НАЦИОНАЛЬНАЯ АКАДЕМИЯ НАУК УКРАИНЫ
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
МЕЖДУНАРОДНЫЙ НАУЧНО-УЧЕБНЫЙ ЦЕНТР
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

В.И. ГРИЦЕНКО, А.А. УРСАТЬЕВ

РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ ШИРОКОГО ПРИМЕНЕНИЯ

КОНЦЕПЦИЯ ОПЫТ РАЗРАБОТКИ И ВНЕДРЕНИЯ

КИЕВ НАУКОВА ДУМКА 2005

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МІЖНАРОДНИЙ НАУКОВО-НАВЧАЛЬНИЙ
ЦЕНТР ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
ТА СИСТЕМ

ГРИЦЕНКО Володимир Ілліч
УРСАТЬЄВ Олексій Андрійович

РОЗПОДІЛЕНІ ІНФОРМАЦІЙНІ СИСТЕМИ ШИРОКОГО ЗАСТОСУВАННЯ

КОНЦЕПЦІЯ. ДОСВІД РОЗРОБКИ і ВПРОВАДЖЕННЯ

(Російською мовою)

Київ, видавництво "Наукова думка"

Оформлення художника
Художній редактор *Є.І. Муштенко*
Технічний редактор *Г.М. Ковальова*
Коректор *В.О. Подолянчук*
Оператори *О.О. Іщенко, О.О. Іщенко*
Комп'ютерна верстка *О.І. Фуженко*

УДК 004.65:004.7:004.738.5:004.9

В монографии предложена концептуальная модель распределенной гетерогенной информационной системы (ИС), положенная в основу построения масштабируемых платформонезависимых экономичных ИС широкого применения на типовых серверных платформах (узлах) с использованием *Internet/intranet*-технологий и распространенных стандартных программных компонентов. Рассмотрены вопросы защиты информации и сетевого мониторинга распределенной ИС. Особое место в монографии занимает импорт/экспорт данных в ИС. Приведены примеры реализации распределенных ИС.

Книга рассчитана на подготовленного читателя, знакомого с проектированием и использованием ИС в практической деятельности.

Для специалистов в области информационных систем и технологий, программного обеспечения, а также студентов соответствующих специальностей.

У монографії запропоновано концептуальну модель розподіленої гетерогенної інформаційної системи (ІС), яку покладено в основу побудови платформонезалежних економічних ІС широкого застосування, що масштабуються, на типових серверних платформах (вузлах) із використанням *Internet/intranet*-технологій і розповсюджених стандартних програмних компонентів. Розглянуто питання захисту інформації і мережного моніторингу розподіленої ІС. Особливе місце в монографії займає імпорт/експорт даних в ІС. Наведено приклади реалізації розподілених ІС.

Книга розрахована на підготовленого читача, знайомого з проектуванням і використанням ІС в практичній діяльності.

Для фахівців у галузі інформаційних систем і технологій, програмного забезпечення, а також студентів відповідних спеціальностей.

A conceptual model of distributed heterogeneous information system (IS) has been proposed in the monograph. It is assumed as a basis of construction of scaled, platform-independent efficient IS of wide use on the type server platforms (nodes) with the use of Internet/intranet-technologies and wide-spread standard software components. The monograph deals with the problems of information defend and network monitoring of the distributed IS. Special place in the work belongs to import/export of the data in IS. Results of implementation of distributed IS are presented.

The book includes results intended for a trained reader, familiar with the problems of IS design and practical use in various subject fields.

For specialists in the field of information systems and technologies, software, as well as for students of the corresponding specialities.

Рецензент доктор технических наук, профессор В.А. ВЛАСЕНКО

Утверждено к печати ученым советом Международного научно-учебного

центра информационных технологий и систем НАН Украины и Министерства образования и науки Украины

Редакция физико-математической и технической литературы

Редактор *С.Е. Ноткина*

ISBN 966-00-0572-5

ПРЕДИСЛОВИЕ

Процессы создания информационного общества и его глобализации интенсивно развиваются. На современном этапе перехода от индустриального к информационному обществу и построения в дальнейшем общества знаний степень использования информационного пространства и новых технологий становится во всех ведущих государствах мира непосредственным фактором экономического роста и социально-политической стабильности.

Общегосударственный информационный ресурс формируется как интегральная составляющая ресурсов, сосредоточенных в информационных системах (ИС) верхнего уровня и низовой инфраструктуры. Без достаточного развития последней невозможно или крайне сложно решение задач социально-экономического развития страны. В этой связи необходимо создание экономных ИС с предоставлением информации разно-образного характера. В основу построения таких систем должен быть положен принцип оптимизации показателя функциональность/стоимость путем выбора технологии построения, обусловленной задачами и целью ИС.

В Международном научно-учебном центре информационных технологий и систем НАН Украины и МОН Украины (в дальнейшем Центр) предложена и реализована оригинальная концепция создания высокоэкономичных распределенных ИС с использованием *Internet/ intranet*-техно-логий и распространенных программных компонентов. При таком подходе затраты на разработку и эксплуатацию ИС, обучение пользователей и персонала, сопровождающего систему, значительно ниже затрат на внедрение таких зарубежных систем, как *Oracle, Sybase* и др. Это дает возможность в сжатые сроки создавать ИС в разных сферах и обеспечивать надежную базу для развития

системы на протяжении ее жизненного цикла. Отечественный и зарубежный опыт подтверждает эффективность этого направления.

Созданная Центром технология использована при разработке Государственной навигационно-гидрографической информационной системы (1997—2000) и сети распределенных гидрометеорологических баз данных (БД) для Государственной гидрометеорологической службы Минэкоресурсов Украины (2000—2002). Так, под сетью распределенных гидрометеорологических баз данных понимается создание и объединение в единое проблемно ориентированное информационное пространство множества специализированных тематических баз гидрометеорологических данных с возможностью доступа к их ресурсам. Эти базы данных, в общем случае, географически или территориально распределены. Они локально формируются и сопровождаются. В БД предполагается сосредоточить *режимно-справочные* архивные материалы наблюдений гидрометеорологической службы, информация в которых во временном отношении делится на срочную (с разным количеством сроков наблюдений за сутки) и обобщенную за суточные и месячные интервалы. Специализированные тематические базы данных в свою очередь группируются в банки данных (БнД) по видам гидрометеорологической информации (актинометрическая, агрометеорологическая, гидрологическая, метеорологическая и т.д.). Так образуется сеть распределенных гидрометеорологических БнД. Поскольку в БнД должны накапливаться архивные материалы наблюдений, то следовало бы говорить о создании *электронного хранилища данных*, в котором данные, не претерпевая в последующем изменений, будут пригодны для использования в автоматизированных ИС. Исходные гидрометеорологические данные, сохраняемые на бумажных, магнитных и других носителях, представленные в разных видах и форматах, являются лишь сырьем соответствующих ресурсов, несмотря на уникальность информации, накапливаемой с 1800 г. Вследствие этого возникает проблема "грязных данных" (данные в силу тех или иных причин неоднозначно идентифицируются), которая в последнее время тревожит Мировое сообщество. Для гидрометеорологических данных, как и для информационного пространства, характерны разнородность (гетерогенность), противоречивость, слабоструктурированность.

Первая очередь сети распределенных гидрометеорологических БнД испытана в Украинском гидрометеорологическом центре. Работа выполнялась в рамках заданий Национальной программы информатизации Украины.

Привязка основных научно-технических решений к структуре гидрографической и гидрометеорологической служб подтверждает их универсальность и возможность широкого применения в

других областях человеческой деятельности и социальной практики.

Авторы глубоко признательны своим сотрудникам, внесшим основной вклад в некоторые специальные области разработки, и специалистам министерств и ведомств Украины за содействие в выполнении прикладных разработок.

Особую благодарность авторы выражают рецензенту, проф. В.А. Власен- ко, высказавшему ряд замечаний, которые авторы постарались учесть.

РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ. СОСТОЯНИЕ. ПЕРСПЕКТИВЫ РАЗВИТИЯ

1.1. Решения проблемы интероперабельности

В условиях интенсивно развивающихся про-цессов создания информационного общества и его глобализации особая роль отводится законам и принципам развития информационного пространства, процессам взаимодействия информационных ресурсов, которые обеспечивают высокий динамизм поиска и общей информационной поддержки в глобальных и локальных средах.

Реализация многообразия задач в мировом сообществе (промышленность, экономика, бизнес, экология, гидрометеорология, гидрография, океанография, социальная сфера и другие области) связана с потребностью совместного использования информации и ресурсов в распределенной среде баз данных с разными структурами и технологией применения [1—4]. Иначе говоря, требуется анализ разнородных данных, которые порождаются, накапливаются, хранятся и используются в информационных системах, ориентированных на различные по функциональности задачи. По сути, речь идет о разнородных БД и различных приложениях, обеспечивающих информационную поддерж-ку тех или иных процессов, но не способных взаимодействовать в общей среде.

Это вызывает дополнительные требования к современным информацион-

нологиям (ИТ) относительно интероперабельности. Создание информационных технологий с такими качествами во многом будет предопределять эффективность использования знаний — новых интеллектуальных ресурсов, являющихся движителями информационного общества и в перспективе общества знаний.

Интеграционные процессы охватывают различные аспекты: от моделирования архитектуры сложных информационных систем [2, 5], позволяющего систематизировать подходы к проектированию корпоративных ИС, описать процедуру их создания, установить взаимодействие участвующих сторон, до применения тех или иных технологий к решению поставленных задач. Например, в промышленности [3, 6] широкое распространение получили программные продукты нового поколения, направленные на интеграцию приложений *EAI* (*enterprise application integration* — интеграция приложений предприятия) путем объединения функ-циональных подсистем *CPM, CRM, ERP, PDM, SCM*, баз данных, хранилищ данных и других внутренних подсистем предпри-ятия. Здесь:

CPM (*corporate performance management*) — управление эффективностью бизнеса. Системы опираются на централизованную БД, образующую единое информационное пространство управленческого планирования и контроля, и ориентируются на оперативную аналитическую обработку *OLAP* (*online analytical processing*) основных параметров хозяйственной и финансовой деятельности предприятия;

CRM (*customer relationship management*) — системы управления отношениями с клиентами;

ERP (*enterprise resource planning*) — системы планирования ресурсов предприятия, которые решают задачи оперативного производственного планирования в целях минимизации затрат на производство продукции или предоставление услуг;

PDM (*product data management*) — управление взаимодействием данных, описывающих структуру промышленного изделия и процессов его производства на любом этапе жизненного цикла;

SCM (*supply chain management*) — управление цепочками поставок.

До настоящего времени почти все проекты *EAI* являются частными решениями. Подсистемы работают на собственном наборе данных, не могут обмениваться данными в режиме реального времени и не образуют единую систему [3].

Одним из примеров такого продукта, реализующего интегрированную информационную среду предприятия *EAI*, может служить система *Windchill Solution* с *Internet*-ориентированной архитектурой. Ее основными характерными признаками являются [7]:

- платформа интеграции корпоративных приложений *EAI*, реализующая метод интеграции приложений "сверху вниз", в соответствии с которым множество источников данных объединяется без программирования (на уровне исходных кодов), благодаря поддержке промышленных стандартов на инфраструктуры ИТ, сетевые протоколы, *API* и интерфейсы *SQL, ODBC, JDBC, LDAP, HTML, XML*. Платформа интеграции содержит широкий набор адаптеров для доступа к информации в приложениях класса *ERP* и *PDM*, а также в практически любых унаследованных системах или БД;
- федеративное [1] (*federative*) представление данных (гибкая модель данных) позволяет работать с различными СУБД (*Oracle, Sybase, Informix* и т.д.), интегрировать существующие БД, без затрат времени и средств на создание общей БД, и быстро внедрять систему на предприятии с единой корпоративной информационной средой. Отсутствие необходимости в централизованной [2] БД [8] особенно важно для реализации сети субподряд-

[1] Федеративное представление данных предполагает доступ к данным, находящимся непосредственно в разнородных источниках, и создание единого виртуального хранилища. Разработчики пишут свои запросы к федеративной системе данных, выполняющей роль посредника, состоящую в том, чтобы абстрагироваться от соединений с различными серверными источниками данных [8].

[2] Централизованный подход к извлечению данных из различных источников предполагает дублирование данных, приведение их к единому формату и последующее накопление в одной БД.

чиков и поставщиков, в которой основной задачей является поддержание единой модели данных.

Ядром системы *Windchill* является модуль *Windchill Founda-tion* — информационная архитектура, поддерживающая *Internet*-ориентированные приложения в распределенных средах. В него включены службы управления документооборотом, потоками заданий и администрирования.

Клиенту *Windchill* для доступа к любому модулю и функциям системы необходим *Web*-браузер. На клиентской машине выполняются *Java*-аплеты, загружаемые из сервера.

В качестве интегрированной среды разработки приложений используется встроенный пакет *Visual Cafe* фирмы *Symantec* с библиотекой компонентов *JavaBean*, поддерживающий также разработку графических интерфейсов пользователя [7].

Процессы, вызываемые функционированием подсистем *CRM, SCM,* выходят за рамки предприятия. Распределенные системы электронного бизнеса *B2B* (бизнес—бизнес), электронные торговые площадки (*eMarketplace*), электронная коммерция требуют значительного расширения корпоративной среды таким образом, чтобы обеспечить единое информационное поле для производства и торговли. Вследствие этого прогнозируется, что со временем системы *B2B* и подобные им могут поглотить существующие системы *EAI* [3].

В технологии федеративного доступа, использующей известные механизмы интеграции приложений (*EAI, JDBC* и др.), также предусмотрены средства одновременного поиска необходимых сведений в ряде разнородных источников информации, что также востребовано в системах *CRM*, работающих с данными из многих источников, и при управлении контентом — информационным наполнением ресурса предприятия, состоящим преимущественно из малоструктурированных данных.

Традиционно поиск сопряжен со значительными затратами времени вследствие адресации запроса лишь одному источнику данных. Для получения данных из нескольких источников необходимы метаданные о хранимых в различных структурах и форматах данных, понятные разнород-

ным системам. Для распознавания этих метаданных и последующего их использования в целях извлечения данных из распределенных серверов ресурсов (рис. 1.1) в решениях, направленных на реализацию технологии федеративного доступа (*Liquid Data BEA Systems, IBM DB2 Infor-mation Integrator (DB2 II), MS Yukon (СУБД SQL Server), СУБД Oracle 9i*) [8], в качестве универсального формата применяют язык *XML* и стандартные средства извлечения информации из источников *XML*-данных — *XQuery* [9]. Используемые в *XML* теги представляют собой семантику данных, распознаваемую разнородными системами. Данные из различных БД автоматически приводятся к стандарту *XML*.

В рамках этой технологии разработан программный инструментарий *IBM Content Manager* на базе СУБД *IBM DB2* [10], ориентированный на создание хранилищ данных (электронных архивов), в которых информация не изменяется после сохранения

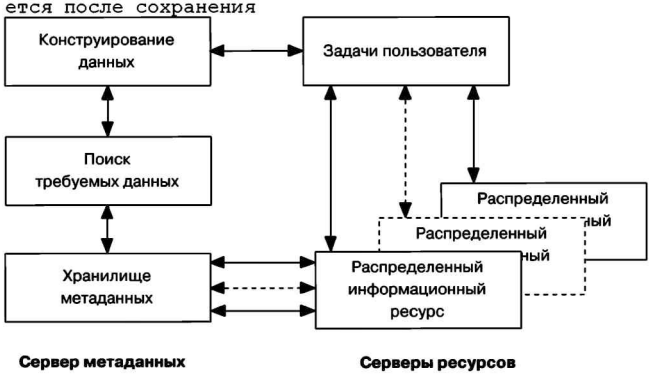


Рис. 1.1. Технология федеративного доступа

(например, мультимедийные коллекции, изображения банковских чеков и др.) и имеет весьма значительный объем. Ресурсные серверы, работающие в общем случае под управлением различных операционных систем, размещаются удаленно, вблизи от групп пользователей, создающих

и поддерживающих информационное наполнение. В ответ на запросы пользователей серверы ресурсов извлекают контент из различных источников, указанных в хранилище.

Архитектурно *Content Manager* представляет собой сервер библиотеки, ресурсные серверы и клиентское место, предоставляющее пользовательский интерфейс. Библиотечный сервер обеспечивает хранение метаданных, индексацию, авторизацию пользователей и управление хранением документов в качестве "объектов" на ресурсных серверах.

Отметим еще один аспект интеграции ресурсов (аппаратный): сети хранения (*SAN — storage area network*) как средство объединения современной инфраструктуры предприятия — переход от *DAS* (*direct attached storage*) хранения данных на персональном компьютере или в сети, на файл-сервере, к их распределению. Сетевое хранение данных по технологии *SAN* предоставляет серверам корпоративной сети консолидированный сетевой ресурс внешней памяти. Консолидация строится на основе создания общего пула ресурсов с едиными принципами доступа в виде виртуальной [3] памяти *VCS* (*virtual consolidated store*). В качестве ресурса хранения информации используют дисковые массивы, ленточные библиотеки. В основе концепции *SAN* лежит возможность соединения по высокоскоростной сети передачи данных любого из серверов с любым устройством хранения данных, работающим по протоколу *Fibre Channel*. Скорость передачи данных в дуплексном режиме составляет 400 Мбайт/с. Новой редакцией стандарта предусматриваются скорости обмена 800 Мбайт/с и 2 Гбайт/с. Для организации распределенных систем хранения требуется применение кабельной сети по типу *Gigabit Ethernet* с волоконно-оптическим соединением устройств (до 10 км). Разнообразные топологии *SAN* в совокупности с системами управления хранением *SAM* (*storage area management*) — управление средой хранения — обеспечи-

[3] Идея виртуализации — обеспечить предоставление серверам ресурсов хранения в виде, независимом от используемых дисковых массивов. В идеале серверы должны "видеть" не устройства хранения данных, а ресурсы.

вают большую гибкость, производительность и надежность [11—15].

Возможности существующих средств, предназначенных для интеграции приложений, определяются конкретной архитектурой (*CORBA, DCOM, EJB* [4] *J2EE, .NET* [5] и др.). Средствами *CORBA* (*OMG*) и *DCOM* (*MS*) строится промежуточное программное обеспечение, независимое от операционных систем и от языков программирования. *CORBA* и *DCOM*, направленные на создание инфраструктуры распределенных вычислений, позволяют организовать единое информационное поле, в котором обмен между приложениями осуществляется на уровне объектов. Однако объекты и методы действуют в рамках своей архитектуры. Предложенная *Sun Microsystems* платформа *J2EE* (*Java 2 Platform, Enterprise Edition*) инициировала появление сервера приложений, на котором сконфигурированы с *Web*-сервером предварительно собранные программные приложения, решающие те или иные задачи и позволяющие разработчику сосредоточить свое внимание на функциональной части решаемой задачи.

До недавнего времени *Web*-технологии играли роль консолидатора гетерогенных систем. При этом *Web* рассматривался как механизм предоставления информации [6], как средство общения людей с ИС: по запросу пользователя предоставлялась статическая или динамическая *html*-страница, формируемая с помощью того или иного программного расширения *Web*-сервера [17,18]. В последнем случае на *Web*-сервер возлагаются задачи не только пересылки файлов по запросу клиента, но и выполнения приложений, на лету создающих *web*-страницы. В последующем выполнение приложений на стороне сервера привело к появлению сервера приложений. В основу

---

[4] *EJB* (*Enterprise JavaBeans*) — компонентная модель программирования (технология), которая изолирует приложения от базовых вычислительных платформ, поддерживая их переносимость.

[5] *.NET* включает в себя средства *XML* как стандарт обмена данными, протокол доступа к объектам *SOAP* для удаленного вызова процедур, технологию создания динамических *web*-приложений *ASP.NET* и средство взаимодействия с БД *ADO.NET* [16].

[6] Вид информационных услуг *Internet—Web*-сервис — предоставление хранящихся на *Web*-серверах гипермедиа-документов другим компьютерам глобальной сети *Internet*.

взаимодействия браузеров и *Web*-серверов положен протокол *HTTP* (*Hyper Text Tra-nsfer Protocol*).

Сегодня интерпретируют *Web*-службы (*Web-service*) [3, 11, 19] как способные обеспечить создание интегрированной платформы. *Web*-службы являются новым средством общения ИС и никоим образом не ориентированы на человека. Вследствие этого приложения, содержащие непохожие и в общем случае разные технологии и стандарты, построенные на разных платформах, могут взаимодействовать. Однако в работах [11, 20] отмечается, что термин *Web*-службы неадекватен, поскольку ориентированную на службы архитектуру *SOA* [7] (*service oriented architecture*) увязывают с понятием *Web*. Правильнее будет говорить о службах, так как в них используются лишь сетевые принципы, сходные с *Internet*.

Новое поколение многозвенной архитектуры (*Web*-службы) ориентировано на слабо связанные приложения (*loosely coupled*), т.е. изменения в одном приложении не должны влиять на другие. Кроме того, приложения должны работать асинхронно, что позволит выполнить процесс обслуживания заявок в течение определенного промежутка времени и тем самым избежать перегрузки *Web* (*web-storm*). Обмен сообщениями должен осуществляться достаточно крупными информационными блоками (*coarsegrain*), в противном случае произойдет закупорка магистралей [11].

В основе технологии *Web*-служб лежит протокол *SOAP* (*simp-le object access protocol*) — простой протокол доступа к объектам, который основан на произвольном

---

[7] *SOA* — архитектура, ориентированная на службы, — тип организации распределенных гетерогенных систем, предполагающей наличие программных агентов, которые, взаимодействуя, обеспечивают заданную функциональность сервисов [20]. На примере трех поколений корпоративных систем [20, 21] показаны произошедшие в них изменения как в архитектуре, так и в функциональности. Как следствие закономерного эволюционного процесса в этой области ИТ и вызвано появление *SOA*.

Вероятно, архитектуры (*CORBA, DCOM* и др.) тоже ориентированы на службы, но при этом каждая из них определяет свои собственные форматы и протоколы, механизмы вызовов, интерфейсы приложений. Недостаток универсальности ограничивает их распространение. В сегодняшней трактовке *SOA* под службами понимают *Web,- Internet*-технологии и развитую инфраструктуру [20].

транспортном протоколе [8]. Протокол *SOAP* использует *XML* как способ представления информации для обмена сообщениями между ИС, корпоративными приложениями и для удаленного вызова процедур *RPC* (*remote procedure call*). *SOAP* имеет специфицированный транспорт и способ упаковки информации, а также способ описания предоставленной функциональности. Стандартизация делает *SOAP* действительно общим и открытым средством общения гетерогенных систем в интернете.

Платформа *Web*-служб, основанная на стеке стандартов *SOAP*, *WSDL* и *UDDI*, разделяется на три части: коммуникационные протоколы, описания служб и поиск услуг.

1. Простой протокол доступа к объектам *SOAP* поддерживает связь между *Web*-службами. Системы взаимодействуют с *Web*-службами способом, указанным в их описании, используя сообщения в стандарте *SOAP*.

Физически *Web*-служба представляет собой фрагмент программного обеспечения, называемый "агентом". Реализуя абстрактную функциональность сервиса, агент способен передавать и принимать сообщения. Один и тот же сервис может быть обеспечен разными агентами [20].

2. Язык *WSDL* (*web services description language*) — язык описания *Web*-служб, формата *XML*; описывает интерфейс *Web*-служб в виде абстрактного определения в терминах сообщений, обмен которыми происходит при обращении к службе, т.е. раскрывает формализованное взаимодействие клиент—служба.

3. Глобальный каталог *UDDI* (*universal description, discovery and integration*) — универсальное описание, информация и интеграция — представляет собой регистр описаний *Web*-служб.

Таким образом, в настоящее время *Web*-службы определяют [20, 22—25] как реализуемый программно-стандартизованный способ межмашинного взаимодействия в сети для интеграции приложений на основе открытых стандартов: *SOAP*, *WSDL* и *UDDI*. Глобальный каталог *UDDI* помогает найти службу, *WSDL* — ее охарактеризовать, а *SOAP* — взаимодей-

[8] В настоящее время *SOAP* используются транспортные протоколы *HTTP*, *FTP* и *SMTP*.

ствовать с ней. Документы в виде *HTML*-страниц, *CGI*-приложения (*CGI*-сценарии), программы *ASP*, *PHP* и др. по определению не являются *Web*-службами, поскольку не придерживаются указанных стандартов.

Существует альтернативный подход — "архитектурный стиль" *REST* (*Representational State Transfer*) [26], который позволяет реализовать функции *Web*-служб без привлечения этих стандартов. *REST* задуман как основа масштабируемого прикладного протокола, обеспечивающего доступ к *информационным ресурсам*, в то время как *Web*-службы на основе протокола *SOAP* не используют возможности механизма адресации информационных ресурсов, основанного на единых идентификаторах ресурсов *URI* (*uniform resource identifier*).

Сторонники *REST* утверждают, что *SOAP* — нижний уровень в стеке протоколов — не что иное, как работающие через *Internet CORBA* или *DCOM*. Действительно, первоначально технологии, подобные *SOAP*, называли *Web*-брокерами, или объектными брокерами, построенными в *Web*. Используемая в них модель доступа к объектам предназначена для замкнутой среды, в которой известен каждый пользователь и данные можно перераспределять между ними. В замкнутой среде возможны прямые коммуникации [26, 27].

Реализуются *Web*-службы имеющимися средствами. Они строятся на стеке стандартов, который со временем, предполагают, претерпит изменения, например будущее поколение *Web*-служб [3, 28], получившее название *Semantic Web* [9] *Services*, в которых при сохранении общей модели отдельные составляющие будут заменены новыми. Прежде всего это относится к описанию функциональности служб и к средствам взаимного поиска соответствующих служб (*service matchmaking*). В итоге может сложиться иной стек стандартов.

Вызвано это тем, что язык расширяемой разметки *XML* и дополняющий его в части описания малоструктуриро-

---

[9] *Semantic Web* (*SWeb*) — семантическая сеть. В полном объеме *SWeb* — это некий идеальный образ виртуального пространства будущего [3, 28].

ванных метаданных *RDF* [10] (*Resource Description Framework*) [29] для приложений семантической сети [11], связанных с представлением и обработкой знаний на базе *Web*, неэффективен при описании отношений между объектами, называемыми в *SWeb* онтологиями. Средства описания службы на уровне *WSDL* не достаточны для того, чтобы с ними могли работать программные агенты *SWeb* [28]. Поэтому предложен новый язык разметки *DAML* (*DARPA Agent Markup Language*) — язык разметки агентов, предусматривающий возможность применения в документах широкого спектра семантических концепций вплоть до эквивалентных и уникальных [3, 31]. Язык *DAML* расширяет *XML* и *RDF*. Последняя версия языка (март 2001) носит название *DAML+OIL* (*Ontology Inference Layer*) — уровень онтологических заключений [31]. *DAML+OIL* дополняют *DAML* онтологическими функциями, позволяющими применять общие концепции и придавать смысл данным в документе. Этот стандарт стал основой деятельности онтологической рабочей группы *Web Ontology Working Group*, образованной недавно в консорциуме *W3C*. Задача группы — разработать стандарт создания и управления онтологией в документах [32]. Предполагается, что стандартом *DAML-S* (*DAML Services*: *DAML-based Web Service Ontology*) [31, 33] можно заменить *WSDL*.

Альтернативой регистру описаний *Web*-служб *UDDI* может стать один из разрабатываемых консорциумом *OASIS*

---

[10] *RDF* — инфраструктура описания *XML*-ресурсов. Стандарт описания метаданных осуществляется на языке *XML*. Консорциумом *W3C* предложен облегченный язык *RDF Schema* для описания классов и свойств.

[11] Под семантической сетью понимается ориентированный граф, состоящий из помеченных вершин и дуг и задающий объекты и отношения предметной области (ПрО). Дуга может соединять две вершины или два графа. Метка и направление дуги конкретизируют семантику.

Компонентами семантической сети являются не только понятия (объекты) и отношения ПрО, но и составленные из них различные ситуации (фреймы).

Семантические сети придают фактическим знаниям графовую организацию, более естественную и структурированную, чем в случае других представлений знаний. Такого рода структура моделирует ассоциативные связи для извлечения каких-либо фактов из других фактов [30].

(*Organiza-ion for the Advancement of Structured Information Standards*) — Организация усовершенствования структурированных информационных стандартов — открытых стандартов *ebXML* в области технологий обмена корпоративной информацией, использующих *Inter-net* в качестве транспортной среды. *OASIS* приводит в соответствие конкурирующие решения для создания *XML*-реализаций, ориентированных на индустрию в целом. В настоящее время рассмотрен и одобрен *ebXML Messaging Service* 2.0 [34].

Разработка спецификаций *ebXML* для электронной коммерции будет продолжена совместно с комитетом *UN/CEFACT* (*Uni-ted Nations Centre for Trade Facilitation and Electronic Business*) — Центр ООН по развитию торговли и электронного бизнеса [34]. Разработчики (*OASIS* и *UN/CEFACT*) считают, что *ebXML* станет международным стандартом и заменит *EDI* [12] (*Electronic Data Interchange*) — электронный обмен данными: передача между ИС электронным способом структурированных сообщений в согласованном стандарте [34, 35].

Вместе с тем корпорации *IBM* и *Microsoft*, ранее принимавшие участие в разработке технологий *Web*-служб, представили стандарт *WS-Inspection* (*Web Services Inspection Language*), который дополняет *UDDI*, является логическим его продолжением и полностью с ним совместим. Как и в случае с *UDDI*, все *web*-решения в рамках новой технологии основаны на *XML*, однако, как утверждается, в основе сервисов будут лежать более совершенные алгоритмы. Если *UDDI*, строго говоря, представляет собой универсальный справочник, то *WS-Inspection* существенно расширяет возможности поиска. Спецификация нового стандарта *WS-Inspection* дополняет глобальную технологию каталогов *UDDI* благодаря оптимизации процесса непосред-

[12] *EDI* отличаются от систем электронного документооборота, функционирующих в рамках одной корпорации или предприятия, тем, что *EDI* — это межведомственные системы обмена электронными документами. В настоящее время *EDI* используют стандарт *UN/EDIFACT* (*United Nations rules for Electronic Data Interchange for Administration, Commerce and Transport* — "Правила ООН электронного обмена документами для государственного управления, торговли и транспорта") [36, 37]. Известно использование технологии *XML/EDI* в электронной коммерции [36].

ственного обнаружения служб на *Web*-сайтах, отсутствующих в реестрах *UDDI*.

Стандарт *WS-Inspection* обмена информацией об услугах, предоставляемых друг другу *Web*-службами, инициирован в первую очередь требованиями обработки информации международных баз данных для предприятий электронной коммерции [38—40].

Представим еще один стандарт консорциума *OASIS*, ориентированный на развитие технологии порталов [13] при решении интеграционных задач. Главная проблема корпоративных порталов на функциональном уровне — это проблема доступа к разнородной, в общем случае пространственно рассредоточенной информации. Они призваны объединить все имеющиеся у организации информационные ресурсы (приложения, базы и хранилища данных, аналитические системы и пр.) и, используя *Web*-интерфейс, предоставить пользователям единый защищенный доступ к корпоративной и внешней информации [41—45]. Распространенные варианты использования порталов — электронный бизнес, информационные *Web*-сайты компаний, интеграция эксплуатируемых функциональных подсистем предприятия (*CPM, CRM, ERP, SCM*) и других узкоспециализированных приложений.

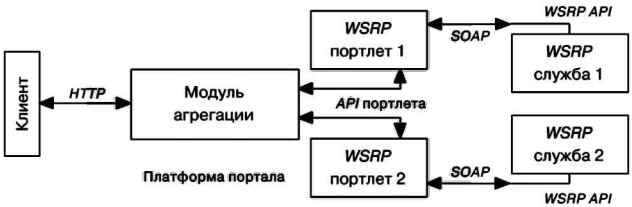
Стандарт *WSRP* (*Web Services for Remote Portlets*) — *Web*-службы для удаленных портлетов [25, 43, 46, 47] — определяет правила взаимодействия порталов со службами, иначе, *WSRP* предлагает ряд интерфейсов *Web*-служб, облегчающих включение в портал дистанционно расположенного контента и прикладной логики. Формализация способа взаимодействия портала с внешним приложением и унификация технологии представления результатов, достигаемых введением для всех *WSRP*-служб [14] одинако-

---

[13] По мнению аналитиков международного исследовательского агентства *IDC (International Data Corporation)*, изучающего рынок информационных технологий и программного обеспечения (ПО), корпоративные порталы должны играть роль центрального шлюза, *единой для всех точки доступа к любой информации, производимой компанией или требуемой ей* [42].

[14] *WSRP*-служба — это презентационно-ориентированная *Web*-служба, которая может быть использована любым заинтересованным в нем клиентским приложением. "Презентационные" интерфейсы

вых "презентационных" интерфейсов, позволяющих любому *WSRP*-совместимому порталу использовать их, предоставляют возможность автоматического [15] подключения уда-



ленных портлетов [16], отвечающих данному стандарту.

Рис. 1.2. Интеграция внешних приложений в порталы по стандарту *WSRP*

Первая задача решалась соглашением о том, что *WSRP*-службы являются *Web*-службами, следовательно, в основе работы с ними лежит стек протоколов *UDDI*, *WSDL* и *SOAP*. Вторая — разработкой и согласованием "презентационных" *WSRP*-интерфейсов. Интеграция по стандарту *WSRP* показана на рис. 1.2.

Здесь стандартом определены структуры данных и интерфейсы, позволяющие удаленным приложениям предоставлять свои данные универсальным способом через *WSRP*-службы. Интерактивная работа поддерживается представителем *WSRP*-службы на портале *WSRP*-портлетом. *WSRP*-модуль может быть установлен и работать не только в среде портала, но и на удаленном сервере, высту-

---

WSRP-службы — интерфейсы, специфицированные в стандарте *WSRP* [47].

15 *WSRP*-совместимый портал берет на себя функции поиска и встраивания *WSRP*-служб. Результатом его работы является автоматически сгенерированный представитель *WSRP*-службы на портале — *WSRP*-портлет (*Portlet pro_xy*). Таким образом, интеграция приложений через *WSRP*-службы осуществляется административными средствами портала.

16 Портлеты, или портальные апплеты, — это своеобразные *API*, предназначенные для подключения корпоративных приложений к порталу, — элементы программ, исполняющие роль плагина для портальной структуры [43].

пая промежуточным звеном между порталом и *WSRP*-службой [25, 43, 46, 47].

