Министерство образования Российской Федерации
Томский государственный университет
факультет информатики
кафедра теоретических основ информатики

К защите в ГАК допустить
Зав. кафедрой, доцент
Ю.Л. Костюк
« » июня 2002 г.

Филатьев Олег Владимирович

Проектирование и реализация инструментов администрирования территориальной информационной системы «ТИСА»

Дипломная работа

Научный руководитель:
к.т.н, доцент
_____Поляков В.И.
Исполнитель:
студент группы 1471
_____Филатьев О.В.

Томск 2002

Реферат

Отчет о дипломной работе на 60 страницах, 5 рисунков, 5 источников, 3 приложения.

Microsoft Repository, COM, ActiveX, Средний уровень информационных систем, Информационные модели.

(1) Создание инструмента настройки пользовательского интерфейса системы «ТИСА».

(2)Создание модели среднего уровня системы «ТИСА».

(3) Изучение технологии COM, изучение средств создания COM – приложений, изучение COM+, проектирование информационной модели, использование CASE – средств.

(4) Данная работа используется в проекте «ТИСА» (© СибГеоИнформатика).

Содержание:

Введение

Целью данной работы является создание инструментов для облегчения использования и придания новых возможностей системе «ТИСА».

Для дальнейшего ознакомления с работой необходимо знать минимальные сведения о системе «ТИСА». Система «ТИСА» это информационная система созданная сотрудниками НПО «СибГеоИнформатика» для информатизации муниципальных учреждений, автоматизации учета имущества, находящегося в муниципальной собственности , и контролем за его использованием.

Система представляет собой клиент - серверное приложение. В качестве Системы Управления Базами Данных используется семейство СУБД Oracle[1]. На сервере управления данными находятся базы данных системы. Схема баз достаточно большая (порядка 900 таблиц и представлений) и наукоемкая.

Клиентская часть системы «ТИСА» является отдельным от сервера приложением. Какие данные будет видеть пользователь, какие действия может выполнять, в каком виде данные будут представлены ему – все это задается декларативно. Часть системы, которая занимается переводом декларативной информации в вызовы определенных модулей, назовем средним уровнем системы или business logic level. Термин средний уровень используется в трех и более уровневых системах. Система «ТИСА» не является полностью трех уровневой системой, но имеет многие черты таких систем. Именно для настройки и управления средним уровнем были спроектированы и реализованы инструменты, описываемые в данной работе

Многие системы подобные системе «ТИСА» имеют достаточно удобные и простые средства настройки диалога с пользователем. Разработчиками системы «ТИСА» в качестве подобного средства использовался продукт корпорации Oracle - Oracle Designer 2000. Designer использовался также для создания и хранения схемы БД, и является сложным и громоздким CASE – средством. Этот инструмент не приспособлен непосредственно для описания диалога с пользователем и использовался немного не для тех целей. Отсюда следовали ограничения – трудности по созданию и поддержке разных

конфигураций пользовательского интерфейса и невозможность пользователями системы изменять ее конфигурацию из-за сложности инструмента используемого для настройки.

Инструменты, о которых пойдет речь в данной работе, позволяют упростить работу проектировщиков и администраторов системы.

Также появилась возможность поддерживать несколько конфигураций диалога с пользователем. Конфигурация создается средством настройки и сохраняется в специфичной БД с помощью Microsoft Repository[2]. Средство установки берет данные из этой БД и устанавливает только необходимые компоненты.

Схема БД, где хранятся данные о конфигурации, создается с помощью Microsoft Visual Modeler поставляемым вместе с Visual Basic Enterprise Edition. Затем с помощью Microsoft Model Development Kit схема переносится в репозиторий. Схема создается не в виде таблиц и ограничений, а в виде интерфейсов (аналогов COM интерфейсов), которые могут наследоваться и связываться друг с другом с помощью связей (relationships) и иметь свои свойства (Property).

Инструменты созданы по технологии ActiveX Document[3], как и часть клиента системы «ТИСА». Использование такой технологии оправдано возможностью подключения данных инструментов к любому ActiveX контейнеру.

Для использования инструментов было создано приложение – контейнер в стиле Windows Explorer, необходимое для объединения управляющих инструментов в одно приложение. К нему достаточно легко добавить новые инструменты, созданные по технологии ActiveX Document.

Для последовательности изложения главы отчет расположены в следующем порядке:

1. Краткое описание системы «ТИСА». Такое описание необходимо для ознакомления с технологией, используемой при создании системы, для которой создаются инструменты, о которых идет речь в данной работе. Описание взято из документации по системе «ТИСА».

2. Модель среднего уровня системы «ТИСА».

3. Оболочка-контейнер для инструментов администрирования.

4. Средство настройки диалога с пользователем.

Глава 1. Краткое описание системы «ТИСА»

(Данная глава взята из документации к системе «ТИСА», более полно смотри [4])

1.1 Место системы «ТИСА» в классификации ИС

Предваряя освещение особенностей проекта системы «ТИСА», охарактеризуем ту "экологическую нишу", которую занимает система в общей классификации ИС.

Прежде всего, определимся с моделью хранения данных создаваемого информационного пространства. Характерными особенностями работы с ним являются:

колоссальные объемы хранимых данных;

сложная семантика предметной области (разнообразие типов объектов и отношений между ними);

многопользовательский, часто удаленный доступ;

критичность ко времени отклика на запрос пользователя;

устойчивость к программно-аппаратным сбоям;

гарантия отсутствия несанкционированного доступа;

наличие богатых возможностей верификации данных.

Наилучшим образом удовлетворяют всему комплексу требований системы управления базами данных (СУБД). Среди многочисленных моделей БД реальных альтернатив две: реляционная и объектно-ориентированная. На стороне первой – основательный теоретический базис, многолетняя история развития, богатый выбор мощных, эффективных программных продуктов. У второй – впечатляющие перспективы и масса нерешенных проблем. Поэтому неудивителен выбор в качестве модели хранения данных реляционной БД.

Единое интегрированное информационное пространство предполагает управление им одним программным продуктом – сервером БД. Это не исключает возможного территориального распределения его по узлам сети, в том числе с

избыточным дублированием данных. Но делаться это должно самой СУБД в строгом соответствии со специально декларированными правилами. Только такой подход гарантирует целостность состояния БД и исключает колоссальные проблемы, связанные с синхронизацией модификаций. Таким образом мы приходим к идее реализации интегрированной системы с клиент-серверной архитектурой.

Важным критерием, по которому различаются системы БД, является то, предназначена ли она для создания и ведения БД или только для ее использования . ИС первого класса принято называть OLTP (On-Line Transaction Processing)-системами. Основная задача этих систем – поддержание актуального целостного состояния БД путем выполнения транзакций на включение, модификацию и удаление данных. Непременным атрибутом транзакций является верификация состояния БД, включающая проверки ограничений целостности, которые декларированы в схеме БД.

Ко второму классу относятся многочисленные виды аналитических ИС – OLAP (On-Line Analytical Processing)-системы, системы поддержки принятия решений, ИС руководства и системы интеллектуального анализа данных. Отличительной особенностью таких систем является то, что они не меняют состояния БД, а используют ее только в режиме чтения. Основное их назначение – извлечение информации из БД и предъявление ее пользователю в виде, максимально удобном для принятия решения.

Поскольку главной задачей системы “ТИСА” является формирование и поддержание в актуальном состоянии общегородской БД, она в первую очередь является OLTP-системой. Но, прекрасно осознавая заинтересованность пользователей в средствах анализа, разработчики предусмотрели в системе ряд возможностей, позволяющих решать большую часть аналитических задач предметной области. К ним относятся:

предопределенные отчеты;

гибкие средства определения новых выходных документов;

простые и вместе с тем мощные средства произвольных запросов.

Это дает право позиционировать систему “ТИСА” как OLTP-систему с элементами анализа.

Основу информационной модели (ИМ) пользователя составляет модель "Сущность-Связь" (Entity-Relationship (ER)-модель)[6](рис.4).

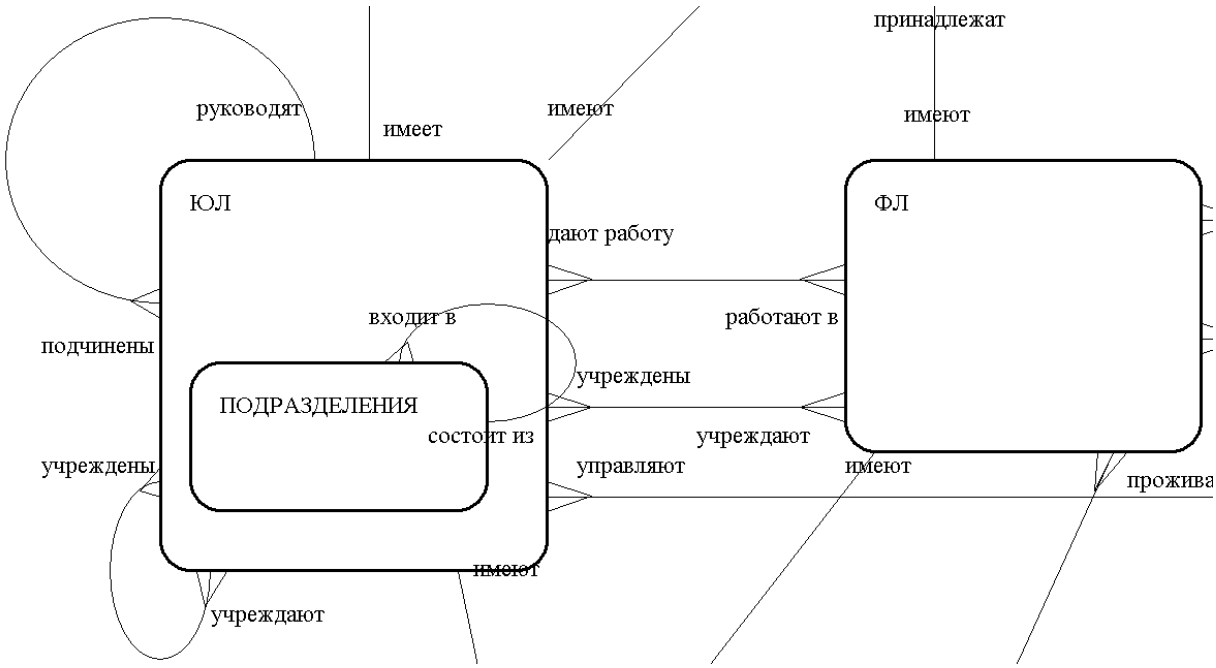

Рис.1. Фрагмент информационной модели пользователя

Она настолько проста и естественна, что неудивительно ее применение как профессионалами (на этапе анализа и проектирования ИС), так и неискушенными в информатике пользователями (при работе с ИС). Специфика использования сказывается лишь на характере получаемого результата.

Проектировщики с ее помощью создают детальную схему БД, в то время как ИМ пользователя оперирует более агрегированными объектами. Так множество сущностей “физические лица” в ИМ пользователя декомпозируется в десяток более мелких множеств сущностей ИМ БД. Аналогичные отличия существуют и на уровне множеств связей.

Дополнительным элементом ИМ пользователя является иерархия обобщения множеств сущностей. Каждый узел дерева обобщения связан со своим родителем отношением “есть некоторый”. Так множество сущностей “участки” является частным случаем множества сущностей “недвижимость”, последний в свою очередь есть некоторый “объект права”. На верхний уровень

иерархии вынесены самые общие “абстрактные” множества сущностей “субъекты права”, “объекты права”, “права” и т.д.

