: Introducing the New Windows File System, March, 2004. http://msdn.microsoft.com/en-us/library/aa480687.aspx