При существующих вариантах интеграции приложений в порталы (рис. 1.3), например в широко используемых программных портальных продуктах *IBM WebSphere Portal* [44, 48] и *Plumtree Portal* [45] компании *Plumtree Software*, приложение должно иметь своего "представителя" на стороне портала, который возьмет на себя функции общения с ним, — адаптер приложения. Непосред-

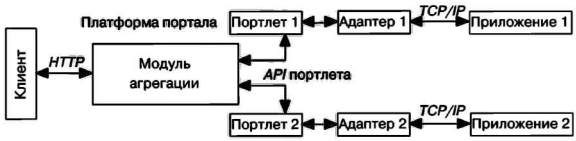


Рис. 1.3. Интеграция удаленного приложения посредством локального портлета

ственно с адаптером общается портлет [17], который через интерфейсы адаптера получает данные, обрабатывает их на уровне логики задачи. Он же отвечает за способ представления результатов. Портал объединяет результаты, полученные от разных портлетов, и отсылает их клиенту.

Таким образом, при "классическом" варианте интеграции приложений в портальные системы, во-первых, требуется обязательное написание портлета, который будет общаться с адаптером приложения. Во-вторых, адаптер приложения, который в ряде случаев необходимо разработать, требуется установить на портал или удаленный сервер. В-третьих, если имеется N приложений, то в общем случае эти действия требуется повторить N раз. И самое главное, все перечисленное выше возможно только тогда, когда это позволяют как портальная платформа, так и само приложение [25]. Поэтому веду-

[17] Портлет называют *локальным*, если он выполняется в среде портала. Для программного обеспечения *IBM WebSphere Portal* такой средой является сервер приложений *IBM WebSphere Application Server* — законченный *Java*-сервлет *Web*-сервер [49, 50].

щие разработчики портального ПО (*IBM, Vignette, SAP* и конкурирующие *Plumtree Software, Documentum, BEA Systems, Sun Microsystems*), наряду с промышленными аналитиками и представителями предприятий, присоединились к поддержке *OASIS* стандартов *WSRP*. Это позволяет предоставлять конечным пользователям доступ к приложениям и данным, расположенным на разных платформах и в гетерогенных сетях, используя обыкновенный *Web*-браузер.

В то же время наблюдается стремительное развитие географически распределенной среды *Grid*-вычислений, называемой будущим *Grid*-компьютинга [18]. Сравнивая *Grid* с *WWW*, можно отметить, что *Grid* выходит за рамки совместного пользования информацией, предоставляя возможности распределенных вычислительных мощностей и систем хранения данных.

Изначально *Grid* [19] использовался академическими структурами в задачах распределенной аналитической обработки больших объемов информации. Приведем некоторые *Grid*-решения, свидетельствующие о жизненности и перспективности этого направления работ: Британская национальная *Grid* (*UK National Grid*); Голландская *Grid*-сеть (*The Netherlands Computing Grid*); американская *Grid*-система *DTP* (*Distributed Terascale Facility*) производительностью 13,6 трлн операций в секунду для проведения исследований в области биологии, медицины, моделирования климатических ситуаций и других дисциплин; вычислительная *Grid*-среда для диагностики и изучения рака молочной железы (США) и др. Работы по *Grid* проводятся Институтом прикладной математики им. М.В. Келдыша РАН. Развитие российского сегмента *Grid* проводится в связи с инициативой ряда институтов физики высоких энергий участвовать в создании европейской инфраструктуры *DataGrid*.

В целом это позволит интенсифицировать научные исследования, объединив вычислительные мощности и дан-

---

[18] *Grid*-компьютинг: отдельный компьютер становится частью виртуальной вычислительной среды, полностью растворяясь в глобальной ИТ-инфраструктуре [51].

[19] *Globus Toolkit*, свободно распространяемый инструментарий, — фактический стандарт конструирования *Grid*-систем.

ные, обеспечив их разделенное использование и финальное объединение результатов. В качестве коммуникационной основы для академических структур *Grid* используются высокопроизводительные сети, относящиеся к *Internet 2*. Под управлением Национального научного фонда США строится трансконтинентальная сеть *Global Terabit Research Network* (*GTRN*), которая физически будет иметь каналы в Европе и Америке; логически она будет объединятся с существующими азиатскими и европейскими сетями. Сеть предоставит четыре уровня услуг: национальную магистраль, магистраль для предприятий, оптическое и медное подсоединение к персональным компьютерам с пропускной способностью 1000, 100, 10 и 1 Гбит/с соответственно [51, 52].

Поддержка приложений электронного бизнеса (моделирование в области дизайна, сборки и управления жизненным циклом производства таких сложных объектов, как самолеты или автомобили, моделирование финансовых ситуаций и др.) расширила сферу применения *Grid* и сблизила с технологиями и стандартами, обеспечивающими взаимодействие на уровне служб. Эту функцию выполняет открытый стандарт *OGSA* (*Open Grid Servi-ces Architecture*). Он интегрирует *Web*-службы и услуги в области *Grid*-вычислений, ресурсы распределенных, гетерогенных, динамических сред и сообществ, что является основой для использования *Grid*-приложений в коммерческих целях. Модель *OGSA* включает в себя три стандарта *Web*-служб: *SOAP, WSDL* и *WS-Inspection*. В отличие от *Web*-служб, *OGSA* допускает использование временных (*transient*) экземпляров служб, инициируемых и завершаемых динамически [51,52].

1.2. Обсуждение вопроса

Итак, прослеживаются три класса ИС, в целом ориентированные на:

- управление производством и анализ многоаспектных данных с целью получения значимой для бизнеса информации [53];
- процесс превращения данных в знания;
- использование среды вычислений и хранения данных.

Каждому классу ИС отвечает присущая ему служба (*Web-, SWeb-, Grid*), обеспечивающая взаимодействие между приложениями в неоднородной вычислительной среде распределенной ИС. Появление архитектур, ориентированных на службы, направлено на создание универсальной платформы для корпоративных приложений. Интеграция служб, их развитие создают предпосылки для слияния классов ИС, точнее, для привлечения тех или иных заимствованных ресурсов в зависимости от специфики возникающих задач. Кроме того, ИС будут в состоянии удовлетворить не только решение целевых задач, но и различные требования приложений электронного бизнеса.

Так, онтологии, играющие значительную роль в архитектуре *Semantic Web*, с успехом используются не только при обработке знаний, но и в электронной коммерции в силу того, что представить номенклатуру реализуемых изделий так же несложно, как и описать, например, хронику последовательных исторических событий. Здесь наблюдаются две различные онтологии: в первом случае в качестве концепта выступает товар, во втором — исторические даты. Онтологии используются для поддержки автоматизированного обмена данными между покупателями и продавцами, для вертикальной интеграции рынков, а также для повторного использования описаний различными электронными торговыми точками [28]. *Grid*-технология применяется для создания распределенной системы хранения медицинской информации о пациентах (объединяет тысячи больниц, расположенных в разных городах), которая фиксирует полный диапазон медицинс-

ких файлов, включая высококачественные медицинские снимки (томограммы, маммограммы), записи и истории болезни. Система обеспечивает управление и хранение огромных файлов с возможностью их быстрого поиска, сравнения и диагностической оценки при гарантированном соблюдении стандартов защиты и конфиденциальности, включая соответствие различным законодательным требованиям [52].

В последнем случае, по мнению авторов, представлять, хранить и обрабатывать клиническую информацию целесообразнее, используя технологии фреймов и семантических сетей, так как медицина относится к слабоструктурированной и трудноформализуемой области знаний [54].

Вместе с тем распределенное хранение данных, развитие служб не только упрощает в известном смысле положение с конструированием ИС, но и выдвигает ряд серьезных задач, связанных с обеспечением взаимодействия на уровне служб. Назовем только некоторые из них: организация обмена сообщениями, работа с очередями, управление потоками работ и необходимость разработки для этого новых языков программирования, трансформация модели данных БД и др. Нельзя не согласиться с мнением авторов работ [3, 11, 51], что решение этих вопросов выходит за рамки инженерных задач и требует привлечения интеллектуальных технологий.

Здесь уместно упомянуть о логических моделях данных в контексте БД и используемых СУБД. Реальные данные и их взаимосвязи крайне сложны, а их отображение на реляционные таблицы часто приводит к крайне неэффективным структурам и запросам с множеством операций соединения, необходимых для воссоздания этих сложных структур. Все это приводит к снижению производительности из-за непомерно больших объемов дополнительной обработки. На смену реляционной технологии пришли объектная технология и объектные БД, в которых оперируют уже не с полями, а с понятиями "класс", "объект" и "свойство". Основным сдерживающим фактором от перехода с реляционной технологии на объектную является

значительный объем существующих наработок на основе реляционных СУБД.

Сегодня традиционным реляционной и объектно-ориенти-рованной моделям противопоставляется сетевая или иерархическая [30], обеспечивающие высокую скорость работы с данными и простоту масштабируемости. Постреляционные СУБД, поддерживающие объектную технологию, используют многомерные структуры данных иерархического или сетевого ядра системы в виде классов объектов и реляционных таблиц, обеспечивая тем самым поддержку обеих традиционных моделей. Многомерная модель (*гиперкуб*) — совокупность разреженных многомерных массивов, называемых *глобалями* (*globals*), — обеспечивающая эффективное и компактное хранение данных в сложных информационных структурах, используется в СУБД *Cache*. Данные,

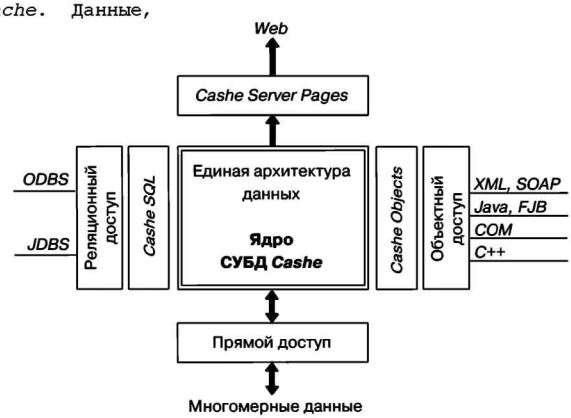


Рис. 1.4. Архитектура СУБД *Cache*

хранящиеся в глобале, могут иметь любое количество индексов, что позволяет описать и хранить произвольно сложную структуру данных. "Постреляционность" *Cache* реализована с помощью так называемой единой архитектуры данных *UDA* (*Unified Data Architecture*), которая

предусматривает единое описание объектов и таблиц, отображаемых непосредственно на многомерные структуры ядра базы данных. В *Cache* предлагается три способа доступа к данным: с помощью объектов, посредством *SQL* или прямой доступ через многомерные структуры (рис. 1.4).

Именно реализация трех моделей данных, поддержка большинства аппаратных платформ и операционных систем (*Windows NT* 4, *XP*, *Server* 2003, *Red Hat Linux (Intel)* и др.) позволяет эффективно использовать СУБД *Cache* не только для хранения данных, но и для интеграции информационной среды предприятия *EAI*. В *Cache* имеется сервер приложений (*Cache Server Pages*) со встроенной технологией создания корпоративных *Web*-приложений *CSP* (*Cache Server Pages*), работающих во многом аналогично *ASP* и *JSP*. Для обмена данными с другими системами в ней реализованы эффективные интерфейсы импорта и экспорта данных в структуры *XML*. Сообщения, которыми обмениваются сервер и клиент, представлены в виде *XML*-документов со структурой, определяемой стандартом *SOAP* [55–57].

В свете изложенного уместно задаться вопросом, какова же перспектива ИС на основе *Web*-технологии? Эти ИС [58] объединяют в проблемно ориентированную информационную среду территориально или географически распределенные, локально сопровождаемые БнД. Основу распределенных ИС широкого применения составляет предложенная и апробированная в Центре концептуальная модель на типовых серверных платформах (узлах), в которую положены: унифицированный интерфейс для пользователя и для задач управления/администрирования взаимосвязанной совокупностью сервисов и служб; широкое использование *Internet/intranet*-технологий и стандартных программных компонентов; независимость от программно-аппаратной платформы [59], вытекающая из принципа взаимодействия компактных элементов ИС на уровне протоколов.

Защита информации от несанкционированного доступа осуществляется с учетом сохранения в ИС информации разных степеней значимости, доступ к которой регламентируется нормативными актами [60].

Распределенные ИС предоставляют информацию разнообразного характера, используя средства динамической генерации *web*-страниц — программные расширения, непосредственно интегрированные в *Web*-сервер [17, 18, 58]. Они экономны и легко осваиваемы, так как не требуют значительных затрат на эксплуатацию, обучение пользователей и персонала, сопровождающего систему. Кроме того, имеется база для постоянной модернизации и развития ИС благодаря тому, что рынок программного обеспечения и технологий для *Internet/intranet* стремительно развивается.

На этих принципах реализованы Государственная навигационно-гидрографическая информационная система (ГосНГИС) и сеть распределенных гидрометеорологических БнД — ИС "Банк гидрометеорологических данных" [4, 60—63]. Системы обеспечивают информационную поддержку принятия решений, осуществляя их на качественно новом технологическом уровне. В обеих системах *Web* используется как средство представления информации.

Правда, в распределенной ИС "Банк гидрометеорологических данных", действующей на основе режимно-справочных архивных материалов наблюдений гидрометеорологической службы, существуют две задачи, строго говоря, выходящие за рамки *Web*-технологий. Первая, достаточно сложная, задача — актуализация БнД архивными данными за предшествующие годы, сохраняемыми на бумажных, магнитных и других носителях в разных форматах (*ASCII*-, *DBF*-файлы электронных таблиц). Положение усугубляется тем, что требуется автоматизация процесса ввода архивных данных в связи с необходимостью внесения значительных объемов данных в тематические БД за длительный период (с 1800 года) наблюдений за разнообразными гидрометеорологическими показателями. Вторая — связана с пополнениями БнД, поступающими в архив после всестороннего анализа, уточнения сомнительных данных, внесения корректив с региональных систем центров сбора и обработки данных [4, 60].

Сегодня автоматизация процесса ввода архивных данных решена путем создания программного продукта (ПП), удовлетворяющего требованиям независимости от структуры БД, инвариантности к используемым СУБД и про-

граммной платформе. Это обеспечило внесение неоднородной информации с таблиц и позволило бы связать БнД информационным потоком с поставщиками данных для архивов, используя *xml*-документы как универсальный формат для обмена информацией между отдельными системами с различными БД и структурами. Задача создания ПП рассматривается как преобразование неоднородной информации из одного представления в другое с нахождением правил описания структур табличных данных и имеет универсальное решение.

Возможно, дальнейшее развитие ИС, ее широкое использование для расчетов прогнозных характеристик потребует вычислений с привлечением *SOAP* для удаленного вызова процедур *RPC* при обращении к другим национальным и международным БнД природного профиля. В этих случаях целесообразно прибегнуть к *Web*- или другим службам. По мнению авторов, выбор технологии должен определяться задачами и целями, возлагаемыми на распределенную ИС, и соизмеряться с экономической целесообразностью применения того или иного решения. Именно здесь уместно вспомнить работу [5], в которой предлагается аппарат моделирования процесса создания сложных систем, позволяющий связать эту область с технологиями.

ГЛАВА

КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ ШИРОКОГО ПРИМЕНЕНИЯ

2.1. Постановка задачи

Модель единого информационного пространства Украины представляет собой органическое объединение отдельных структур в одно стройное целое. Это справедливо и по отношению к информационным системам. Основой любой ИС служат базы данных (знаний), которые поддерживаются той или иной СУБД. Однако использование высокопроизводительных, с широкими функциональными возможностями, зарубежных СУБД (*IBM* (*Informix*), *Oracle* и др.) во всех структурах вряд ли целесообразно из-за высокой стоимости владения и сложности эксплуатации. Эти СУБД имеют поддержку мультимедийных и *XML* типов данных, включая изображения и видео, приложения оперативной аналитической обработки данных *OLAP* и вспомогательные инструментальные средства (средства проектирования, графические средства просмотра БД, генераторы отчетов и т.д.). Однако, несмотря на лидирующее положение на мировом рынке БД, современные СУБД сложны; ОС, под управлением которых они функционируют, многообразие и мощность ресурсов компьютеров усложняют и затрудняют администраторам управление и мониторинг. Кроме того, их развитие и поддержка весьма

Рис. 2.1. Лидирующие СУБД на Мировом рынке

Возможная территориальная или географическая разобщенность объекта автоматизации и связанная с этим рассосредоточенность информации не всегда требуют применения распределенных СУБД с присущими им следующими свойствами:

- дорогостоящая поддержка, вызванная в основном тем, что распределенные СУБД достаточно сложны в сопровождении, в особенности в администрировании;
- чрезмерная загрузка коммуникационной среды [1], при которой производительность прикладной системы может упасть ниже допустимого предела, поскольку при работе с распределенной СУБД по сети может передаваться большой объем информации, крайне необходимой для качественного управления БД (обеспечение целостности

[1] Размещение информационных ресурсов возле мест их обработки позволяет максимально оптимизировать трафик сети.

данных, поддержка механизма "прозрач-ности" запросов, сообщения о результатах выполнения операций и состоянии узлов сети, в которых расположены БД, и др.);

Рис. 2.2. Информационные ресурсы

• непосредственная реализация гипертекстовых структур в СУБД реляционного типа сложна и избыточна.

Образ ИС должен отвечать поставленной цели, а функциональные возможности СУБД должны соответствовать решаемым задачам.

На верхнем уровне управления, в частности в банковской деятельности, где информация распределена и постоянно меняется, подобного рода системы себя оправдывают. Однако в таких областях деятельности, как Госгидрография, в которой базисные предприятия привязаны к портам Украины, события в своем большинстве не столь динамичны; в службе гидрометеорологии при анализе архивных данных и пр., по-видимому, нецелесообразно использовать такого рода программный про-

дукт. Например, информационный ресурс (рис. 2.2) гидрографической службы Украины складывается из огромного количества материалов, необходимых для навигационно-гидрографического обес-печения мореплавания [62,63]. Это всевозможные справочники, документы по морскому законодательству, пособия, руководст-ва, инструкции, правила, лоции, атласы, навигационные электронные карты, извещения мореплавателям, материалы Всемирной службы навигационных предупреждений и международных морских организаций различной сферы деятельности (по изучению Мирового океана, морскому судоходству, рыболовству и т.д.), техническое и материальное обеспечение Гидрографической службы Украины и др. Навигационно-гидрографическая информация характеризуется разнообразием видов и структур данных, представленных в виде текстов на бумажных носителях (книги, документы); графических материалов (схемы, карты, рисунки, фотографии и т.п.); цифровых (компьютерных) файлов на магнитных носителях; видеоматериалов и т.д.

Очевидно, ИС, включая такой распределенный объект, долж-на по запросам пользователей извлекать из территориально разобщенных служб и предоставлять потребителю необходимую информацию разнопланового характера, используемую для информационного обеспечения процесса управления безопасностью мореплавания, иначе говоря, для информационной поддержки принятия решений на качественно новом технологическом уровне.

Информационное поле может быть представлено документами различного вида: от строго структурированных (реляционная модель данных) до слабоструктурированных, таких, как гипертекстовые и мультимедийные (карты, чертежи, схемы, фрагменты технических фильмов и т.п.). БД в таких системах могут быть территориально или географически распределены и функционировать локально, т.е. сопровождение баз данных, их актуализация ведется по месту их расположения. ИС должна объединить эти локальные, функционирующие независимо друг от друга, БД в единую проблемно ориентированную информационную среду с возможностью предоставления данных разнопланового характера.

Если допустить, что на основе этих территориально разнесенных источников информации БД формируются локально и при этом существующие БД не должны восприниматься пользователем как единое целое (одна логическая БД), то вряд ли целесообразно моделировать сетевые данные и применять мощные зарубежные СУБД, поддерживающие распределенные БД. Альтернативный подход представляется в создании распределенной ИС с сопровождаемыми локально БД и решении вопросов построения механизмов "прозрачного" (т.е. надежного, эффективного и устойчивого) взаимодействия удаленных узлов ИС и централизованного управления ими.

Таким образом, для низовой инфраструктуры Украины назрела необходимость создания экономной ИС широкого применения. В основе последней должна быть оптимизация показателя функциональность/стоимость за счет применения менее дорогостоящих программных продуктов и недорогой поддержки их функционирования. Достичь этого можно выбором соответствующей прогрессивной ИТ, адекватной цели создания ИС и возлагаемым на нее задачам, и сопоставлением экономической целесообразности применения того или иного решения. Экономные ИС могли бы быть применены для обеспечения информационной поддержки принятия решений, направленных на управление хозяйственным комплексом страны на качественно но-вом технологическом уровне.

Важным этапом проектирования такой распределенной ИС является построение ее модели, включающей в себя не только перечень необходимых для ИС сервисов и служб, но и механизмы их взаимодействия в различных сетевых средах. Выбор модели напрямую связан с составом компонентов ИС. Поэтому уже на первых этапах разработки ИС детальная проработка модели является весьма значимым фактором создания распределенной ИС в целом.

В основу такой модели положена идея построения ИС на типовых серверных платформах — "узлах". Каждый из узлов должен быть элементарным полнофункциональным модулем, содержащим все необходимые элементы и сервисы (службы). Узел не всегда может состоять из одного физического сервера, но обязательно должен представ-

лять собой совокупность компонентов, воспринимаемую клиентами и другими узлами как единое целое.

Такое решение позволит максимально быстро создавать распределенные ИС без настройки специфических сервисов на местах. Узел сможет функционировать автономно, например на этапе отладки, предшествующем объединению с другими узлами, входящими в состав распределенной ИС. Кроме того, каждый узел будет содержать все необходимые средства управления как локальными, так и удаленными (расположенными на территориально удаленных узлах) сервисами и службами распределенной ИС. Это снимает потребность в специальных выделенных серверах, реализующих функции управления элементами распределенной ИС, позволяет структурировать и децентрализировать, в случае необходимости, задачи управления, повышая надежность и снижая стоимость системы.

Поясним некоторые определения, вытекающие из выбранного подхода к распределенным ИС.

Базовая программно-инструментальная платформа — совокупность базовых сервисов (служб) и технология их взаимодействия для различных сетевых сред, применяемых при построении распределенных ИС.

Гетерогенная сетевая среда — коммуникационная среда, образованная сетями различной архитектуры в совокупности с размещенными в них сервисами (службами), реализованными на разнообразных программно-аппаратных платформах.

Информационная система широкого применения — распределенная многопользовательская ИС, интегрирующая в БД информационные ресурсы различных форм и моделей представления.

Распределенная информационная система — фиксированное множество территориально разнесенных, сопровождаемых локально баз данных, связанных унифицированными средствами визуализации и управления в проблемно ориентированную информационную среду.

2.2. Модель распределенной информационной системы

Область применения распределенной ИС следующая:

• ИС, интегрирующие в своих базах ресурсы различных форм и моделей представления данных;

• распределенные ИС, в которых приложения не являются распределенными и локально сопровождаемые БД не образуют единую логическую структуру;

• БД, не требующие унифицированных специфических средств сопровождения.

Общие подходы к проектированию ИС [2] сформировались достаточно давно, но пока универсальной технологии их построения не существует. Вместе с тем известны и хорошо разработаны технологии создания крупномасштабных корпоративных сервисов, которые могут быть использованы в качестве распределенных ИС. Каждое из подобных решений ориентировано на

[2] Как правило, это базирующиеся на *WWW Internet*-сервисы, имеющие некоторые свойства, характерные для распределенных ИС, однако в действительности таковыми не являются.

конкретную программную (а часто и аппаратную [3]) платформу и использует сложную, в большинстве случаев уникальную иерархическую модель взаимодействия отдельных компонентов. Хотя подобные системы, как правило, можно интегрировать друг с другом, это требует значительных затрат на дополнительное оборудование и/или программное обеспечение. Такие системы имеют высокую начальную стоимость, требуют для своего обслуживания высококвалифицированный персонал, поэтому применение их экономически целесообразно только в крупномасштабных проектах.

Отсюда следует, что разработка модели распределенной ИС, инвариантной по отношению к программным [4] и аппаратным [5] платформам ее компонентов, является важным этапом на пути создания универсальной технологии построения таких систем, что позволит значительно ускорить и удешевить их создание, сопровождение и развитие.

Для решения этих проблем в разрабатываемую модель были положены следующие концепция и принципы.

1. *Концепция единого интерфейса* как для решения задач, поставленных перед пользователем (возможность доступа к информации любого формата, хранящейся в БД ИС), так и для задач управления/администрирования ИС.

В любой системе удобный и интуитивный пользовательский интерфейс является одним из важнейших компонентов. Поскольку ИС может содержать разнородную по форматам и способам хранения информацию, необходим либо индивидуальный пользовательский интерфейс, либо унифицированный, поддерживающий все используемые в ИС форматы хранения данных и способы организации доступа к ним.

Проблема унификации пользовательского интерфейса в задачах централизованного управления распределенными ИС также весьма актуальна и непроста. Причем дело от-

---

[3] Характерно для решений, предлагаемых *IBM*.

[4] Программная платформа — ОС в совокупности со специфичным для нее системным ПО, а также прикладное ПО, реализующее сервисы распределенной информационной системы.

[5] Аппаратная платформа — архитектура физического сервера.

нюдь не в «эстетичности» такого интерфейса, а в простоте и эффективности его применения пользователями, не имеющими квалификации сетевого и/или системного администратора.

Реализация концепции единого интерфейса позволит, с одной стороны, легко наращивать количество рабочих мест пользователей с минимальными затратами, с другой — существенно ускорить подготовку как пользователей, так и сопровождающего персонала (администраторов) ИС.

2. *Принцип взаимодействия* компактных элементов [6] распределенной ИС на уровне протоколов. Это означает, что на программно-аппаратную платформу как ИС в целом, так и отдельных ее компонентов не накладывается жестких ограничений.

3. *Принцип максимального использования* стандартных программных компонентов. Это позволит значительно ускорить проектирование и внедрение создаваемых ИС.

4. *Использование Internet/intranet-технологий* в основных сервисах ИС. Поскольку данные технологии получили статус стандарта де-факто для глобальных ИС, их применение решает проблему интеграции разрабатываемых ИС в глобальное информационное пространство. Именно поэтому были использованы стандартные для *Internet* протоколы и механизмы взаимодействия компьютерных систем (*Internet* выбран каналом связи для узлов распределенной ИС).

Предложенные решения обеспечат:

- инвариантность к разным программно-аппаратным платформам;
- интеграцию информационных ресурсов разных форм и моделей представления данных и доступ к ним через единый *Web*-интерфейс;
- объединение ИС с Мировым информационным сообществом;
- эффективный поиск информации;

[6] Компактный элемент — БД, сервисы (службы) и другие компоненты ИС, территориально размещенные в одном месте.

• незначительные затраты на эксплуатацию ИС, обучение пользователей и персонала, сопровождающих систему, по сравнению с внедрением зарубежных систем аналогичного профиля;

• надежную базу для постоянной модернизации и развития ИС благодаря тому, что рынок программного обеспечения и технологий для *Internet/intranet* стремительно развивается.

Изложенные принципы создают условия для построения распределенных экономичных систем с предоставлением информации различного характера.

2.3. Модель типового "узла" распределенной ИС

По аналогии с локальными вычислительными сетями (ЛВС) возможны два подхода к построению распределенных ИС: модель с выделенными серверами и одноранговая модель. На этом, однако, все аналогии с ЛВС заканчиваются. Критерии выбора модели для ИС должны быть иными, поскольку гетерогенная сетевая среда, в которой система должна функционировать, по надежности значительно уступает современным архитектурам ЛВС. Целесообразно, чтобы каждый элемент распределенной системы содержал в себе все используемые сервисы, а также все необходимые средства для управления такой ИС. В рамках модели такой элемент называется *узлом*. Это означает, что даже в случае нарушения связи между регионами, в которых находятся элементы распределенной ИС, или отдельными узлами она остается работоспособной (по крайней мере, частично) и управляемой. Кроме того, данная модель отвечает новейшим тенденциям построения ИС, в ней используется одноранговая модель *Internet* (в отличие от модели *IBM/Microsoft*, ориентированной на работу с выделенными серверами). О поддержке такой концепции в построении ИС недавно объявила корпорация *Sony* — один из лидеров на рынке ИТ. По-видимому, это решение существенно повлияет на выбор стандартной архитектуры для глобальных распределенных ИС ближайшего будущего, так как *Sony* имеет значительный потенциал для продвижения своих технологий (например, доходы *Sony* в 3,5 раза превышают доходы *Microsoft* [64]).

Программные компоненты узла распределенной ИС

Программные компоненты, необходимые для построения распределенной ИС (исключая ОС), делятся на четыре основные группы:

- информационные сервисы;
- службы управления / администрирования;
- коммуникационные службы;
- вспомогательные службы.

Информационные сервисы обеспечивают доступ клиентов к информации, хранящейся в БД ИС, а также служат для организации интерактивного взаимодействия с пользователем (например, в задачах контекстного поиска). Сюда же относятся СУБД

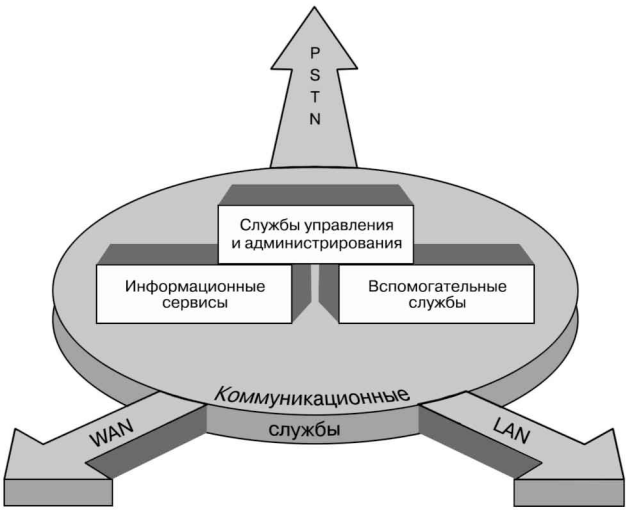


Рис. 2.3. Архитектура узла ИС: *LAN* — локальные сети; *WAN* — глобальные сети (*Internet*); *PSTN* — телефонные линии

и средства организации взаимодействия между различными сервисами данной группы.

Службы управления/администрирования предназначены для выполнения задач удаленного системного администрирования и управления информационными сервисами.

Коммуникационные службы обеспечивают связь между удаленными узлами ИС через глобальную сеть (*Internet*), а также предоставляют доступ к ИС по выделенным и телефонным линиям связи для удаленных клиентов.

На рис. 2.3 изображена иерархическая структура сервисов (служб) узла ИС относительно базового (с точки зрения взаимодействия с клиентами) уровня — коммуникационных служб и протоколов.

Отметим, что четкую границу между первыми двумя группами сервисов (служб) провести нельзя. Не вдаваясь в подробности, заметим, что это положение является следствием кон-

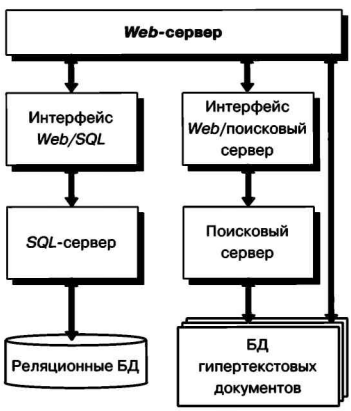


Рис. 2.4. Информационные сер-висы узла ИС

цепции данной модели — обеспечение унифицированного пользовательского интерфейса как для доступа к разнообразной информации, так и для задач управления/администрирования.

Информационные сервисы. В рассматриваемой модели основные информационные сервисы реализуются на основе *Web*-серве-ра, *SQL*-сервера и поискового сервера (рис. 2.4). *Web*- сервер [7] предоставляет доступ клиентам к информационной базе документов: гипертекстовые или *html*-документы. *html*-Докумен-ты представляют собой тексты в стандарте *ASCII*, содержащие команды специального платформонезависимого языка *HTML* (*hyper text markup language*) — языка разметки гипертекста. Использование формата *HTML* позволяет интегрировать в единое целое не только текстовые фрагменты, но и средства мультимедиа. В спецификацию языка *HTML* вклю-

[7] В общем случае сервером называется любой постоянно выполняющийся системный процесс. В терминологии *UNIX* для обозначения сервера используется термин "демон" (*Daemon*) в сочетании с названием протокола, поддержку которого он реализует. Например, в качестве синонима для обозначения *Web*-сервера часто используется *HTTP Daemon (HTTPD)* или, иначе, "демон" поддержки протокола передачи гипертекста.

чены специальные дескрипторы, предназначенные для создания заполняемых пользователем форм. С помощью этих форм пользователь формирует необходимый ему запрос, предположим, к реляционной БД из *html*-документа. Результат запроса оформляется также в виде *html*-документа и возвращается пользователю [65—67].

Таким образом, гипертекстовые страницы являются не просто унифицированным средством представления разнообразной информации, хранящейся в базах данных ИС. *Web*-сервер вместе с рядом специфических программных компонентов реализует единый интерфейс конечного пользователя (задачи поиска/отображения информации) и администратора ИС (задачи системного администрирования и управления сервисами ИС). Он также (совместно с различными компонентами ОС) обеспечивает аутентификацию клиентов и реализацию разнообразных механизмов защиты информации.

SQL-сервер выбран в качестве СУБД для распределенной ИС по двум основным причинам: во-первых, он гибок и универсален как в задачах, характерных для клиентов (поиск/измене-ние/ввод информации), так и в задачах сопровождения реляционных БД; во-вторых, он позволяет достаточно просто реализовать клиентскую часть реляционной БД в контексте единого пользовательского интерфейса — *Web*-браузера.

Поисковый сервер осуществляет просмотр информационной базы документов с целью нахождения элементов с заданными пользователем свойствами. Его архитектура будет зависеть от конкретной ИС, хотя основные принципы построения остаются неизменными.

Служба управления и администрирования. Службы управления/администрирования предназначены для дистанционного управления основными информационными сервисами, расположенными на узле распределенной ИС, и удаленного системного администрирования. Все службы этой группы либо обеспечивают интерфейс с пользователем через *Web*-сервер, либо сами являются сетевыми серверными приложениями, использующими для взаимодей-

ствия с клиентом механизм *Sockets*[8] [68]. В последнем случае службы должны поддерживать протокол *HTTP* (*hyper text transfer protocol*) — протокол обмена гипертекстом [8].

Возможны различные подходы к реализации таких служб с точки зрения механизма взаимодействия клиента и контролируемого объекта [70, 71]. Наиболее целесообразным представляется использование сценариев *CGI* [9] в "чистом" виде, т.е. без комбинации с другими возможными механизмами. Преимущества данного подхода — надежность (сценарий *CGI* не может повлиять на устойчивую работу сервера) и платформонезависимость (правильно написанные сценарии переносимы на различные платформы и работают с большинством распространенных *Web*-серверов). Поскольку *CGI* опирается на заполняемые пользователем формы [65—67], дескрипторы которых включены в спецификацию языка *HTML*, он совместим с любыми *Web*-браузерами, в том числе с текстовыми типа *Lynx*. Это дает возможность управлять ИС практически с любых клиентских платформ, включая мобильные (блокнотные ПК и даже электронные записные книжки, оснащенные модемом и программным эмулятором терминала), что необходимо, например, для оперативного устранения неполадок в работе ИС.