Основное назначение иерархии множеств сущностей – облегчение поиска необходимого исходного типа объектов. При наличии в ИМ пользователя более 300 множеств сущностей это непростая задача.

Поскольку конкретные сущности связаны со своим множеством также отношением “есть некоторый” (Иванов есть некоторое физическое лицо), они также находят свое место в дереве обобщений, и родителем для них является узел соответствующего множества сущностей. Таким образом, чтобы создать, изменить, удалить или просто посмотреть на описание конкретной сущности (физического лица, участка и т.д.) надо знать множество сущностей, экземпляром которого она является.

Но носителями информации являются не только сущности, эту роль выполняют и связи. Так можно работать с сущностью типа “права” и с сущностью типа “правоизменяющие документы”, но так и не получить ответа на вопрос, какие права устанавливает или аннулирует конкретный договор купли-продажи. Эту информацию обеспечивают как раз связи между сущностями указанных типов. Выбрав требуемый экземпляр правоизменяющего документа, необходимо пройти по его связям типа “устанавливает” к сущностям типа “права” и таким образом селектировать те из них, которые устанавливаются данным договором. Теперь их можно анализировать как сами по себе, так и далее по их связям с объектами права, субъектами права и т.д. Таким образом реализация любой информационной задачи в этой модели выливается в навигацию по множествам сущностей и связей.

Диалоговые формы являются инструментом пользователя для навигации по ИМ системы, и поэтому они используют те же основные понятия – множества сущностей и связей. Основное окно системы разделено на две части – слева представлено дерево множеств сущностей, отражающее ИМ пользователя, правая часть предназначена для показа множества сущностей в виде списка, либо одной сущности в развернутом виде (рис.2).

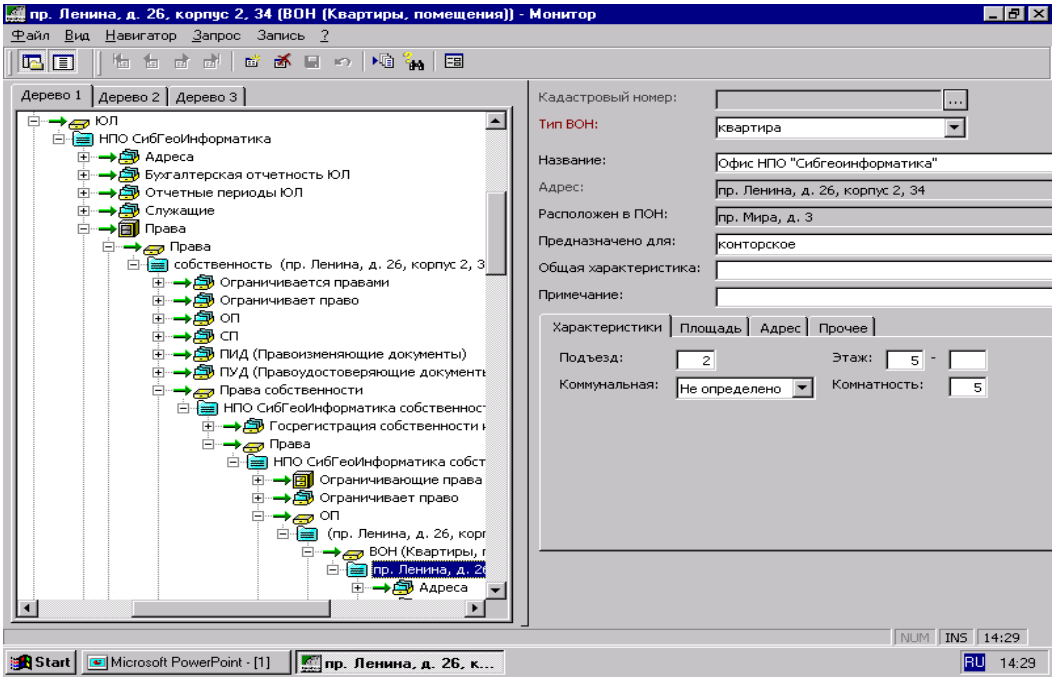

Рис.2. Основное окно системы

На начальной стадии диалога пользователь опускается до нужного ему множества сущностей (например, юридических лиц), раскрывая свернутые ветви дерева. Далее он может попросить систему показать все экземпляры этого множества или задать условие, по которому произойдет фильтрация сущностей. И в том, и в другом случае в правом окне ему будет предъявлен список юридических лиц. Выбрав одну из сущностей, пользователь в этой же части окна получит о ней полную информацию.

При этом в левом окне появится возможность осуществить переход по связям этой сущности с сущностями других типов. Предположим, что нам необходимо просмотреть все права, которыми обладает данное юридическое лицо. Мы выбираем связь “имеет ПРАВА”. Система автоматически переходит к множеству сущностей “права”, предлагая в правом окне список всех прав выбранного юридического лица. Далее, указав конкретное право и осуществив переход по связи “распространяется на ОБЪЕКТЫ ПРАВА”, мы можем работать с тем объектом права (участком, зданием и т.д.), которое относится к этому праву.

По мере необходимости с экземплярами множеств сущностей можно выполнять следующие действия – создавать новые, изменять или удалять существующие.

Следует отметить, что вал информации не будет обрушиваться на пользователя без необходимости. Можно выделить очень узкие информационные пространства для каждой задачи, каждого исполнителя. Естественно, что при этом и диалог будет свернут до манипулирования только теми сущностями и связями, которые необходимы. Неиспользуемые части дерева и переходы будут просто невидимыми. Но в любой момент информационные потребности задачи могут быть расширены без особого труда.

Такую гибкость инструмента способны обеспечить предлагаемый проект ИМ и способов ведения диалогов с пользователем. Проигрывая в специализации, мы выигрываем в комплексности и гибкости. К тому же администратору системы легче поддерживать в работоспособном состоянии единообразно спроектированные подсистемы различных организаций и служб.

Полезные свойства ИМ и диалога, на которых мы акцентировали внимание ранее, - это только видимая “надводная часть айсберга”. Гибкость, масштабируемость – генеральная стратегия системы “ТИСА”. Естественно, что она нашла свое отражение в выборе базовых средств разработки и в архитектуре системы.

Ключевым решением, повышающим гибкость системы, является использование максимально обобщенных программных модулей. Вся специфика конкретного использования системы определяется декларативным образом и хранится в специальных таблицах многочисленных настроек.

На этапе внедрения системы описания ИМ подсистем, задач, пользователей с детальным определением прав на действия с БД каждого участника информационных процессов заносятся в эти таблицы. В них же определяются глобальные параметры конкретной установки системы.

Программным ядром системы является менеджер диалога с пользователем. При запуске системы конкретным пользователем осуществляется его идентификация указанием имени и пароля. По этой информации менеджер определяет полномочия пользователя, его ИМ и форму дерева множеств сущностей. Далее менеджер осуществляет стандартный диалог по принципам,

описанным в предыдущем параграфе. По мере необходимости им вызываются специализированные модули, отображающие конкретные множества сущностей и их экземпляры. Причем их связь с узлами дерева также декларируется в системных таблицах, а не в коде менеджера. Что обеспечивает их простую замену в случае развития системы.

Глава 2. Модель среднего уровня

2.1 Использование модели

Модель среднего уровня, которая будет здесь описана, пока не используется на этапе непосредственно работы системы. Модель используется на этапе настройки системы. Структуры, полученные на основе этой модели, используются для хранения описания системы, из хранилища информация переносится в специализированную схему БД на сервере данных.

Клиент системы «ТИСА» работает, используя информацию из этой схемы.

Технология, которая используется при построении модели, позволяет достаточно легко и без изменений в самой модели создать полноценную трех уровневую систему, гибкую и масштабируемую. Клиент будет получать описание пользовательского интерфейса независимо от сервера данных. Появится возможность сегментировать систему, где одни и те же данные будут использоваться клиентами с различным пользовательским интерфейсом.

Такая схема взаимодействия облегчает перенос системы на другое (отличное от СУБД Oracle) хранилище данных.

2.2 Технологии и инструменты, используемые при построении модели

Основным инструментом хранения модели и описания является Microsoft Repository (сокращенно MSR) [2]. Основными выдающимися свойствами MR являются:

Возможность определять и хранить в одном месте и модель, т.е. метаданные и описание, построенное по этой модели, т.е. непосредственно данные. В инструменте, который использовался ранее для аналогичных целей (Oracle Designer 2000), возможность определения модели была ограничена. В MSR модель м.б. ограничено только Repository Type Information Model (RTIM).

RTIM накладывает не слишком сильные ограничения и с ее помощью можно смоделировать совершенно разные системы различной сложности.

RTIM представляет модель пользователя в виде COM интерфейсов, COM объектов и связей (relationship) между ними. Один из путей использования MSR является использование его хранилища для сохранения состояния объектов, и использование описаний классов и интерфейсов для создания COM объектов стандартными средствами COM встроенными в MS Windows 9x и NT/2000.

MSR хранит структуру модели в виде реляционной схемы и свойства объекта являются атрибутами сущностей. СУБД для этой схемы является система MS Jet поставляемая как обязательная часть операционной системы Windows.

Также MSR легковесен, если сравнивать с другими подобными системами. Из-за того , что в него не встроена какая-либо другая модель, кроме RTIM, MSR является достаточно универсальным и гибким инструментом.

Для построения модели использовался еще один продукт Microsoft – MS Visual Modeler, поставляемый вместе с MS Visual Basic Enterprise Edition. Для переноса модели созданной в MS Visual Modeler в MSR использовались инструменты из MS Model Development Kit.

Visual Modeler использует язык UML для хранения пользовательских моделей . Пользователю предоставляется возможность определить модель в виде интерфейсов, классов, связей, компонент и перечислений. Модель представляется в виде диаграмм, у классов и интерфейсов задаются свойства и связи. У связей задаются роли, которые играют объекты в связи. Visual Modeler имеет в своей основе иную модель, чем модель MSR. Для переноса модели используется инструменты из MDK, в частности инструмент Check Model, который можно вызвать в главном меню Visual Modeler, в пункте меню Tools.

Этот инструмент проверяет модель на соответствие RTIM и при желании пользователя переносит модель в MSR.

Также в MDK есть инструмент для создания type library в виде файла с расширением .tlb и создания заголовочных файлов с константами различных идентификаторов для Visual Basic и C++.

Для просмотра содержимого MSR используется Repository Browser поставляемый в составе Repository SDK.

Как видно, существует полный набор необходимых инструментов для работы с репозиторием.

Репозиторий является средством хранения моделей и данных и не имеет средств редактирования данных. Для редактирования разработчик приложения, которое использует репозиторий, должен создать интерфейс для работы с пользователем и последующим переносом данных в репозиторий.

Ядро репозитория выполнено в виде COM объектов, с помощью которых можно модифицировать как данные, так и модель. Т.к. ядро выполнено в виде COM объектов, то оно легко и непринужденно подключается к Visual Basic и Visual C++, и к любой среде программирования, которое использует COM объекты.

MSR является интересным, эффективным и конкурентно-способным инструментом. Его поддерживают около 60 фирм разработчиков ПО. В репозитории реализована функциональность, которая не используется в наших инструментах, но которая представляет интерес для разработчика – это версионность, рабочие пространства и т.д. Также реализована транзакционность и возможность откатов. Этот инструмент позволяет упростить разработку среднего уровня приложения и облегчает непосредственно работу с ним.