Основные недостатки *CGI* — снижение производительности сервера и ограниченная функциональность – в данном случае не существенны. Задачи управления/администрирования составляют незначительную часть общего объема задач, решаемых ИС, следовательно, реальное снижение производительности системы невелико.

---

[8] Забегая вперед, поясним: термином "*socket*" обозначают уникальный идентификатор, присвоенный каждому соединению *TCP/IP*. Определенному виду сервиса соответствует свой номер логического *TCP* порта, который в совокупности с *IP*-адресом называется номером сокета (*socket number*). Он уникально идентифицирует сервис в сети. Прикладные программы, использующие протокол *TCP/IP*, устанавливают связь с определенным сервисом по номеру логического *TCP* порта. *Sockets* — стандарт программного интерфейса.

[9] *CGI (common gateway interface)* — механизм передачи параметров из *HTML*-формы приложению (сценарию *CGI*). *CGI*-сценарии обеспечивают доступ средствами *www* к реляционным БД [65—67].

Ориентированность *CGI* на обработку форм запросов и генерацию ответов также не является препятствием — по существу все задачи управления и контроля можно представить как последовательность типа запрос-ответ.

Механизм взаимодействия клиента и контролируемого объекта, расположенного на узле ИС, поясняет рис. 2.5.

Контролируемыми объектами могут быть информационные сервисы (*Web*-сервер, поисковый сервер, *SQL*-сервер), конфигурационные параметры различных компонентов ОС и устройств, а также статистическая информация и регистрируемые события.

Коммуникационные службы. Они предназначены обеспечить прозрачное взаимодействие клиентов с узлами ИС и узлов друг с другом. Для реализации такой задачи в рамках гетерогенной сетевой инфраструктуры распределенной ИС коммуникационные службы, очевидно, должны иметь достаточно сложную многоуровневую архитектуру.

Как отмечалось выше, одним из основных принципов рассматриваемой модели является использование *Internet*-техно-логий и, как следствие, стандартных для *Internet* коммуникационных протоколов семейства *TCP/IP* (*transmission control protocol/ Internet protocol*) [72, 73]. Реализуют поддержку этих протоколов коммуникационные службы. Поэтому многоуровневая архитек-

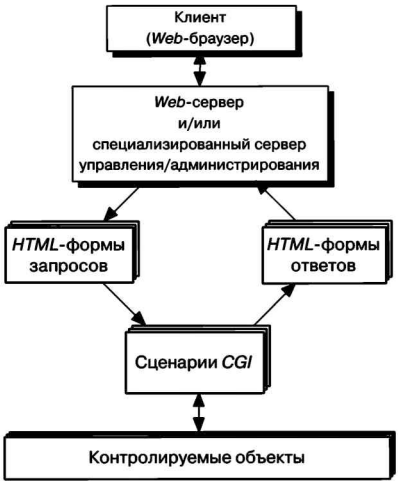


Рис. 2.5. Управление узлом ИС

тура коммуникационных служб имеет непосредственную тесную связь с многоуровневой моделью стека протоколов *TCP/IP* [74] — промышленного стандарта для глобальных сетей.

Рассмотрим место коммуникационных служб распределенной ИС в многоуровневой модели взаимодействия открытых систем *ISO/OSI* [10] [75] и стека протоколов *TCP/IP* (рис. 2.6).

Нижний, IV, уровень стека *TCP/IP* соответствует физическому (1) и канальному (2) уровням модели *OSI* и поддерживает разнообразные стандарты [11], относящиеся к этим уровням, в частности, *ethernet* (локальные сети)

[10] Данная модель не имеет прямого отношения к стеку *TCP/IP*, поэтому соответствие уровней достаточно условно. Однако введение такого соответствия необходимо, поскольку модель *ISO/OSI* принята в качестве стандарта взаимодействия двух систем (компьютеров) через вычислительную сеть и все современные сетевые технологии и системы рассматриваются в ее контексте.

[11] Перечень стандартов и их состав в настоящее время не регламентированы.

и *PPP* (*Point-to-Point Protocol*) [76] — протокол двухточечного соединения, используемый в глобальных сетях. На этом уровне используются драйверы или

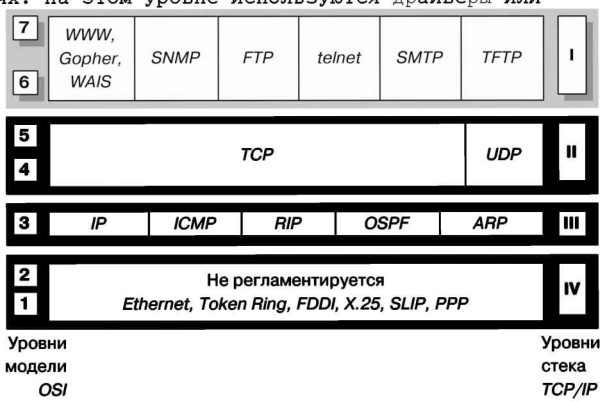


Рис. 2.6. Стэк TCP/IP

модули (если ядро ОС имеет модульную структуру), поддерживающие разнообразные физические устройства и обеспечивающие инкапсуляцию [12] *IP*-пакетов в кадры соответствующей физической сети (рис. 2.7).

---

[12] Инкапсуляция (*encapsulation*) — продвижение пакета информации (данные с соответствующим сетевым заголовком) по уровням сверху вниз с добавлением заголовка каждого нового уровня. В модели межсетевого соединения взаимодействие *TCP* и протоколов нижнего уровня не специфицировано. Однако должен существовать механизм, который обеспечивал бы асинхронную передачу информации от одного уровня к другому. Результатом работы этого механизма является инкапсуляция протокола более высокого уровня в тело протокола более низкого уровня. Каждый *TCP*-пакет вкладывается в пакет протокола ниже лежащего уровня, например *IP*. Получившаяся таким образом дейтаграмма содержит в себе *TCP*-пакет аналогично тому, как *TCP*-пакет содержит пользовательские данные.

Метод согласования сетей на уровне транспортных протоколов сводится к тому, что инкапсуляция может быть использована, ко-

Уровни II и III — это, как правило, "монолитный" компонент стека. Их состав неизменен. Уровень III — это уровень межсетевого взаимодействия, который занимается передачей пакетов, используя транспортные технологии сетей различной при-

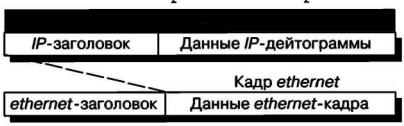


Рис. 2.7. *Инкапсуляция IP-пакетов в ethernet-кадры*

роды (*ethernet*, *X.25*, *ATM* и др.). В качестве основного протокола сетевого уровня (в терминах модели *OSI*) в стеке используется протокол *IP* (*RFC*791), который является дейтаграммным протоколом: при передаче информации по протоколу *IP* каждый пакет передается от узла к узлу сети и обрабатывается в узлах независимо от других пакетов. Уровень II называется основным. На этом уровне функционируют протокол управления передачей *TCP* (*transmission control protocol*) и протокол дейтаграмм пользователя *UDP* (*user datagram protocol*). Протокол *TCP* обеспечивает надежную передачу сообщений между удаленными прикладными процессами за счет образования виртуальных соединений. Протокол *UDP* обеспечивает передачу прикладных пакетов дейтаграммным способом, как и *IP*, и выполняет функции связующего звена между сетевым протоколом и многочисленными прикладными процессами [72—74].

На I уровне располагаются всевозможные сервисы, реализующие протоколы прикладного уровня (*Web-*, *FTP-*, *SMTP*-сервер и др.). Их состав, естественно, может варьироваться. На первый взгляд все коммуникационные службы должны были бы охватывать только I уровень, поскольку все остальные уровни, по существу, являются частью ОС. Однако такое представление является слишком упрощенным, по причинам, рассмотренным ниже.

гда две сети с одной транспортной технологией необходимо соединить через сеть, использующую другую транспортную технологию [72—74].

В предлагаемой модели для соединения узлов и клиентов используются ЛВС (*LAN*) *ethernet* и глобальные (*WAN*) сети. В последнем случае, а также для соединения по выделенным или коммутируемым телефонным линиям используется протокол *PPP*. Если архитектура *ethernet* реализована на уровне аппаратного обеспечения и драйверов, которые можно считать частью ОС, то с *PPP* дело обстоит иначе. Реализующая его служба состоит из нескольких частей. Одна часть представляет собой драйвер и обеспечивает инкапсуляцию дейтаграмм для последовательных линий связи (IV уровень *TCP*-стека). Другие — выполняют разнообразные функции, в частности поддержку аутентификации, и реализуются службами прикладного уровня. Вместе они образуют как клиентскую, так и серверную часть, т.е. обеспечивают входящие и исходящие соединения [13] — *PPP*-сервер или службу удаленного доступа [14] *RAS* (*remote access service*).

К коммуникационным службам относится также *Proxy*-сер-вер. Его основное назначение — обеспечить связь локальных и удаленных клиентов узла ИС с другими узлами. Рассмотрим принцип работы *Proxy*-сервера.

Узел ИС имеет несколько сетевых интерфейсов (*ethernet*, последовательные порты) для соединения с ЛВС, подключения удаленных клиентов и связи с *Internet*. Сетевые интерфейсы компьютера отделены друг от друга, и, таким образом, трафик одного интерфейса недоступен другому интерфейсу, если только нет специального программного обеспечения, которое перенаправляет трафик с одного интерфейса на другой. Такой процесс называется маршрутизацией. Для того чтобы маршрутизация работала, все соединяемые посредством *Internet* компьютеры должны находиться в одном адресном пространстве, т.е. должны иметь зарегистрированные *IP*-адреса.

---

[13] Разделение клиент—сервер здесь достаточно условно, поскольку *PPP* — одноранговый протокол. Для установления соединения не имеет значения, какая сторона (клиент или сервер) его инициирует.

[14] Службой удаленного доступа часто называют сервис, обеспечивающий только входящие соединения.

Используя *Proxy*-сервер, можно "переключать" трафик между сетевыми интерфейсами. Следует, однако, подчеркнуть, что *Pro-xy*-сервер не является маршрутизатором *IP*-пакетов, это программный шлюз - своего рода ретранслятор услуг, работающий не на уровне сетевого протокола, а на уровне прикладных программ пользователя и соответствующих им протоколов. Он принимает запрос от какой-либо машины из локальной сети и передает его по другому сетевому интерфейсу. *Proxy*-сервер должен поддерживать протоколы сервисов, доступ к которым он обеспечивает. Например, широко распространенный *Proxy*-сервер *Squid* для ОС *Unix* обеспечивает доступ только к *FTP*- и *HTTP*-серве-рам, в то время как *WinGate*, ориентированный на ОС *Windows*, поддерживает также протоколы *Telnet, POP, SMTP, RealAudio* и др. Таким образом, соединяемые с *Internet* через *Proxy*-сервер компьютеры могут иметь любые *IP*-адреса, однако рекомендуется использовать адреса из специально отведенного диапазона [77]. Вследствие использования незарегистрированных *IP*-адресов такое соединение получается "односторонним", т.е. никакие сервисы, расположенные за *Proxy*-сервером, из *Internet* недоступны.

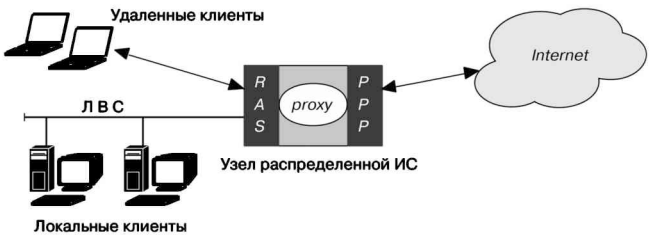


Рис. 2.8. Коммуникационные службы узла распределенной ИС

На рис. 2.8 условно изображены коммуникационные службы узла ИС и их взаимодействие с клиентами.

Вспомогательные службы. Службы [15], не являющиеся *абсолютно* необходимыми для нормальной эксплуатации базового варианта распределенной ИС, называются вспомогательными. Они в значительной части специфичны для конкретной платформы, на которой реализован узел ИС, и состав их может варьироваться. К таким относятся службы:

- используемые на этапе отладки ИС или для восстановления после сбоев;
- расширяющие функциональные возможности базового варианта ИС или предоставляющие дополнительные удобства ее пользователям;
- улучшающие функциональные характеристики компактного элемента распределенной ИС.

В первую очередь выделим в данной группе службы, которые не являются характерными для какой-либо платформы. Это *FTP-*, *SMTP-* и *POP*-серверы, а также сервер управления электропитанием [16].

---

[15] Как следует из определения модели распределенной ИС, все службы данной группы должны использовать для связи с соответствующими им клиентами протоколы семейства *TCP/IP*. Исключение могут составлять лишь службы, используемые только локальными (относящимися к компактному элементу ИС) клиентами.

[16] В терминологии *UNIX* обычно называется *PowerD (Power Daemon)*. Основное назначение — отслеживать наличие напряжения в сети электропитания посредством взаимодействия с источником

Основное назначение *FTP*-сервера в рамках задач, связанных с управлением/администрированием, — обеспечить передачу [17] файлов от одного узла ИС к другому. *SMTP*- и *POP*-серверы предназначены для обмена электронной почтой между пользователями ИС. *SMTP*-сервер обеспечивает пересылку почты от отправителя адресату, а *POP*-сервер организует доступ клиентов к "почтовым ящикам". Кроме того, они могут использоваться для передачи сгенерированных системным ПО сообщений о различных неполадках заранее определенным адресатам (эта возможность широко реализована в системах на платформе ОС *UNIX*).

Среди специфичных для *UNIX* необходимо отметить *telnet*- и *NFS*-серверы. *Telnet*-сервер реализует одноименный протокол связи с удаленным текстовым терминалом. Поскольку практически все задачи, связанные с администрированием ОС *UNIX*, могут быть выполнены с использованием интерфейса командной строки, данный сервер будет чрезвычайно полезным средством на этапе отладки удаленных узлов ИС.

NFS-сервер вместе с соответствующими клиентами является стандартным средством построения распределенных файловых систем, что повышает общую производительность и надежность систем хранения информации.

Это наиболее важные вспомогательные сервисы, которые имеют практическую направленность в рамках рассматриваемой модели.

2.4. Взаимодействие компонентов распределенной ИС

бесперебойного питания (*UPS*). В случае исчезновения электропитания *UPS* переходит в режим работы от встроенных аккумуляторов и посылает соответствующий сигнал *PowerD*. Последний принимает решение о завершении работы ("выгрузке" ОС) с таким расчетом, чтобы это произошло до разряда аккумуляторов *UPS*. *PowerD* является неотъемлемым элементом всех критичных к сбоям систем, поскольку восстановление работы большинства многозадачных ОС зависит от корректного завершения.

[17] Здесь и далее под передачей подразумевается пересылка файла от клиента серверу (*Upload*), а под приемом — от сервера клиенту (*Download*). В качестве клиента (как для приема, так и для передачи) может использоваться *Web*-браузер. Однако единственным стандартным средством, поддерживающим возможность передачи файла от клиента серверу, является *FTP*-сервер.

Взаимодействие компонентов внутри узла. Как отмечалось выше, узел ИС не обязательно реализуется на одном аппаратном сервере. Сервисы узла ИС могут располагаться в гетерогенной сети с достаточной для их прозрачного взаимодействия пропускной способностью. При этом механизм взаимодействия основных информационных сервисов должен, во-первых, обеспечивать прозрачное взаи-модействие сервисов в гомогенной и гетерогенной сетевых средах, а также в операционной среде узла ИС, во-вторых, использовать стандартные средства, позволяющие просто и максимально эффективно реализовать програм-мный интерфейс между сервисами.

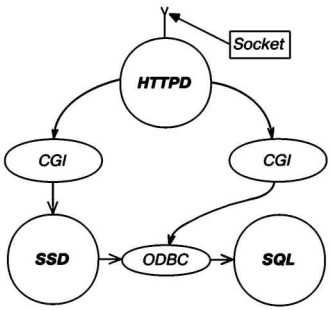


Рис. 2.9. Взаимодействие информационных сервисов распределенной ИС, размещаемых на одном физическом сервере

Сначала рассмотрим механизм взаимодействия информационных сервисов для случая, когда все они выполняются в однородной операционной среде на одном физическом сервере (рис. 2.9). *Web*-сервер (*HTTPD*) обеспечивает интерфейс клиентов (*Web*-браузеров) с информационными сервисами (*SSD* и *SQL*) и взаимодействует с последними с помощью специально разработанных *CGI*-скриптов. Для обращения поискового сервера (*SSD*) и клиентов к *SQL*-серверу используются драйверы и библиотека функций стандартного интерфейса *ODBC*, позволяющего приложению осуществлять доступ к различным СУБД [78]. Интерфейс *ODBC* реализован для различных платформ (включая *Windows* и *UNIX*). Его преимущество — простота организации эффективного взаимодействия приложений с СУБД, недостаток — возможность взаимодействия с СУБД только в гомогенной сетевой среде с использованием однотипных клиент-сер-верных платформ. Это обусловлено тем, что стандартная сетевая библио-

тека, используемая *ODBC* для связи с *SQL*-сервером, как правило [18], реализует механизм межпроцессного взаимодействия посредством именованных каналов (*named pipes*) [19]. Поэтому для клиентов, использующих *ODBC*, необходим доступ к файловой системе *SQL*-сервера, что трудно осуществить в случае, если клиент и сервер реализованы на разных платформах.

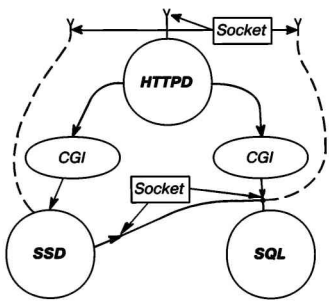


Рис. 2.10. Универсальный механизм взаимодействия информационных сервисов распределенной ИС с использованием механизма *Sockets*

Если информационные сервисы находятся на различных физических серверах (они могут отличаться программно-аппаратной платформой), механизм их взаимодействия должен быть универсальным (рис. 2.10). Каждый информационный сервис принимает запросы клиентов (ими могут быть другие информационные сервисы), используя механизм *Sockets*. Напомним, это предполагает, что каждый информационный сервис имеет определенный номер логического *TCP*-порта, который в совокупности с *IP*-адресом называется номером сокета (*Socket*). Он уникально идентифицирует сервис в сети. Используя этот механизм, сервисы и клиенты могут прозрачно взаимодействовать как в однородной операционной среде, так и в любой гетерогенной сетевой среде, поддерживающей протокол *TCP/IP*. Кроме того, данный механизм, в отличие от рассмотренного выше, позволяет клиентам обращаться к информационным сервисам непосредственно, — минуя *HTTPD*. К недостаткам этого механизма следует отнести некоторое снижение эффективности взаимодейст-

[18] В частности, это справедливо для реализаций *ODBC* в *ОС Windows*.

[19] *Named pipes*, или *FIFO*, — специальные файлы, используемые для обмена данными между независимыми процессами.

вия сервисов, что обусловлено избыточностью информационного обмена с использованием протоколов *TCP/IP* (помимо данных передается большой объем служебной информации), и относительную сложность реализации.

Учитывая особенности двух рассмотренных механизмов взаимодействия информационных сервисов, можно прийти к выводу, что в узле распределенной ИС они должны использоваться совместно. С одной стороны, это позволит добиться максимально эффективного взаимодействия между информационными сервисами, размещенными в однородной операционной среде, с другой — обеспечит возможность взаимодействия сервисов в гетерогенной сетевой среде.

Помимо совместного использования рассмотренных выше механизмов, возможна также их "комбинация". Пример такой ком-

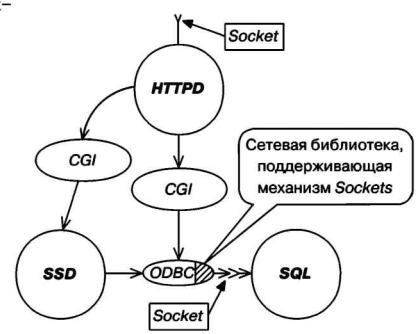


Рис. 2.11. Использование нестандартной сетевой библиотеки в *ODBC*

бинации приведен на рис. 2.11. Здесь для связи с *SQL*-сервером *ODBC* использует специальную сетевую библиотеку, поддерживающую механизм *Sockets*. В общем случае *ODBC* позволяет одновременно использовать различные сетевые библиотеки.

Взаимодействие узлов друг с другом. Взаимодействие узлов распределенной ИС представляет собой комплекс-

ную задачу. С одной стороны, необходимо рассмотреть механизмы взаимодействия коммуникационных служб, с другой — такие механизмы можно детально рассматривать только в контексте конкретной сетевой архитектуры ИС. Существенное влияние на архитектуру ИС оказывают также задачи управления и защиты информации, которые должны рассматриваться в комплексе вопросов взаимодействия узлов ИС.

Рассмотрим два базовых варианта механизма взаимодействия узлов распределенной ИС, отличающихся реализацией сетевой инфраструктуры. Взаимодействие клиентов с узлами распределенной ИС в свою очередь будет зависеть от примененного механизма взаимодействия узлов.

Оба варианта предполагают использование коммуникационной инфраструктуры *Internet* для организации взаимодействия сервисов распределенной ИС. Физическое соединение обеспечивается аппаратурой коммуникационных служб и представляет собой выделенную двух- или четырехпроводную телефонную линию, объединяющую узел с провайдером услуг *Internet* (*ISP*) по протоколу *PPP*. Данный тип соединения является предпочтительным с точки зрения материальных затрат, но архитектура коммуникационных служб позволяет реализовать практически любой тип соединения при наличии соответствующего физического интерфейса. При достаточной пропускной способности (типичное значение 115,2—128 *Kbps* в зависимости от физического интерфейса) выделенная линия значительно дешевле, проще в реализации и доступнее (обеспечивается практически всеми *ISP*) альтернативных вариантов (*Ethernet, ISDN*, беспроводные микроволновые каналы связи).

Первый базовый вариант предполагает построение сетевой инфраструктуры узла в виде двух изолированных контуров. На физическом и канальном уровнях каждый контур представляет собой сегмент сети *Ethernet*. Первый контур находится в адресном пространстве *Internet* и объединяет информационные сервисы. При этом сервисы узлов взаимодействуют друг с другом непосредственно, используя механизм *Sockets*. Локальные (по отношению к сервисам узла) клиенты находятся в изолированном адресном пространстве второго контура. Сервисы удален-

ного доступа (*RAS*), обеспечивающие работу удаленных клиентов (преимущественно по коммутируемым телефонным линиям), также подключены к этому контуру. "Мостом" между двумя контурами является *Proxy*-сервер, обеспечивающий доступ всех клиентов к информационным сервисам узла, а также к удаленным узлам. Кроме того, использование *Proxy*-сервера с гибким механизмом кэширования позволяет существенно повысить эффективность работы клиентов с удаленными узлами.

Как разновидность этого варианта может использоваться подключение всех информационных сервисов узла к обоим контурам одновременно (рис. 2.12). Такая топология сетевой инфраструктуры узла позволяет повысить эффективность обслуживания большого числа локальных клиентов. Характерной особенностью данного варианта является возможность непосредственного взаимодействия с информационными сервисами узла из зоны *Internet*. Это удобно в случае, если потенциальный клиент уже имеет соединение с *Internet*.

Второй вариант базируется на технологии виртуальных частных сетей (*VPN*) [20] [79] и направлен на максимальную защиту

[20] В общем случае *VPN* — это объединение локальных сетей или отдельных компьютеров, подключенных к общедоступной глобальной сети *Inter-net*, в единую виртуальную сеть, обеспечивающую конфиденциальность и целостность передаваемой по ней информации, а также безопасное вхождение в нее удаленных пользователей.

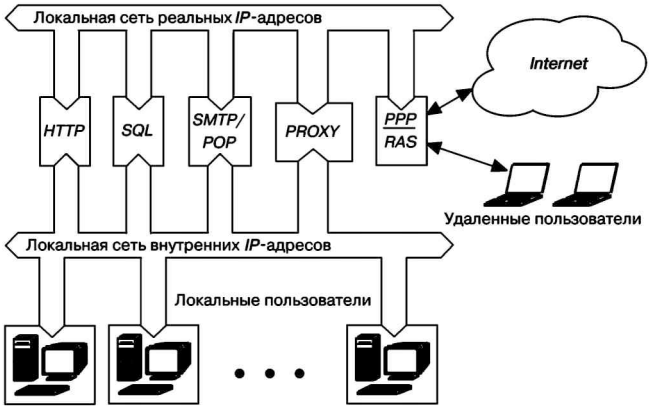


Рис. 2.12. Первый базовый вариант взаимодействия узлов

информационного обмена в распределенной ИС. При этом как информационные сервисы узла, так и клиенты находятся в изолированном адресном пространстве. Исключение составляют почтовые службы узла, которые находятся в адресном пространстве *Internet* и обеспечивают получение/отправление сообщений электронной почты внутри распределенной ИС и за ее пределами. Ключевым звеном здесь являются коммуникационные службы, расположенные в адресном пространстве *Internet*. Они обеспечивают организацию виртуальных выделенных каналов между узлами и криптографическую защиту передаваемых данных (рис. 2.13). Эти службы создают двусторонний канал связи между парой узлов распределенной ИС, следовате-льно, каждый узел образует N-1 каналов, где N — число узлов распределенной ИС. Коммуникационные службы комбинируе-

ются с *Proxy*-сервером, обеспечивая клиентам доступ к сервисам и службам *Internet*.

Естественно, при таком механизме взаимодействия непосредственный доступ из зоны *Internet* к сервисам распределенной ИС невозможен, но в случае необходимости один или несколько узлов могут служить "порталами" для доступа к сервисам распределенной ИС. Поскольку сервисы распределенной ИС доступны через *web*-интерфейс, для этого достаточно вынести *Web*-серверы узлов — "порталов" — в адресное пространство *Internet*.

Выводы

Разработанная модель распределенной гетерогенной системы [58, 61] является основой для создания универсальной технологии построения масштабируемых, платформонезависимых ИС широкого применения. Модель базируется на концепции типовых узлов, инвариантных по отношению к программно-аппа-ратным платформам.

Использование *Web*-технологий для представления информации и интерактивного взаимодействия дает возможность в сжатые сроки и с минимальными затратами обеспечить сопровождение и работу конечных пользователей с сервисами ИС, а также интеграцию созданной системы в существующее информационное пространство.

Предусмотренные моделью механизмы взаимодействия компонентов позволяют оперативно наращивать функциональность ИС, интегрировать ее с существующими БнД. Механизмы сетевого взаимодействия на основе открытых стандартов реализуют защищенное мультипротокольное взаимодействие сервисов, служб и клиентов в *IP*-сетях, обеспечивая надежную базу для развития системы на протяжении ее жизненного цикла.

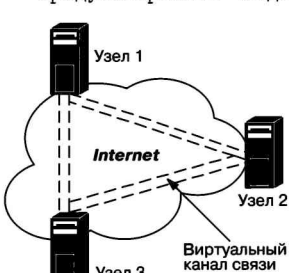


Рис. 2.13. Виртуальные выделенные каналы

Основным достоинством модели является низкая начальная стоимость создаваемых на ее основе ИС, что позволяет проектировать систему в сжатые сроки.

ГЛАВА

КРИТЕРИИ ВЫБОРА ПРОГРАММНОЙ ПЛАТФОРМЫ ДЛЯ СЕРВИСОВ И СЛУЖБ РАСПРЕДЕЛЕННОЙ ИС ШИРОКОГО ПРИМЕНЕНИЯ

Как было отмечено в гл. 2, одним из преимуществ модели распределенной гетерогенной ИС является платформонезависимость ее базовых компонентов — узлов. Это означает, что информационные сервисы и службы узла (коммуникационные, управления/ администрирования и т.п.) могут быть реализованы на различных программно-аппаратных платформах — важно лишь, чтобы они поддерживали механизмы взаимодействия, предусмотренные моделью. Вместе с тем разработка модели является частью задачи создания технологии построения распределенной ИС и, следовательно, носит практическую направленность. Поэтому необходимо сформулировать основные критерии выбора программной платформы для сервисов и служб распределенной ИС и в их контексте провести сравнительный анализ этих платформ.

На основе проведенного анализа по каждой из перечисленных выше групп сервисов и служб можно принять решение о возможности использования ОС в качестве интегрированной платформы для сервисов и служб узла распределенной ИС или плат-формы для отдельных сервисов и/или служб.

С практической точки зрения интерес представляют наиболее распространенные

и перспективные (на отечественном рынке) программные и аппаратные платформы. К программным платформам следует отнести системы на базе ОС *Windows NT* и клонов *UNIX* (прежде всего *Linux*), к аппаратным — *IBM PC*.

3.1. Особенности *Windows NT* и ОС клона *UNIX* в контексте задач, решаемых распределенной ИС

Вопрос о выборе серверной платформы на основе *Microsoft Win-dows NT Server* или одной из ОС клона *UNIX* (в него входят *AIX, BSDI, Digital UNIX, FreeBSD, IRIX, Linux, NetBSD, OpenBSD, Solaris* и др.) поднимается достаточно часто. Вместе с тем сравнительная характеристика ОС в общем случае проблематична — каждая система имеет свои особенности, ограничения, преимущества и недостатки. Корректное сравнение возможно лишь в случае, если реализованные на базе той или иной ОС программные комплексы имеют одинаковое функциональное назначение. При этом следует учитывать, что каждая система разрабатывалась для решения определенного круга задач: можно, конечно, на ОС *Windows 95/98* установить *Web*-сервер, но, поскольку производитель изначально позиционировал ее как платформу для домашнего использования и игр [80], рассматривать такие аспекты ее применения в сравнении со специализированными *Web*-серверами бессмысленно. Из множества статей и обзоров [80—92], посвященных критике и сравнению *Windows NT Server* и ОС клона *UNIX* (в частности, *Linux*), к сожалению, большинство носит субъективный характер и отражает, как правило, личное мнение автора. В дискуссию включились не только отдельные сторонники либо противники той или иной системы, но и целые компании. Так, *Microsoft*, долгое время игнорировавшая существование *Linux*, выпустила документ с красноречивым названием "Мифы Линукс", затрагивающий и ОС клона *UNIX* в целом [81]. Отдельные положения в нем справедливы, некоторые спорны, иные не соответствуют действительности. Например, утверждение, что "*Linux* является разновидностью *UNIX* и, как следствие, несет на себе бремя давно устаревших технологий, так как его архитектура была разработана 30 лет назад"

[82], может вызвать лишь недоумение. В общем, среди подобных материалов можно встретить диаметрально противоположные мнения, причем дискуссии ведутся как в специализированных изданиях, так и в СМИ. Критерии, на основе которых авторы работ [80—92] приходят к тому или иному выводу, существенно различаются и могут быть условно разделены на четыре группы.

В первую группу входят такие критерии, как простота установки, освоения и использования системы. Эти критерии применимы для сравнения "персональных настольных систем". Поскольку позиционирование ОС *Linux* (не говоря уже о других ОС клона *UNIX*) как персональной системы [1] нельзя считать верным, практическая ценность выводов, опирающихся только на эти критерии [83], представляется сомнительной.

Ко второй группе критериев относятся экономические показатели — стоимость дистрибутива ОС, стоимость сопровождения, совокупная стоимость владения[2] (*total cost of ownership, TCO*) и т.п. Эти показатели важны, однако серьезных исследований относительно долей затрат, составляющих *TCO* рассматриваемых ОС на отечественном рынке, насколько нам известно, не проводилось. Исследования такого рода проводились для рынка США [80], но они не учитывали потерь, связанных с вынужденными простоями оборудования в результате сбоев, т.е. фактора надежности ОС. Вообще говоря, такие сравнительные показатели, как производительность и надежность, могут быть корректно оценены только при использовании одинаковых аппаратных средств, но поскольку все равно будут сказываться особенности конфигурации программного обеспечения (ПО) и ОС, объек-

[1] По крайней мере, это справедливо на данный момент. Большинство таких систем используется в качестве серверных платформ, в частности для информационных *Internet*-сервисов. Например, по данным *Netcraft (http://www.netcraft.com/survey)*, доля *Web*-серверов *Apache* (устанавливаются преимущественно на системах с ОС клона *UNIX*) на июнь 2000 г. составила 62,53 % от общей численности *Web*-серверов в *Internet*.

[2] TCO включает в себя стоимость оборудования, ПО и эксплуатации [19].

тивное оценивание крайне проблематично. В этом случае полезными могут быть и статистические данные, собранные на различном оборудовании. По нашему мнению, критерии, выработанные на основе экономических показателей, безусловно, необходимо учитывать, но применительно к конкретным задачам, позволяющим на практике получать количественные оценки.

Третья группа представляет собой довольно широкий спектр критериев оценки ОС, которые условно можно назвать техническими показателями. К ним прежде всего относятся требования к аппаратному обеспечению [85], особенности архитектуры [86],

Рис. 3.1. Надежность и масштабируемость ОС

надежность и производительность системы [87], различные аспекты безопасности [88], доступность ПО [89], открытость системы [90].

Эта группа критериев является наиболее важной. Несомненный интерес вызывает достаточно основательная работа [87], которая опирается на результаты целевых исследований третьих фирм, статьи из технической периодики и по существу представляет мнение специалистов, имеющих опыт системного и сетевого администрирования обеих систем в промышленных масштабах. В работе [87] преимущественно сравниваются функциональные возможности, надежность, производительность и системное управление. Кратко приведем результаты оценки некоторых характеристик. Отмечаются высокая надежность *UNIX*-систем и недостаточная устойчивость *Windows NT* (рис. 3.1). Обращается внимание на случаи остановки системы, когда единственный выход — полная переза-

грузка, и анализируются вероятные причины этого явления. Напротив, если *UNIX*-серверы и останавливаются, то это почти всегда вызвано сбоем аппаратных средств. Любое приложение, вызывающее проблемы, в среде *UNIX* обычно проявляется через некоторое время, иногда в виде общего падения производительности системы, предоставляя администратору достаточно времени для выявления причины, исправления ее и остановки/перезапуска конкретного процесса (очень редко всей системы).

Производительность. Вычислительная мощность в большей степени зависит от аппаратного обеспечения компьютера, нежели от ОС. Поэтому сравнение производительности *NT*-сервера проводилось с *Linux* и *FreeBSD*, так как эти ОС работают на платформе *Intel*. Общее мнение специалистов таково, что *Linux* и *FreeBSD* превосходят *NT*. Объясняется это тем, что их ядра могут перекомпилироваться таким образом, чтобы исключить поддержку неиспользующихся возможностей, — любая ОС, требующая меньше ресурсов, будет производительнее. Кроме того, *UNIX*-подобным системам, в отличие от *Windows NT*, для функционирования не нужен графический интерфейс, для реализации которого необходимы большие объемы дискового пространства и памяти.