Работать непосредственно с репозиторием достаточно легко, но было решено создать дополнительную прослойку между пользовательским интерфейсом и репозиторием. В роли прослойки служит набор объектов и коллекций, представляющие в реальном виде, в виде COM компонент, объекты репозитория являющиеся абстракцией и интерфейсом для доступа к хранилищу атрибутов. Дополнительная прослойка нужна для ускорения работы, так как информация в процессе выполнения находится в памяти, а не в хранилище на жестком диске, также с COM объектами проще работать при реализации приложений на основе среднего уровня: нет необходимости детально знать, как программировать с использованием репозитория, с COM объектами передается информация о типах, методах и свойствах, что помогает при программировании.

2.3 Модель среднего уровня системы «ТИСА»

Управляющая информация в системе «ТИСА» хранится в реляционном виде на сервере данных. Реляционную схему необходимо было расширить и переработать в RTIM, которая является некоторым расширением объектной и COM модели. В результате вместо 7 типов сущностей в реляционной модели (RM) в RTIM было определено 18 интерфейсов, 16 классов и множество различных связей. Такое большое количество различных типов позволяет фактически исключить пустующие значения каких-либо атрибутов сущностей необходимых для хранения управляющей информации. Такая проблема была при представлении схемы в RM. Также убраны различные поля обозначающие различные типы сущностей в RM. В RTIM создается новый тип. Наличие связей и ролей между классами вносят элемент самодокументируемости и облегчают понимание схемы.

В RTIM есть возможность наследования между интерфейсами. Это наследование определений, а не реализаций, такое же, как и в COM. Свойства объектов определяются через интерфейсы, объекты могут реализовывать несколько интерфейсов, связи могут устанавливаться только между интерфейсами. Основная семантика описывается интерфейсами и связями. На них мы остановимся поподробнее.

Основным интерфейсом является интерфейс ITreeNode. ITreeNode - это абстрактный интерфейс, он не имеет непосредственной реализации в виде класса. Этот интерфейс представляет собой узел в дереве навигации. У интерфейса ITreeNode имеются свойства DisplayName, где хранится отображаемое имя модуля, и свойство DataBaseId , где хранится идентификатор соответствующей записи в системных таблицах системы «ТИСА», такое свойство есть еще у нескольких интерфейсов. ITreeNode связан с интерфейсом INodeLink через связи NodeIncludeOther и NodeIncludedFromOther. Интерфейс INodeLink нужен для хранения свойств связей между сущностями предметной области. Можно было бы просто сделать связь, но у связей между сущностями есть свойства, а у связей в RTIM нет возможности определять свойства.

В INodeLink определено свойство DisplayName в котором задается отображаемый заголовок в дереве для узла в который мы переходим по связи. Свойство RestrictivWhere хранит часть SQL запроса, в котором определяется,

какие данные надо показывать после перехода по связи. Свойство UsageSequence указывает, в каком порядке необходимо отображать связи сущности в дереве.

От интерфейса ITreeNode наследуются три интерфейса:

ITreeFolder, ITool, IScreenView. Рассмотрим эти интерфейсы.

ItreeFolder:

Интерфейс, описывающий узел, который не представляет собой сущность и является контейнером для других узлов. Имеет свойство IsNavigator, означающее то, что данный узел является навигатором (см. Главу 1.) и свойство FixedColumn текстового типа по которому отбираются связанные сущности.

ITool:

Этот интерфейс описывает узлы, представляющие не сущность предметной области , а различные формы - инструменты необходимые для выполнения каких-либо специфических операций над данными, для настройки различных частей системы, для генерации отчетов и т.д. Интерфейс Itool связан с IdialogLink – аналог InodeLink, разница в том, что связывает узел дерева не с другим узлом дерева, а с IdilalogView представляющем отображение информации в виде окон диалога. ITool Связан с интерфейсом IExecuted связью Implies. COM и , соответственно, RTIM не позволяют множественное наследование интерфейсов (в отличии от CORBA) и связь Implies говорит о том, что необходимо, чтобы с каждым объектом реализующим ITool был обязательно связан объект реализующий IExecuted.

У интерфейса IExecuted есть единственное свойство ProgID здесь сохраняется идентификатор исполняемого объекта, который будет создаваться при активизации узла .

IDialogView:

В интерфейсе IDialogView не имеет свойств, крома DataBaseId.

От IDialogView наследуется два других интерфейса - IFormView и IQueryDialog.

IFormView описывает форму для представления данных. Не имеет собственных свойств.

IQueryDialog представляет форму –диалог запроса для выборки данных по критерию. Имеет свойство SQLAdapter – начало SQL запроса до слова where включительно, остальное дописывается во время исполнения в зависимости от желания пользователя.

IQueryDialog и IFormView связаны связью Implies с интерфейсом IExecuted, описанным выше.

IScreenView является интерфейсом, описывающим окно-таблицу. Основная масса описаний и, соответственно, большая часть модели связана с IScreenView. IScreenView имеет следующие свойства:

AllowInsert – типа Boolean, при значении True разрешается вставлять новые записи;

AllowDelete – типа Boolean, при значении True разрешается удалять записи;

AllowUpdate – типа Boolean, при значении True разрешается обновлять записи;

SqlWhere – типа String, содержит ограничение на выборку из связанной таблицы.

SqlFrom – типа String, содержит либо имя таблицы откуда берутся данные или содержит запрос откуда берутся данные.

IScreenView связан с IDBColumnSet связью SVGetValuesFrom определяющем из какой таблицы берутся значения для отображения пользователю, и связью SVHasBaseTable ,которая связывает с таблицей в которую новые значения или записи вставляются. Также ISreenView имеет связь ConsistOf с интерфейсом IColumn представляющим столбец в окне-таблице. Кроме этого IscreenView связанна с IDBColumn через связи SVHasPK и SVHasDescrColumn первая связь указывает на столбец - идентификатор записи из базовой таблицы , откуда берутся и куда записываются значения. А второе свойство показывает столбец откуда будет браться текстовая информация для отображения в дереве навигации.

IColumn связан связью GetValuesFrom с IDBColumn, представляющим столбец таблицы в базе данных. Следующие свойства определены в IColumn:

Alias – здесь хранится алиасное имя столбца в SQL запросе.

DisplayName – отображаемое имя столбца. Именно оно будет в заголовке в таблице-окне.

UsageSequence – место столбца в таблице.

DefaultValue – значение по умолчанию (полезно для выпадающих списков и т.д.)

DisplayLength – указывает ширину столбца в таблице-окне.

От IColumn наследуют три интерфейса: ItextColumn, IcontrolColumn, IlookupColumn.

ItextColumn представляет текстовый столбец. Может содержать текстовую информацию, даты и числовые значения. О том, какой тип информацию содержит столбец, говорит свойство Type.

IcontrolColumn представляет встраиваемый в окно-таблицу элемент управления. Для каких-то полей необходима дополнительная обработка данных вводимых пользователем. Для таких полей как Адрес, Кадастровый номер и некоторых других необходимо вначале создать сущность в других таблицах и , затем, связать с текущей сущностью. Такую работу выполняют созданные группой разработчиков системы «ТИСА» элементы управления встраиваемые в окно-таблицу. Интерфейс не имеет свойств. Связан с IExecuted связью Implies.

ILookupColumn представляет столбец реализующий связи между таблицами типа 1 к 1 и 1 к М. Столбец отображается в виде выпадающего списка. В таблице хранится только идентификатор связанной сущности, а в окне-таблице показывается более дружественная пользователю информация из связанной таблицы. Свойство Type указывает способ получения данных. В системе “ТИСА” существует два способа получения данных из связанных таблиц: напрямую из таблицы и через карман. Напрямую из таблицы берутся все данные из классификаторов, где число сущностей ограниченно. При связях с таблицами с большим количеством сущностей все данные получать неэффективно и для такой ситуации был создан карман, куда пользователем помещаются необходимые ему сущности. И уже из кармана данные попадают в выпадающий список представляющий связь. Свойство LookupInfo содержит

управляющую информацию, например запрос по которому выбирают данные из связанной таблицы или название столбца, который необходимо выбрать из кармана.

IDBColumnSet описывает таблицу-источник данных .Все интерфейсы с префиксом IDB представляет собой сущности реляционной модели. IDBColumnSet имеет свойство Cashed – сохранять в памяти данные для более быстрого доступа.

Рассмотрим часть модели связанную с IDBColumnSet. Вначале рассмотрим все интерфейсы связанные с IDBColumnSet , а затем связи между ними.

С IDBColumnSet связан интерфейс IDBColumn представляющий столбец в базе данных. IDBColumn имеет свойства ColumnType, определяющее тип столбца (number, varchar2 и другие типы определенные в СУБД Oracle), и свойство Length для хранения длинны столбца, если тип имеет длинну.

Между IDBColumnSet и IDBColumn имеется связь.Связь называется ColumnSetConsistOfColumn. Связь устанавливает из каких столбцов состоит таблица. Связь типа агрегации, т.е. один элемент является составной частью другого.

Также между IDBColumnSet есть связи ColumnSetHasDescription и ColumnSetHasPK, через первую связь обозначается, какой столбец таблицы берется для описания строки. В технологии создания БД для системы «ТИСА» принято осуществлять доступ к данным через представления, а не напрямую к таблицам и в каждом представлении создается агрегирующий столбец, в котором собирается основная информация о сущности предметной области. И такой агрегирующий столбец является описанием. Вторая связь обозначает первичный ключ таблицы.

INodeLink и IDBColumn связаны через FixedColumnForLink. Данная связь определяет из какого столбца нужно брать значения(как правило это ID) для отображения в дереве навигации только сущностей, связанных с первоначальной сущностью. В дополнение к этой связи в интерфейсе INodeLink присутствует свойство RestrictivWhere , с помощью которого накладываются дополнительные ограничения.

Мы закончили рассмотрение модели в виде интерфейсов. Рассмотрим модель классов.

Модель классов достаточно простая – каждый класс, за исключением абстрактных (IdialogView, ITreeNode, IColumn), реализуют по одному интерфейсу и называются также как интерфейсы только без префикса “I”. Реализация в Visual Modeler обозначается связью типа Dependency и называется Refines.

Также каждый класс реализует интерфейс INamedObject, который имеет только одно свойство Name. Данный интерфейс необходим для поиска объекта по имени.

Для доступа к объектам в репозитории существует класс ReposRoot. Но для доступа к объектам репозитория необходимо, что бы класс ReposRoot реализовывал интерфейс, который определяет связи с другими объектами модели. В нашей модели такую роль играет IRootInterface, имеющий связи с интерфейсами с которых начинается навигация по репозиторию (ITreeFolder,IScreenView,ITool,IFormView, IQueryDialog,IDBColumnSet).

2.4 API для доступа к репозиторию

Общая схема классов и связей между ними отличается от схемы хранения информации в репозитории. Такие отличия связаны прежде всего с тем, что VB не поддерживает полноценного наследования. В VB имеется наследование определений, но нет наследования реализации. По этой причине пришлось отказаться от многоуровневого наследования и применить возможность множественного наследования.

В VB можно делать реализацию в классах предках и есть возможность описывать абстрактные классы (по теминолгии COM - интерфейсы). В данном случае был выбран способ реализации через абстрактные классы, потому что в некоторых случаях необходимо знать объект какого класса обрабатывает информацию методы для работы, с которой описаны в классе – предке. Соответственно, нужно или включать проверки в методы на принадлежность к какому-то классу-потомку (что дает потери времени при выполнении) или

реализовывать абстрактные метода в классах-потомках (что создает дублирование кода), более логично было пойти по пути дублирования кода.

В VB факт наследования от какого-либо класса описывается выражением Implements <имя класса>, при этом при реализации какого-либо метода класса-предка необходимо указать префикс – название класса предка. Обратится к методам класса – предка можно через присвоение ссылки на класс – потомок , ссылке на класс предок. Пример:

Определяем ссылку на класс-предок

Dim Node As TreeNode

Создаем объект класса-потомка

Dim SV As New ScreenView

Присвоение ссылки

Set Node = SV

Использование члена класса – предка

Node.Name = “Пример”

Без присвоения ссылки невозможно обратится к члену класса-предка.К сожалению,в VB нет приведения типов без дополнительных переменных.

В VB, как и во многих других современных языках программирования, есть такое понятие как свойство (Property). Свойство это упрощение для пользователя, свойству можно присваивать значение и просматривать значение. Работа со свойствами идет как с переменными, но при присвоении или просмотре свойства есть возможность выполнить код, то есть свойство разбивается на два метода типа Get и Let (для объектов - Set) При присвоении значения свойству вызывается метод типа Let и в нем можно реализовать обработку нового значения, соответственно, и при получении значения вызывается метод типа Get.

Далее подробно рассмотрим реализацию нескольких классов, которые дадут достаточное представление о способах реализации API. Остальные классы будут описаны менее подробно.

Класс ScreenView.

Данный класс реализует обработку информации об окне-диалоге.

Класс ScreenView наследуется от класса TreeNode. Как было отмечено выше, реализация методов класса-предка находится в классе – потомке. Соответственно

все методы определенные в TreeNode реализованы в ScreenView и будут рассмотрены в этом разделе.

Рассмотрим, как объект общается с репозиторием.

Для работы с репозиторием необходимо иметь доступ к объекту типа RepositoryObject, являющимся постоянным хранилищем информации. В случае ScreenView, объект репозитория должен реализовывать интерфейс IscreenView. При существующем объекте репозитория свойству класса ScreenView RepObject присваивается этот объект, при создании нового объекта вызывается метод Create,которому в качестве параметра передается ссылка на репозиторий.

При работе со свойствами можно по-разному реализовать получение значения свойства, можно получать значение при обращении к свойству, то есть обращаться к хранилищу каждый раз – но это очень накладно, можно при первом обращении реализовывать инициализацию переменной в памяти, а потом брать из переменной, но при этом при обращении к свойству необходима проверка на инициализированность переменной. Можно сосредоточить инициализацию всех переменных в одном месте, но для этого нужно отдельная функция. В случае, рассматриваемом здесь каждому объекту необходимо присвоить свойство RepObject , при обработке этого свойства происходит инициализация переменных. При обращении к свойствам для получения значения берется значение из переменной в памяти, при присвоении значения значение записывается в переменную в памяти и в репозиторий.

Рассмотрим код соответствующий описанному выше:

```
Public Property Set RepObject(RO As RepositoryObject)
On Error GoTo RepObject_Err
Сохраняем значение объекта репозитория с которым будем работать.
Set mRO = RO
Инициализируем значения свойств.
mName = mRO.Name
mDisplayName = NvlStr(mRO("ITreeNode").DisplayName)
mAllowDelete = NvlBool(mRO("IScreenView").AllowDelete)
mSQL_Where = NvlStr(mRO("IScreenView").SQL_Where)
mAllowUpdate = NvlBool(mRO("IScreenView").AllowUpdate)
```

```
mAllowInsert = NvlBool(mRO("IScreenView").AllowInsert)
```

В репозитории все связи даже связи один-к-одному представлены в виде коллекций:

```
If RO("IScreenView").GetValuesFrom.Count > 0 Then
If mGetValuesFrom Is Nothing Then
Set mGetValuesFrom = New BaseTable
End If
Set mGetValuesFrom.RepObject = RO("IScreenView").GetValuesFrom.Item(1)
End If

Dim ItRO As RepositoryObject
Dim ControlColumn As ControlColumn
Dim TextColumn As TextColumn
Dim LookupColumn As LookupColumn
```

Здесь инициализируется коллекция столбцов. Инициализация осложнена тем, что столбцы могут быть разных типов, и необходимо их различать и по разному обрабатывать.

```
For Each ItRO In RO("IScreenView").ConsistOf

If SupportInterface(ItRO, "ITextColumn") Then
Set TextColumn = New TextColumn
Set TextColumn.RepObject = ItRO
ConsistOf.Add TextColumn

End If

If SupportInterface(ItRO, "IControlColumn") Then
Set ControlColumn = New ControlColumn
Set ControlColumn.RepObject = ItRO
ConsistOf.Add ControlColumn
End If

If SupportInterface(ItRO, "ILookupColumn") Then
```

```
Set LookupColumn = New LookupColumn
Set LookupColumn.RepObject = ItRO
ConsistOf.Add LookupColumn
End If

Next
```

Здесь инициализируются коллекции входящих и выходящих связей окна-диалога.

```
Dim theNodeLink As NodeLink
For Each ItRO In RO("ITreeNode").Including
Set theNodeLink = New NodeLink
Set theNodeLink.RepObject = ItRO
mIncluding.Add theNodeLink
Next

For Each ItRO In RO("ITreeNode").Included
Set theNodeLink = New NodeLink
Set theNodeLink.RepObject = ItRO
mIncluded.Add theNodeLink
Next
Exit Property

RepObject_Err:
```

Обработка ошибок, вызывается функция отображающая информацию об ошибке

```
ErrorMsg "MidleTier", "ScreenView/RepObject"
```

Ошибка передается выше, в вызвавшую процедуру.

```
Err.Raise Err.Number, Err.Source, Err.Description
End Property
```

Ниже приведен пример обработки простого свойства.

При получении значения берем переменную из памяти.

```
Public Property Get SQL_Where() As String
SQL_Where = mSQL_Where
```

```
End Property
```

При записи сохраняем новое значение в репозитории

```
Public Property Let SQL_Where(ByVal Value As String)
mSQL_Where = Value
mRO("IScreenView").SQL_Where = Value
End Property
```

Рассмотрим, как происходит обработка более сложных свойств, представляющие связи.

Связи могут быть представлены двумя способами – как ссылка на объект, или как коллекция ссылок.

Рассмотрим случай, когда связь 1 к 1 или 1 к М и на уровне VB является ссылкой на один объект.

При получении значения получаем ссылку на уже существующий объект в памяти

```
Public Property Get GetValuesFrom() As BaseTable
Set GetValuesFrom = mGetValuesFrom
End Property
```

При установке значения свойства необходимо сохранить изменения в репозитории

```
Public Property Set GetValuesFrom(Value As BaseTable)
Set mGetValuesFrom = Value
```

Вначале удаляем существующие ссылки. Как было описано выше, в репозитории любая связь представлена в виде коллекции, и , соответственно, работать приходится как с коллекцией.

```
Do While mRO("IScreenView").GetValuesFrom.Count > 0
        mRO("IScreenView").GetValuesFrom.Remove
        mRO("IScreenView").GetValuesFrom.Count
Loop
```

Добавляем в коллекцию новую ссылку.

```
mRO("IScreenView").GetValuesFrom.Add mGetValuesFrom.RepObject
End Property
```

Теперь рассмотрим более сложный случай связи М к 1, когда и на уровне VB и на уровне репозитория связь представлена в виде коллекции. В данном случае возникает проблема, состоящая в том, что необходимо знать объект репозитория, к которому добавляется связь, и если реализовывать добавление на уровне объекта-коллекции, то нужно передавать ссылку на объект репозитория и при этом в объекте-коллекции нужно проверять тип объекта репозитория и в зависимости от типа добавлять новую связь в коллекцию связей. Описанная выше операция сложна и ограничивает гибкость применения коллекции. Для преодоления проблемы добавление новой связи на уровне объекта репозитория, связь добавляется в специальном методе, который назевается Add <Имя коллекции>.

Пример добавления к коллекции связей.

В примере добавляются объекты абстрактного типа Column, для добавления реальных объектов репозитория необходимо привести ссылки на тип Column в ссылки на объекты имеющие реализацию.

```
Public Sub AddColumn(Item As Column)
On Error Resume Next
Dim TextColumn As TextColumn
```

Здесь для определения типа используется метод присвоения, при присвоении происходит приведение типов и если объект реализует такой интерфейс, то ссылка получает значение, иначе ссылка остается пустой.

```
Set TextColumn = Item
If Not TextColumn Is Nothing Then
mRO("IScreenView").ConsistOf.Add TextColumn.RepObject
mConsistOf.Add TextColumn
Exit Sub
End If

Dim LookupColumn As LookupColumn
Set LookupColumn = Item
If Not LookupColumn Is Nothing Then
mRO("IScreenView").ConsistOf.Add LookupColumn.RepObject
mConsistOf.Add LookupColumn
```

```
Exit Sub
End If

Dim ControlColumn As ControlColumn
Set ControlColumn = Item
If Not ControlColumn Is Nothing Then
mRO("IScreenView").ConsistOf.Add ControlColumn.RepObject
mConsistOf.Add ControlColumn
Exit Sub
End If
End Sub
```

Выше были рассмотрена реализация всех существующих в схеме репозитория типов связей и свойств. Обобщив, можно сказать, что в данной реализации применяется подход двойной инициализации объекта и полная инициализация свойств, перемещение данных с постоянного носителя в память происходит при втором шаге инициализации (присвоение свойства RepObject). Только в одном случае данное правило нарушено – это инициализация объектов отвечающих за связи между модулями, объектов типа NodeLink и DialogLink. Основными связями этих объектов являются связи с модулями, и для инициализации связей необходимо, что бы объекты представляющие модуля уже существовали в памяти. Создание и инициализация объектов типа NodeLink и DialogLink происходит при инициализации соответствующего объекта-модуля, при этом не все модуля имеют представление в памяти в виде объектов, соответственно, некоторые ссылки невозможно установить. Во избежание этого инициализация связей объектов типа NodeLink и DialogLink с объектами-модулями происходит во время первого обращения к соответствующим связям. Пример:

```
Private mIncluded As TreeNode
Public Property Get Including() As TreeNode
Dim tRO As RepositoryObject
Проверка на не инициаллизированность.
If mIncluding Is Nothing And mRO("INodeLink").Including.Count > 0 Then
```

```
Set tRO = mRO("INodeLink").Including(1)
If SupportInterface(tRO, "IScreenView") Then
Set mIncluding = m_SVCol(tRO.Name)
End If

If SupportInterface(tRO, "ITreeFolder") Then
Set mIncluding = m_TreeFolderCol(tRO.Name)
End If

If SupportInterface(tRO, "ITool") Then
Set mIncluding = m_ToolCol(tRO.Name)
End If
End If
Set Including = mIncluding
End Property
```

Выше рассмотрена реализация объектов, представляющих собой одну сущность, но кроме таких объектов еще необходимы объекты-коллекции для представления множественных связей.

В VB имеется тип Collection, который реализует работу с множеством ссылок на объекты, тип Collection реализует методы для добавления, удаления и подсчета числа элементов. Естественно, есть свойство для доступа к элементам, причем возможен доступ по ключу, как порядковому номеру, так и по строке являющейся идентификатором ссылки.

Для каждого типа связи (соответственно для каждого типа элементов) был заведен

свой тип коллекции. Все коллекции агрегируют в себе объект типа Collection и являются своеобразной прослойкой облегчающей работу с элементами конкретного типа .Ниже будет рассмотрена коллекция типа SVCol, содержащая объекты ScreenView, на примере данной коллекции можно получить представление о работе остальных коллекций.