Итак, *UNIX* — это зрелая, технически более совершенная группа высокопроизводительных ОС, отличающихся надежностью и безопасностью работы в сетевой среде. Подразумевается, что читатель должен *сам* сделать этот вывод на основе изложенных в работе [87] фактов. В том, что факты надежны, его должны убедить более чем 150 ссылок на технические статьи, официальные издания, рефераты. *ОС Windows NT*, по мнению автора, подойдет для малого бизнеса с числом пользователей меньше 250 (при условии, что нагрузка будет распределена на несколько физических серверов), где не используются критичные к сбоям процессы.

В четвертую группу входят такие критерии, как перспективы развития ОС, эффективность сопровождения и поддержки, психологические факторы [91], вплоть до гражданской сознательности [92]. Последнее, по мнению автора работы [92], подразумевает покупку коммерче-

ского ПО у солидного поставщика, который платит налоги, предоставляет рабочие места программистам и менеджерам, занимается благотворительностью. Понятно, что на основании таких спорных и субъективных критериев невозможно прийти к какой-либо согласованной оценке.

В заключение можно сказать, что сравнительный анализ ОС самих по себе (не в контексте решаемых ими в составе программных комплексов задач) неэффективен, а полученные результаты необъективны.

3.2. Принципиальные различия *Windows NT* и *Linux* и их значимость в контексте построения серверной платформы распределенной ИС

Как *Windows NT*, так и *Linux* являются универсальными серверными платформами. Это значит, что они способны выполнять функции:

- информационного сервера, в частности *SQL*- и *Web*-сервера;
- коммуникационного сервера — сервера удаленного доступа и сервера, выступающего в качестве маршрутизатора, межсете-

вого экрана *(firewall)*, *Proxy*-сервера и средства построения виртуальных частных сетей (*virtual private network, VPN* [3] [79]).

Для реализации этих сервисов и служб существует достаточно большой выбор ПО (бесплатного, коммерческого или поставляемого в дистрибутиве ОС [4]). Однако основное внимание целесообразно уделить не преимуществам или недостаткам каких-либо конкретных программных продуктов, а "степени пригодности" программной платформы на базе конкретной ОС для поддержки перечисленных сервисов и служб.

Кроме особенностей поддержки ОС тех или иных сервисов и служб, следует учитывать и существенные различия в их архитектуре и пользовательском интерфейсе. Многие из них, традиционно считающиеся недостатками той или иной системы, в контексте модели распределенной ИС не являются критичными. А различия, на которых обычно мало акцентируется внимание, могут иметь принципиальное значение. Например, наиболее существенное отличие ОС *Windows NT* от систем клона *UNIX* и, в частности, *Linux*, заключающееся в том, что первая не является *многопользовательской* системой,[5] нельзя считать недостатком в рамках архитектуры распределенной ИС. Использование клиент-серверной архитектуры и унифицированных клиентов (*Web*-браузеров) позволяет отказаться от серверов приложений, где необходим многопользовательский доступ. В то же время особенности

---

[3] Напомним, технология *VPN* позволяет объединять территориально разнесенные локальные сети в одну логическую сеть, используя коммуникационную инфраструктуру глобальных сетей.

[4] Каждая из рассматриваемых ОС существует в нескольких вариантах и может поставляться вместе с дополнительным ПО. В данной работе речь идет о *Windows NT Small Business Server* из состава пакета *Microsoft Back Office* и дистрибутиве *RedHat Linux*.

[5] Войдя в сеть *NT*, можно читать файлы и печатать. Зайдя на *UNIX*-сервер, можно присутствовать на этой машине и делать все, что можно было бы делать с ее клавиатурой и мышью. *Windows NT Terminal Server Edition* [98] едва ли можно назвать многопользовательской системой — скорее, это сервер приложений, который поддерживает только клиентов на платформе *Microsoft Windows*. Для поддержки иных платформ необходимо ПО третьих производителей.

реализации пользовательского интерфейса *Windows NT*, конфигурации системы и приложений, обычно не являющиеся препятствием для применения системы в гомогенных локальных сетях, могут накладывать существенные ограничения на использование ее как ключевого коммуникационного звена распределенных гетерогенных систем. Например, конфигурирование сетевых интерфейсов в *Windows NT* осуществляется через *GUI* [6], а оперативные изменения в таблицу маршрутов можно внести только через *CLI* [7]. Далее, в *Windows* вся информация о конфигурации системы и приложений хранится централизованно в *Registry* — иерархической базе данных специального формата [8]. Поскольку параметры конфигурации приложений, сервисов и служб считываются с *Registry* только при загрузке системы, большинство изменений в конфигурации системы или установка/обновление приложений требуют перезагрузки. Сам принцип централизованного хранения конфигурационной информации, особенно в специфическом формате, имеет не только несомненные преимущества, но и ряд существенных недостатков:

- для внесения изменений в *Registry* необходим специальный редактор;
- повреждение одного из файлов *Registry*, как правило, приводит к полной или частичной потере работоспособности системы;
- все изложенное выше затрудняет создание интерфейсов для дистанционного управления системой и приложениями.

В *UNIX*-подобных ОС (и *Linux*, в частности) все обстоит иначе. Конфигурационные параметры самой ОС и всех приложений хранятся в отдельных текстовых файлах, как правило, самодокументируемых, т.е. содержащих примеры и пояснения. Они могут быть изменены с помощью любого текстового редактора; создать интерфейс для управления системой и приложениями в этом

---

[6] *GUI* (*graphical user interface*) — графический интерфейс пользователя.

[7] *CLI* (*command line interface*) — интерфейс командной строки.

[8] Физически *Registry* представляет собой несколько файлов.

случае сравнительно просто. Однако следует учитывать, что эти файлы содержатся в различных каталогах файловой системы. Несмотря на то что *FHS* [9] [93] предполагает некоторую унификацию в структуре дерева каталогов файловой системы для клонов *UNIX*, различия в местоположении, названии и структуре системных конфигурационных файлов, даже в пределах одного клона, могут быть довольно существенными. Кроме того, синтаксис и структура файлов конфигурации приложений обычно меняются при появлении версии, радикально отличающейся от предшествующих. Такое положение, на первый взгляд, должно чрезвычайно затруднять создание приложений, поддерживающих более одного типа ОС (не говоря уже о поддержке многочисленных версий), в действительности же этого не наблюдается. Самый простой выход — создание таких модульных приложений, в которых отдельные модули могут меняться или добавляться в процессе работы. Примером является *Webmin* — *Web*-интерфейс для администрирования ОС клона *UNIX* [94]. Это приложение поддерживает более десяти различных ОС, включая коммерческие. Новый модуль или даже новую версию *Webmin* можно загрузить с *Web*-сервера и сразу включить в работу.

Едва ли не единственное изменение, требующее перезагрузки системы в *Linux*, — переход на новую версию ядра. Установка или обновление ПО перезагрузки не требует. Архитектура ядра допускает использование загружаемых драйверов устройств, что позволяет производить замену или установку новых аппаратных средств без перезагрузки. Например, для установки дополнительной сетевой карты (конечно, если ее конструкция предусматривает "горячую установку") необходимо лишь подгрузить соответствующие драйверы. Это свойство может использоваться для построения систем, в которых повышенная надежность достигается избыточностью аппаратных средств, а также в случаях, когда изменение их конфигурации не должно приводить к перерывам в работе. Так, в пресс-релизе от 15 августа 2000 г. сообщается, что *Motorola Computer Group* и *Red Hat, Inc.* объединяют усилия по продвижению ОС *Red Hat Linux* на

[9] *Filesystem hierarchy standard* — стандарт иерархической структуры файловой системы.

новой линии коммуникационных серверов *CPX*8000, которая, благодаря пакету *HA-Linux* (*high-availability software for linux*) от *Motorola*, обеспечит гарантированную устойчивую работу (99,999 % от общего времени), т.е. время простоев — не более пяти минут за год [95]. Для сравнения: простои мэйнфрейма *IBM S/390* с архитектурой сисплекс (кластер процессоров) составляют в год около 10 минут, а простои ПК с ОС *Windows NT* — в среднем 224,5 часа [96].

Важным отличием ОС *Linux* от *Windows NT* является наличие открытого исходного кода первой, что позволяет увеличить производительность и снизить ресурсоемкость системы путем оптимизации под конкретное аппаратное обеспечение с учетом функционального назначения сервера. Поскольку, как заверяет *Microsoft*, большинство сбоев *Windows NT* происходит "при использовании серверов и рабочих станций с ресурсами, заведомо неадекватными запускаемым на них задачам", типичной ситуацией является "недостаток ОЗУ или слабый процессор" [80]. Можно предположить, что это отчасти вызвано избыточностью, заложенной в "раздутом" ядре системы.

3.3. Критерии выбора программной платформы для сервисов и служб распределенной ИС

К сожалению, существующие методики оценки качества ПО не могут быть непосредственно использованы для выбора программной платформы [97] по нескольким причинам. Во-первых, такие методики ориентированы на количественные характеристики качества ПО, что вызвано стремлением иметь объективные результаты. Однако эти оценки могут быть получены только для ограниченного набора свойств ПО, поскольку не всем свойствам можно поставить в соответствие измеримые показатели. Кроме того, для получения по существующим методикам количественных оценок некоторых параметров необходимо наличие исходного текста программы, что применительно к ПО для *Windows NT* невыполнимо. Во-вторых, эти методики дают "абсолютные" оценки программного продукта, в то время как нас интересует сравнительная оценка. В-третьих, оцениваются отдельные программы, а не программные комплексы, к которым относится распределенная ИС. Сложность оценки комплекса ПО состоит в том, что его компоненты могут образовывать иерархическую структуру (например, ОС—системное ПО—прикладное ПО) и на окончательный результат будет влиять оценка каждого из них.

Однако набору свойств, который в идеале должен быть оценен количественно, можно дать и качественную оценку. Взяв за основу обобщенное дерево свойств, используемых для оценки программного продукта [97], необходимо представить набор критериев, на основании которых производится оценка пригодности программной платформы для построения узла распределенной ИС в целом.

Такой набор качественных критериев (рис. 3.2) включает в себя:

- *дистанционное управление/администрирование*. Наличие средств дистанционного управления для сервисов и служб узла распределенной ИС и средств удаленного си-

стемного администрирования, а также средств мониторинга состояния узлов ИС;

• *простоту интеграции*. Сервисы и службы, входящие в состав узла, должны взаимодействовать друг с другом и с другими узлами эффективно и использовать при этом стандартные механизмы, предусмотренные моделью распределенной ИС;

• *безопасность*. Узел должен быть защищен от несанкционированного доступа и обеспечивать защиту информационного обмена с другими узлами;

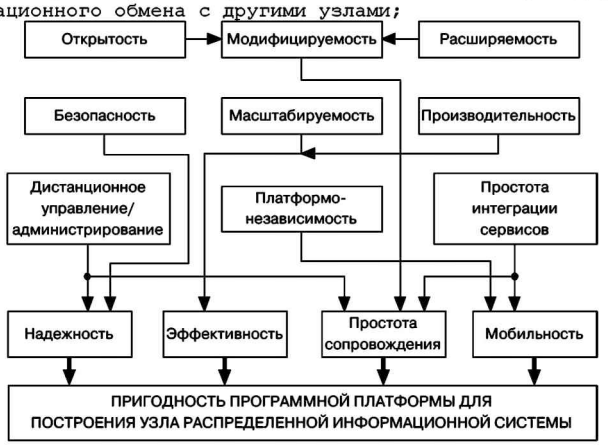


Рис. 3.2. Критерии оценки программной платформы узла распределенной ИС

• *открытость* предполагает наличие исходного программного кода и средств разработки;

• *расширяемость* — возможность добавления в состав узла новых сервисов и служб;

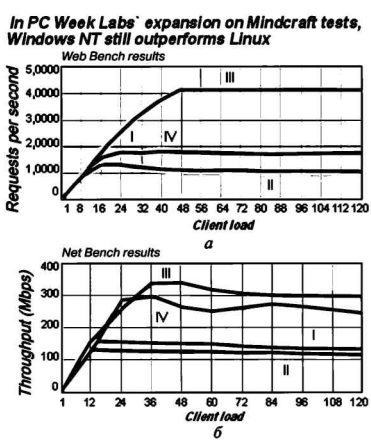
• *платформонезависимость* — возможность функционирования на различных программно-аппаратных платформах;

• *производительность* (применительно к узлу) целесообразно было бы измерять числом одновременно поддерживаемых клиентов при фиксированной нагрузке. Од-

нако провести измерение такого параметра весьма сложно, поскольку при этом необходимо имитировать реальную работу клиентов со всем комплексом сервисов и служб, входящих в состав узла, с учетом коэффициента загрузки каждого из них. В настоящее время существуют тесты, имитирующие реальную работу клиентов со "средне-ста-тистическим" *Web*-сервером [98];

- *масштабируемость* традиционно связывается с возможностью увеличения производительности программных средств за счет наращивания аппаратных ресурсов, а узел информационной системы — с возможностью увеличения производительности узла путем "распараллеливания" сервисов, достигаемого, например, с помощью их кластеризации;
- *модифицируемость* — следствие открытости и расширяемости; характеризует возможность внесения изменений в существующие сервисы и службы, а также добавления новых сервисов и служб при необходимости расширения функциональности узла.

Рассмотрим подробно некоторые из приведенных выше критериев.



Производительность. Как ни парадоксально, даже при наличии объективных тестов производительности системы пока нельзя получить однозначный ответ на вопрос, какая же платформа в этом отношении более предпочтительна. Например, считалось (подтверждено результатами многих тестов), что при небольших нагрузках производительность *Web*-серверов на ОС *Linux*

равна или больше производительности на *Windows NT* при использовании серверов нижнего (*Low-End*) уровня (типичная конфигурация — один процессор, один сетевой адаптер, 128-512 *Mb* ОЗУ). Однако при наращивании числа процессоров производительность *Web*-серверов на ОС *Linux* увеличивается незначительно, в то время как *Windows NT* демонстрирует хорошую масштабируемость (рис. 3.3) [99]. Кроме того, тесты показывали, что при использовании больше одного сетевого адаптера производительность *Web*-серверов на ОС *Linux* также падает. Поскольку качественная картина не изменялась в зависимости от используемого ПО *Web*-сервера, установленного на *Linux*, логично предположить, что причиной снижения производите-льности явились ограничения, накладываемые ядром ОС.

Рис. 3.3. Сравнительные характеристики производительности *Web*-серверов на ОС *Windows NT* и *Linux*:
а — I — *Linux* (4 *processors*); II — *Linux* (1 *processors*); III — *NT* (4 *processors*); IV — *NT* (1 *processors*); *б* — I — *Linux* (*4 processors; NT clierts*); II — *Linux* (*4 processors; Win 95 clierts*); III — *NT* (*4 proces-sors; Win 95 clierts*); IV — *NT* (4 *pro-cessors; NT clierts*)

В 2000 г. компанией *Dell Com-puter Corp.* были проведены сравнительные испытания *Web*-серверов под *Windows* 2000 *Advanced Server (Internet information server* 5.0) и *Red Hat Linux* 6.2 *(TUX "threated linux web server"* 1.0) на аппаратных серверах собственного производства [100]. В комментариях специалистов [101] отмечается, что незначительные различия в составе аппаратных средств серверов с *Windows* и *Linux* не могли оказать заметного влияния на результаты тестов.

Рис. 3.4. Результаты теста *SPECweb*99: — *Red Hat* 6.2; — *Windows* 2000

Для исследования применялся тест *SPECweb*99 [102] — новая разработка *Standard Performance Evaluation Corporation*, осуществляющей сопровождение

стандартизированных наборов тестов и показателей для оценки производительности компьютерных систем. Данные тестирования [100] представлены в виде графика (рис. 3.4). Как видно из результатов исследования, *Linux*-система по производительности значительно превосходит систему на базе *Windows* 2000 и, кроме того, имеет хорошую масштабируемость, чего нельзя сказать о последней. Во всех тестах были использованы сетевые адаптеры *Gigabit Ethernet*, обеспечивающие пропускную способность сети 1*Gb/s*. Число сетевых адаптеров в каждом тесте равнялось числу процессоров. На 8-процессорном сервере испытания для *Windows* 2000 не проводились, но, судя по результатам, полученным на серверах с меньшим числом процессоров, рост производительности на этом сервере должен быть незначительным из-за недостаточной масштабируемости.

Согласно опубликованным данным по состоянию на октябрь 2000 г. *Standard Performance Evaluation Corporation*, результат 8-процессорной *Linux*-системы (приведенной выше конфигурации) был превзойден (с показателем *SPECweb*99=7288) только 12-процессорным (600*MHz RS*64-*III*) *IBM eServer pSeries* 680 с ПО *Web*-сервера *Zeus* 3.3.6 от *Zeus Technology Ltd.* (доступен для многих современных ОС клона *UNIX*, в том числе и для *Linux*) [103].

Таким образом, можно отметить значительный прогресс *Linux*-систем, как, впрочем, и то, что ситуация постоянно меняется, поэтому трудно предсказать, какая из систем покажет лучшие результаты тестов в ближайшем будущем.

Еще одним аспектом, позволяющим скептически относиться к практической ценности результатов оценки производительности той или иной системы, являются затраты на ее оптимизацию для получения максимальных показателей. Они будут включать в себя не только дорогостоящее аппаратное обеспечение, но и тщательную настройку ПО (вплоть до перекомпиляции ядра ОС), выполняемую высококвалифицированными, а следовательно, высокооплачиваемыми программистами.

Открытость. Под открытостью подразумеваются доступность исходного кода ПО и особенности лицензирования, допускающие использование этого кода в новых разработках. ОС производства *Microsoft* являются коммерческими продуктами, исходные коды которых закрыты, это исключительная собственность владельца торговой марки. Отсюда следует, что по чисто правовым соображениям на основе программных продуктов *Microsoft* не может быть создан дистрибутив технологии построения распределенных ИС. Однако подобная технология может и должна допускать использование в рамках ИС коммерческого ПО при соблюдении условий его лицензирования.

Большинство ПО в дистрибутивах *Linux* и само ядро ОС (в дистрибутиве *Debian GNU/Linux* все ПО) подпадают под действие *GNU General Public Licence* (*GNU GPL*) [104], основные положения которой следующие:

- все ПО должно распространяться со своими исходными кодами;
- допускаются любые изменения исходного кода ПО и коммерческое использование полученного в результате программного продукта при условии, что он также подпадает под действие *GNU GPL*.

Следовательно, на основе *GNU GPL* может быть разработан лицензионно чистый дистрибутив распределенной ИС. Кроме того, доступность исходного кода позволит легко модифицировать и оптимизировать имеющееся ПО.

Таким образом, открытость ПО является необходимым условием для создания дистрибутива распределенной ИС, но не может быть критерием, регламентирующим использование той или иной программной платформы для создания конкретного узла этой системы.

Безопасность. Это одна из важнейших характеристик систем для хранения и передачи информации. В ИС главным аспектом безопасности является защита информации от несанкционированного доступа. При анализе безопасности системы следует принимать во внимание используемые механизмы доступа к информации и их реализацию.

Если реализация механизмов доступа к информации не допускает надежного блокирования сервисов и служб,

присутствующих в операционной среде, но не предусмотренных архитектурой ИС, то очевидно, что функционирование этих средств также будет влиять на защищенность ИС.

Как *Windows NT*, так и *Linux* позволяют реализовать следующие стандартные для сетевой ОС механизмы защиты информации:

- разделение прав доступа к объектам файловой системы;
- криптографическую защиту информационного обмена;
- ограничение доступа на сетевом уровне.

Рассмотрим особенности реализации перечисленных уровней защиты для каждой из этих ОС.

В *Windows NT* для предоставления пользователям доступа к объектам ОС (и, в частности, к объектам файловой системы) используется список контроля доступа (*access control list, ACL*). Он позволяет в случае необходимости назначить каждому пользователю индивидуальные права доступа к объекту ОС, хотя базовая политика опирается на предоставление доступа группам пользователей. В стандартных ядрах *Linux* поддержка *ACL* не реализована. Доступ к объектам файловой системы может быть установлен для владельца/группы/остальных пользователей. На уровне объектов ОС подобный механизм не реализован, однако для контроля доступа к исполняемым файлам могут быть использованы встраиваемые модули аутентификации (*pluggable authentication modules, PAM*). В виде дополнений к ядру *Linux* реализованы механизмы с большей функциональностью, чем *ACL* в *Windows NT*. Существующая система контроля доступа на основе набора правил для *Linux* (*rule set based access control for linux, RSBAC*) [105] обеспечивает уровень безопасности *Linux*-системы, примерно соответствующий классу *B1* по "Оранжевой книге" [106], в то время как *Windows NT* соответствует классу безопасности *C2* — самому низкому. Кроме того, используя распространенные средства и приемы, в ОС *Linux* несложно создать криптографическую "файловую систему" с применением достаточно надежных алгоритмов шифрования, например *DES*. Такая "файловая

система" физически представляет собой файл, содержимое которого шифруется при записи и дешифруется при чтении. Стандартными средствами его можно смонтировать в любом месте файлового дерева (подобно файлу с образом компакт-диска формата *ISO*9660), как обычный раздел файловой системы. Повышенная безопасность хранения данных в таком разделе достигается за счет жесткого контроля за процедурой его монтирования.

Криптографическую защиту информационного обмена в обеих системах обеспечивает дополнительное ПО. В сетях *Microsoft Network* пароли, передаваемые *Windows*-клиентами на сервер *Windows NT*, шифруются, но это распространяется только на стандартные службы этих сетей — службы доступа к файлам, принтерам и удаленного доступа.

На сетевом уровне стек *TCP/IP Windows NT* позволяет контролировать доступ к сервисам и службам, разрешая или запрещая соединение с клиентами в зависимости от их *IP*-адреса. Современные дистрибутивы *Linux* имеют в своем составе средства, поддерживаемые на уровне ядра. Они обеспечивают более "тон-кую" настройку правил доступа к сервисам и службам, в частности, позволяют блокировать определенные протоколы. Однако можно утверждать, что ни одна из этих систем не гарантирует стопроцентную безопасность и надежность.

Дистанционное управление/администрирование. Стабильность функционирования распределенной ИС, построенной на разнородных по своей архитектуре компонентах, в первую очередь будет зависеть от возможности оперативного контроля и (в случае необходимости) коррекции ее работы. При этом для узла первостепенное значение имеет наличие средств дистанционно-го управления/администрирования. Такие средства должны охватывать как системное ПО, так и основные сервисы и службы распределенной ИС, а также иметь *Web*-интерфейс. Именно наличие и функциональность таких средств для той или иной программной платформы будет являться главным критерием, определяющим возможность ее использования для размещения сервисов и служб распределенной ИС.

ОС *Windows NT* имеет службу дистанционного системного администрирования с *Web*-интерфейсом — *Windows NT Web Administrator*, функциональность которого может быть достаточна лишь в случае управления файл- и принт-сервером. Учитывая особенности конфигурации *Windows NT*, а также отсутствие стандартных средств разработки дополнительных компонентов для этой службы, можно прийти к выводу о нецелесообразности использования этой ОС как базовой платформы для коммуникационных служб распределенной ИС. Стандартный для *Windows NT Internet Information Server* 4.0 также имеет средства дистанционного администрирования с *Web*-интерфейсом, функциональность которых вполне достаточна для большинства задач, характерных для сопровождения *Web*-сервера.

Для ОС *Linux* доступны несколько утилит системного администрирования с *Web*-интерфейсом. Наиболее распространенные — *Webmin* [94] и *Linuxconf* [107] — имеют модульную структуру и поставляются со средствами разработки новых модулей. Важной особенностью *Webmin* является непосредственная поддержка криптографически защищенного протокола обмена гипертекстом *HTTP Secure* (*https*). В настоящее время для *Webmin* разработано большое количество модулей, предназначенных для управления различными сервисами и службами. Среди них есть модули для управления и конфигурирования *Web*-сервера *Apache*, *SQL*-сервера *MySQL* и т.п. *Linuxconf* больше ориентирован на системное и сетевое администрирование, где его функциональности в сочетании с гибкой системой включения новых модулей достаточно для задач управления и контроля коммуникационными службами распределенной ИС.

Выводы

Предлагаемый набор критериев качественной оценки программных платформ распределенных ИС и проведенное на их основе сравнение двух ОС не претендуют на полноту, поскольку всегда существуют требования, специфичные для конкретной ИС. Однако такие критерии, как производительность, открытость, безопасность и наличие

средств дистанционного управления/администрирования, по мнению авторов, должны всегда учитываться при выборе программной платформы для сервисов и служб распределенной ИС.

ЗАЩИТА ИНФОРМАЦИИ И СЕТЕВОЙ МОНИТОРИНГ РАСПРЕДЕЛЕННОЙ ИС

4.1. Защита информации средствами ОС и *Microsoft Internet Information Server*

Принципиально различаются три вида защиты информации: защита информации, хранимой на сервере (в виде файлов), от несанкционированного доступа; криптографическая защита информации, передаваемой по сети; защита на сетевом уровне. Первый вид защиты используется для аутентификации пользователей и разделения прав доступа, второй — для защиты от перехвата информации при передаче по сети, третий — позволяет контролировать доступ к ресурсам сервера из конкретных сетей. Ниже рассмотрены принципы конфигурации *Windows NT* 4.0 и *Internet Information Server* (*IIS*) 4.0 для обеспечения защиты первого и третьего типа и дополнительные меры защиты данных от несанкционированного доступа средствами шифрующей файловой системы *EFS* (*encrypted file system*), реализованной в пятой версии *NTFS* и используемой в *Microsoft* ОС *Windows* 2000 и *XP*.

Принципы назначения пользователям прав доступа [108–110]. Назначение прав доступа осуществляется на двух уровнях, где учитываются права доступа файловой системы *NTFS* (естественно, в том случае, если ресурсы *IIS* находятся на *NTFS* разделе) и задаются права доступа к разделяемым (*sha-red*) ресурсам, аналогично тому, как это делается при назначении сетевых дисков. Для

Рис. 4.1. Окно менеджера пользователей с бюджетом *IUSR_имя*. Имя сервера — *INTERINTRAS*

предоставления анонимного доступа к ресурсам через *Web*- или *FTP*-интерфейс используется специальное имя пользователя — *IUSR_имя*, где *имя* — название сервера (рис. 4.1). Бюджет с этим именем создается на локальной машине или в домене (зависит от роли сервера [1]) в процессе инсталляции *IIS*. Сервисы *IIS* при этом исполняются конкретно для каждого пользователя и полномочия клиента ограничиваются его правами, поэтому набор полномочий этого пользователя в домене *Windows NT* должен быть минимальным. Как правило, данный пользователь является членом групп *Domain Users* и *Guests*.

Поскольку права доступа, назначаемые на уровне файловой системы, могут не совпадать с правами доступа, предоставляемыми *IIS* (например, к домашнему каталогу и/или отдельным его файлам), общий принцип защиты состоит в том, что из предоставленных прав доступа используются наиболее строгие. Алгоритм проверки полномочий пользователя сервером приведен на рис. 4.2.

Интерфейсы, используемые для конфигурации *IIS*. Доступ к параметрам конфигурации *IIS* версий 3.0 и 4.0 возможен с использованием как *Web*-интерфейса, так и специального интерфейса. В первом случае конфигурация возможна с любого удаленного терминала, имеющего сетевое соединение с *IIS* и оборудованного *Web*-браузером. Доступ к параметрам конфигурации осуществляется через специальный *URL*: *http://сервер/iisadmin*, где сервер —

[1] Отдельный сервер (*Standalone Server*) или основной контроллер домена (*Primary Domain Controller*).

IP-адрес или полностью определенное доменное имя (*FQDN*).

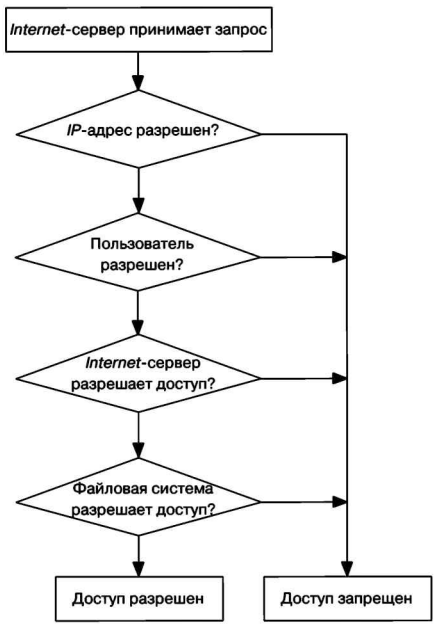


Рис. 4.2. Алгоритм проверки полномочий поль-зователя сервером *IIS*

Данный способ имеет одно существенное ограничение. Воспользоваться *Web*-интерфейсом для конфигурации *IIS* можно только в случае, если возможность анонимного доступа к ресурсам сервера отключена.

Оригинальный интерфейс для управления и конфигурации *IIS* 3.0 — диспетчер служб *Internet Microsoft* — позволяет конфигурировать сервисы, входящие в состав *IIS* (*Web*, *FTP*, *Gopher*) в пределах локальной сети. Такой же диспетчер используется и в службе *Peer Web Service* — аналоге *IIS* для рабочих станций *Win-dows NT*.

В *IIS* 4.0 применяется *Microsoft Management Console* — консоль управления *Microsoft* (рис. 4.3), — обеспечивающая единый интерфейс управления и конфигурации для различных сервисов (не только для входящих в состав *IIS*) в пределах локальной сети.

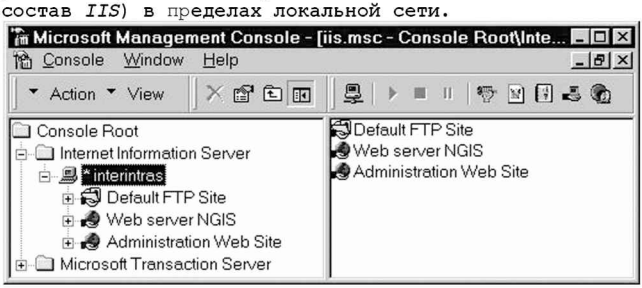


Рис. 4.3. Консоль управления *Microsoft*

С помощью диспетчера служб *Internet Microsoft* можно также конфигурировать и *IIS* 4.0, но нельзя получить доступ к конфигурации *IIS* 3.0 из *Microsoft Management Console.*

Конфигурация средств защиты информации в *IIS* 4.0. Для организации доступа к ресурсам через *Web*- или *FTP*-интерфейс используется механизм виртуальных каталогов. Подобно тому, как при назначении разделяемого ресурса — сетевого диска — ему дается имя, которое может быть никак и не связано с его местоположением на физической дисковой системе сервера, виртуальный путь также не отражает реального местоположения ресурса. При этом для клиента, использующего *Web*-браузер или *FTP*-клиент, реальный путь к ресурсу (файлу или каталогу), расположенному на локальном (или сетевом) диске сервера, заменяется виртуальным путем. Виртуальный путь — это та часть *URL*, которая записывается справа от *IP*-адреса или *FQDN*-сервера. Например, в *URL http://сервер* виртуальным каталогом является "/" — так называемый домашний (*home*) каталог ("/" в конце не всегда отображается, но всегда подразумевается), реальный путь к которому для физической дисковой системы сервера выглядит как

имя_диска:\InetPub\wwwroot. При этом права доступа для пользователей к ресурсам сервера назначаются на уровне файловой системы, подобно тому, как это делается для разделяемых ресурсов. Для идентификации пользователя используется имя и пароль его бюджета.

Настройка системы защиты сервера производится с помощью опции *Directory Security* окна *Web Server Properties* (рис. 4.4). Поле *"Anonymous Access and Authentication Control"* предназначено для выбора методов аутентификации клиента.

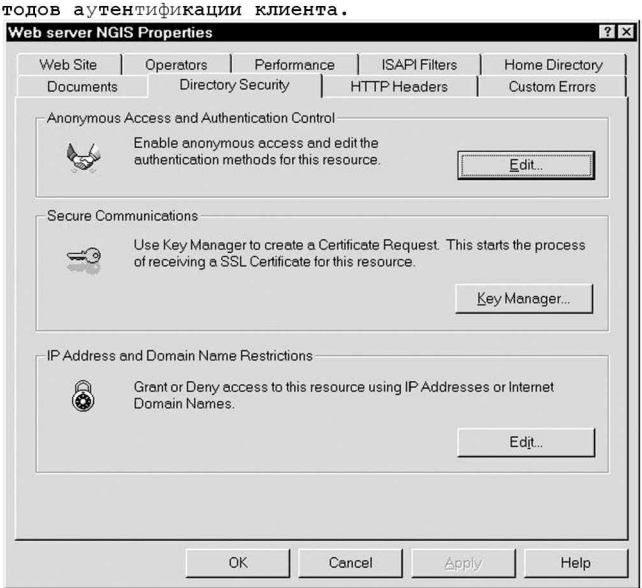


Рис. 4.4. Настройка средств безопасности *Web*-сервера *IIS*

Выбор метода аутентификации осуществляется через вложенное окно (рис. 4.5). Представляющие практический интерес комбинации методов и их характеристики приведены в таблице 4.1.

Использование базового метода аутентификации несет в себе скрытую угрозу, так как имя и пароль, передаваемые через сеть, кодируются с использованием примитивного алгоритма *base*64 и могут быть легко перехвачены и дешифрованы. Однако это единственный метод проверки, поддерживаемый браузерами, отличными от тех, которые выпускает *Microsoft*. Метод *Challenge/Response* является наиболее защищенным. При использовании базового метода или метода *NT Challenge/Response IIS* проверяет имя и пароль через базу учетных записей пользователей и порождает процесс, исполняющийся в контексте данного пользователя.

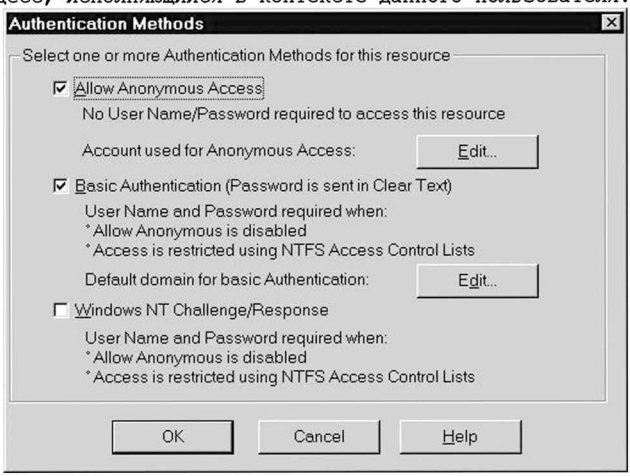


Рис. 4.5. Выбор методов аутентификации клиента

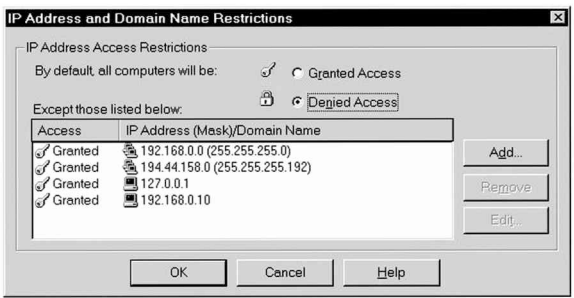


Рис. 4.6. Конфигурация контроля доступа на сетевом уровне

Все права на ресурсы в таком случае определяются набором полномочий этого пользователя.