В первую очередь, как и объекты-сущности, коллекции должны быть инициализированы. Инициализация происходит в метода InitAllNames:

```
Public Sub InitAllNames(rep As CRepository)
```

```
Dim RO As RepositoryObject

Dim SV As ScreenView
Обновление коллекции.
Set mCol = Nothing
Set mCol = New Collection
Создание новых объектов и  сохранение ссылок.
For Each RO In rep.RootObject.Interface("IRootInterface").ScreenViews
Set SV = New ScreenView
Set SV.RepObject = RO
mCol.Add SV, RO.Name
Next
End Sub
```

Не во всех типах коллекций есть такой метод - некоторые коллекции никогда не инициализируются из внешнего приложения - они заполняются при инициализации объекта содержащего такую коллекцию.

При добавлении в коллекцию также возникает проблема – надо ли добавлять новую сущность в репозиторий или она уже там есть. Для решения проблемы в метод Add кроме ссылки на добавляемый объект передаются флаг типа Boolean, который говорит новая сущность или она уже есть. Пример:

```
Public Sub Add(Item As ScreenView, rep As CRepository, Optional IsNewSV As Boolean = True)
Dim Node As CNode
Set Node = Item
If IsNewSV Then
rep.RootObject.Interface("IRootInterface").ScreenViews.Add Item.RepObject
mCol.Add Item, Node.Name
Else
mCol.Add Item, Node.Name
End If
End Sub
```

Обратная проблема стоит при удалении сущности из коллекции - надо ли удалять из репозитория или только из коллекции в памяти. Для решения проблемы был реализован метод PersistantRemove, который удаляет из

репозитория, в дополнение к методу Remove, который удаляет ссылку из коллекции.

Одной из возможностей коллекций в VB является возможность использования итераторов [3], которые позволяют перебирать все элементы коллекции без индекса или ключа. Это очень удобно при полном переборе коллекции. Для того чтобы у коллекции была такая возможность необходимо реализовать скрытый метод [3] NewEnum:

```
Public Property Get NewEnum() As IUnknown
Set NewEnum = mCol.[_NewEnum]
End Property
```

На этом описание коллекций законченно, как и описание всего API. Естественно рассмотрены не все объекты, но рассмотрены все возможные типы свойств и все нюансы и проблемы, возникавшие при разработке API. Все остальные объекты используют такие же механизмы, как и описанные выше и, поэтому нет необходимости рассматривать их все. Для более полного понимания можно посмотреть схему взаимодействия между объектами в Приложении 2, схема дана в нотации UML.

Глава 3. Оболочка-контейнер для инструментов администрирования системы «ТИСА»

3.1 Описание технологий используемых в программе

В первую очередь необходимо уточнить причины создания дополнительной программы. Ведь можно было реализовать средство настройки и средство установки как отдельные приложения или объединить их в одно нерасширяемое приложение. Но при этом пострадали бы расширяемость, гибкость и удобство. Под расширяемостью понимается следующее – при дальнейшем развитии системы могут появиться новые инструменты администрирования, которые легко могут быть встроены в общую оболочку, если делать отдельные приложения, то достаточно быстро администраторы запутаются в большом количестве приложений. В случае отдельных приложений необходимо следить за синхронизацией данных. Отдельное приложение требует больше ресурсов. Также разработчикам труднее поддерживать большое количество маленьких приложений, чем одно большое.

Инструменты администрирования, выполненные по технологии ActiveX Document, имеют такую же функциональность , что и обыкновенное Windows-приложение, но могут отображаться любым другим контейнером, поддерживающим данную технологию. Таким контейнером является MS Internet Explorer, клиентское приложение системы “ТИСА” и многие другие контейнерные приложения.

По выше изложенным причинам было решено создать приложение в стиле Windows Explorer. Тем более, что клиент системы “ТИСА” также выполнен в стиле Windows Explorer.

3.2 Руководство пользователя

Окно приложения состоит из двух частей. Левое окно – это окно отображающее список подключенных средств, правое окно – окно для отображения инструментов администрирования. При двойном щелчке на каком-

либо названии в левом окне, в правом окне появляется инструмент соответствующий выбранному названию, который представляет собой стандартный Windows-диалог с элементами управления. Такое поведение привычно для пользователей Windows, т.к. многие приложения выполнены в таком стиле (например, Microsoft Management Console).

При запуске приложения первым появляется окно-приглашение содержащее только текстовую информацию.

В меню “Help” присутствуют два пункта: “О программе” – стандартные сведения о программе, версии, фирме – производителе, “Помощь” – вызывается окно с информацией об использовании приложения. При завершении работы с приложением подключение к серверу данных закрывается.

3.3 Руководство программиста

Основной инструмент при создании данного приложения – это среда разработки Visual C++. Соответственно, приложение написано на языке C++ с использованием библиотеки классов MFC (Microsoft Foundation Classes)[3].

С помощью мастера приложений создается каркас приложения, в котором создаются классы, наследуемые из классов MFC, обеспечивается инициализация и минимальные действия фактически не изменяющиеся от одного приложения к другому. Также обеспечивается уничтожение объектов, освобождение памяти, загрузка и выгрузка DLL. MFC и AppWizard облегчают работу программиста избавляя его от написания большого количества кода.

При написании приложения использовалась технология хранения и представления данных - Document/View(Документ/Представление). Документ представляет собой объект, отвечающий за общение с источником данных(т.к. файлы, БД и т.д.). Также через него сохраняются и загружаются перманентные свойства объектов. Представление является средством для отображения данных пользователю и диалога с ним. У одного документа м.б. много представлений.

В нашем приложении создаются два представления для одного документа. Представление, находящееся в левой части, наследуется из класса CTreeView, класс называется CLeftView. Класс CTreeView обеспечивает для

окна, с которым связан объект этого класса, возможности стандартного элемента управления TreeControl. TreeControl позволяет отображать данные в виде древесной структуры с раскрывающимися узлами. Добавление и удаление узлов реализуется методами CTreeView AddItem и DeleteItem и др [3].

CLeftView содержит два объекта CStringArray (массивы строк) в первом из них хранятся ProgID ActiveXDocument'a, который будет отображаться в правом окне, во втором хранятся названия инструментов, которые будут отображаться в дереве.

ProgID необходим для создания ActiveXDocument'a. ProgId представляет собой строковый атрибут в виде ИмяМодуля.ИмяОбъекта. В системном реестре при регистрации COM объекта создается ключ с таким же названием, как и ProgID объекта. С этим ключом связывается CLSID – уникальный идентификатор класса, состоящий из 20 шестнадцатеричных цифр. Также создается ключ с названием, соответствующим CLSID, и, в связанном с ним поле, хранятся данные о местоположении сервера ActiveX Document'a.

Для подключения ActiveX Document'a используются функция Win32 API CLSIDFromProgID, которая принимает ProgID и выдает CLSID, соответствующий ProgID.

Непосредственно ActiveX Document представляет класс CadminToolsCntrItem наследуемый от CdocClientItem. В MFC поддержка ActiveX Document'ов как бы наложено на технологию OLE. Класс CadminToolsDoc наследуется из класса COLEDocument, который реализует документы со встроенными объектами (например, как документы Word). Специфичное управление ActiveX Document'ов, отличное от управления встраиваемыми объектами, сосредоточено в классе CadminToolsDoc.

В класс CadminToolsCntrItem добавлен метод Create в котором вызывается метод Create предка, передается CLSID и метод предка создает ActiveX Document в памяти. Для перевода ProgID в CLSID реализован приватный метод Initialize. При этом объекты только создаются, а отображаются они методом DoVerb с параметром OLE_SHOW. Данный вызов происходит после двойного щелчка мыши на названии инструмента в левом окне, при этом активизируется ActiveX Document, связанный с этим узлом дерева. Связь налаживается при создании - у каждого узла есть свойство ItemData в которое

записывается ссылка на ActiveX Document в памяти. По этой ссылке ActiveX Document активизируется. Если кликнуть два раза по другому узлу, то де активизируется уже загруженный ActiveX Document.

При закрытии приложения подключение также закрывается методом Close().

Глава 4. Инструменты настройки системы

На данный момент реализовано два приложения для настройки системы «ТИСА». Приложение для визуализации и изменения содержимого репозитория является основным инструментом настройки, именно в этом приложении можно редактировать и создавать новые модули и связи. Приложение, инициализирующее информацию о базе данных в репозитории носит служебный характер и необходимо для инициализации информации о схеме, хранящейся в базе данных.

Благодаря сокрытию множества сложностей связанных с подключением репозитория, созданием ActiveX Document'ов, упрощению доступа к COM объектам репозитория с помощью Automation, программирование инструмента на Visual Basic достаточно просто и программисту нет необходимости отвлекаться от логики выполняемой инструментом работы

4.1 Приложение для визуализации и изменения содержимого репозитория

Кроме самого репозитория и API доступа к его данным для полноценной системы необходимо приложение, визуализирующее и позволяющее изменять содержимое репозитория. Приложение создано в среде VB и представляет собой COM объект, удовлетворяющий требованиям технологии ActiveDocument. Для пользователя нет никакого различия между простым windows – приложением и ActiveDocument, в обоих случаях пользователь видит привычные окна диалога с привычными управляющими элементами и полями ввода.

4.1.1 Руководство пользователя

Приложение делится на две части – дерево связей между модулями и набором закладок с атрибутивной информацией (см рис 3). В дереве отображаются названия модулей и связи между ними, связи бывают входящие и выходящие, соответственно этому даны названия ветвей. От модулей у которых могут быть связи идут ветви с входящими и выходящими связями, листьями этих ветвей являются модуля связанные с текущим, при нажатии мышкой происходит

переход в дереве к связанному модулю и появляется возможность отследить его связи и атрибуты.

В начале работы необходимо подключиться к репозиторию и , если потребуется взаимодействие с системными таблицами , к БД. Параметры подключения задаются на закладке «Начало», здесь можно задать путь к файлу репозитория , данные о пользователе репозитория (имя, пароль) и параметры подключения к серверу Oracle.

Отдельного внимания требует часть программы взаимодействующая с системными таблицами системы «ТИСА». В программе реализована возможность загружать данные системных таблиц в репозиторий и обратно. Эта возможность необходима для редактирования уже существующих настроек системы. При загрузке и выгрузке информация об ошибках и предупреждения сохраняются в текстовые файлы, путь к этим файлам указывается в меню «Параметры». Для загрузки и выгрузке информации в системные таблицы системы «ТИСА» необходимо активизировать кнопки «Загрузить в БД» и «Загрузить из БД».

Для задания связей имеется кнопка ”Создать связь” расположенная под деревом. После активизации этой кнопки появляется окно диалога, в котором можно задать модуль, с которым будет установлена связь, здесь же можно задать атрибуты связи. В этом окне для выбора модуля необходимо определить, с каким типом модуля нужно установить связь и выбрать тип модуля в самом верхнем выпадающем списке, при этом в списке названий модулей будут только названия модулей определенного типа. После названия модуля идет поле ввода, где можно задать отображаемое имя модуля. Если тип модуля будет окно-таблица, то можно задавать ограничения на выборку сущностей и задавать фиксированный столбец . Также в специальном поле можно задать очередность отображения модуля.

В процессе работы возникает необходимость не только создавать, но и удалять связи, для этого нужно перейти на связанный модуль в ветке выходящих связей и удалить связь. Удаление связи производится кнопкой “Удалить связь”.

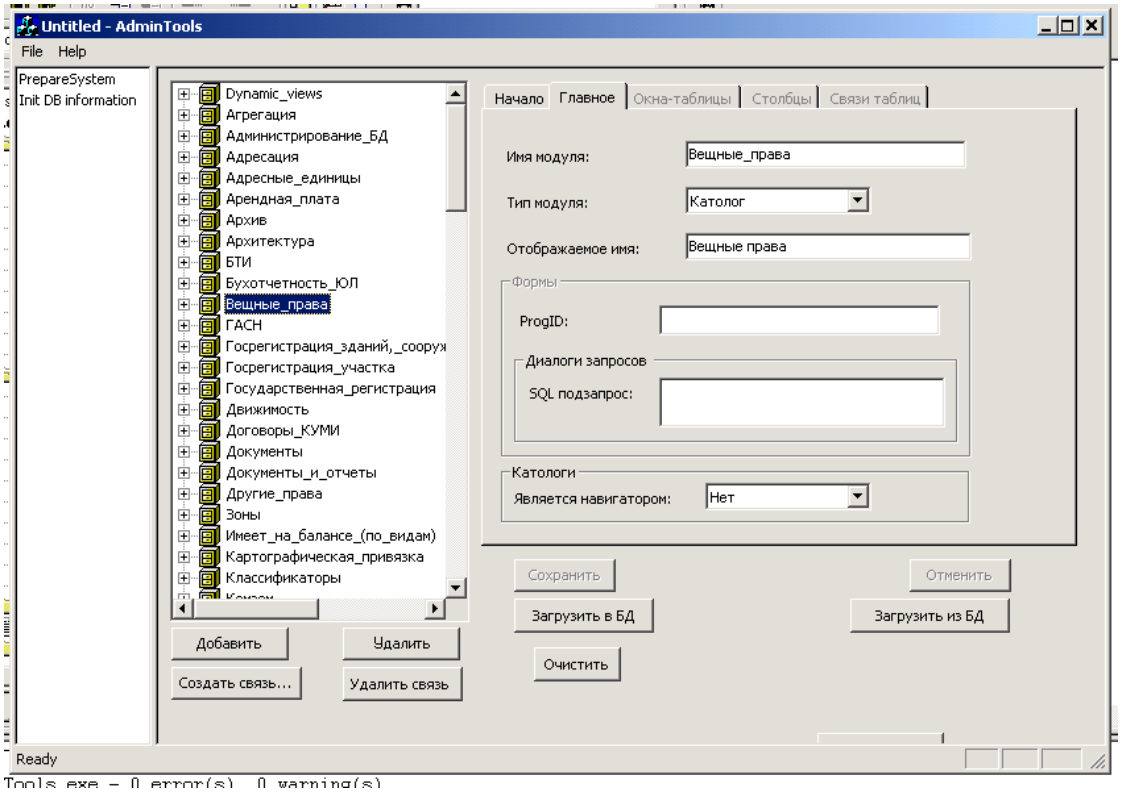

Рис. 3 Окно приложения.

Для добавления нового модуля необходимо нажать кнопку “Добавить”. При этом активизируется закладка “Главное”, где можно задать основные атрибуты модуля.

Для удаления модуля нужно нажать кнопку “Удалить”, и текущий модуль будет удален. Соответственно, будут удалены и все его связи.

Кроме связей и названий у модулей имеется множество других атрибутов.Количество и семантика атрибутов зависит от типа модуля. В приложении предусмотрена транзакционность, то есть, для сохранения всех изменений нужно нажать кнопку “Сохранить”, для отката к последнему сохраненному состоянию нужно нажать кнопку “Отменить”

Атрибуты модуля типа форма.

У формы, как и у любого модуля, имеется **имя модуля и отображаемое имя**.

Имя модуля – для использования администратором системы, отображаемое имя - для пользователя. Для вызова соответствующего объекта-формы необходимо задать **ProgID** – имя объекта в реестре MS Windows.

Атрибуты модуля типа диалог запроса

В дополнение к атрибутам формы у диалога запросов есть атрибут **SQL подзапрос.**В этот атрибут заносится начало запроса на выборку данных.

Атрибуты модуля типа каталог

У каталога атрибуты только **имя** и **отображаемое имя**, но каталог может иметь и входящие и выходящие связи.

Атрибуты модуля типа инструмент

У инструмента атрибуты те же что и у формы (**имя**, **отображаемое имя** и **ProgID**), но каталог может иметь и входящие и выходящие связи.

Атрибуты модуля типа окно-таблица

Окно-таблица имеет намного больше атрибутов, чем остальные модуля. Атрибуты остальных модулей умещаются на одну закладку, и только для окна-таблицы пришлось добавлять еще три.

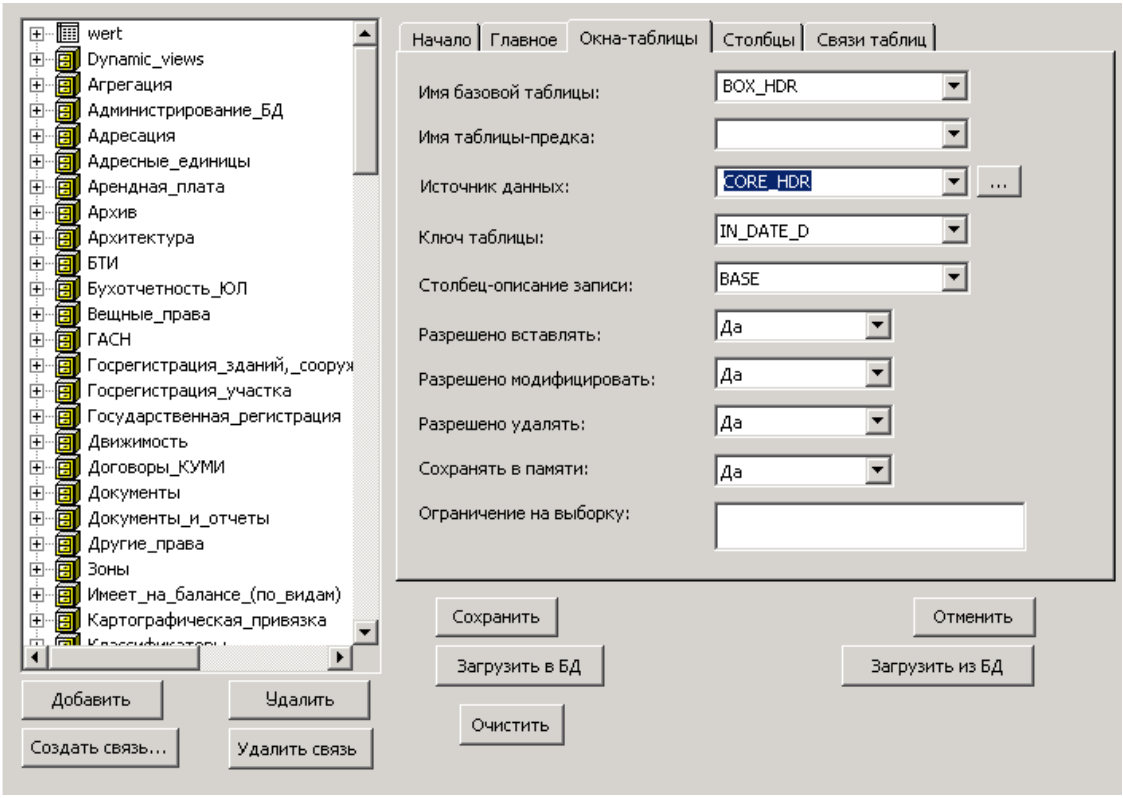

Рис.4 Закладка «Окна-таблицы»

На закладке с названием “окна-таблицы” можно задать атрибуты таблицы из которой берутся основные значения (см рис 4). В выпадающем списке “Имя базовой таблицы” нужно выбрать имя таблицы, куда будут сохранятся обновленные значения, в выпадающем списке «Источник данных» выбирается таблица, откуда берутся значения для отображения, в выпадающем списке «Таблица предок» задается таблица являющаяся предком для базовой (Например ОП(Объект права) является предком для таблицы УЧАСТОК) . В выпадающем списке «Ключ таблицы» задается столбец из базовой таблицы, который является ключом записи, , в выпадающем списке «Столбец-описание записи» задается столбец из таблицы-источника данных, который будет использоваться для отображения в дереве навигации. В находящихся ниже выпадающих списках со значениями “Да”, ”Нет” можно установить ограничения на операции с таблицей (например - запретить удалять).

На закладке с названием “Столбцы” задаются столбцы, которые будут отображаться пользователю. Столбцы задаются в элементе управления – таблице, в таблице можно задать **имя столбца**, его **тип**, **имя столбца в БД**, **очередность использования**, **алиасное имя** для запроса к БД, **длину столбца**.

Для столбцов представляемых не стандартными элементами управления можно задать **ProgID**.

Для запоминания данных о столбце нужно, после введения значений нажать клавишу “Enter”, при этом строка значений перейдет в основную часть таблицы, после введения всех описания всех столбцов нужно нажать кнопку “Сохранить” при этом все изменения будут сохранены в репозитории. Сохранять необходимо для работы с атрибутами на следующей закладке.

На закладке “Связи таблиц” можно задать атрибуты для столбцов, берущих значения из других таблиц. Такие столбцы необходимы для отображения семантического значения, тогда как в исходной таблице хранится числовое значение – идентификатор записи в связанной таблице. В выпадающем списке “Для столбца” выбирается один из столбцов, для которого будут заданны атрибуты. В выпадающем списке “Тип связи” задается тип связи, связь может быть двух типов: через таблицу или через карман **(см выше)**. При связи через таблицу необходимо задать в выпадающем списке имя таблицы, с которой будет связь, в двух других выпадающих списках необходимо задать первичный ключ таблицы и столбец, из которого будут браться семантические значения. При связи с карманом в поле ввода “Источник данных” необходимо ввести название источника данных в “кармане”, который совпадает с отображаемым именем окна-таблицы, из которого берутся значения в карман.

4.1.2 Руководство программиста

Приложение реализовано в среде VisualBasic и представляет собой COM объект, удовлетворяющий требованиям технологии ActiveDocument **(См выше)**. Приложение визуально состоит из двух частей – дерева связей модулей и закладки с атрибутивной информацией. Дерево отображается элементом управления типа TreeView(1), являющимся ActiveX элементом управления поставляемым вместе с VB. В коде данный элемент управления называется ctlTree. Атрибутивная информация отображается на закладках, управляемых ActiveX элементом управления типа SSTab, также поставляемого вместе с VB. В коде данный элемент называется ctlTab.

Для управления действиями предусмотрены несколько кнопок являющимися стандартными элементами управления.

Основная часть приложения называется FrmPrepareSystem. Также с ним связанно окно диалога типа Form с названием frmAddLink. Для удобства вывода информации имеется окно диалога типа Form, с названием frmInformation.

Реализация FrmPrepareSystem.

Компонента FrmPrepareSystem. включается в проект VB и, соответственно, в исполняемый модуль PrepareSystem. Для работы со средним уровнем (исполняемый модуль MiddleTier) и работы с репозиторием (исполняемый модуль RepositoryShell) необходимо загрузить их в память при выполнении, эту работу на себя берет VB. Для указания среде VB, какие компоненты необходимо загрузить, нужно задать соответствующие компоненты в окне диалога Reference [3].