Поле *"IP Address and Domain Name Restrictions"* опции *Direc-tory Security* предназначено для контроля доступа к ресурсам сер-

Таблица 4.1

Методы контроля доступа к ресурсам *Web*-сервера *IIS* и их характеристика

Метод	Характеристика
Allow anonymous. Access	Доступ разрешен только анонимным пользователям. Права доступа клиента к ресурсам соответствуют правам пользователя *IUSR_имя*. Уровень защиты информации высокий, так как по сети не передаются пароли
Allow anonymous. Access Basic Authentication WindowsNT Chal-lendge/ Responce	Разрешен доступ как анонимным, так и зарегистрированным пользователям (имеющим бюджеты на сервере или в домене). Для аутен-фикации пользователей может использоваться базовый метод или метод *Challenge/Response* в зависимости от того, какой из них поддерживается клиентом. Возможен перехват и дешифрование пароля при использовании базового метода аутентификации
Allow anonymous. Access WindowsNT Chal-lendge/ Response	Разрешен доступ как анонимным, так и зарегистрированным пользователям (имеющим бюджеты на сервере или в домене). Для аутентификации пользователей может

	использоваться только метод *Challenge/Response*. Поддерживается только клиентами *Microsoft*
WindowsNT Challendge/ Response	Анонимный доступ запрещен. Для аутентификации пользователей может использоваться только метод *Challenge/Response*. Поддерживается только клиентами *Microsoft*. Обеспечивается высокий уровень защиты информации от несанкционированного доступа за счет надежности алгоритма шифрования

вера из *IP*-сетей. Выбор метода осуществляется через вложенное окно, представленное на рис. 4.6.

Положение *Granted Acces* переключателя *"By default all com-puters will be:"* позволяет ввести *IP*-адреса сетей или их групп, *FQDN* отдельных компьютеров, доступ которых к ресурсам сервера будет запрещен.

Положение *Denied Acces* переключателя *"By default all compu-ters will be:"* позволяет ввести *IP*-адреса сетей или их групп, *FQDN* отдельных компьютеров, доступ которых к ресурсам сервера будет запрещен, кроме списка из окна *"Except those listed below"*.

При использовании *FTP*-сервиса следует учесть, что пароли и имена пользователей передаются по сети нешифрованными, поэтому любой пользователь, имеющий средства сетевого мониторинга, может их получить. В этом случае наиболее защищенным является режим анонимного доступа, когда пользователь регистрируется под именем *anonymous* и указывает вместо пароля почтовый адрес. Рекомендуется использовать только анонимный доступ.

4.1.1. Защита данных от несанкционированного доступа в файловой системе *NTFS*

Microsoft добавила в *Windows* 2000/*XP* новые свойства и компоненты (службу каталогов *Active Directory;* возможности управления хранением данных, такие, как шифрование данных посредством *EFS* и квотирование дискового пространства, подсоединение каталогов, монтиро-

вание томов; разреженные файлы; средства развертывания приложений с помощью объектов групповой политики и др.), обеспечиваемые файловой системой *NTFS* 5.0 (*NTFS*5).

Обезопасить локальные ресурсы (файлы, расположенные на жестком диске компьютера) средствами ОС можно двумя способами: с помощью разрешений (*permissions*) *NTFS*5 [111—115] и с помощью шифрующей файловой системы *EFS* [111, 115—127]. Разрешения доступа, предоставляемые *NTFS*5, действуют как на сетевые подключения, так и на локально зарегистрировавшихся пользователей. Однако защита действует до тех пор, пока отсутствует доступ посторонних лиц к компьютеру или диску на физическом уровне. Любой пользователь, получивший эту возможность, может обойти встроенную систему управления доступом файловой системы *NTFS*5 и с помощью специальных средств[2] прочесть информацию с жесткого диска.

Шифрующая файловая система — тесно интегрированная с *NTFS*5 служба, располагающаяся в ядре ОС *NTFS5* [115,117], — использует архитектуру открытых ключей для шифрования файлов, каталогов или томов с помощью *EFS*, которая обеспечивает защиту данных от несанкционированного доступа. Структурно *EFS* выполнена в виде драйвера устройства (*\winnt\system32\driversefs.sys*),

[2] Доступ к файлам, находящимся в разделе *NTFS*, из операционных систем *MS-DOS* или, например, *Windows* 98 в обход системы безопасности *NTFS* может быть получен с помощью программ *NTFSDOS* или *NTFS*98.

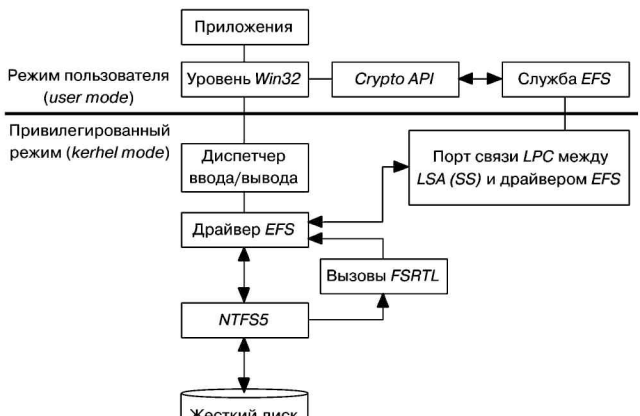


Рис. 4.7. Архитектура *EFS*

тесно связанного с *NTFS*5 и работающего в режиме ядра (*kernel mode*); *FSRTL* (*File System Run Time Library*) — библиотеки реального времени файловой системы *EFS* (рис. 4.7). Когда *NTFS*5 встречает шифрованный файл, она вызывает функции[3] из драйвера *EFS*, зарегистрированные им в *NTFS*5 при инициализации *EFS*. Поэтому средства шифрования доступны только на разделах *NTFS*5.

Когда пользователь включает шифрование файла, служба *EFS*, работающая в локальной подсистеме безопасности ОС *LSA(SS)* (*Local Security Authority* (*Subsystem*) — *\winnt\system32\lsass.exe*)), обращается к функциям *Crypto API* [4] (*Application Programming Inter-face*) [128], чтобы выделить пару из открытого и закрытого,

[3] Драйвер *EFS* регистрирует подпрограммы, напрямую вызываемые *NTFS*5 (*callss*), и *NTFS* может передавать операции с шифрованными файлами драйверу *EFS*.

[4] *Crypto API* обеспечивает взаимодействие приложений с криптографическими модулями ОС.

личного, ключей пользователя[5], которая применяется *EFS* для шиф-

Рис. 4.8. Хранение зашифрованных ключей

рования. Каждый файл шифруется с помощью случайно сгенерированного *EFS* криптографического ключа [6] (шифровальный ключ файла *FEK* (*file encryption key*)). Подобный подход значительно затрудняет осуществление большого набора атак, основанных на криптоанализе.

Файл шифруется блоками с помощью полученного *FEK* и усовершенствованного алгоритма *DESX* (*data encryption standard eXtended*). *DESX* — симметричный алгоритм (для шифрования и расшифровки используются одинаковые ключи), поэтому в процессе восстановления также используется *FEK*, и *EFS* должна защитить его от несанкционированного доступа. Для криптозащиты *FEK*, *EFS* шифрует его открытым ключом пользователя с помощью асимметричного алгоритма *RSA* и сохраняет зашифрованный *FEK* в файле (рис. 4.8.). С целью обеспечения возможности доступа к файлу со стороны нескольких пользователей создается связка (*key ring*) зашифрованных ключей *FEK*.

При *каждом* шифровании файла данных ключ *FEK* также шифруется с помощью открытого ключа *EFS агента восстановления данных DRA* (*data recovery agents*). Эта копия ключа *FEK* добавляется в файл (см. рис. 4.8).

---

[5] Ключ шифрования данных может быть известен многим, поэтому его обычно называют открытым ключом (*public key*), а ключ для расшифровки, известный только одному лицу, — личным, или закрытым, ключом (*private key*).

[6] Длина ключа составляет 128 бит для США и Канады. В международной версии *EFS* используется уменьшенная длина ключа 56 или 40 бит.

EFS хранит зашифрованные ключи *FEK* в специальном атрибуте файла[7] *$LOGGED_UTILITY_STREAM*, содержащем информацию *EFS*. Хранящиеся здесь данные состоят из поля *дешифрования*

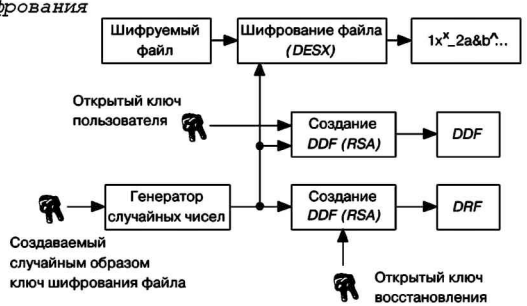


Рис. 4.9. Схема процесса шифрования содержимого файла данных

данных DDF (*data decryption field*) и *поля восстановления данных DRF* (*data recovery field*). В поле *DDF* хранится *FEK*, зашифрованный с помощью открытого ключа *EFS* пользователя, а *DRF* содержит копию *FEK*, который зашифрован открытым ключом *DRA*.

Схематически процессы шифрования и дешифрования файла показаны на рис. 4.9. и 4.10.

Восстановление данных (см. рис. 4.10) производится следу-ющим образом:

- с помощью личного, закрытого ключа пользователя или агента восстановления данных из элемента *DDF* или *DRF* извлекается зашифрованный *FEK* и дешифруется;

[7] Число элементов ключей *DDF* хранится в заголовке зашифрованного файла. Например, если файл совместно используют несколько пользователей, то *DDF* состоит из соответствующего количества элементов, совокупность которых образует связку ключей. Каждый элемент зашифрован открытым ключом одного из пользователей. Но во всех элементах *хранится один и тот же FEK*, *уникальный* для данного файла.

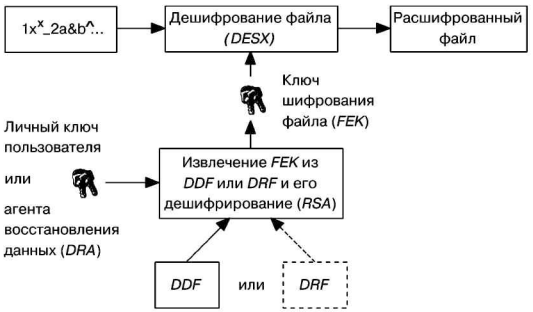


Рис. 4.10. Схема процессов дешифрования содержимого файла данных пользователем или агентом восстановления данных *DRA*

- расшифрованный *FEK* используется для дешифрования данных файла.

Для повышения производительности *NTFS*5 кэширует *FEK* в неоткачиваемую область памяти, т.е. ключи шифрования хранятся в резидентном пуле памяти, что исключает несанкционированный доступ к ним через файл подкачки. Там же она декодирует данные шифрованного файла и вновь шифрует их при записи на диск.

Восстановление данных в *EFS* — закрытая операция. В процессе восстановления расшифровываются данные, но не ключ пользователя, с помощью которого эти данные были зашифрованы.

То, что в общем случае список зашифрованных ключей *FEK* хранится в полях *DDF* и *DRF*, означает, что обращаться к зашифрованным файлам могут те, кто их зашифровал, и назначенные один или несколько агентов восстановления данных. Агент восстановления данных используется для аварийного восстановления данных на диске (например, администратор по неосторожности удалил профиль владельца данных вместе с закрытым ключом, ОС инсталлирована заново, удалена учетная запись уволенного сотрудника, зашифровавшего папку, и др.). Проблема возможной потери данных не менее остра, чем защита их конфиденциальности, поэтому и введен *DRA*[8].

[8] *DRA* является обязательной составляющей *EFS*. Если удалить последнего *DRA* в системе, то *EFS* отключается. В *Windows XP*

Итак, чтобы расшифровать ключ *FEK*, необходимо знать закрытый ключ *EFS*. После расшифровки ключа *FEK* его можно использовать для расшифровки файла. Для расшифровки файлов в случае утраты закрытого ключа используется агент восстановления данных. Закрытый ключ агента восстановления данных позволяет расшифровать ключ *FEK*, а следовательно, и сам файл данных.

Пара криптографических ключей (открытый и закрытый) хранится в профиле пользователя. Вследствие этого достигается полная прозрачность операций шифрования/дешифрования для пользователя. Когда пользователь обращается к зашифрованному файлу, *EFS* проверяет наличие у него секретного ключа. Если ключ отсутствует или не подходит ни к одному элементу связки ключей, то система запрещает доступ к объекту.

Пару ключей служба *EFS* генерирует для конкретного пользователя, когда тот впервые шифрует папку или файл данных[9]. Открытый ключ сертифицируется в центре сертификации *Certi-ficate Authoritie*, а если таковой недоступен (например, в рабочей группе или на домашнем компьютере), то *EFS* сама подписывает открытый ключ[10]. Открытый ключ оформляется в виде цифрового сертификата и записывается в папку *Documents and Settings\<username>\ApplicationData\Microsoft\SystemCertificates\My\Certificates*. Информация цифрового сертификата открытая. Закрытые ключи располагаются в папке *Documents and Settings\<username>\ApplicationData\Microsoft\Crypto\RSA*. Эта папка не может быть переименована или перенесена в другое место. Для надежной защиты закрытых ключей применяется многоуровневое шифрование. Все ключи в папке *RSA* автоматически шифруются с помощью сгенерированного случайным

удаление *DRA* приводит к невозможности шифрования новых файлов, хотя обращаться к уже зашифрованным — допустимо.

[9] В этом можно убедиться, если после инсталляции ОС запустить оснастку ***Сертификаты*** и раскрыть папку ***Личные***. Эта папка будет пуста. Если затем зашифровать некоторый файл или папку и вернуться в оснастку ***Сертификаты***, то увидим, что в папке ***Личные*** появился сертификат, выданный текущему пользователю.

[10] *EFS* создает собственный сертификат для стандартной записи *Administrator*.

образом симметричного ключа, называемого основным ключом пользователя (*user's master key*). Основной ключ в свою очередь автоматически шифруется службой *Protected Storage* [11] и сохраняется в *Documents and Settings*\<*username*>*ApplicationData**Microsoft**Protect* [116, 120].

Цифровой сертификат и закрытый ключ, как было сказано выше, хранятся в профиле пользователя на жестком диске. Агенты восстановления данных могут хранить свои открытые и личные ключи в безопасном месте вне системы: на диске, съемном носителе (например, на смарт-картах) или других хорошо защищенных устройствах. Для этого при экспорте ключей желательно удалить ключи с жесткого диска.

Шифрование данных. С помощью *EFS* можно зашифровать как отдельные файлы[12] данных, так и папки с вложенными ди-

[11] Служба *Protected Storage* также шифрует все основные ключи пользователей компьютера, а также различные другие ключи, используемые операционной системой, с помощью системного ключа (*system key*). Системный ключ уникален для каждого компьютера, начиная с *Windows* 2000. По умолчанию этот ключ хранится непосредственно на компьютере и разбросан (*scatter*) по реестру с использованием некоторого сложного алгоритма [116, 120].

[12] При шифровке отдельных файлов система предлагает зашифровать также и каталог, в котором находится выбранный файл, так как в противном случае шифрование будет автоматически отменено при первой модификации такого файла. Шифрование отдельных файлов разрешается, но не рекомендуется из-за непредсказуемого поведения приложений.

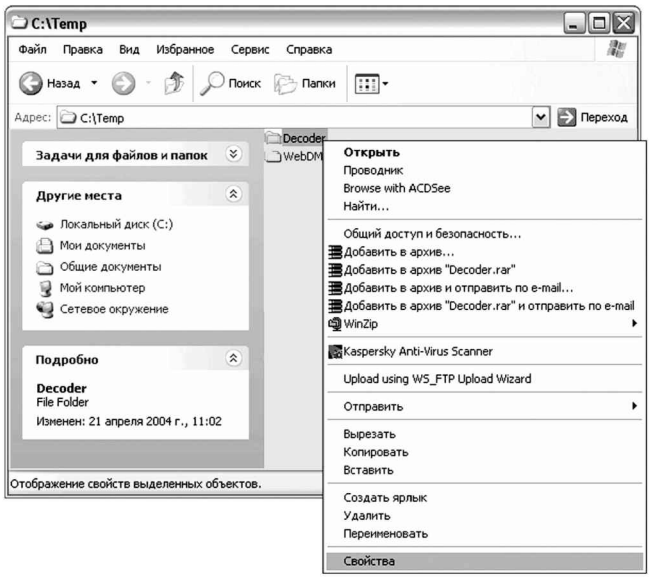


Рис. 4.11. Контекстное меню при нажатии правой кнопки мыши на папке

ректориями. В качестве примера зашифруем папку *Decoder* из директории *Temp*. Для этого нажмем правую кнопку мыши на папке *Decoder* и выберем пункт ***Свойства*** (рис. 4.11).

В открывшемся окне, на вкладке ***Общие***, выберем ***Другие*** для задания папке дополнительных атрибутов (рис. 4.12).

В окне ***Дополнительные атрибуты*** диалогового окна свойств папки выберем ***Шифровать содержимое для защиты данных*** (рис. 4.13). Отметим опцию ***К этой папке и ко всем вложенным папкам и файлам*** (рис. 4.14) и нажмем ***ОК*** в окне подтверждения заданных атрибутов. На этом процедура шифрования папки закончена.

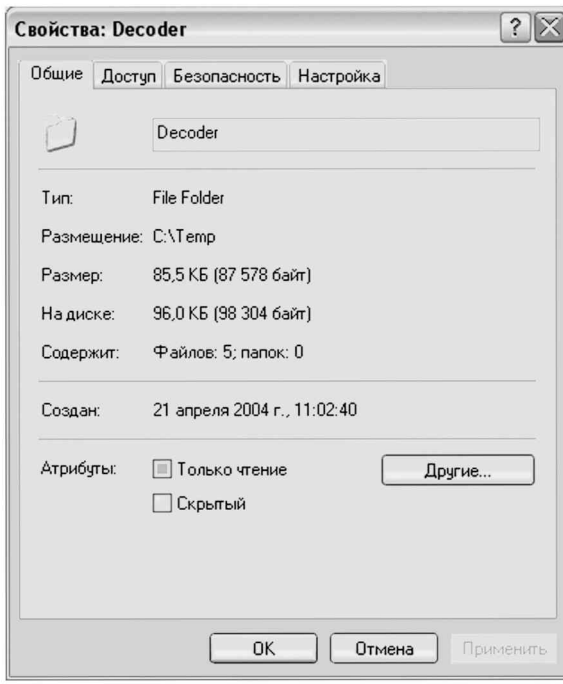


Рис. 4.12. Окно диалога ***Свойства папки***

Теперь получить доступ к папке *Decoder* и вложенным в нее файлам могут только зашифровавший ее пользователь и агент восстановления данных. Следует отметить, что над файлами, зашифрованными с помощью *EFS*, невозможно выполнить какие-либо действия, кроме удаления [13], но можно просмотреть содержимое всех вложенных папок.

[13] Чтобы исключить возможность несанкционированного удаления файлов, лучше, в случае необходимости, использовать ***Особые***

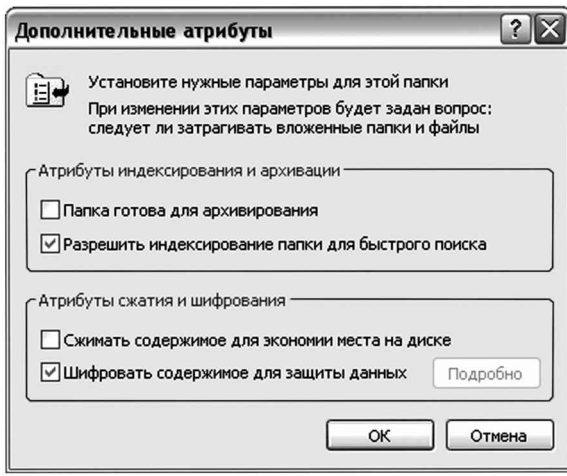


Рис. 4.13. Окно диалога ***Дополнительные атрибуты***

Работа с зашифрованными файлами в *EFS* не требует от пользователя каких-либо специальных действий по декодированию данных — *EFS* исключает необходимость предварительного расшифровывания данных при доступе к ним. Эти операции выполняются автоматически при считывании и записи данных на диск.

Для аварийного восстановления данных на диске, как уже отмечалось, используется агент восстановления данных. В *Windows* 2000 агентом восстановления данных по умолчанию является локальный администратор. Если компьютер входит в состав домена *Windows* 2000, то агентом восстановления по умолчанию является администратор домена.

По умолчанию *EFS* автоматически сгенерирует ключи для агента восстановления данных, создаст собственный

разрешения, при которых пользователи получают тот же набор разрешенных действий, что и при ***Полном доступе***, но без права удаления (см. далее "Некоторые особенности управления хранением данных в файловой системе *NTFS*5").

сертификат для учетной записи администратора[14] и сохранит его при первом

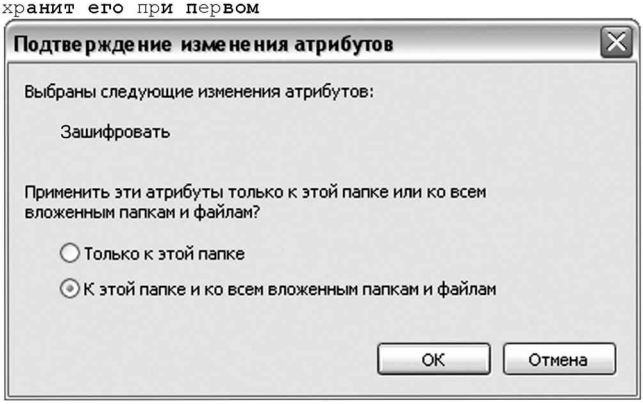


Рис. 4.14. Окно диалога ***Подтверждение изменения атрибутов***

входе в систему. Возможность создания сертификатов для других учетных записей отсутствует [122]. В *Windows XP* не создается используемый по умолчанию агент восстановления данных и, следовательно, пользователь сам должен позаботиться о сохранности своей информации, создав агента восстановления данных. Таким агентом может быть назначен каждый, кто имеет учетную запись[15] на компьютере.

Экспорт и импорт ключей агента восстановления данных. Для создания агента восстановления данных прежде всего необходимо экспортировать сертификат пользова-

[14] Под "учетной записью по умолчанию *Administrator*" разработчики *Mic-rosoft* подразумевают созданную во время инсталляции ОС *Windows* учетную запись администратора, а не любого другого члена локальной группы *Administrators*.

[15] В *Windows* 2000 администратор может сбросить пароль учетной записи пользователя, войти в систему под именем этого пользователя и получить доступ к зашифрованным файлам. В *Windows XP* после выполнения операции *reset password* доступ к файлам *EFS* будет заблокирован. Спасти ситуацию сможет только агент восстановления данных [116].

теля из хранилища сертификатов в файл на жесткий диск либо сразу же в безопасное место вне компьютера.

Управление сертификатами, их экспорт и импорт осуществляются с помощью контекстных меню оснастки ***Сертификаты*** (рис. 4.15). Пользователи имеют возможность управлять только своими собственными сертификатами.

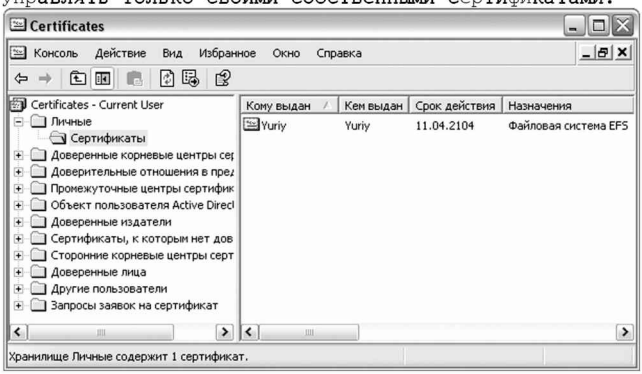


Рис. 4.15. Окно ***Certificates***

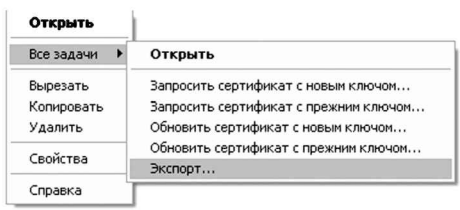


Рис. 4.16. Меню сертификата

Для этого в правой части окна сертификатов (см. рис. 4.15) щелкните правой кнопкой мыши на сертификате и в открывшемся меню (рис. 4.16) в разделе ***Все задачи*** выбирите ***Экспорт…***

В появившемся окне (рис. 4.17) мастера экспорта сертификатов установите флажок ***Да, экспортировать закрытый ключ*** и нажмите кнопку ***Далее***. В следующем окне (рис. 4.18) выберите опцию ***Удалить закрытый ключ после успешного экспорта***. В последующих окнах необходимо ввести пароль для доступа к экспортируемому файлу, дать ему имя (без расширения) и с помощью кнопки ***Обзор***... выбрать для него месторасположение. По

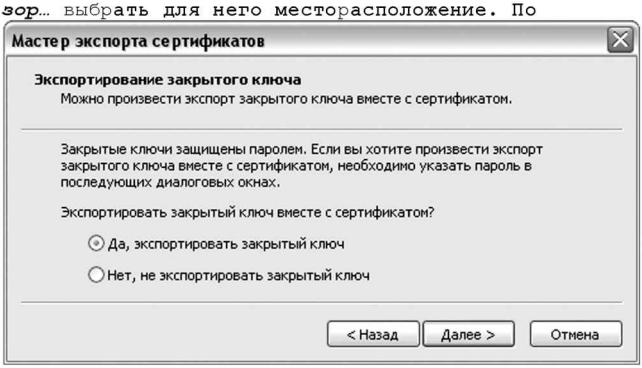


Рис. 4.17. Окно диалога ***Мастер экспорта сертификатов***

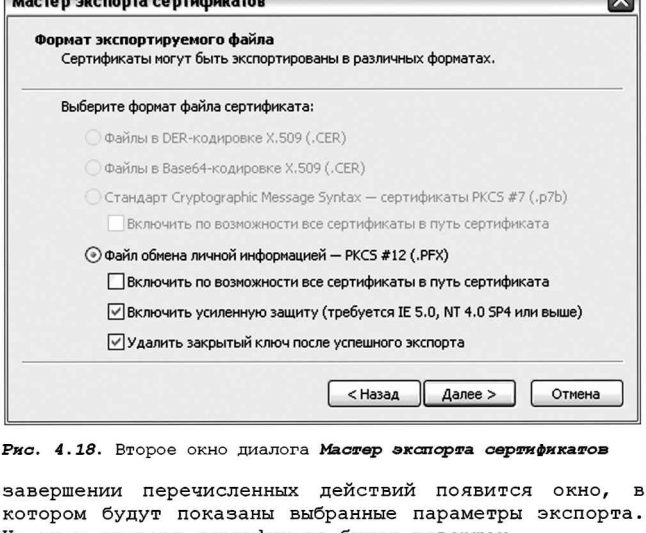


Рис. 4.18. Второе окно диалога ***Мастер экспорта сертификатов***

завершении перечисленных действий появится окно, в котором будут показаны выбранные параметры экспорта. На этом экспорт сертификата будет завершен.

Таким образом, сертификат и личный ключ пользователя *Yuriy* будут экспортированы в файл с расширением ***.pfx***, т.е. мы делегировали права пользователя *Yuriy* на доступ к файлу созданному агенту восстановления данных. Этот файл находится в папке, указанной при операции экспорта сертификата. Как уже отмечалось, файл с расширением ***.pfx*** вместе с паролем защиты данного файла можно записать на каком-либо съемном носителе информации и спрятать в сейф либо вручить доверенному лицу, чтобы оно могло восстановить данные, если, например, учетная запись пользователя *Yuriy* будет утеряна.

В таких случаях доступ к зашифрованным данным восстанавливается с помощью импорта сертификата (файла ***.pfx***). Для этого необходимо навести курсор "мыши" и дважды щелкнуть на файле с расширением ***.pfx***, затем следовать дальнейшим инструкциям. При импорте, так

же, как и при экспорте, потребуется пароль, незнание которого не позволит получить доступ к зашифрованным данным. После успешного импорта доступ к данным будет разрешен.

В этом случае пользователь сам позаботился о сохранности своих данных и создал агента восстановления данных. Теперь рассмотрим пример, в котором администратор в целях предотвращения потери данных создает для каждого пользователя компьютера агента восстановления данных либо сам становится таковым.

Администратор при создании новой учетной записи пользователя проделывает описанные выше операции экспорта-импорта сертификата из профиля пользователя в профиль администратора. Сделать это можно двумя путями.

Первый путь предусматривает шифрование какой-либо папки (файла) в целях получения сертификата пользователя и дешифрирование ее после получения сертификата. Эти действия описаны выше. Второй — предполагает использование служебной программы ***cipher***, запускаемой из основного меню *Windows*: ***Пуск — Выполнить***. Команда с ключом ***r*** имеет вид: ***cipher /r:имя файла без расширения*** (рис. 4.19). Необходимо ввести пароль для файлов сертификата. При успешном выполнении этой команды в профиле пользователя (***C:\Documents and Settings\Пользователь***) появятся два файла — ***.cer*** и ***.pfx***.

Далее, уже в профиле администратора, из меню ***Администрирование*** запустите ***Локальные политики безопасности*** (рис. 4.20). В окне диалога ***Локальные политики безопасности*** откройте группу

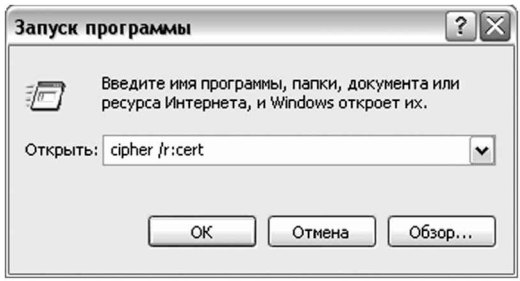
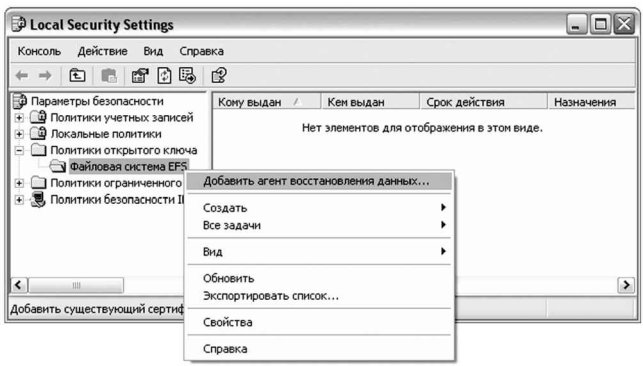


Рис. 4.19. Окно запуска программ



Рим. 4.20. Окно диалога ***Локальные политики безопасности***

Политики открытого ключа и выберите пункт ***Файловая система EFS.*** Правой кнопкой мыши откройте меню для этого пункта и выберите в нем ***Добавить агент восстановления данных…*** (см. рис. 4.20). Откроется окно мастера добавления агентов восстановления данных.

Нажмите ***Далее*** и в следующем окне с помощью кнопки ***Обзор папок…*** выберите ранее созданный файл с расширением ***.cer*** из профиля пользователя (***C:\Documents and***

Settings\Пользователь). В окне появится строка, говорящая о том, что *USER_UNKNOWN*

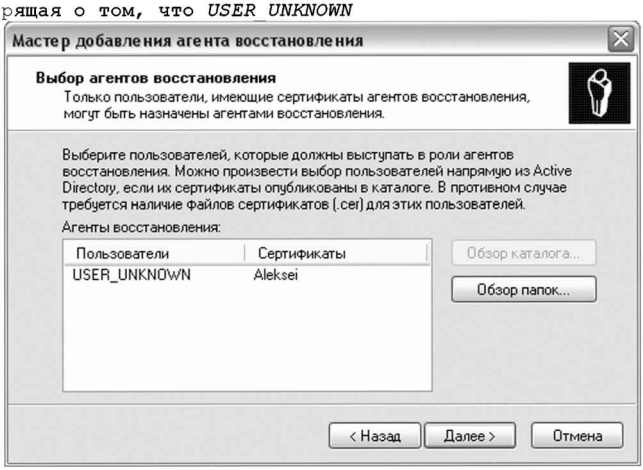


Рис. 4.21. Окно ***Мастер добавления агента восстановления***

по сертификату такого-то пользователя станет агентом восстановления данных (рис. 4.21). При нажатии кнопки ***Далее*** появится окно подтверждения, что пользователь *USER_UNKNOWN* действительно стал агентом восстановления данных. На этом процедура создания агента восстановления данных закончена.

Итак, в файловой системе *EFS* имеется встроенная функция восстановления зашифрованных данных в результате исполнения соответствующего требования *политики восстановления*. Это требование заключается в том, что политика восстановления должна быть установлена до того, как пользователи получат возможность шифровать файлы. Политика восстановления представляет собой тип политики открытого ключа, в которой можно назначить в качестве агента восстановления данных одну или несколько учетных записей пользователей. Используемую по умолчанию политику восстановления автоматически применяют при первом входе администратора в си-

стему, назначая администратора агентом восстановления данных. Для автономных (*stand-alone*), или изолированных, входящих в рабочую группу компьютеров используемая по умолчанию политика восстановления настраивается локально.

Различают политику трех типов: политика агентов восстановления; пустая политика восстановления (*empty policy*) — эквивалентна отключению работы *EFS*; отсутствие политики восстановления (*no policy*) — для восстановления зашифрованных данных в условиях утраты личного ключа используются *локальные* политики восстановления, существующие на каждом компьютере, при этом процессом восстановления управляет локальный администратор компьютера [125].

Особенности реализации политики восстановления в *Win-dows* 2000 делают *EFS* пригодной для использования преимущественно в корпоративной сети, где есть возможность избавиться от локального администратора в роли агента восстановления данных. Использование *EFS* в *Windows* 2000 на компьютерах вне домена небезопасно из-за подверженности *off-line* атакам [116].

Криптографическая стойкость *EFS* достаточно высока. Система *EFS* зашифровывает данные, используя устойчивую криптографическую схему, основанную на открытых ключах. С криптографической точки зрения выбран стандартный подход к шифрованию данных, суть которого сводится к сочетанию симметричного *DESX* и асимметричного *RSA* алгоритмов шифрования. Использование симметричного блочного шифра существенно снижает вычислительную нагрузку на процессор. В свою очередь, применение асимметричных методов криптографии позволяет надежно защитить *FEK*.

В *Windows XP* [16] включена дополнительная поддержка более стойкого симметричного алгоритма *3DES*. Ключ для

[16] ОС *Windows XP* — единственный из продуктов *Microsoft*, прошедший сертификацию. РФ первая в мире подписала с корпорацией *Microsoft* соглашение о начале программы сотрудничества *GSP* (*Government Security Program*), в соответствии с которой в марте 2003 г. в Гостехкомиссию РФ для проведения сертификации были переданы ОС *Windows XP* и *Windows Server* 2003. Сертификация проводилась в соответствии со стандартом *ISO* 15408 — "Общие критерии" и не предполагала анализа исходных текстов. Спустя

этого алгоритма имеет длину 168 бит. Включение *3DES* позволяет пользователям обращаться к файлам, зашифрованным *DESX*, но при модификации эти файлы будут шифроваться *3DES* [116].

Система *EFS* поддерживает резервное копирование и восстановление зашифрованных файлов без их расшифровки с помощью утилиты *NtBackup*.

Некоторые особенности управления хранением данных в файловой системе *NTFS*[5]. Пользователь, создавший папку или файл, становится владельцем данного объекта и всегда может прочитать или изменить информацию о разрешениях. Для предоставления пользователю или группе разрешения на доступ к определенной папке или файлу необходимо:

• указать папку "мышью" и нажать правую кнопку. Выбрать пункт ***Свойства*** контекстного меню. В появившемся окне свойств папки перейти на вкладку ***Безопасность*** (рис. 4.22); в верхней половине окна — раздел ***Группы или пользователи*** — показан список пользователей и групп, которым уже предоставлены разрешения для этой папки;

• ввести нажатием кнопок ***Добавить/Удалить*** новых пользователей или новые группы либо удалить прежних. В группе ***Разрешения***... с помощью указателей ***Разрешить*** и ***Запретить*** установить соответствующие права доступа к папке.

Для более тонкой настройки разрешений необходимо нажать кнопку ***Дополнительно***. Появится диалоговое окно ***Дополнительные параметры безопасности для***... (рис. 4.23). В нем можно дополнительно предоставить особые разрешения (рис. 4.24), настроить политику аудита, изменить (просмотреть) информацию о владельце папки, добавить/удалить пользователей, имеющих право доступа к файлу. Именно здесь установка флажков в полях ***Уда-***

---

10 месяцев был выдан сертификат, согласно которому ОС *Microsoft Windows XP Prof.* обладает первым оценочным уровнем доверия (ОУД). Различают семь уровней ОУД для работы с конфиденциальной информацией. $ОУД_{5-7}$ соответствуют государственной тайне и регламентируются национальными законами РФ. Максимальный ОУД, достигнутый *Microsoft* в иностранных государствах, — 4^{+} [129].

ление подпапок и файлов и ***Удаление***, снятие флажка в поле ***Полный доступ*** исключают возможность несанкционированного удаления зашифрованного файла или папки и в то же время предоставляют пользователю достаточно широкие полномочия.

Таким образом, файловая система *EFS* зашифровывает файлы и папки так, что неавторизированный пользователь не может их просмотреть или прочесть. Разрешения *NTFS* для файлов и каталогов, используемые совместно с *EFS*, направлены на достижение консолидированной безопасности информации от несанкционированного доступа. Об этом упоминалось, когда речь шла о возможности несанкционированного удаления зашифрованного файла.

Отслеживание квот. Базовые функции квотирования, присутствующие в *NTFS*5, предоставляют системному администрато-
ру инструмент для отслеживания и ограничения дискового пространства на сервере, занимаемого отдельными пользователями. Администратор может указать предупреждения и ограничить дисковое пространство, предоставляемое в распоряжение пользователя на томе *NTFS*, т.е. ввести квоту на использование диска определенному пользователю. После установки квот пользователь может хранить на томе ограниченный объем данных, в то время как на этом диске может оставаться свободное простран-

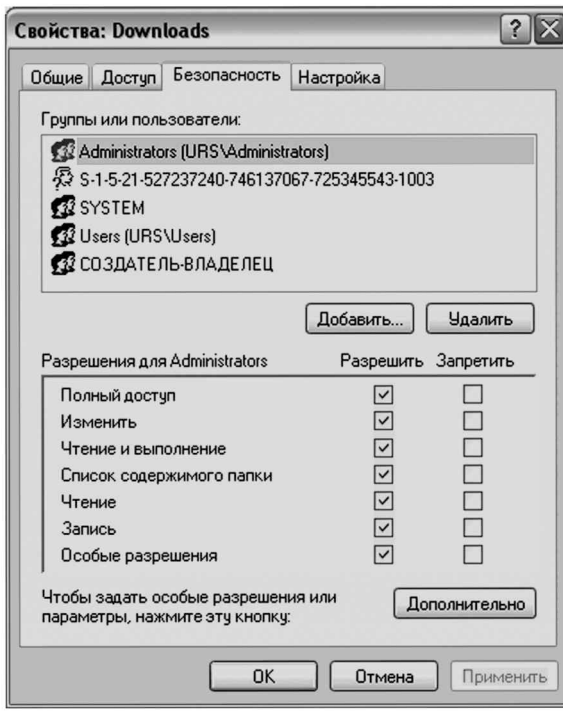


Рис. 4.22. Окно диалога ***Свойства папки***

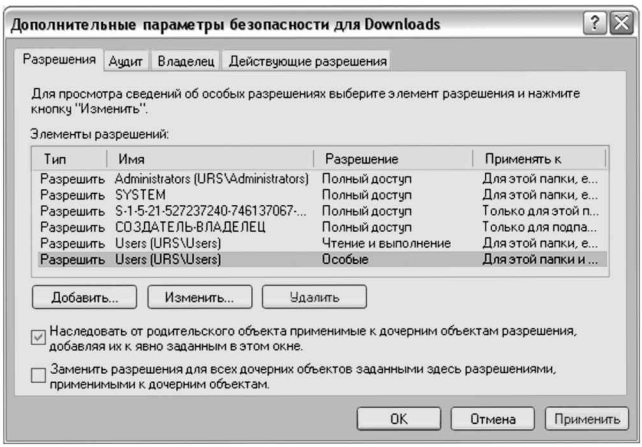


Рис. 4.23. Диалоговое окно ***Дополнительные параметры безопасности***

ство. Если пользователь превысит выданную ему квоту, то в журнал событий будет внесена соответствующая запись. Тем самым пользователь вынужден будет следить за актуальностью информации и своевременно удалять ненужные файлы, освобождая дисковое пространство.

Приложения, работающие на компьютере пользователя, также не смогут получить информацию о реальном объеме свободного пространства на диске и создать временные или кэш-файлы, исходя из объема всего доступного пространства. Это особенно важно, поскольку считается, что приложение, создающее такие файлы, имеет права на бóльший объем дискового пространства, чем объем, который система позволяет задействовать данному пользователю [111].

Точки монтирования томов, подсоединения каталогов NTFS и др. [111, 114, 115]. Они позволяют пользователю определить различные, не связанные между собой папки и даже диски в системе как один диск или одну

папку. Это важно для сосредоточения в одном месте разнородной информации, находящейся в системе. Файлы и папки, созданные таким образом, имеют уни-

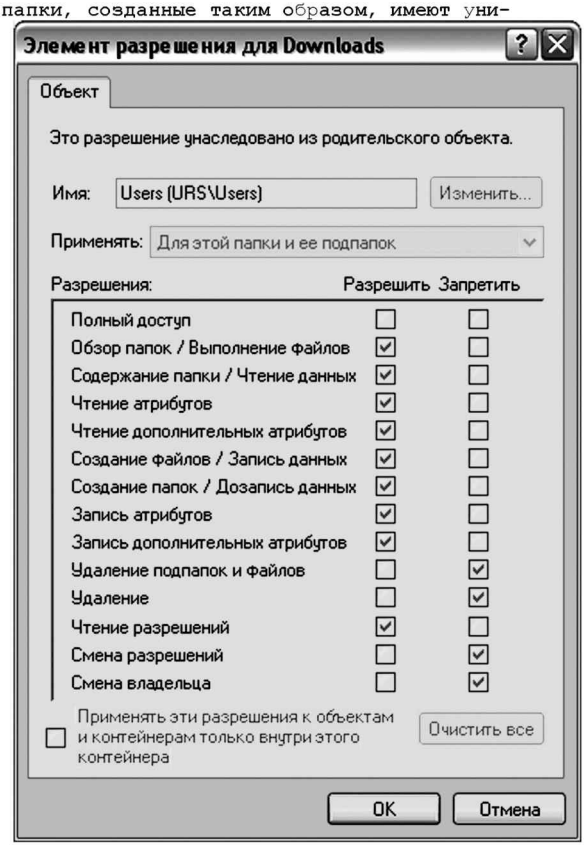


Рис. 4.24. Диалоговое окно *Особые разрешения*

кальный идентификационный номер, что гарантирует их правильное нахождение в системе.

Рассмотренные выше способы защиты информации и удобства ее расположения на дисковом пространстве не требуют до-

полнительных программных или аппаратных средств и обеспечивают достаточный для работы в локальных сетях уровень защиты. Для криптографической защиты информации при передаче ее по сети следует применять протокол *IPSec* (*IP Security*) — протокол защиты сетевого трафика на *IP*-уровне [130]. Сетевой протокол *IPSec* шифрует данные на передающем компьютере и обеспечивает механизм дешифрования их только на принимающем компьютере.

В целях обеспечения безопасности передач по протоколу *TCP/IP* в *IPSec* использованы принятые в качестве отраслевых стандартов алгоритмы шифрования и комплексный подход к управлению безопасностью. Это позволило включить *IPSec* в ОС *Microsoft® Windows®* 2000 *Server* и обеспечить надежную защиту данных, передаваемых по сетям (*intranet* или *Internet*). Протокол *IPSec* обеспечивает защиту целостности данных (за счет взаимной аутентификации хостов и целостности передаваемых пакетов), проверку прав доступа (служба аутентификации предотвращает возможность перехвата данных при использовании неправильно объявленных данных идентификации) и конфиденциальность (шифрование данных исключает несанкционированный доступ к данным при передаче их между взаимодействующими сторонами) [131].

Одно из самых больших преимуществ включения протокола *IPSec* в *Windows* 2000 состоит в способности противостоять как внутренним, так и внешним угрозам[17].

4.2. Обеспечение безопасности в виртуальных частных сетях (*VPN*)

4.2.1. Уязвимые места на пути передачи данных

Основное внимание в данном разделе уделяется вопросам защищенного взаимодействия узлов — ключевых элементов архитектуры распределенной ИС [58]. Поскольку главной информационной магистралью, связывающей узлы, служит *Internet* — общедоступная, незащищенная сеть, в кото-

[17] В большинстве стратегий безопасности сетей основное внимание уделяется предотвращению внешних атак на сеть предприятия.

рой имеется множество способов и возможностей нарушения целостности данных (данные при передаче не должны быть искажены, потеряны или продублированы), конфиденциальности (данные должны передаваться в форме, предотвращающей их несанкционированный просмотр) и аутентичности (данные должны быть переданы тем отправителем, который доказал, что он тот, за кого себя выдает), средства защиты должны иметь комплексную архитектуру, способную противостоять подобным угрозам на всем пути передачи данных от источника к адресату. Такая защита может быть реализована в рамках технологии виртуальных сетей *VPN* (*virtual private networks*) [132]. Эта технология позволяет эффективно объединять территориально разнесенные компьютерные сети, что делает ее идеально подходящей для организации взаимодействия узлов распределенной ИС. Перед рассмотрением реализаций *VPN* целесообразно выявить возможные угрозы информационной безопасности в сетях, использующих данную технологию.

Internet — популярная и достаточно дешевая базовая транспортная инфраструктура. Его повсеместная доступность дает многим потребителям возможность построения безопасной *VPN*. Проблема проектирования *VPN* состоит в использовании общедоступных ресурсов *Internet* как для построения внутренней коммуникационной структуры ИС, так и для ее взаимодействия с внешним миром при сохранении традиционной для корпоративных сетей (*intranet*) безопасности, конфиденциальности и самоуправляемости.

Использование *VPN* создает дополнительные (по сравнению с традиционными корпоративными сетями) проблемы защиты информации. Типичный тракт передачи данных с использованием *VPN* может содержать:

- оборудование, находящееся вне контроля организации (например, сервер доступа поставщика услуг *Internet—ISP* (*Internet service provider*), а также маршрутизаторы в пределах сети *Internet*);

• шлюз безопасности (*security gateway*) или шлюз защиты: МСЭ [18] — межсетевой экран (он же файрволл (*firewall*), брандмауэр) или маршрутизатор, располагающийся между внутренним *(intranet)* и внешним *(Internet)* сегментами;

• внутренний сегмент, содержащий компьютеры и маршрутизаторы, через которые будет проходить внутренний и внешний трафик организации;

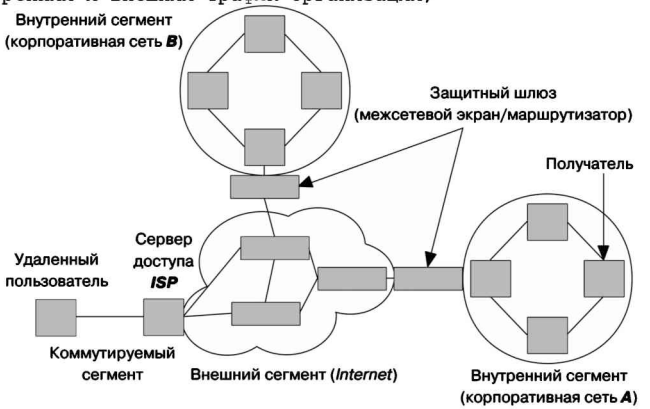


Рис. 4.25. Типичные элементы пути передачи данных

• *Internet*, который обеспечивает прохождение трафика не только от сети рассматриваемой организации, но и от других источников.

Некоторые из фрагментов сети могут быть использованы злоумышленниками, так как в подобной разнородной среде имеется множество способов и возможностей перехватить, изменить содержание или адрес назначения дейтаграммы, осуществить атаку типа "отказ в обслужи-

[18] МСЭ позволяет контролировать использование портов и протоколов, "прятать" неиспользуемые порты для исключения атаки через них; запрещать/разрешать доступ определенным приложениям к конкретным сетевым адресам, а также использование *cookies, ActiveX control, Java applets* и др.

вании". Однако существуют решения, позволяющие противостоять этим угрозам.

Для понимания проблем, связанных с безопасностью *VPN*, рассмотрим элементы пути передачи данных. Некоторые из них являются общими для любой конфигурации *VPN* (рис. 4.25). Трафик от узла-отправителя до узла-получателя обычно проходит через три основных сегмента: коммутируемый (телефонная линия), внешний (*Internet*) и внутренний (*intranet*).

Как видно из рис. 4.25, путь передачи данных может включать в себя телефонное соединение с провайдером *Internet — ISP*, который использует *Internet* для передачи трафика пользователя к шлюзу, находящемуся в периметре корпоративной сети связи. Путь, по которому осуществляется связь узлов *A* и *B*, включает в себя два отдельных сегмента *intranet*.

Рассмотрим три основных сегмента пути трафика.

Коммутируемый сегмент. Это часть коммуникационной инфраструктуры ИС, которая необходима в случае, если ИС рассчитана на широкий круг пользователей, а также если требуется обеспечить оперативный доступ к ее ресурсам. Коммутируемый сегмент простирается от машины удаленного пользователя до сервера доступа *ISP*. Протоколы и процедуру соединения устанавливает *ISP*. Как правило, используется протокол двухточечного соединения *PPP*.

Внешняя сеть (*Internet*). *Internet* не управляется какой-то одной структурой, находящейся в чьей-либо собственности. Она представляет собой совокупность доменов маршрутизации, сопровождаемых уполномоченными на то организациями. Объединяет домены набор стандартизированных протоколов межсетевого взаимодействия *IP (the Internet protocol suite)*, предложенный группой *IAB* [19] (*Internet architecture board*) — Совет по вопро-

[19] *IAB* — одна из групп, отвечающая за решение инженерных задач и выпускающая документы, содержащие описания набора протоколов *Internet*, а также связанную с ними информацию (так называемые *RFC* (*request for comments*) — запрос комментариев), используемую производителями для внедрения стандартов в архитектуру *TCP/IP*. *IAB* – *http://www.iab.org/iab/*.

сам архитектуры *Internet* — комиссии *IETF* [20] (*Internet engineering task forces*) — инженерная комиссия *Internet*. Протоколы *IP* на сетевом уровне направляют поток данных по пути, который может проходить через несколько доменов маршрутизации. Поскольку *IP* — технология без установления соединения, пользовательские дейтаграммы могут следовать различными путями. На практике это означает, что трафик различных организаций может проходить через один и тот же маршрутизатор *Internet*, т.е. информационные потоки различных *VPN* смешиваются и, в отличие от трафика корпоративной сети, не могут рассматриваться как изолированные от внешнего мира.

Внутренняя сеть (*intranet*): этот сегмент находится на конечном участке пути передачи данных и полностью контролируется владельцем. Однако учитывая необходимость предоставления доступа к внутренним серверам сторонним организациям, а также широкое использование *IP*-протоколов в корпоративных сетях, *VPN* должна обеспечивать защиту как внешнего, так и внутреннего сегментов.

На пути передачи данных могут встречаться четыре типа машин (см. рис. 4.25):

- компьютеры удаленных пользователей, использующие коммутируемые телефонные линии;
- стационарные компьютеры (источники и получатели данных или клиенты и серверы);
- сервер доступа *ISP*;
- защитные шлюзы.

Жизнеспособность решений по защите *IP*-трафика организации можно обеспечить средствами, находящимися под ее контролем, но они не должны зависеть от серверов доступа *ISP* или маршрутизаторов, используемых на пути передачи данных в *Internet*. Таким образом, эти

---

[20] Техническая работа *IETF* выполняется на общественных началах в рамках рабочих групп, организуемых по темам. Темы объединяются по областям исследований (например, маршрутизация, передача данных, безопасность). Цели и задачи *IETF* определены в *RFC* 1160. В частности, одной из ее задач является выработка стандарта безопасного обмена сообщениями, обеспечивающего конфиденциальность электронного общения. *IETF* — *http://www.ietf.org/*.

средства могут охватывать удаленных пользователей, стационарные компьютеры и защитные шлюзы.

Уязвимые места в коммутируемом сегменте. Коммутируемый сегмент (см. рис. 4.25) передает трафик от пользователя к серверу доступа *ISP*. Если эти данные не зашифрованы, то *ISP* легко может получить доступ к конфиденциальной информации пользователя или же злоумышленник может прослушать коммутируемую телефонную линию.

Шифрование на уровне передачи данных (*link-layer*) между удаленным пользователем и *ISP* может предохранить от прослушивания, но не защитить от злонамеренных действий со стороны *ISP*, где поток данных пользователя может быть расшифрован.

Уязвимые места в *Internet*. В качестве примера возможных атак на конфиденциальность информации пользователя рассмотрим (не вдаваясь в технические подробности) сценарий организации удаленного доступа, при котором *ISP* формирует туннель[21], увеличивающий протяженность *PPP*-соединения таким образом, чтобы его конечными точками были сервер доступа *ISP* и защитный шлюз. Если при этом протокол туннелирования[22] не

[21] Прокладка виртуального канала сквозь территориальную сеть или глобальную сеть.

[22] Туннелирование (*tunneling*) — процесс, в котором пакет одного типа инкапсулируется в пакет, используемый другим протоколом. Это позволяет передавать первоначальный пакет по другому типу сети.

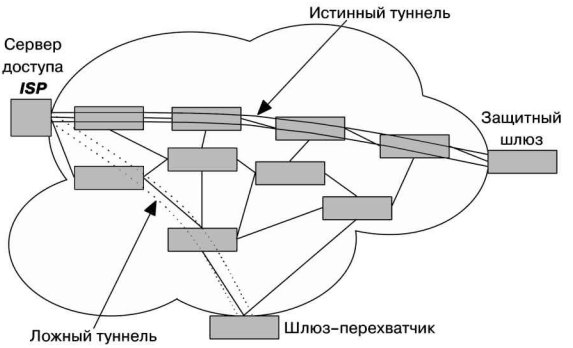


Рис. 4.26. Уязвимые места во внешнем (*Internet*) сегменте

обеспечивает надежную защиту, то злоумышленники на стороне *ISP* легко могут сформировать туннель, который не ведет к нужному шлюзу (рис. 4.26). Таким образом, данные пользователя могут быть переданы через ложный туннель на шлюз-перехват-чик, где их могут исследовать и даже подменить.

Угроза нарушения конфиденциальности существует также при следовании дейтаграмм по корректному туннелю. При перемещении от начальной к конечной точке (см. рис. 4.26) дейтаграммы пользователя проходят через маршрутизаторы *Internet*. Если дейтаграммы не зашифрованы, то на любом из этих маршрутизаторов можно исследовать или изменить дейтаграмму. Кроме того, линии связи на пути следования данных могут прослушиваться злоумышленниками.

Шифрование "*от соединения к соединению*" на каждом сегменте магистрали *Internet* может помешать прослушиванию, но не может защитить данные пользователя, так как на каждом маршрутизаторе можно расшифровать поток данных. Подобное шифрование не защитит и от ложного туннеля, поскольку конечный шлюз-перехватчик также имеет доступ к расшифрованным данным.

Забегая вперед, заметим, что даже некоторые из распространенных туннельных протоколов (например, *Layer 2 Tunnel Proto-col* [133]) недостаточно защищены. Поэтому комиссия *IETF* рекомендовала защищать туннелируемый трафик с помощью протокола *IPSec* [130].

Уязвимые места в защитном шлюзе. Основная задача защитного шлюза — усилить политику контроля доступа (пропускать желаемый входящий трафик, отклонять нежелательный и предотвращать бесконтрольную передачу внутреннего трафика корпоративной сети во внешнюю сеть). Защитный шлюз находится под полным контролем организации-владельца корпоративной сети, однако он может иметь брешь в защите: злоумышленник, находящийся в организации, имеет возможность исследовать любой трафик, который шлюз расшифровывает и передает во внутреннюю сеть.

Некриптографические методы защиты информации, например контроль проникновения/утечки нежелательного трафика в/из сети, безусловно, способны обеспечить некоторый уровень защиты. Наиболее распространенным методом является использование паролей, фильтрация пакетов, трансляция сетевых адресов. Однако некоторые виды защиты можно преодолеть с помощью известных атак (например, подмена адреса отправителя (*address spoofing*) или перехват сеанса[23] (*session hijacking*)), другие — дают перманентные решения. Например, как только разрабатывается новый алгоритм фильтрации пакетов против известной атаки, хакеры изобретают новую атаку, которая в свою очередь требует создания нового алгоритма и т. д.

Надежным средством защиты конфиденциальной информации от взлома являются криптографические методы. Они требуют больших затрат машинного времени и использования высокопроизводительных компьютеров, а следовательно, в ряде случаев являются недопустимо дорогими для злоумышленников.

---

[23] По окончании начальной процедуры аутентификации установленное соединение, скажем, с почтовым сервером, переключается на новый хост-компьютер, а исходному серверу выдается команда разорвать соединение.

Шифрование "*от соединения к соединению*" не может предотвратить перехват, подмену конфиденциальных данных или изменение маршрута их следования, так как каждое промежуточное звено имеет доступ к расшифрованным сообщениям. Даже шифрование "*пользователь-шлюз*" имеет тот же недостаток, поскольку шлюз также имеет доступ к расшифрованным данным.

Уязвимые места в корпоративной сети. Несмотря на распространенное мнение, что большинство угроз информационной защите сосредоточено в *Internet*, нельзя не принимать во внимание внутренние источники многих атак. В случае, если какой-нибудь из компьютеров, шлюзов или маршрутизаторов в пределах корпоративной сети (см. рис. 4.26) не является абсолютно надежным, его нужно модифицировать таким образом, чтобы получить возможность контролировать, изменять или перенаправлять дейтаграммы, передающиеся по корпоративной сети. Когда данные от источников в разных сетях смешиваются в пределах одной корпоративной сети (например, в случае использования *VPN*, связывающей корпоративную сеть поставщика информационных услуг с корпоративными сетями нескольких потребителей), угрозы информационной защите в пределах корпоративной сети должны быть исключены. Даже если организация *A* (см. рис. 4.25) считает, что ее собственная корпоративная сеть безопасна, то клиент, чей трафик будет через нее проходить, не может на это полагаться. Фактически конфиденциальность данных клиента оказывается под угрозой, если корпоративная сеть компании *A* каким-либо образом скомпрометирована.

Таким образом, угрозы информационной защите имеются на всем пути следования данных от источника к потребителю — на телефонной линии, сервере доступа *ISP*, в *Internet*, на межсетевом экране или маршрутизаторе и даже в корпоративной сети. Пользователь воспринимает сеть как надежно защищенную среду только в том случае, если он уверен, что его "собеседник" именно тот, за кого он себя выдает (аутентификация сторон), передаваемые пакеты не просматриваются посторонними лицами (конфиденциальность связи) и

получаемые данные не подвергаются изменению в процессе передачи (целостность данных).

Наиболее распространенными атаками являются: подмена *IP*-адреса отправителя другим адресом; перехват сеанса по окончании процедуры аутентификации; создание ложного туннеля для перенаправления данных на шлюз-перехватчик; перехват пароля, передаваемого по сети в незашифрованной форме, путем "прослушивания" канала (*password sniffing*).

Перечисленные сетевые атаки возможны в следующих случаях: во-первых, когда аутентификация отправителя осуществляется исключительно по его *IP*-адресу; во-вторых, процедура аутентификации выполняется только на стадии установления соединения, а в дальнейшем подлинность принимаемых пакетов не проверяется; в-третьих, данные, имеющие отношение к системе защиты, передаются по сети в незашифрованном виде.

4.2.2. Основные протоколы защищенного доступа в *VPN*

Существующие реализации *VPN* могут быть классифицированы по нескольким признакам. Считается, что наиболее важным является уровень стека протоколов, на котором реализована *VPN* (рис. 4.27). Как правило, это канальный, сетевой и транспортный уровни семиуровневой модели *OSI* (см. рис. 2.6 и 4.27).

Существуют и другие протоколы защищенного доступа, которые работают на верхних уровнях модели взаимодействия сетей *OSI*, дополняя *VPN*-решения. Например, протоколы прикладного уровня, обеспечивающие информационную безопасность таких приложений, как электронная почта *PEM* [24] (*privacy enchanced mail*), *S-MIME* [25]

---

[24] *PEM* (*RFC* 1421—1424) рекомендован как единый стандарт в *Internet* для защиты почтовых сообщений. Особенностью применения *PEM* является использование службы распределенной сертификации открытых ключей в соответствии с *X*.509 (система распределения и сертификации открытых ключей для защиты *E-mail*-обмена). Защищенные объекты *PEM* могут входить в состав сообщения в формате *X*.400.

[25] *S-MIME* (*RFC* 2630, 2632, 2633, 2634) — стандарт, позволяющий использовать X.509. *S-MIME*, представляет собой реализацию криптографической системы с асимметричным ключом. Система может быть использована как для внедрения электронной подписи

(*secure multipurpose internet mail exten-sion*), *PGP* (*open pretty good privacy*)-*MIME* [134], применяющие алгоритмы шифрования и электронной подписи; *WWW — S-HTTP* (*secure HTTP*), *SSL* (*secure sockets layer*) и его развитие *TLS* (*transport layersecurity*) [135], защищающие *Web*-трафик на транспортном уровне; протокол *SOCKS* [137].

Протокол *S-HTTP* (*RFC* 2660) обеспечивает необходимые сервисы безопасности — конфиденциальность, целостность данных и аутентификацию — между конечными системами[26] за счет использования криптографических механизмов *RSA* при обмене сообщениями на прикладном уровне.

SSL и *TLS* — наиболее эффективные протоколы, ставшие общепринятыми стандартами. Они обеспечивают на уровне криптографической защиты конфиденциальность обмена между

Рис. 4.27. Архитектура *TCP/IP* и протоколы, используемые при построении защищенных *VPN*

двумя прикладными процессами клиента и сервера на транспортном уровне, например защищенный доступ к

(*digital signature*) в почтовые сообщения, так и для шифрования последних.

[26] Для технологий безопасной передачи данных по общедоступной (незащищенной) сети используется обобщенное название — защищенный канал (*secure channel*). Термин "канал" подчеркивает тот факт, что защита данных обеспечивается между двумя узлами сети (хостами или шлюзами) вдоль некоторого виртуального пути, проложенного в сети с коммутацией пакетов.

Internet-ресурсам. *SSL* использует комбинацию симметричного и асимметричного шифрования для обеспечения сквозной безопасности. Для инициализации протокола в качестве префикса адресов *web*-страниц используется аббревиатура *https*. В ответ на это обращение сервер высылает клиенту свой открытый ключ, затем происходят проверки и обмен ключами для сеанса связи. Протокол *SSL*/*TLS* применяется и для защиты передачи файлов при использовании протокола *FTP*.

Протокол *TLS* (*RFC* 2246) базируется на спецификации протокола *SSL* 3.0, опубликованного *Netscape*. Различие между протоколами незначительны, но они вполне достаточны, чтобы сделать *TLS* 1.0 и *SSL* 3.0 несовместимыми[27] [136].

Протокол *TLS* имеет два уровня: протокол записей и протокол диалога. На нижнем уровне, работающем поверх транспортного протокола (*TCP*), размещается протокол записей *TLS*. Этот протокол обеспечивает безопасность соединений, которые имеют два основных свойства.

1. Соединение является конфиденциальным. Для шифрования данных используется симметричная криптография (например, *DES*, *TripleDES* и др.). Допускается не применять шифрование.

2. Соединение является надежным. Процедура передачи сообщения включает в себя проверку целостности с помощью вычисления *MAC* [28]. Для расчета применяются хэш-функции (например, *SHA*, *MD5* и др.).

Протокол записей *TLS* используется для инкапсуляции различных протоколов высокого уровня. Один из них — протокол диалога *TLS*, который позволяет серверу и клиенту аутентифицировать друг друга и согласовать алгоритм шифрования и крипто-ключи до начала процесса передачи данных.

Протокол диалога *TLS* обеспечивает безопасное соединение, при котором:

• идентичность партнеров может быть выяснена с помощью асимметричной криптографии (например, *RSA*, *DSS*

---

[27] *TLS* 1.0 имеет механизм, с помощью которого приложения могут поддерживать *SSL* 3.0 [136].

[28] MAC (*message authentication code*) — код аутентификации сообщения, применяемый в механизме аутентификации сообщений с использованием хэш-функций [138, 139].

и др.). Аутентификация может быть опционной, но она необходима, по крайней мере, для сервера;

• выявление общего секретного кода безопасно, а диалог надежен: атакующий не может модифицировать обсуждаемое соединение без того, чтобы быть обнаруженным партнерами обмена.

Одним из преимуществ *TLS* является то, что он не зависит от протокола приложения [29]. Протоколы верхнего уровня, такие, как *Telnet, FTP, SMTP, SNTP, DNS* и др., могут прозрачно размещаться поверх протокола *TLS* и использовать преимущества защищенного соединения. Наибольшее применение протокол *SSL* (*TLS*) получил в обеспечении защиты *HTTP*-коммуникаций. Защищенный вариант использует адреса (*URL*), начинающиеся с *https*, и порт по умолчанию 443.

Протокол *SOCKS* [30] работает также на транспортном уровне. Он поддерживает удобное и безопасное использование сервиса сетевых файрволлов (МСЭ) для приложений типа клиент-сер-вер, работающих по протоколу *TCP*, таких, как *TELNET, FTP*, и протоколов обмена информацией: *HTTP, WAIS* и *GOPHER*.

Существующий протокол *SOCKS v* 4 предназначен для работы через МСЭ без аутентификации. Протокол *SOCKS v* 5 (*RFC* 1928) расширяет модель *SOCKS v* 4, добавляя к ней поддержку протокола *UDP* [31] и обеспечение универсальных схем строгой аутентификации [32]. Некоторые реа-

---

[29] Стандарт *TLS* не специфицирует то, как протоколы увеличивают безопасность с помощью *TLS*. Решение того, как инициализировать *TLS*-диалог и как интерпретировать сертификаты аутентификации, отдается на усмотрение разработчиков протоколов и программ, работающих поверх *TLS*.

[30] Реализация протокола *SOCKS* обычно влечет за собой перекомпиляцию клиентских программ, работающих по протоколу *TCP*, для использования соответствующих функций *SOCKS*-библиотеки.

[31] Создавать посредников для сервисов на базе *UDP* весьма затруднительно, так как протокол *UDP* не предполагает установления соединения: каждый пакет передается как отдельное сообщение. *SOCKS* 5 решает эту проблему посредством установления соединения *TCP* с последующей передачей по нему данных *UDP*.

[32] Когда работающий по *TCP* клиент хочет соединиться с объектом, доступным только через МСЭ, он должен открыть *TCP*-соединение с соответствующим *SOCKS*-портом *SOCKS*-сервера. Сервис *SOCKS* обычно находится на *TCP*-порту 1080. Если соединение

лизации *VPN* используют только протоколы верхнего уровня (обычно комбинацию *SOCKS v5* и *SSL*).

В общем случае преимущество протоколов верхних уровней в том, что они могут работать в любых сетях — защищенный канал инвариантен тому типу сетей, которые применяются для транспортировки данных (*IP, IPX, Ethernet* или *ATM*). Недостаток в том, что приложение в той или иной степени зависит от конкретного протокола защиты. Так, продукты этой категории зачастую привязаны к единственному приложению и протокол защищает только вполне определенный сетевой сервис или службу — файловую, гипертекстовую или почтовую (например, протоколы *S-HTTP, S-MIME*). Каждая служба требует разработки соответствующего защищенного протокола.

Общим недостатком подобных решений является их привязанность к определенному типу приложений. Чем ниже в стеке реализованы средства защищенного канала, тем проще их сделать прозрачными для приложений и прикладных протоколов. На канальном уровне зависимость приложений от протоколов защиты исчезает полностью, но здесь иная проблема — зависимость протокола защиты от конкретной сетевой технологии. В виртуальной сети, как уже отмечалось, используются разные канальные протоколы, поэтому проложить защищенный канал через эту гетерогенную среду с помощью единого протокола канального уровня невозможно.

На этом уровне (см. рис. 4.27) преимущественно используются такие протоколы защищенного канала: *PPTP — Point-to-Point Tunneling Protocol* (*Microsoft, 3Com* и *Asced Communications*), *L2F — Layer 2 Forwarding* (*Cisco*) и *L2TP — Layer 2 Tunneling Protocol* (*IETF*). Протокол *PPTP* основан на протоколе *PPP* [33], широко ис-

прошло успешно, клиент начинает переговоры о используемом методе аутентификации, затем проходит аутентификацию по выбранному методу и посылает свой запрос. *SOCKS*-сервер обрабатывает запрос и либо пытается установить соответствующее соединение, либо отказывает в нем.