Работа приложения начинается с подключения к репозиторию. Для этого создается объект типа CRepository m_Repos и вызывается метод OpenRepository, куда передается путь к файлу репозитория. Все это происходит в процедуре OpenRep. Там же, после открытия репозитория происходит инициализация коллекций разных типов модулей, это происходит вызовом метода InitAllNames (см описание MiddleTier) у объектов - коллекций. Затем происходит вызов процедуры LoadTree , которая заполняет дерево содержимым коллекций. Пример заполнения:

```
For Each Node In ScreenViews
Set SV = Node
Set TreeNode = ctlTree.Nodes.Add(, , Node.Name, Node.Name)
FetchRefTo ctlTree.Nodes.Add(TreeNode.Key, tvwChild, , strRefTo), SV
FetchRefBy ctlTree.Nodes.Add(TreeNode.Key, tvwChild, , strRefBy), SV
Next
```

Процедуры FetchRefTo и FetchRefBy, для каждого модуля, заполняют в дереве ветки с выходящими и входящими связями.

После загрузки дерева необходимо инициализировать выпадающие списки содержащие названия таблиц, что происходит в процедуре FillTableName:

```
Dim theColumnSet As DBColumnSet
BColumnSets.InitAllNames m_Repos
For Each theColumnSet In DBColumnSets
```

```
cmbTableName.AddItem theColumnSet.Name
Next
```

После выполнения этих процедур начальная инициализация приложения завершена. Дальнейшее исполнение приложение зависит от действий пользователя.

При активизации какого-либо модуля выполняется процедура SetCurrentModule. В этой процедуре по имени модуля из коллекций получается ссылка на объект-модуль с таким именем, и вызываются процедуры заполняющие элементы диалога значениями атрибутов модуля. Рассмотрим, как реализуется вышеописанное для окон-таблиц(ScreenViews):

```
Set SV = ScreenViews.Item(ModuleName)
Проверяем, принадлежит ли модуль к типу окно-таблица
If Not SV Is Nothing Then
Инициализация коллекций для временного хранения информации
If m_ColumnDescCol Is Nothing Then
Set m_ColumnDescCol = New Collection
Else
Set m_ColumnDescCol = Nothing
Set m_ColumnDescCol = New Collection
End If
m_bChangedColumn = False
Вызов функции заполняющей элементы диалога
LoadScreenView SV
SetSaveChancelEnables False
Set m_CurModule = SV
m_CurModuleType = psTypeScreen
Exit Sub
End If
```

Как видно основную работу выполняет процедура LoadScreenView. В этой процедуре просматриваются атрибуты ScreenView, а также связанных с ним объетов типа BaseTable, Column и других. Рассмотрим как это реализовано:

```
Private Sub LoadScreenView(SV As ScreenView)
```

```
        On Error GoTo LoadScreenView_Err
        обнуляем все значения  эле ментов диалога
        ClearAllControls
        Dim Node As CNode
        Set Node = SV
Отображаем значения   атрибутов Name и DisplayName
        txtModuleName.Text = Node.Name
        txtDisplayName.Text = Node.DisplayName
        With SV
        cmbModuleType.ListIndex = psTypeScree
Отображаем значения атрибутов разрешающих операции над данными
        If .AllowInsert Then
        cmbAllowInsert.ListIndex = 0
        Else
        cmbAllowInsert.ListIndex = 1
        End If

        If .AllowUpdate Then
        cmbAllowUpdate.ListIndex = 0
        Else
        cmbAllowUpdate.ListIndex = 1
        End If

        If .AllowDelete Then
        cmbAllowDelete.ListIndex = 0
        Else
        cmbAllowDelete.ListIndex = 1
        End If

        txtSQL_Where.Text = .SQL_Where
```

Если существует связанный объект типа BaseTable, то отображаем его имя и заполняем именами столбцов связанных с объектом типа BaseTable выпадающие списки в элементе диалога-таблице.

```
        If Not .GetValuesFrom Is Nothing Then
```

```
cmbTableName.Text = .GetValuesFrom.Name
FillDBColumn .GetValuesFrom.Name
End If


Dim TextColumn As TextColumn, LookupColumn As LookupColumn, ControlColumn As Executed
Dim TempDescr As ColumnDescription
Dim SVColumn As Column

On Error Resume Next
```

Заполняем временную коллекцию столбцов в памяти. Временная коллекция делается для упрощения чтения и записи элементом диалога типа GridEx. Компонент GridEx фирмы Janus реализуют отображение и редактирование данных в виде таблицы.

```
For Each SVColumn In .ConsistOf
Set TempDescr = New ColumnDescription
TempDescr.Alias = SVColumn.Alias
TempDescr.Name = SVColumn.Name
TempDescr.DefaultValue = SVColumn.DefaultValue
TempDescr.DisplayLength = SVColumn.DisplayLength
TempDescr.UsageSequence = SVColumn.UsageSequence
TempDescr.DBColumnName = SVColumn.GetValueFrom.Name

If TypeName(SVColumn) = "TextColumn" Then
Set TextColumn = SVColumn
TempDescr.ColType = TextColumn.ColumnType
End If

If TypeName(SVColumn) = "ControlColumn" Then
Set ControlColumn = SVColumn
TempDescr.ProgID = ControlColumn.ProgID
TempDescr.ColType = mtControlColumn
End If
```

```
If TypeName(SVColumn) = "LookupColumn" Then
Set LookupColumn = SVColumn
TempDescr.LookupInfo = LookupColumn.LookupInfo
TempDescr.ColType = mtLookupColumn
End If
m_ColumnDescCol.Add TempDescr, TempDescr.Name
Next
ctlColumnsGrid.ItemCount = m_ColumnDescCol.Count
ctlColumnsGrid.RefreshRowIndex 1
```

В функции FillLookups инициализируется выпадающий список cmbColumnsForLink в, котором можно выбрать столбец для связи с другой таблицей. В список заносятся имена только объектов типа LookupColumns

```
FillLookups SV
cmbColumnsForLink.ListIndex = TextToIndex(cmbColumnsForLink, m_CurLookupColName)
End With
m_bChangedModuleType = False
m_bChangedTableLink = False
Exit Sub
LoadScreenView_Err:
ErrorMsg "FrmPrepareSystem", "LoadScreenView"
End Sub
```

Обработка других типов модулей примерно также происходит, только проще в связи с меньшим количеством атрибутов.

При сохранении результата происходит обратная операция, то есть из элементов диалога информация переносится в репозиторий. Такая операция активизируется при нажатии кнопки “Сохранить”, в процедуре btnSave_Click. В этой процедуре, в зависимости от того, новый ли модуль или редактируется старый, вызываются процедурыCreateNode или UpdateNode. В процедуре CreateNode создается объект соответствующего типа и вызывается метод Create

(см описание MiddleTier). После создания объекта вызывается процедура, заполняющая атрибуты объекта соответствующего типа. В процедуре UpdateNode определяется тип модуля и вызывается соответствующая процедура, заполняющая атрибуты объекта. Детально рассматривать реализацию не имеет смысла, так как она достаточно проста и очевидна.

Из отдельных операций осталось рассмотреть только создание связи. Создание связи происходит после активизации кнопки “Создать связь”. Создание связи осложняется тем, что есть два типа связи: между модулями, представляющими узла дерева и между окном-таблицей и формой или диалогом запроса. В зависимости от типа связи создается объект типа NodeLink или DialogLink. Для получения атрибутов связи вызывается функция frmAddLink.ShowDlg. В эту функцию по ссылке передаются переменные, которые после выполнения содержат значения атрибутов связи. Если в frmAddLink была активизирована кнопка “Ок”, то ShowDlg возвращает True. Если ShowDialog вернула True, то связь будет создаваться, иначе нет.

Создание связи отличается от редактирования атрибутов тем, что при создании связи результат изменений сохраняется сразу, а при редактировании атрибутов нужно нажимать кнопку “Сохранить”.

Модули экспорта и импорта (ExportUtils и ImportUtils).

Отдельно от программы редактирования стоят модули экспорта и импорта данных из и, соответственно, в системные таблицы системы «ТИСА», хранящихся на севере БД. Функции , необходимые для экспорта и импорта, вынесены в отдельные модуля для лучшего понимания кода.

Импорт данных происходит посредством запроса данных из системных таблиц и созданием соответствующих объектов репозитория. Для запроса данных используются объекты ADO[3]. Объект ADODB.Connection общий для всех модулей и инициализируется в функции FrmPrepareSystem.btnLogin_Click. Для получения данных используется ADODB.Command, для хранения и обработки ADODB.Recordset. Данные получаются от сервера в виде набора записей. По набору записей в цикле проходит указатель на текущею запись [3] и, в соответствии с данными, создаются объекты и связи в репозитории. При импорте возможны противоречия между данными в репозитории и данными в системных таблицах. Например, когда у окно-таблицы прописана базовая

таблица, информации о которой нет в репозитории. В этом случае описание ошибки сохраняется в текстовом файле (см. Руководство пользователя). Информации о пути к файлу хранится в реестре.

Экспорт данных происходит посредством пакеджа Oracle написанном не PL/SQL [1]. В функции пакеджа , посредством ADODB.Command, посылаются данные об объектах репозитория, и в функции эти данные вставляются в новые или заменяют старые записи соответствующих таблиц. Пример функции:

procedure create_module (InId in out number,ModuleType char,DisplayName varchar2,BaseTableID number default null,KeyColumn varchar2 default null, DescrColumn varchar2 default null, SqlWhere varchar2 default null, AllowInsert char default 'N',AllowUpdate char default 'N',AllowDelete char default 'N') is

flag number; SQLAutoQuery char(1); SQLFrom varchar2(256);

begin

if ModuleType = 'D' then

select cached,name into SQLAutoQuery,SQLFrom from t$sys_base_table where id = BaseTableID;

else

SQLAutoQuery := 'N';

SQLFrom := null;

end if;

Смотрим, есть ли уже запись с таким Id.

select decode(count(*),null,0,1,1) into flag from t$sys_module where t$sys_module.id = InId;

--insert into test(num) values(flag);

Если есть то заменяем данные, если нет то вставляем с новым Id.

if flag > 0 then

update t$sys_module set module_type = ModuleType,display_name = DisplayName,base_table_id = BaseTableID,sql_col_key = KeyColumn,sql_col_descr

= DescrColumn,sql_from =SQLFro ,sql_where = SQLWhere, sql_auto_query = SQLAutoQuery,allow_insert = AllowInsert, allow_update = AllowUpdate,allow_delete = AllowDelete where id = InId;

else

select export_utilites_seq.nextval into InId from dual;

insert into t$sys_module (id,module_type,display_name,base_table_id,sql_col_key,sql_col_descr,sql_from,sql_where,sql_auto_query,allow_insert,allow_update,allow_delete) values(InId,ModuleType, DisplayName,BaseTableId, KeyColumn,DescrColumn,SQLFrom,SQLWhere,SQLAutoQuery,AllowInsert,AllowUpdate,AllowDelete);

end if;

begin

insert into t$sys_module_ace(module_id,grantee) values(InId,'S_ALL') ;

exception

when DUP_VAL_ON_INDEX then

null;

end;

end;

Большинство работы по обработке и сохранению данных делается на сервере. На стороне клиента в цикле проходится по объектам репозитория и вызываются соответствующие функции сервера. Из функции возвращается Id записи либо новый или старый и присваивается свойству DataBaseId объекта репозитория. Свойство DataBaseId необходимо для поддержания целостности при процедуре экспорта – импорта. При работе экспорта ошибки также записываются в файл. Но здесь ошибки фатальны, то есть обработка прекращается при появлении ошибки.

Как можно заметить из вышеописанного, в многих местах необходима информация о схеме базы данных, для работы с которой настраивается клиентская часть системы.

Ниже будет описано приложение, инициализирующее информацию о схеме БД.