[33] *PPP Authentication Protocol* определен документом *RFC*1334, в котором описаны два протокола аутентификации: *PAP* (*password authentification protocol*) — протокол аутентификации паролей и *CHAP* (*challenge handshake authentification*

пользуемом в соединениях "точка-точка", например при работе по выделенным линиям. Протокол *PPTP* обеспечивает прозрачность средств защиты для приложений и служб прикладного уровня. Однако поскольку протокол *PPP* используется далеко не во всех сетях (в большинстве локальных сетей на канальном уровне работает протокол *Ethernet*, а в глобальных — протоколы *ATM*, *frame relay*), то *PPTP* нельзя считать универсальным средством.

При реализации *VPN* на уровне передачи данных (канальном уровне модели *OSI*) предполагается туннелирование *PPP*-соеди-нений через другую сеть. Таким образом, протяженность соединения как бы "растягивается" и появляется возможность передачи мультипротокольного трафика (протоколов сетевого уровня, поддерживаемых *PPP*, через *IP*-сеть) за счет инкапсуляции в *IP*-пакеты как трафика *IP*, так и других различных видов трафика. Средства *VPN*, используемые на нижнем уровне, обеспечивают независимость создаваемой сети от конкретной платформы, поскольку в данном случае клиентские системы обычно не требуется оснащать специальными программными или аппаратными средствами, за исключением адаптера, поддерживающего доступ по коммутируемой линии.

Подобная техника была разработана прежде всего для обеспечения доступа к корпоративным сетям удаленных пользователей по коммутируемому соединению. Она предлагает пользователю соединение с локальным (по отношению к его настоящему местонахождению) *ISP*, вместо дорогостоящей междугородней связи. Этот же прием может быть применен для объединения сетей, каждая из которых использует *PPP*-соединение с *ISP*. Его преимуществом в данном случае является доступность подобного соединения как относительно его поддержки различными *ISP*, так и стоимости. К отрицательным сторонам

---

protocol) — протокол аутентификации с квитированием связи. *CHAP* считается более надежным. Он построен на базе алгоритма шифрования, требующего, чтобы две соединенные между собой системы периодически идентифицировали друг друга.

этого подхода следует отнести его ориентацию исключительно на одиночных

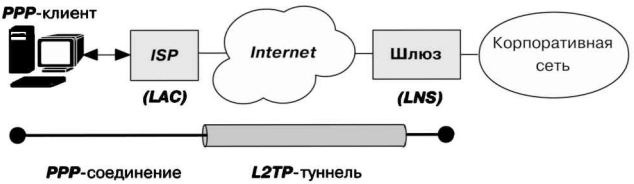


Рис. 4.28. Типичное применение *L2TP* :
LAC – концентратор доступа *L2TP*; ***LNS*** – сервер доступа к сети *L2TP*

удаленных пользователей и отсутствие гибкости, необходимой администраторам для управления связью между локальными сетями [140].

Протокол *L2TP* (рис. 4.28) объединяет функциональность *PPTP* и *L2F* и представляет собой их дальнейшее развитие.

Несмотря на то что *L2TP* обеспечивает экономичную реализацию *VPN*, поддерживающую различные протоколы транспортного уровня, сам по себе он не может быть использован для реализации защищенного взаимодействия узлов распределенной ИС по следующим причинам:

- аутентификация обеспечивается для субъектов, расположенных на концах туннеля, но не для пакетов, передающихся по туннелю. Следовательно, возможны реализация атак типа "встреча посредине" и подмены адреса;
- поскольку отсутствует поддержка контроля целостности для каждого передаваемого пакета, возможна атака типа "отказ в обслуживании" за счет генерации ложных управляющих сообщений, что может привести к разрушению туннеля или обрыву *PPP*-соединения;
- сам по себе *L2TP* не обеспечивает шифрование трафика, что может привести к нарушению конфиденциальности информационного обмена;
- несмотря на то что содержимое *PPP*-пакета может быть зашифровано, протокол *PPP* не имеет механизма для автоматической генерации криптографических ключей и

их обновления. Поэтому существует повышенный риск взлома шифра из-за прослушивания линии связи.

Однако для защиты соединения, установленного с помощью *L2TP*, вполне могут быть использованы протоколы *IPSec*. Туннель создается путем инкапсуляции кадра *L2TP* в *UDP*-пакет, который, в свою очередь, инкапсулируется в *IP*-пакет, адреса источника и получателя которого соответствуют концам туннеля (рис. 4.29). Поскольку внешним инкапсулирующим протоколом является *IP*, к такому пакету применимы все протоколы *IPSec*. Таким образом, данные, передающиеся через туннель, могут быть надежно защищены [133].

Учитывая изложенное, использование *L2TP* совместно с *IPSec* позволяет построить эффективную и экономичную среду для защищенного информационного обмена между узлами распределенной ИС.

Работающий на сетевом уровне протокол *IPSec* является компромиссным вариантом. С одной стороны, он прозрачен для приложений, а с другой, — может работать практически во всех сетях, так как основан на широко распространенном протоколе *IP* (в настоящее время в мире только 1 % компьютеров не поддерживает *IP*, остальные 99 % используют его либо как единственный протокол, либо как один из нескольких протоколов [141]).

Самый радикальный способ преодоления указанных ограничений сводится к тому, чтобы создавать систему защиты не для отдельных классов приложений, а для сети в целом. Применительно к *IP*-сетям это означает, что системы защиты должны действовать на сетевом уровне модели *OSI* (см. рис. 2.6 и 4.27). Сетевой уровень модели *OSI* (*IP* в случае стека *TCP/IP*) является самым низким уровнем, который может обеспечить защиту данных на всем протяжении их следования от источника до адресата. Протоколы защиты сетевого уровня обеспечивают полную безопасность для всех данных верхнего (прикладного) уровня, размещенных в теле *IP*-дейтаграммы, и не требуют модификации приложений.

Преимущество такого выбора подтверждает тот очевидный факт, что в *IP*-сетях именно данный уровень отличается наибольшей гомогенностью: независимо от протоколов, физической

Рис. 4.29. Инкапсуляция *L2TP* в *IP* для построения туннеля

среды передачи и технологии канального уровня транспортировка данных по сети не может быть произведена в обход протокола *IP*. Поэтому реализация защиты сети на третьем сетевом уровне автоматически гарантирует как минимум такую же степень защиты всех сетевых приложений, причем без какой-либо модификации последних. Для пользователей процедуры защиты окажутся столь же прозрачными, как и сам протокол *IP*.

Несмотря на отсутствие универсальной договоренности о том, что же должна включать в себя сеть *VPN* и как ее следует строить, и о том, что рынок средств предлагаемого оборудования для *VPN* должен быть сегментирован в соответствии с тремя уровнями (вторым, третьим и четвертым) семиуровневой модели *OSI*, все же доминируют продукты для *VPN* третьего сетевого уровня: маршрутизаторы, брандмауэры, программное обеспечение и выделенные аппаратные средства. Ожидается, что инвестиции в постепенно набирающую силу технологию *VPN* третьего уровня ускорят получение законченного решения *VPN*, способного охватить все проблемы заказчика: подключить удаленных пользователей к корпоративной сети, организовать защищенные туннели для передачи трафика от филиалов компании через *Internet* в центральный офис, обеспечить шифрование наиболее критичных компонентов внутреннего трафика и создать качественный сервис в управлении (администрировании) вновь созданными *VPN* [140].

Сетевой протокол защищенного доступа IPSec

При реализации протокола *IPSec* [130, 141–145] в сетевой архитектуре *TCP/IP* появляется *единый защищенный канал*, через который проходит трафик всех без исключения сетевых приложений, сервисов и служб (см. рис.

4.27). *IPSec* производит аутентификацию и шифрование на уровне *IP* (межмашинное "*host-to-host*"-взаимодействие), в отличие от других протоколов, защищающих отдельные виды соединения или сетевые приложения (например, *SSL* защищает только лишь конкретный прикладной сокет; *PGP* защищает определенный файл или электронное письмо). *IPSec* шифрует всю информацию, передаваемую между двумя машинами.

Архитектура защиты *IPSec* является первым всесторонним и непротиворечивым решением, способным достичь безопасности всего трафика между заданными *IP*-адресами или группами *IP*-адресов, т.е. на всем пути от источника к потребителю. В результате представляется возможным защитить сетевую инфраструктуру, не затрагивая отдельных приложений. *IPSec* предлагается как чисто программная модернизация самой сетевой инфраструктуры. Такой подход позволяет реализовать систему защиты, не прибегая к дорогостоящим модернизациям прикладных программ. Важно и то, что *IPSec* обеспечивает интероперабельность различного сетевого оборудования, компьютеров и устройств коммуникации.

IPSec формирует стандартную платформу для построения защищенных сетей и электронных туннелей, соединяющих различные машины. Туннели могут создаваться как между удаленными пользователями, так и в рамках локальной сети. Благодаря этому удается организовать надежные соединения, по которым можно, не опасаясь злоумышленников, передавать пакеты данных. Протокол предусматривает заключение каждого пакета данных в новый пакет, содержащий информацию, необходимую для настройки, сопровождения и отключения туннеля, когда надобность в нем отпадает.

IPSec может быть применен:

- *в транспортном режиме* (сквозные защищенные каналы между произвольными *IP*-хостами и/или группами хостов);
- *в туннельном режиме* (защищенные каналы между шлюзами отдельных подсетей).

Для каждого *IP*-соединения, т. е. для каждой пары *IP*-адре-сов респондентов (отправителя и получателя) или групп *IP*-адре-сов, задается своя политика безопасности, определяющая принципы применения *IPSec* и на-

бор входящих в его состав средств обеспечения безопасности соединения и защиты данных. Как туннельный, так и транспортный режимы *IPSec* широко используются в *VPN*, обеспечивая безопасность и конфиденциальность взаимодействия отдельных фрагментов или подсетей территориально распределенной *VPN*, которые связаны между собой не напрямую, а через другие сети.

Архитектура IPSec

Протокол *IPSec* был создан как базовый протокол обеспечения безопасности данных на уровне *IP*-соединений, дополнение к применяющемуся в настоящее время стандарту *IPv* 4 и неотъемлемая часть перспективного *IPv*6. Архитектура (рис. 4.30) и спецификации протокола *IPSec* описаны в документах тематической рабочей группы по безопасности *IP* (*IP Security protocol working group*)

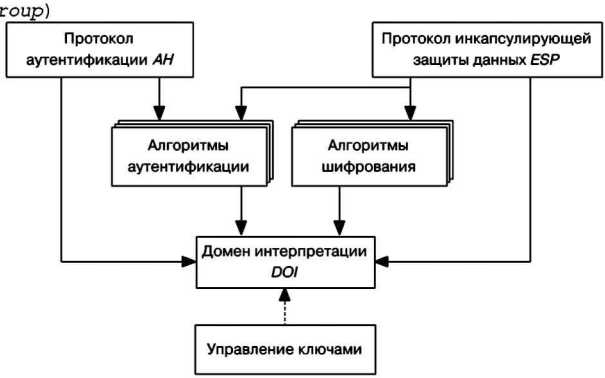


Рис. 4.30. Архитектура *IPSec* согласно *RFC*2411

[130] *IETF* (*RFC*2401 — *RFC*2412). Фактически *IPSec* не является протоколом как таковым. Это набор из двух протоколов: протокола аутентификации *AH* (*authentication header*) — аутентифицирующий заголовок — и протокола инкапсулированной защиты данных *ESP* (*Encapsulating Security Payload*) — инкапсуляция за-

шифрованных данных. Каждый из них выполняет определенные функции по защите *IP*-пакетов. Протокол *AH* обеспечивает аутентификацию, а *ESP* — конфиденциальность данных. Оба протокола также имеют средства контроля целостности *IPSec*-пакетов и защиты от воспроизведения трафика. Защита данных *AH* и *ESP* осуществляется с помощью криптографических алгоритмов аутентификации и шифрования.

Протокол AH. Протокол аутентификации *AH* (*RFC*2402) осуществляет аутентификацию и защиту целостности данных *IP*-пакетов, а также защиту от воспроизведения *IP*-трафика (*replay attack*). Аутентификация и целостность данных обеспечивается за счет применения к ним специальных алгоритмов, генерирующих код аутентификации сообщения *HMAC* (*keyed-hashing for message authentication code*) — идентификация сообщения с помощью хэширования[34]. Генерация кода аутентификации происходит на основании так называемого дайджеста (*digest*) или хэша сообщения — значения, полученного после применения к сообщению хэш-функции[35]. *HMAC* (*RFC2104*) — это алгоритм хэширования или, применительно к данному случаю, аутентификации с секретным ключом. Обеспечиваемая целостность данных и аутентификация источника зависит от масштаба распространения секретного ключа. Если ключ *HMAC* известен только передающей и принимающей сторонам, то это обеспечит и аутентификацию источника данных, и целостность пакетов данных, пересылаемых между двумя сторонами. Так, получатель пакета может, проверив хэш отправителя, убедиться в том, что пакет был отправлен именно тем, кем нужно, и не модифицировался на пути следования. Если при передаче сообщение было изменено, то значение хеш-функции изменится, и *IP*-пакет будет отвергнут.

---

[34] Хэширование (*hashing*) — преобразование массива данных произвольного размера в блок данных фиксированного размера, служащий (в некоторых случаях) заменителем исходного массива. Хэширование выполняется с помощью хэш-функций [139].

[35] Хэш-функция (*hash-function*) — это функция, устойчивая к коллизиям. Под устойчивостью к коллизиям понимается тот факт, что невозможно найти два разных сообщения m_1 и m_2, таких, что $H(m_1) = H(m_2)$, где H – хэш-функция.

В настоящее время предусмотрена возможность применения хэш-функций *MD*5, *SHA*1 и хэш-функции, базирующейся на алгоритме шифрования *DES*. Соответствующие алгоритмы генерации кодов аутентификации сообщений *HMAC-MD*5 и *HMAC-SHA*1 описаны в стандартах *RFC*2403 и *RFC*2404. Их действие основано на применении обоими участниками сеанса общего секретного ключа длиной 128 бит в случае *MD*5 (*message digest version* 5, *RFC* 1321) и 160 бит в случае *SHA*1 (*secure hash algorithm version* 1, *FIPS* 180-1). Стандарта или формального описания алгоритма *HMAC-DES* не существует, его реализация и поддержка необязательна.

Аутентификация источника данных (т.е. подтверждение отправителем собственной легитимности) осуществляется включением им в состав хэшируемого сообщения предварительно согласованного с респондентом секретного кода. Защита от воспроизведения трафика обеспечивается использованием строго последовательной нумерации исходящих *IP*-пакетов в заголовке *AH*.

Протокол *AH* обеспечивает целостность данных, но ничего не предпринимает для безопасности потока данных, кроме случая, когда он не используется вместе с *ESP*.

Протокол *ESP*. Протокол инкапсулирующей защиты данных *ESP* (*RFC*2406), как и протокол *AH*, обеспечивает защиту *IP*-пакетов в целом, за исключением переменных служебных полей, которые могут быть модифицированы в процессе его передачи (*mutable fields*). Основное назначение *ESP* — обеспечение конфиденциальности данных, для чего применяются следующие алгоритмы шифрования: *BLOWFISH*, *CAST*, *DES*_IV64, *DES*_IV32, *DES*, *IDEA*, 3*IDEA*, *NULL*, *RC*4, *RC*5 и *Triple DES* (3*DES*).

Алгоритм шифрования *DES* с явно заданным вектором инициализации (*initialization vector* — IV) применяют в протоколе *ESP* по умолчанию. Обязательным к реализации и поддержке, кроме *DES*_IV64, является режим "без шифрования" — *NULL*. В качестве альтернативы *DES* определены следующие алгоритмы: *ARCFour*, *Blowfish*, *CAST*-128, *IDEA*, *RC*5 и *Triple DES*.

Многие пользователи считают алгоритм *CAST* (*RFC*2144) таким же стойким, как и алгоритм *Triple DES* с 128-битовым ключом. Кроме того, он быстрее алгорит-

ма *DES*. *RC5* (*RFC*2040) — алгоритм шифрования потока данных, использующий ключ переменной длины. По мнению большинства, стойкость *RC*5 зависит от длины ключа, которая может достигать 256 бит. Алгоритм *IDEA* (*international data encryption algorithm*) рассматривают как "быстрый" эквивалент *Triple DES*. Еще одним алгоритмом, использующим ключ переменной длины, является *BLOWFISH*.

Последний алгоритм, *ARCFour*, является общедоступной версией алгоритма *RC*4 [145].

В соответствии с *RFC*2451 рекомендуемыми алгоритмами шифрования являются *BLOWFISH*, *CAST*-128, *IDEA*, 3*DES* и *RC*5, как обеспечивающие большую криптографическую стойкость и производительность. Определяется также режим применения блочных шифров *CBC* (*cipher block chaining*).

Выбор алгоритмов, кроме обязательных, целиком зависит от разработчика. Возможность выбора алгоритма шифрования предоставляет ему дополнительное преимущество, ведь злоумышленник должен не только вскрыть шифр, но и определить, какой именно шифр ему надо вскрывать. Возникает необходимость подбора ключей, а это, скорее всего, оставляет ему слабую надежду на своевременную расшифровку данных [145].

Помимо шифрования данных, протокол *ESP* предоставляет все возможности, имеющиеся у протокола *AH*: аутентификацию, защиту целостности данных *IP*-пакетов и защиту от воспроизведения *IP*-трафика. Аутентификация и целостность данных при применении протокола *ESP* обеспечивается теми же средствами, что и в протоколе *AH*. Аутентификация источника данных (т.е. подтверждение отправителем собственной легитимности) также осуществляется посредством включения им в состав хэшируемого сообщения предварительно согласованного с респондентом секретного кода. Защита от воспроизведения трафика обеспечивается использованием строго последовательной нумерации исходящих *IP*-пакетов в заголовке *ESP*.

Протокол *ESP*, как и *AH*, может быть применен в туннельном или транспортном режиме. Каждый из этих двух

протоколов может использоваться как самостоятельно, так и одновременно с другим, следовательно, в тех случаях, когда шифрование из-за действующих ограничений[36] применять нельзя, систему можно использовать только с протоколом *AH*. Естественно, защита данных в этом случае будет недостаточной. От несанкционированного просмотра по пути следования данных протокол *AH* защитить не может, так как не шифрует их. Для шифрования данных необходимо применять протокол *ESP*, который может также проверить их целостность и аутентичность.

Домен интерпретации *DOI* (*domain of interpretation*). Это специальная справочная база данных, содержащая полный набор предопределенных зарегистрированных параметров протокола *IPSec*, представленных стандартными идентификаторами (*RFC*2407). *DOI* содержит идентификаторы протоколов *AH* и *ESP*, *IKE*, применяемых ими алгоритмов аутентификации и шифрования, опциональных алгоритмов компрессии данных *IP*-пакетов. В частности, *IPSec DOI* применительно к протоколу *AH* определяет три идентификатора хэш-функций: *AH_MD*5, *AH_SHA* и *AH_DES*, идентификаторы алгоритмов шифрования протокола *ESP* — *ESP_BLOWFISH*, *ESP_CAST*, *ESP_DES*_IV64, *ESP_DES*_IV32, *ESP_DES*, *ESP_IDEA*, …, *ESP_NULL* и др.

Данные *DOI* используются всеми протоколами, входящими в состав *IPSec* и *IKE*.

Управление ключами. Протоколы *AH* и *ESP* обеспечивают аутентификацию и криптографическую защиту передаваемых по сети данных, но не имеют собственных *средств установления* защищенных соединений и согласования параметров сессии (на-пример, какие алгоритмы и с какими ключами будут применяться для аутентификации хостов и шифрования данных, какова

[36] Продвижение *IPSec* существенно сдерживается экспортными ограничениями, распространяемыми при реализации криптографических протоколов, обеспечивающих конфиденциальность данных путем шифрования.

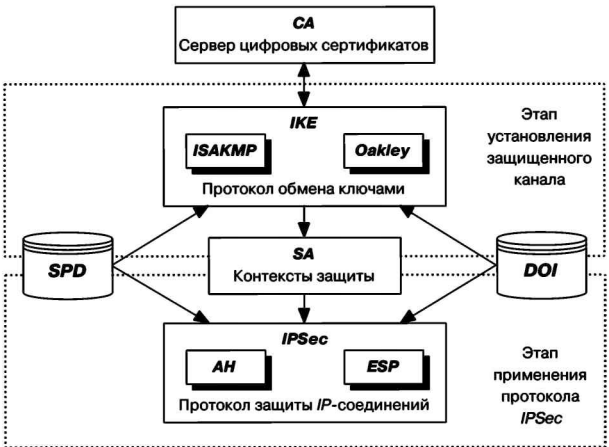


Рис. 4.31. Инфраструктура *IPSec*-системы согласно *RFC*2401:
AH, ESP – протоколы аутентификации и защиты данных; *IKE* – протокол управления ключами *IKE* : *ISAKMP* и *OAKLEY*; *DOI* – справочная БД предопределенных стандартных констант (идентификаторы зарегистрированных протоколов, алгоритмов шифрования, аутентификации и режимов их применения); *SA* (*Security Association*) – "безопасная ассоциация", иначе, контекст безопасности или защиты логического соединения; *SPD* (*Security Policy Database*) – БД политик безопасности, управляющая функционированием *IPSec*-системой в целом; *CA* (*X*.509 *Certificate Authority*) – внешняя система цифровых сертификатов, используемая для первичной аутентификации участников информационного обмена

продолжительность действия или частота смены ключей и др.) по открытым каналам связи. Для этого используется протокол *IKE* (*Internet key exchange*) – протокол управления ключами для *Internet*, являющийся комбинацией протоколов *ISAKMP* (*Internet security association and key management protocol* – протокол контекстов безопасности и управления ключами для *Internet* – и *OAKLEY*. Протокол *IKE* является внешним по отношению к архитектуре *IPSec*, но без его участия на этапе установления соединения функционирование *IPSec* невозможно.

Учитывая это, далее протокол *IKE* целесообразно рассматривать в составе построенной на основе *IPSec* функционально полной системы обеспечения безопасности данных [142] (рис. 4.31).

Функционирование *IPSec*-системы начинается после получения запроса на установление защищенного *IPSec*-соединения и состоит из подготовительного этапа и целевого применения.

1. *Подготовительный этап (установление защищенного канала). IKE* позволяет участникам информационного обмена проверить аутентичность респондентов, согласовать все необходимые параметры *IPSec*-сессии: договориться об общих применяемых алгоритмах аутентификации и шифрования и обменяться ключами до того, как будет начат обмен данными. При этом *IKE* использует справочную информацию домена интерпретации *DOI* и цифровые сертификаты абонентов, полученные от сервера *CA*.

2. *Целевое применение*. Протокол *IPSec* защищает входящий и исходящий трафик данного *IP*-соединения в соответствии с согласованными контекстами безопасности *SA*.

Установление защищенного канала и защита трафика осуществляется в соответствии с правилами, определенными в базе политик безопасности *SPD* для данного *IP*-соединения.

Рассмотрим (см. рис. 4.31) назначение элементов и их роль в функционировании *IPSec*-системы.

Протокол *IKE* (*RFC* 2409). Он включает в себя протокол *ISAKMP* (*RFC* 2408), применяемый для согласования параметров сессии, и произвольный протокол обмена ключами (*key determi-nation protocol*), в качестве которого де-факто выступает протокол *Oakley* (*RFC* 2412). Спецификация *ISAKMP* описывает механизмы согласования атрибутов используемых протоколов, в то время как протокол *Oakley* позволяет устанавливать сессионные ключи на компьютеры сети *Internet*. Основное назначение *IKE* [37] — установление защищенного логического соедине-

[37] Роль инфраструктуры в *IPSec*, обеспечивающей распределение ключей и согласование протоколов между участниками обмена, выпол-

ния между участниками информационного обмена (переговорный процесс формализован в рамках протокола *IKE*) и, как следствие, выработка соответствующего *контекста безопасности или защиты SA*, определяющего состав протоколов *IPSec* (*AH* и/или *ESP*), секретные ключи, используемые в работе протоколов, режимы их применения и другие необходимые параметры.

Функционирование протокола *IKE* начинается при получении им запроса на установление соединения от приложения или сервиса верхнего уровня. На подготовительном этапе переговоры двух взаимодействующих систем (компьютеров или шлюзов безопасности) организуются в виде двух фаз (фаза I и фаза II), в каждой из которых формируется свой набор *SA*. Назначение фазы I заключается в первоначальном обмене информацией, необходимой для взаимной аутентификации хостов. Фаза II отвечает за безопасность основного канала и переговоры о передаче данных, а также за процесс обновления сессионных ключей, используемых в данном сеансе.

Соответственно в фазе I устанавливаются соглашения *SA* для обмена данными по протоколу *IKE*, а в фазе II — по протоколу *IPSec*. В фазе I создается временный защищенный канал *IKE* с выработкой контекста защиты *SA*, специфичного для данного протокола (*IKE SA*), и происходит взаимная аутентификация респондентов.

При первичной идентификации сторон *IKE* использует их цифровые сертификаты (*X*.509 *digital certificates*), полученные от внешнего сервера *CA* центра сертификации *CA* (*certification au-thority*).

няет протокол *IKE*. Это название в 1998 г. пришло на смену более раннему — *ISAKMP/Oakley*. Протокол *ISAKMP* был разработан в 1997 г. в Управлении национальной безопасности США. Протокол *Oakley* служит для непосредственного обмена ключами.

Протокол *ISAKMP/Oakley* не был специально разработан для совместного использования с протоколом *IPSec*, поэтому возникла необходимость в так называемой области интерпретации (*Domain Of Interpretation*), обеспечивающей совместную работу протоколов *IPSec* и *ISAKMP/Oakley*. Чтобы другие протоколы также могли использовать *ISAKMP/Oakley*, они должны иметь собственные *DOI*-области. В настоящий момент таких областей для других протоколов не существует, но ситуация может и измениться, если какой-либо разработчик решит использовать этот механизм [145].

Цифровые сертификаты включают в себя три части: уникальный идентификатор, позволяющий однозначно опознать участника будущего обмена, открытый ключ, используемый при подписании электронных документов, и открытый ключ *CA*, дающий возможность подписать, а затем идентифицировать весь сертификат.

Параметры контекста безопасности *SA* должны устраивать обе конечные точки защищенного канала. Поэтому при использовании автоматической процедуры установления *SA* протоколы *IKE*, работающие по разные стороны канала, выбирают параметры в ходе переговорного процесса. Успешная взаимная аутентификация — необходимое условие для продолжения переговоров, так как все меры безопасности теряют смысл, если данные передаются или принимаются не тем пользователем или не от того пользователя. Защищенный виртуальный канал, или логическое соединение *SA* (фаза I), в стандартах *IPSec* называемое *безопасная ассоциация,* используется непосредственно для защиты обмена информацией в фазе II, во время которой и формируются окончательные *SA* для работы *IPSec*: выбираются протоколы защиты данных, *AH* и/или *ESP*, определяются их функции (например, только аутентификация и проверка целостности или еще и защита от ложного воспроизведения), соответствующие алгоритмы аутентификации и шифрования, обмена ключами и согласовываются другие параметры *IPSec*, в частности, формируется контекст безопасности *SA* для избранного протокола (например, *AH SA* или *ESP SA*). В этой фазе (фаза II) создаются две *SA*: одна для исходящих пакетов, вторая — для входящих. Подготовительный этап на этом заканчивается и респонденты могут обмениваться защищенными пакетами *IP*.

На этапе целевого применения протокол *AH* или *ESP* функционирует уже в рамках установленного логического соединения *SA*, с его помощью осуществляется требуемая защита передаваемых данных с использованием выбранных параметров. Для каждой задачи, решаемой протоколами *AH* и *ESP*, предлагается несколько схем аутентификации и шифрования, отражаемых в *SA*. Объединение обширной служебной информации в рамках *SA*

предоставляет пользователю возможность сформировать разные классы защиты, предназначенные, например, для электронного общения с разными респондентами. Другими словами, применение структур *SA* открывает путь к построению множества виртуальных частных сетей, различающихся своими параметрами.

Генерация ключей. Согласно политике *IPSec*, срок жизни фазы I *SA*, как правило, значительно больше, чем фазы II *SA*. В связи с этим различают *основной режим* (*main mode*) функционирования протокола *IKE* и *быстрый режим* (*quick mode*) соответственно. Напомним, что в основном режиме оба участника обмена устанавливают *SA*-соглашения для безопасного общения друг с другом по протоколу *IKE*. В быстром режиме *SA*-соглашения устанавливаются уже от имени протокола *IPSec* или любой другой службы, которой необходимы данные для генерации ключей или согласования параметров. Разбиение на фазы оптимизирует процесс переговоров, а также обеспечивает очень мощный защитный механизм обмена ключами.

Для соглашения о совместно используемом ключе применяется метод Диффи—Хеллмана (*Diffie—Hellman* (*D—H*))[38]. Метод *D—H* считается основополагающим в технологии шифрования с открытым ключом. Этот метод является механизмом распределения открытых ключей для защищенного обмена данными без какой-либо информации, заранее известной обеим сторонам, иначе, участники обмениваются открытыми блоками информации, на основе которых каждый из них самостоятельно вычисляет секретный ключ.

Обмен ключами по методу *D—H*, производимый в основном режиме, сопряжен с интенсивными вычислениями. Быстрый режим обеспечивает согласование параметров основного канала *SA* и генерацию новых ключей. Здесь

[38] Метод *D—H* активно используют для установления безопасных сеансов связи в тех случаях, когда необходима динамическая защита и конечные системы не принадлежат одной и той же системе административного управления. Например, его используют в электронной коммерции при установлении соединения для передачи транзакций между двумя компаниями.

операция обмена не нуждается в большом объеме вычислений, так как при этом используется набор простых математических операций. В отличие от симметричной криптографии протоколов *AH* и *ESP* с простыми в реализации блочными шифрами, криптография с открытым ключом базируется на сложном математическом аппарате и требует вычислительных операций над числами большой длины.

Контекст защиты SA. Контекст защиты *SA* [*RFC*2408] содержит всю необходимую информацию, которую должны иметь системы, участвующие в информационном обмене с использованием протоколов *IPSec*. Он представляет собой формализованную запись, определяющую способы обеспечения безопасности при информационном обмене. Контексты защиты автоматически генерируются при установлении защищенного соединения системой управления ключами *IKE* либо задаются вручную.

Каждый контекст защиты имеет уникальный идентификатор, состоящий из трех полей:

- индекса параметров безопасности (защиты) *SPI* (*Security Parameter Index*);
- *IP*-адреса узла назначения;
- идентификатора протокола защиты.

Индекс параметров безопасности *SPI* обеспечивает уникальность каждого контекста защиты, например в случае наличия двух одинаковых по протоколу безопасности и *IP*-адресу получателя контекстов защиты, использующих разные алгоритмы аутентификации/шифрования или разные их ключи.

Каждый *SA* определяет способ защиты трафика только в одном направлении, заданном *IP*-адресом получателя. Это означает, что *SA* является однонаправленным. Для обеспечения безопасности двунаправленного соединения необходимо иметь два отдельных контекста защиты[39].

---

[39] *SA* могут быть одинаковыми и отличаться только направлением (*IP*-адресом получателя). Следовательно, двум респондентам для обеспечения безопасности трафика в обоих направлениях с применением одновременно протоколов *AH* и *ESP* потребуется по четыре контекста защиты каждому (и еще столько же контекстов защиты протокола *ISAKMP* должно было быть временно создано для

Контекст защиты *SA* включает в себя идентификатор протокола безопасности и поэтому является специфичным для каждого протокола. В *IPSec*-системе есть три зарегистрированных в *DOI* протокола — *ISAKMP*, *AH* и *ESP* — и, соответственно, существует три вида контекстов защиты — *ISAKMP SA*, *AH SA*, *ESP SA*. Для каждого соединения с использованием перечисленных протоколов должен быть в наличии соответствующий контекст защиты.

Контексты защиты *SA*, согласованные на этапе установления соединения, хранятся в специальной базе данных — *SAD* (*security association database*). *SAD* формируется и поддерживается на каждом узле, где установлено программное обеспечение *IPSec*. Выбирая ту или иную запись из этой БД, приложение определяет способ защиты *IP*-пакетов, который затем реализуется в рамках согласованного канала *SA*.

База данных политик безопасности *SPD*. База данных политик безопасности *SPD* (*RFC*2401) определяет правила применения *IPSec*-системы и требования по обеспечению безопасности.

SPD задает политики безопасности при установлении защищенного соединения (*connection management SPD*) и политики защиты *IP*-трафика (*traffic SPD*) для определенных *IP*-адресов, т.е. отдельных хостов, шлюзов или маршрутизаторов, или групп *IP*-адресов, т.е. сетей.

В соответствии с определенными в *SPD* правилами к каждому *IP*-пакету может быть применено одно из следующих действий:

- *IP*-пакет может быть передан далее после обработки протоколом *IPSec*;
- *IP*-пакет может быть передан далее без изменений (т.е. пропущен);
- *IP*-пакет может быть уничтожен.

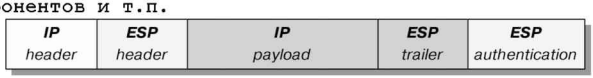
База данных политик безопасности является сугубо специфичной для каждой реализации *IPSec*-системы. Ее содержимое зависит от целевого назначения элемента вычислительной сети с установленной *IPSec*-системой

установления защищенного канала на этапе выработки *AH SA* и *ESP SA*).

(оконечный хост, межсетевой экран, шлюз или маршрутизатор).

Транспортный и туннельный режимы протокола *IPSec*. Эти режимы позволяют создавать *IPSec*-системы, предназначенные для решения задач установления защищенных соединений как между конечными адресатами (хост—хост), так и между узлами и шлюзами сетей. В зависимости от типа соединения (хост—хост, хост—шлюз, шлюз—шлюз) различают транспортный и туннельный режимы использования протокола *IPSec*. Оба режима *IPSec* обеспечивают защиту *IP*-пакетов[40]. Режим применения определяет формат *IPSec*-пакета, полученного в результате обработки исходного *IP*-пакета протоколом *AH* или *ESP*.

Транспортный режим применяется чаще всего для защиты трафика между хост-компьютерами, т.е. между двумя конечными точками соединения. Передача *IP*-пакета через сеть выполняется с помощью оригинального заголовка этого же пакета. Этот режим используется для шифрования поля данных *IP*-пакета, содержащего протоколы транспортного уровня (*TCP*, *UDP*), которое в свою очередь включает в себя информацию прикладных служб. Примером применения транспортного режима является передача электронной почты. Все промежуточные узлы на маршруте пакета от отправителя к получателю используют только открытую информацию сетевого уровня. Недостатком транспортного режима является отсутствие механизмов скрытия конкретных адресов отправителя и получателя пакета, а также отсутствие возможности проведения анализа трафика. Результатом такого анализа может стать информация об объемах и направлениях передачи информации, области интересов абонентов и т.п.