4.2 Приложение, инициализирующее информацию о базе данных в репозитории

Приложение выполнено по таким же технологиям, как и приложение для редактирования и просмотра репозитория.
Данное приложение берет информацию из системных таблиц базы данных и переносит соответствующую информацию в схему репозитория.

4.2.1 Руководство пользователя

Приложение состоит из двух закладок: на одной заносится информация для подключения к репозиторию (см рис 5) и к базе данных, на другой – показывается информация из репозитория.

Для подключения к базе данных надо в соответствующие поля ввести имя, пароль и имя базы данных [1]. При этом можно пароль не вводить и по умолчанию пароль такой же, как и имя пользователя.

Для подключения к репозиторию нужно ввести путь и имя файла, для удобства есть кнопка “Найти” при активизации, которой вызывается диалог навигации по файловой системе. После этого нужно активизировать кнопку “Подключится к репозиторию”. Если есть уже открытое подключение, то можно использовать его.

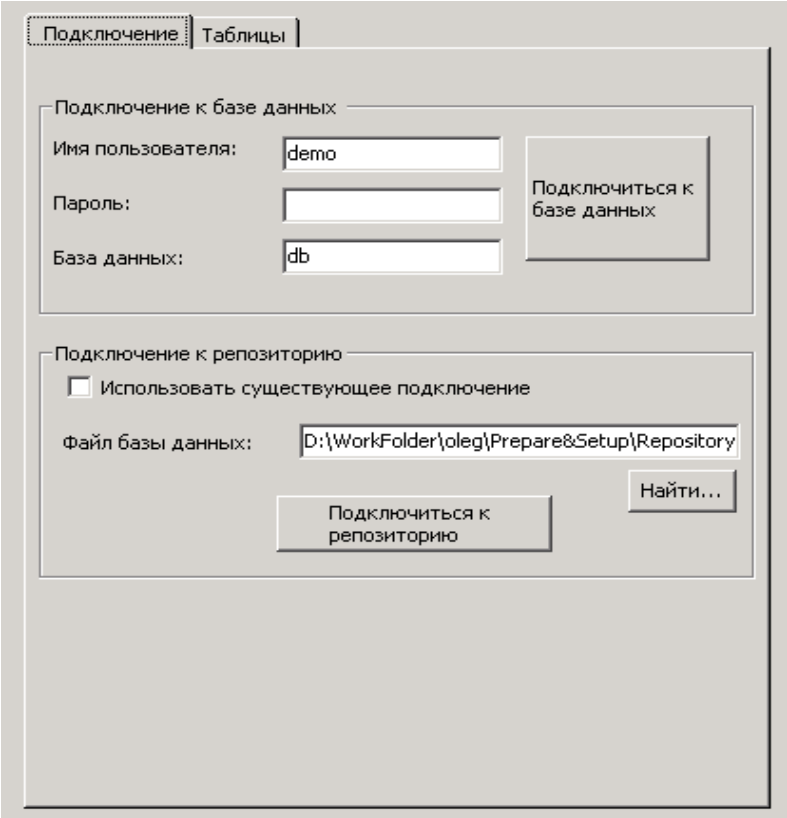

Рис 5. Закладка подключения к БД и к репозиторию.

По умолчанию значение параметров подключения берется из реестра ОС Windows.В реестре хранятся параметры последнего подключения.

На второй закладке находится элемент диалога дерево и в нем отображается информация об именах таблиц и их столбцов содержащаяся в репозитории. Для заполнения этой информации нужно активизировать кнопку “Загрузить из БД”.

При большом количестве таблиц в схеме базы данных, возможно достаточно длительное ожидание результата загрузки.

4.2.2 Руководство программиста

Приложение выполнено в среде VB и соответствует технологии ActiveDocument (см выше и [3]). Исходный код находится в проекте PrepareSystem. Имя компоненты, являющейся приложением – FrmInitInstallDB.

При инициализации приложения вызывается обработчик события UserDocument_Initialize, где получается информация из реестра. Информация из реестра получается с помощью встроенной функции VB GetSetting. Здесь же элементам диалога присваиваются значения по умолчанию.

Подключение к базе данных происходит в обработчике события btnLogin_Click

Подключение к базе данных происходит посредством набора компонент ADO [3]. Для подключения используется информация, введенная пользователем. При возникновении ошибки вызывается диалог, отображающий информацию об ошибке.

При успешном подключении в реестре запоминаются параметры подключения.

Подключение к репозиторию происходит в обработчике события btnRepConnect_Click. Для подключения создается объект типа CRepository и вызовом метода OpenRepository, куда передается путь к файлу репозитория, открывается репозиторий. После открытия репозитория вызывается функция, заполняющая элемент диалога дерево ctlTree.

Дерево заполняется функцией FetchTree. В этой функции вызывается метод InitAllNames объекта с именем mColumnSets типа DBColumnSetCol (см Middle Tier).

После этого просматривается содержимое коллекции, и имена таблиц и столбцов заполняются в дерево.

В обработчике события btnLoad_Click происходит заполнение информации о таблицах в репозиторий. Для этого, посредством объекта ADODB.Command и его метода Execute, выполняется следующий запрос: select TABLE_NAME,COLUMN_NAME,DATA_TYPE,DATA_LENGTH from sys.user_tab_columns where user_tab_columns.table_name in (select view_name from user_views). Метод Ezxecute возвращает объект ADODB.Recordstet содержащий результат запроса. После этого вызывается процедура FetchColumnSetAndColumn в которой удаляются имеющаяся в репозитории информация, а после добавляется новая полученная по запросу. Для ускорения удаления и добавления информации, после обработки информации о каждых ста таблицах происходит завершение транзакции и начало новой. Это необходимо для уменьшения потребления ресурсов. Незавершенная транзакция занимает достаточно много памяти и возможны даже сбои в работе приложения.

Заключение

В современных информационных системах очень большую роль отводят масштабируемости и гибкости. К повышению масштабируемости и гибкости систем ведет трех и более уровневая структура, альтернатива обычным клиент-серверным приложениям [5].

Создание среднего уровня для системы “ТИСА” и инструментов для управления бизнес-логикой – это существенный шаг, повышающий гибкость и возможность масштабируемости системы. На данный момент бизнес-логика распределена между сервером и клиентом. Разработанная модель среднего уровня может быть использована для создания полноценного среднего уровня (при соответствующей доработке ядра системы).

Средний уровень позволит создавать сектора системы, в которых функциональность клиента не зависит напрямую от сервера данных, сервер данных поставляет только данные и никакой управляющей информации. При этом упростится настройка и поддержка системы.

Управлять объектами среднего уровня можно с помощью COM+ - технологии расширяющей COM и поставляющей множество сервисов управления и создания объектов среднего уровня. В операционную систему Windows 2000 интегрирована технология COM+. Более подробно про COM+ смотри в [5].

Также средний уровень позволяет создать специализированные для других СУБД OLTP системы. Достаточно легко перенести репозиторий на MS SQL Server, так как имеется поддержка репозиторием хранения своей информации на данной СУБД. Соответственно, меньше проблем для создания специализированного для данной СУБД клиента. Данное направление достаточно перспективно с точки зрения возможного спроса на рынке информационных технологий. Систему “ТИСА” в современном состоянии невозможно продавать как “коробочный продукт” не требующий затрат времени со стороны разработчиков. Для внедрения системы необходима квалификация и опыт разработчика системы. Такое положение определяется сложностью системы, отсутствием специализированных систем настройки и другими причинами. При создании соответствующих инструментов настройки системы и создания нового ядра системы, удовлетворяющего требованиям “коробочного

продукта”, возможно распространение продукта без привлечения разработчиков. Естественно, возможно сделать такую систему для СУБД Oracle, но Oracle достаточно дорогая система и многие предпочитают более “легкую” и дешевую СУБД SQLServer.

Список литературы

1. Oracle 8 Documentation Library. Oracle Corporation.

2. Microsoft Repository SDK. Microsoft Corporation.

3. Microsoft Developer Network Library. Microsoft Corporation.

4. Концепция построения муниципальной информационной системы “Монитор”. П. П. Андрющенко, А.М. Бабанов, М.В. Вотяков и др. В сб. “Геоинформатика. Теория и практика. Вып. 1 / Под ред. А. И. Рюмкина, Ю. Л. Костюка. – Томск: Изд-во Том. ун-та, 1998, с. 151 – 163.

5. Р.Дж. Оберг Основы Com +. Технологии и программирование. –М.,С-Пб,Киев: Издательский дом Вильямс, 2000.

Приложение 1. Информационная модель среднего уровня системы “ТИСА”

Приложение 2. Обьектная модель среднего уровня системы “ТИСА”

Приложение 3. Список файлов

Оболочка - конитейнер	Средний уровень	Инструменты настройки
AdminTools.cpp	UtilMidTier.bas	ExportUtils.bas
AdminToolsDoc.cpp	Utlites.bas	ImportUtils.bas
AdminToolsException.cpp	BaseTable.cls	UtlMod.bas
AdminToolsView.cpp	BaseTableCol.cls	ColumnDescription.cls
CntrItem.cpp	Column.cls	TableLinkDescription.cls
ConnectDialog.cpp	ColumnCol.cls	PrepareSystem.dll
LeftView.cpp	ControlColumn.cls	FrmInitInstallDB.dob
MainFrm.cpp	DBColumn.cls	FrmIsmonContact.dob
StartDialog.cpp	DBColumnCol.cls	FrmPrepareSystem.dob
StdAfx.cpp	DBColumnSet.cls	FrmInitInstallDB.dox
AdminTools.dep	DBColumnSetCol.cls	FrmPrepareSystem.dox
AdminTools.dsp	DBForeingKey.cls	PrepareSystem.exp
AdminTools.dsw	DBForeingKeyCol.cls	frmAddLink.frm
AdminTools.h	DBPrimaryKey.cls	frmColumnForLink.frm
AdminToolsDoc.h	DialogLink.cls	frmInformation.frm
AdminToolsException.h	DialogLinkCol.cls	frmLogFiles.frm
AdminToolsView.h	DialogView.cls	frmLoginDlg.frm
CntrItem.h	Executed.cls	frmSQLFromText.frm
ConnectDialog.h	FormView.cls	frmAddLink.frx
LeftView.h	FormViewCol.cls	frmColumnForLink.frx
MainFrm.h	GlobalConstants.cls	PrepareSystem.RES
resource.h	LookupColumn.cls	FrmInitInstallDB.vbd
StartDialog.h	LookupTable.cls	FrmPrepareSystem.vbd

StdAfx.h	Node.cls	PrepareSystem.vbg
AdminTools.mak	NodeLink.cls	PrepareSystem.vbp
AdminTools.ncb	NodeLinkCol.cls	PrepareSystem.vbw
AdminTools.opt	QueryDialog.cls	Utilites.bas
AdminTools.plg	QueryDialogCol.cls	CRepository.cls
AdminTools.rc	ScreenView.cls	RepositoryUtilites.vbp
mssccprj.scc	SVCol.cls	RepositoryUtilites.vbw
vssver.scc	Table.cls	
msado15.tlh	TextColumn.cls	
	Tool.cls	
	ToolCol.cls	
	TreeFolder.cls	
	TreeFolderCol.cls	
	TreeNode.cls	
	PrepereAndMiddleTier.vbg	
	MiddleTier.vbp	
	MiddleTier.vbw	
	MonitorModel.mdl	
	TISAMiddle-TierModelVB.mdl	
	P_EXPORT_UTILITES.pck	
	MonRep.mdb	