[40] Напомним, стандартный *IP*-пакет протокола *IP* (*v* 4) состоит из двух частей: заголовка и поля данных. Заголовок *IP*-пакета содержит информацию, необходимую для его передачи по сети и доставки адресату. Поле данных содержит пакет стоящего выше сетевого уровня (для архитектуры *TCP/IP* это будет *TCP*-пакет), который и является "полезной нагрузкой" *IP*-пакета.

Рис. 4.32. Модификация *IP*-пакета при применении протокола *ESP*
в транспортном режиме

IPSec-пакет формируется следующим образом (рис. 4.32). Заголовок *IP*-пакета (*IP header*) сохраняется и становится заголовком *IPSec*-пакета. За ним следуют: заголовок *ESP* (*ESP hea-der*), поле данных *IP*-пакета (*IP payload*) и так называемый *ESP*-трейлер, обеспечивающий выравнивание шифруемой части *IPSec*-пакета на необходимую длину (для применения алгоритмов шифрования требуется, чтобы данные имели определенную длину). Последним является поле аутентификации *ESP* (*ESP au-thentication*), содержащее контрольную информацию по всем пре-дыдущим полям *IPSec*-пакета, за исключением заголовка.

Поле данных и *ESP*-трейлер зашифрованы алгоритмом, который задан в контексте безопасности *SA* для данного соединения. Этим обеспечивается конфиденциальность информации, содержащейся в *IP*-пакете. Аутентичность и целостность информации обеспечивается полем аутентификации.

В случае использования протокола *AH* исходящий *IPSec*-пакет аутентифицируется алгоритмом *HMAC* как единое целое. Код аутентификации сообщения записывается в заголовок *AH*; поле данных передается в открытом виде, т.е. незашифрованным (рис. 4.33).

Формат заголовка *AH* приведен на рис. 4.34. Он состоит из 96-битового заголовка и данных переменной длины (*Authentica-tion Data* (*variable*)), состоящих из 32-битовых слов. Названия полей достаточно ясно отражают их содержимое: *Next Header* указывает на следующий заголовок, *Payload Len* представляет длину пакета, *SPI* является указателем на контекст безопасности и *Sequence Number Field* содержит последовательный номер пакета.

Это поле формируется отправителем и служит для защиты от атак, связанных с воспроизведением трафика. Так как сеть *Internet* не гарантирует порядок доставки пакетов, получатель должен хранить информацию о максимальном последовательном

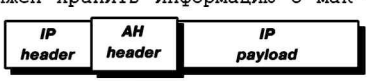


Рис. 4.33. Модификация *IP*-па-кета при применении протокола *AH* в транспортном режиме

Next Header	Payload Len	RESERVED
Security Parameters Index (SPI)		
Sequence Number Field		
Authentication Data (variable)		

Рис. 4.34. Формат заголовка *AH*

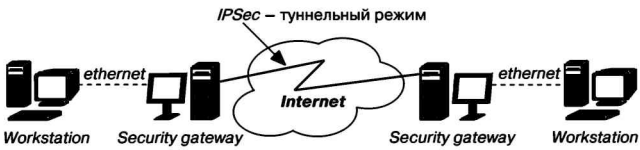


Рис. 4.35. Применение протоколов *IPSec* в туннельном режиме

Рис. 4.36. Модификация *IP*-пакета при применении протокола *ESP* в туннельном режиме

номере пакета, прошедшего успешную аутентификацию, и о получении некоторого числа пакетов, содержащих предыдущие последовательные номера (обычно это число 64). Содержащийся в заголовке *AH* индекс параметров безопасности (*SPI*) позволяет респонденту выбрать соответствующий данному сообщению контекст защиты *SA*.

Туннельный режим устанавливает защищенный канал между двумя промежуточными узлами — шлюзами безопасности (*security gateway*), на каждом из которых работает протокол *IPSec* (рис. 4.35).

Туннельный режим предполагает шифрование всего пакета, включая заголовок (*Orig. IP header*) сетевого уровня (рис. 4.36). Этот режим применяется для защиты данных и скрытия информационного обмена организации с внешним миром. Последнее достигается заменой исходного заголовка *IP*-пакета (*Orig. IP header*) новым заго-

ловком (*New IP header*) теперь уже *IPSec*-пакета. За ним следуют заголовок *ESP* и исходный заголовок *IP*-пакета. Формат оставшейся части *IPSec*-пакета такой же, как и в транспортном режиме (см. рис. 4.32).

Конечные узлы (см. рис. 4.35) — клиентские станции (*Work-station*) или серверы — не обязаны поддерживать *IPSec* и передают незащищенный *IP*-трафик через заслуживающую доверие *in-tranet* сеть. Трафик, направляемый в общедоступную сеть *Inter-net*, проходит через шлюз безопасности, который от своего имени обеспечивает его защиту с помощью *IPSec*. Шлюз помещает каждый *IP*-пакет в "оболочку или конверт" *IPSec*, зашифровывая его содержимое (*IP payload*) вместе с исходным *IP*-заголов-ком. Чтобы обеспечить возможность маршрутизации получившегося пакета, шлюз снабжает его новым *IP*-заголовком и только после этого отправляет в сеть *Internet*. Оригинальный заголовок передается в зашифрованном виде и восстанавливается на конечной точке туннельного *IPSec*-соединения: шлюз дешифрует пакет и передает его на оконечное устройство в первоначальном виде. Описанная процедура, в которой пакет одного типа (в данном случае *IP*-пакет) инкапсулируется в пакет, используемый другим протоколом (в данном случае *IPSec*), и называется туннелированием. Это позволяет передавать первоначальный пакет по сети, обеспечивающей информационную безопасность.

Процесс туннелирования включает в себя три шага: инкапсуляцию, декапсуляцию и управление туннелем. В узле инкапсуляции, или входной точке туннеля, *IP*-пакет размещается в поле данных *IPSec*-пакета подобно тому, как происходит инкапсуляция пакетов (рис. 4.37). Полученный в результате пакет *IPSec* содержит и заголовок *IPSec*-пакета (*New IP header*), и исходный заголовок *IP*-пакета (*Orig. IP header*) плюс всю

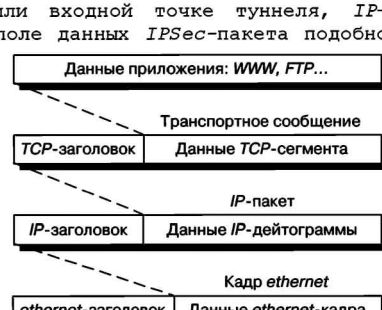


Рис. 4.37. Пример инкапсуляции пакетов в стеке *TCP/IP*

информацию верхних уровней, такую, как заголовок *TCP*, прикладные данные и др., что составляет "полезную нагрузку" *IPSec*-пакета.

На рисунке каждый *TCP*-пакет вкладывается в "конверт" протокола нижнего уровня (*IP*). Получившаяся таким образом дейтаграмма содержит в себе *TCP*-пакет, так же, как последний содержит пользовательские данные. Реализуется механизм инкапсуляции через интерфейс вызовов между *TCP* и *IP*.

Что касается управления, то узел инкапсуляции поддерживает информацию о конфигурации, относящуюся к устанавливаемому туннелю, например согласовывает параметры информационного обмена. Прежде всего определяется максимальный размер ячейки данных или так называемая максимальная единица передачи *MTU* (*maximum transfer unit*), которую можно посылать по данному туннелю. Новый заголовок (*New IP header*) считывают промежуточные маршрутизаторы, перебрасывающие пакет по сети *IPSec* к узлу декапсуляции, или конечной точке туннеля. Здесь заголовок *New IP header* проверяется и затем отбрасывается; остается заголовок *IP* (*Orig. IP header*) плюс данные, которые затем доставляются хосту назначения *IP*.

В результате туннелирования перехват *IPSec*-трафика между шлюзами безопасности не позволит восстановить архитектуру расположенной за ними сети, проследить реальный маршрут *IP*-пакета и установить адреса респондентов.

Удаленный хост может использовать при отправке пакетов шлюзу как транспортный, так и туннельный режим, шлюз же отправляет пакет хосту только в туннельном режиме. Туннельный режим на оконечных системах наиболее часто применяется для поддержки удаленных и мобильных пользователей. И хотя большая часть данных конечных пользователей пересылается шлюзами через туннели, транспортный режим используется на шлюзах для защиты внутренних связей между одноранговыми шлюзами. Последнее может быть применимо для защиты удаленного управления маршрутизаторами, коммутаторами *ATM* [41], межсетевыми экранами и другими ключевыми компонентами инфраструктуры сети.

Для обеспечения возможности комбинированного применения протоколов *AH* и *ESP* в разных режимах (туннельный и транспортный) контексты защиты объединяются в так называемые связки (*SA-bundle*). Имеется два способа связывания контекстов защиты — транспортная связка и повторное туннелирование. Транспортная связка используется на конечных точках соединения и применяет для защиты трафика сразу оба протокола (*AH* и *ESP*) в транспортном режиме. Повторное туннелирование позволяет многократно применять протоколы *AH* и *ESP* к уже защищенному в туннельном режиме трафику [142].

Таким образом, протокол *IPSec*, предназначенный для обеспечения информационной безопасности частных коммуникаций в общедоступной сети *Internet*, формирует *стандартную* платформу для построения защищенных сетей, в частности, своеобразных электронных туннелей, используемых в сетях *VPN*. Корпоративные пользователи, развертывающие набор протоколов *IPSec*, также могут защитить свою сетевую инфраструктуру, не затрагивая

---

[41] *ATM* (*asynchronous transfer mode*) — асинхронный режим передачи — универсальная транспортная сеть для передачи голоса, данных и видео. Имеет скорости передачи 25, 155, 622 и 2400 Мбит/с. Первые две разновидности могут работать по двум витым парам категории 5; аппаратура на 155, 622 и 2400 Мбит/с использует в качестве среды передачи оптический кабель. Сети *ATM* — это коммутируемые сети с установлением соединений (*connection-oriented*). *ATM* служит эффективным связующим звеном между ЛВС и *WAN*-сетями [147, 148].

отдельных приложений [146]. О доминирующем положении протокола *IPSec* в большинстве реализаций *VPN* говорится и в работе [140]. Однако, несмотря на полноту функций, протокол имеет ряд недостатков. Приведем основные из них.

Перехват *IP*-трафика для выполнения *IPsec*-обработки пакетов требует внедрения в стек *TCP/IP* дополнительного кода, иначе, реализация протокола *IPSec* требует замены или модификации существующего стека *TCP/IP*, что является нетривиальной задачей. Вызвано это тем, что расположение протокола *IPSec* на сетевом уровне архитектуры *TCP/IP* определяет жесткую привязку его программной реализации к внутренней архитектуре базовой операционной системы. Вместе с тем это дает возможность реализации *IPSec* в специализированных аппаратно-программных модулях *Cisco*. В ряде случаев это приводит к увеличению трафика локального *TCP/IP* стека и заметному снижению скорости обмена данными по сети [142].

Определенный вклад в снижение скорости обмена вносит и протокол *IKE* — сюда относятся затраты времени на криптографию с открытым ключом, на сложности согласования алгоритмов защиты, на процедуры проверки сертификатов и обмена ключами сессии и т.п. Все это негативно сказывается на производительности систем с алгоритмом *IPSec*, особенно в сетях с высокой интенсивностью трафика. Учитывая то, что в *IPSec* не установлено единственного стандартного способа распределения

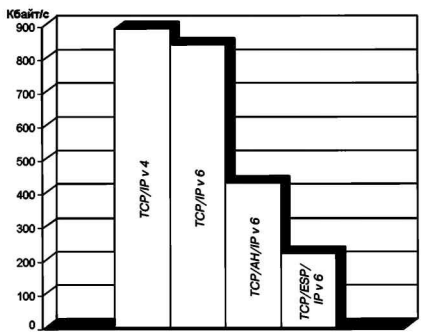


Рис. 4.38. Производительность протоколов *IP*

ключей[42], в целях сокращения временных затрат, применяют протокол *SKIP* [43] (*simple key-management for Internet protocols*) — простой протокол обмена ключами для *Internet* разработки фирмы *Sun* [148], который существенно проще, чем *IKE*.

При использовании *SKIP* ключ сессии содержится в самом пакете, в заголовке *SKIP*. Для того чтобы пакет мог расшифро-ваться только адресатом, этот ключ зашифровывается. Алго-ритм и ключ шифрования выбран так, чтобы его легко можно было вычислить без предварительного обмена (необходимо, ко-нечно, знать секретный и публичный ключи). Ключ сессии мо-жет быть разным в разных пакетах, передаваемых по сети [149].

Небезынтересными являются оценки производительности ре-ализаций протоколов *IP* различных версий и *IPSec*, приведенные в работе [150].

Протокол *IPSec* (AH) обеспечивает аутентификацию пакета *IP* в целом, тем самым предотвращая манипулирование полями *IP*-заголовка во время прохождения пакета. По этой причине протокол *AH* нельзя применять в среде, где используется меха-низм трансляции сетевых

[42] Определено, что обязательно должны поддерживаться ручное распределение ключей и протокол *IKE*. Каждый поставщик вправе дополнить этот набор собственными протоколами обмена ключами.

[43] В настоящее время *SKIP* в реализациях *VPN* доступен для *Solaris*, *Sun OS*, *FreeBSD* и *Linux*.

адресов *NAT* (*network address translation*), так как манипулирование *IP*-заголовками необходимо для его работы. Кроме того, *IPSec* шифрует номера портов *TCP* и *UDP*,

по которым можно распознать типы приложений; информацию о кадрировании, используемую для резервирования пропускной способности, и данные о последовательности передачи пакетов, необходимые для удаленного мониторинга. Некоторые новые маршрутизирующие коммутаторы снабжены функцией классификации сетевого трафика. Однако протокол защиты информации *IPSec* маскирует данные, на основе которых коммутаторы определяют приоритетность трафика [154].

Несмотря на недостатки, на сегодняшний день протокол *IPSec* — одно из наиболее эффективных средств обеспечения сетевой безопасности. Он является неотъемлемой частью следующего поколения протокола *IP* — *IP v* 6 и становится стандартом для *Internet*. Подтверждением этому является реализация протокола безопасности *IPSec* в *MS Windows* [131] и маршрутизаторах *VPN Cisco* [140, 151]. Известен также ряд практических работ, например работы [152, 153], в которых изложен процесс самостоятельного создания защищенного соединения узлов сети, использующих в качестве ОС *FreeBSD*, а в среде *Windows* — 2000/*XP* и *FreeBSD*.

4.3. Перспективные кросс-платформенные средства управления и защиты распределенных ИС

4.3.1. Управление узлами распределенной ИС

В настоящее время существует большое количество программных систем управления сетями. Как правило, это многофункциональные средства, что затрудняет их классификацию. Тем не менее, подобные системы могут быть разделены по функциональным признакам на две группы.

Первая группа состоит из систем, преимущественно ориентированных на управление сетевым оборудованием и/или компьютерами. Наиболее типичными функциями таких систем являются инвентаризация аппаратных средств, контроль версий ПО, управление конфигурацией приложений. Примером может служить ПО системного администрирования и управления корпоративного масштаба для ОС *Linux Caldera Volution* [155].

Во вторую группу входят средства мониторинга сервисов и ресурсов сети, предназначенные для профилак-

тики и быстрого выявления отказов оборудования и сбоев ПО.

Иногда к системам управления сетями относят анализаторы (или сканеры) сетевой безопасности, позволяющие заблаговременно выявлять уязвимые для сетевых атак места.

К сожалению, все существующие системы управления корпоративными сетями имеют ряд недостатков [156]. В частности, отмечается их недостаточная оснащенность для управления гетерогенными сетями, что наряду с высокой стоимостью делает нецелесообразным использование подобных систем в рамках предлагаемой архитектуры построения распределенной ИС.

Вместе с тем многие из существующих средств мониторинга, среди которых заметное место занимают программные продукты с открытым исходным кодом, хорошо приспособлены для работы в гетерогенной сетевой среде и имеют гибкую модульную архитектуру, позволяющую создать на их базе полноценные средства контроля и управления распределенной ИС.

Задачи управления распределенной ИС возлагаются на отдельную службу, состоящую из двух подсистем: администрирования и мониторинга.

Подсистема администрирования — служба, совмещающая в себе функции системного администрирования и управления сервисами ИС, размещается на каждом узле распределенной ИС (в случае, если узел состоит более чем из одного аппаратного сервера, — на каждом сервере). Эта служба имеет модульную структуру, состав модулей может быть адаптирован под конкретный набор сервисов, размещенных на сервере узла. Служба администрирования имеет *Web*-интерфейс и поддерживает защищенный протокол обмена гипертекстом *HTTPS* (протокол должен поддерживаться на стороне клиента — *Web*-браузера). Каждый узел выступает также в качестве портала для доступа к сервисам администрирования *всех* узлов распределенной ИС.

Подобное решение является компромиссным по отношению к варианту, когда задачи администрирования и управления сервисами на разнородных платформах должны

выполняться единой службой, создание и сопровождение которой являются крайне проблематичными по причинам, изложенным в работе [156].

За основу службы администрирования для узлов, базирующихся на *UNIX*-подобных системах, выбрана система *Webmin* [94], полностью отвечающая перечисленным выше требованиям.

Подсистема мониторинга осуществляет мониторинг работы сервисов распределенной ИС. К ней предъявляются два основных требования. Во-первых, она должна быть масштабируемой, т.е. эффективность и надежность ее применения не должна снижаться с увеличением числа узлов ИС. Во-вторых, служба мони-торинга должна поддерживать пассивный и активный контроль сервисов и ресурсов. В активном режиме опрос состояния инициируется службой мониторинга. Этот режим используется для контроля *Web-*, *FTP-*, *SMTP* и других *Internet*-сервисов.

В пассивном режиме информация о состоянии контролируемых объектов собирается и передается службе мониторинга внешними приложениями — агентами. Данный режим необходим для контроля асинхронных событий, например прерываний *SNMP*, а также в случае, если контролируемый объект находится за межсетевым экраном. Возможно совместное использование пассивного и активного режимов проверки для контроля одних и тех же сервисов и ресурсов. Такая избыточность повышает достоверность и надежность работы службы мониторинга.

Подобное решение используется широко распространенными сетевыми мониторами *Big Brother* [157] и *NetSaint* [158], доступными для многих современных клонов ОС *UNIX* [44].

В качестве платформы для службы мониторинга распределенной ИС выбран сетевой монитор *NetSaint* [158], полностью удовлетворяющий названным выше требованиям. Архитектура службы мониторинга представлена на рис. 4.39.

---

[44] Серверы *Big Brother* доступны также для *Windows NT*/2000, а агенты поддерживаются ОС *Novell NetWare, MacOS, OpenVMS* и др.

Непосредственно проверка состояния осуществляется модулями, каждый из которых отвечает за конкретный объект (сервис, службу или ресурс). Такое решение позволяет вводить поддержку новых объектов контроля добавлением соответствующих модулей, не затрагивая при этом остальные компоненты системы.

В активном режиме базовая логика службы мониторинга генерирует запрос проверки состояния и интерпретирует полученные результаты. Опрос состояния в активном режиме может осуществляться непосредственно (для локальных и доступных по сети объектов) или косвенно. В последнем случае с помощью специального модуля запрос передается подсистеме поддержки модулей[45], размещенной на удаленном узле, которая, в свою очередь, передает запрос соответствующему модулю. Таким образом, в активном режиме можно реализовать контроль объектов, недоступных по сети.

[45] Эта подсистема также обеспечивает шифрование передаваемой информации.

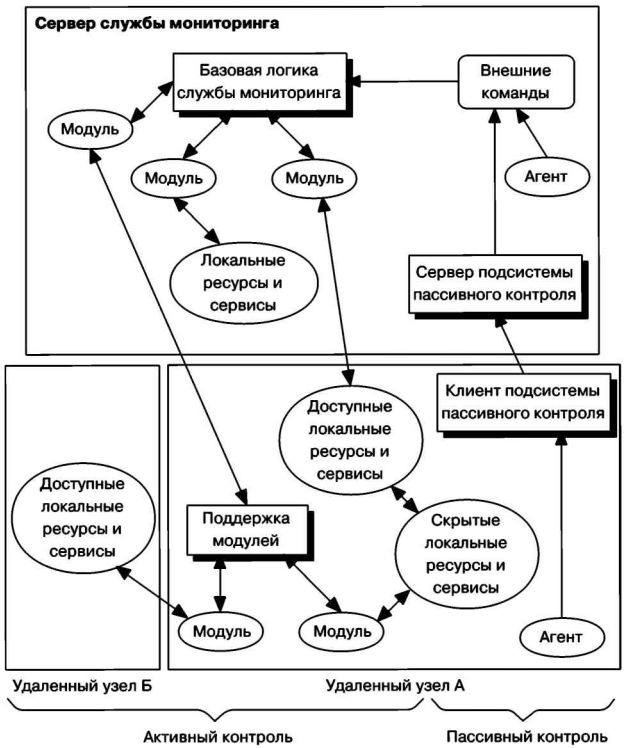


Рис. 4.39. Архитектура службы мониторинга

В пассивном режиме информация о состоянии сервисов собирается программными агентами и передается базовой логике службы мониторинга в виде внешних команд (через файл *FIFO*). Если агент располагается на удаленном узле, то для передачи команды задействуется клиент-серверная подсистема пассивного контроля, которая

также обеспечивает шифрование передаваемой информации.

Результаты активных и пассивных проверок попадают в одну очередь событий для последующей обработки базовой логикой службы мониторинга. Обработка этих результатов происходит по одним и тем же правилам, т.е. для базовой логики службы мониторинга нет никакой разницы, в каком режиме выполняется проверка состояния объекта.

С ростом числа узлов в распределенной ИС нагрузка на сервер мониторинга быстро возрастает (каждый узел содержит множество объектов, состояние которых необходимо контролировать). В данной ситуации возможно использование нескольких серверов мониторинга, при этом один из них назначается центральным. Задачей центрального сервера является интерпретация и отображение результатов, получаемых от других серверов.

Приведем алгоритм работы распределенной службы (см. рис. 4.39 и 4.40).

1. Сервер мониторинга после завершения проверки обращается к специальному локальному модулю.

2. Локальный модуль осуществляет передачу результата проверки клиенту подсистемы пассивного контроля.

3. Клиент подсистемы пассивного контроля передает информацию серверу подсистемы, расположенному на центральном сервере распределенной службы мониторинга.

4. Сервер подсистемы пассивного контроля принимает данные, содержащие результат проверки, и помещает их в файл внешних команд.

5. Базовая логика службы мониторинга считывает команды из файла и обрабатывает полученную информацию, сохраняя результаты в файле состояния. Содержимое последнего интерпретируется и отображается *Web*-интерфейсом службы.

В случае, если данные о контролируемом объекте перестают поступать на центральный сервер, возникает неопределенность в определении его состояния. Для ее преодоления служит подсистема контроля тайм-аутов. Если в течение заданного интервала времени не поступает информация о контролируемом объекте, подсистема

выдает базовой логике службы мониторинга центрального сервера команду выполнить активную проверку объекта с неопределенным состоянием. В результате выполнения активной проверки появляется достоверная информация о состоянии и доступности контролируемого объекта.

Таким образом, предлагаемая модель службы управления распределенной ИС содержит подсистему мониторинга и под-

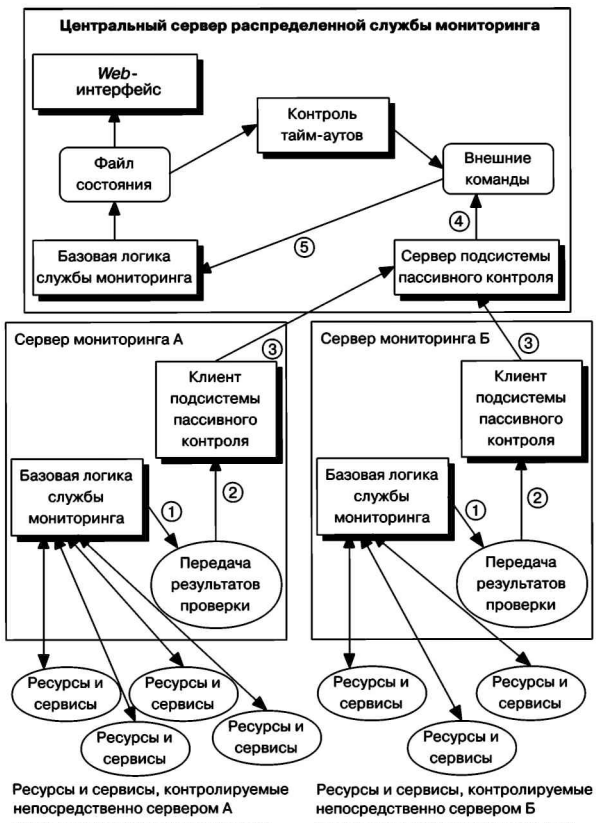


Рис. 4.40. Архитектура распределенной службы мониторинга

систему администрирования. Подсистема мониторинга охватывает узлы ИС и отображает с заданной детализа-

цией релевантную информацию о состоянии контролируемых объектов (сервисов, служб, ресурсов). Полнофункциональные серверы мониторинга размещаются на каждом узле ИС, отличаясь друг от друга только конфигурацией. Один из серверов назначается центральным. Он контролирует объекты узла, на котором расположен, а также принимает, интерпретирует и визуализирует результаты проверок, выполненных серверами мониторинга других узлов. Функции центрального сервера службы мониторинга может выполнять любой из узлов ИС. Изменение статуса сервера не требует приостановки работы ИС или ее отдельных узлов.

Подсистема администрирования, в отличие от подсистемы мониторинга, состоит из отдельных, не связанных друг с другом функционально идентичных программных компонентов[46], размещаемых на управляемых объектах (физических серверах, другом активном оборудовании). Отсутствие интегрированной среды администрирования в значительной мере компенсируется интегрированной средой мониторинга, способной выявить и локализовать неисправность или сбой в масштабе всей ИС.

4.3.2. Использование средств сетевого мониторинга и контроля трафика для увеличения уровня безопасности информационного обмена

Управление компьютерными сетями как частью распределенной ИС все более усложняется и требует для своего осуществления автоматизированных инструментальных средств. В рамках проблемы управления и защиты распределенных ИС выполнены исследования средств мониторинга и измерения трафика в локальных сетях, которые могут быть использованы для обнаружения нарушений политики безопасности, сетевых атак, а также для учета использования сетевых ресурсов (трафика) в задачах оптимизации и планирования сети.

[46] Службы, работающие под управлением ОС сервера или другого оборудования, на котором они размещены.

Выбранная основа типовой платформы мониторинга имеет модульную структуру и, в отличие от большинства существующих решений, может использоваться для количественного учета входящего, исходящего и внутрисетевого трафика, будучи расположенной в произвольном месте контролируемой сети, а не только на шлюзе или маршрутизаторе.

Средства мониторинга могут применяться в коммутируемых сегментах локальных сетей, причем использование коммутаторов с портом мониторинга (современные управляемые коммутаторы, как правило, снабжены портом, на который может транслироваться трафик всей сети) позволяет минимизировать потерю функциональности, вызванную особенностями данной топологии.

Станция мониторинга может взаимодействовать с другими аналогичными станциями и активным сетевым оборудованием по протоколам *sFlow* и *netFlow* (например, маршрутизаторами *Cisco*) для визуализации информации, собранной в разных сегментах сети, и генерации общего отчета. Доступ к собранной статистике и отчетам, а также управление станцией могут быть реализованы через *Web*-интерфейс с использованием защищенного протокола *SSL*. Информация может сохраняться в *SQL*-СУБД для последующей обработки, анализа и генерации отчетов посредством специально разработанных запросов. Возможен также экспорт данных в текстовый и *xml*-форматы.

Мониторинг сети с использованием Ntop

Ntop — это простое, свободно распространяемое, кросс-платформенное средство измерения и контроля трафика, которое поддерживает различные задачи управления сетями, включая оптимизацию и планирование, а также обнаружение нарушений политики безопасности [159].

Измерение трафика. Суть измерения состоит в определении количественных характеристик, существенных для анализа сетевой активности. *Ntop* отслеживает использование сети, генерируя статистику для каждого узла в подсети и для сети в целом. Информация собирается компьютером, на котором запущен *ntop*, посред-

ством наблюдения трафика в сети. Такое построение системы мониторинга разгружает сетевые узлы, перенося все необходимые операции по обработке и сбору данных на выделенный компьютер. Все передаваемые в сети пакеты фиксируются и связываются с отправителем/получателем. Благодаря этому возможно проследить весь трафик заданного узла сети.

В табл. 4.2 представлена информация, регистрируемая *ntop* для каждого узла, включенного в (широковещательную) сеть, а в табл. 4.3 — глобальная статистика, которую дает *ntop* о трафике.

Таблица 4.2

Информация, выдаваемая *ntop* для каждого узла сети

Параметр	Описание
Data Sent/Received	Общий трафик (объем и количество пакетов, отправленных или принятых узлом сети). Классифицируется по сетевым протоколам (*IP, IPX, AppleTalk* и т.д.) и по *IP*-протоколам (*FTP, HTTP, NFS* и т.д.)
Used Bandwidth	Актуальное, среднее и пиковое значение загрузки сети
IP Multicast	Общее количество многоадресного трафика, сгенерированного или принятого узлом сети
TCP Sessions History	Активные в данный момент *TCP*-сессии, осуществленные/принятые узлом сети, и соответствующая статистика по связанному с ними трафику
UDP Traffic	Общее количество *UDP*-трафика по портам
TCP/UDP Used Services	Список сервисов, базирующихся на *IP*-протоколах (т.е. открытые и активные порты), предоставляемых узлом сети со списком последних пяти узлов, которые их использовали
Traffic Distribution	Локальный, исходящий и входящий трафики (локальные сетевые узлы находятся в широковещательной сети)
IP Traffic Distribution	Отношение *UDP*-трафика к *TCP*-трафику и соответствующее распределение *IP*-протоколов, отсортированное по имени узла сети

Таблица 4.3

Общая статистика, регистрируемая *ntop*

Характеристика трафика	Описание
Traffic Distribution	Отношение локального трафика подсети к удаленному (за пределами заданной или

	локальной сети), удаленного к локальному
Packets Distribution	Общее число пакетов, отсортированных по размеру, отношение одновещательного пакета к широковещательному, многовещательного пакета и *IP* к не *IP*-трафику
Used Bandwidth	Актуальное, среднее и пиковое значение заг-рузки сети
Protocol Utilization and Distribution	Распределение зафиксированного трафика (локального к удаленному) по протоколам и источнику/получателю
Local Subnet Traffic Matrix	Зафиксированный трафик между каждой парой узлов в подсети
Network Flows	Статистика трафика по заданным пользователем потокам (трафик, классифицируемый по специальным параметрам, представляющим особый интерес)

Версии *ntop*, начиная с 2.0, позволяют устанавливать модули расширения, дающие детальную статистику специфических протоколов, поддержка которых отсутствует в базовой версии. Примерами могут служить модули поддержки протоколов *NetBIOS* и *NFS*. *Ntop* также генерирует статистику компьютера, на котором он выполняется, выдавая перечень открытых сетевых соединений, объем посланных/полученных данных, а также статистику узлов, с которыми связаны процессы, породившие открытые сетевые соединения.

Контроль трафика. Это, прежде всего, возможность распознавания ситуаций, при которых сетевой трафик не подчиняется заданной политике безопасности или когда некоторые из его характеристик превышают допустимые значения. Сетевые администраторы устанавливают политику, которая определяет поведение элементов в контролируемой сети, но всегда существует вероятность того, что некоторые сетевые узлы не будут придерживаться предписанной политики. Типичные причины такого поведения связаны с ошибками в конфигурации операционных систем, сетевых интерфейсов, приложений и т.п. *Ntop* позволяет обнаружить следующие проблемы конфигурации сети:

- использование повторяющихся *IP*-адресов;

• обнаружение сетевых узлов, работающих в режиме перехвата пакетов;

• обнаружение ошибок в конфигурации приложений путем анализа данных трафика протокола;

• неправильное использование сетевых сервисов;

• обнаружение сетевых узлов, которые не используют предписанные им прокси-серверы;

• использование непредусмотренных сетевых протоколов;

• выявление сетевых узлов, использующих ненужные протоколы;

• обнаружение маршрутизаторов;

• обнаружение рабочих станций, действующих вследствие неправильной конфигурации как маршрутизаторы;

• чрезмерное использование полосы пропускания сети.

Оптимизация и планирование сетевой инфраструктуры. Неоптимальная конфигурация сетевых узлов может негативно влиять на производительность сети. *Ntop* позволяет администратору идентифицировать потенциальные источники непроизводительного использования полосы пропускания, особенно использование ненужных протоколов и неоптимальной маршрутизации. Косвенным путем, через характеристики трафика и его распределение, можно пересмотреть политику работы сети для достижения более рационального использования полосы пропускания.

Обнаружение нарушений политики безопасности и сетевых атак. Если верить статистике, большинство атак на защиту сети исходит из самой сети. *Ntop* обеспечивает отслеживание атак в реальном времени и выявление потенциальных брешей в защите, включая подмену *IP*-адреса, обнаружение сетевых плат, работающих в режиме перехвата пакетов, атаки типа отказа в обслуживании, троянские программы (которые используют хорошо известные порты) и сканирование сети.

В случае обнаружения нарушения защиты сети или выявления ее неправильной конфигурации *ntop* сгенерирует сообщение для оператора сети, сигнализирующее о проблеме (через электронную почту, прерывание *SNMP* или *SMS*), и выполнит определенные действия для блокирова-

ния атаки. Поскольку существует возможность сохранения информации о трафике в базе данных, можно провести детальный анализ атаки для ее предотвращения в будущем.

Вместе с тем *ntop*, так же, как и другие средства мониторинга, может оказаться брешью в системе защиты сети, если он не сконфигурирован должным образом. Неконтролируемый доступ к *Web*-интерфейсу *ntop* позволит любому пользователю просматривать информацию о сети, которую невозможно было бы получить иным способом.

Инсталляция ntop. В настоящее время *ntop* доступен в версии 3.1. Он распространяется по лицензии *GNU GPL* [160] и может быть бесплатно загружен из домашнего сайта [159]. Поддерживаемые платформы и протоколы приведены в табл. 4.4.

Прежде всего необходимо выбрать узел — предполагаемую станцию мониторинга, — на котором будет установлен *ntop*. Трафик, захватываемый сетевым интерфейсом узла, может быть

Таблица 4.4

Основные характеристики *ntop*

Платформа	Среда передачи	Протокол	*IP*-протокол
Win32, UNIX	*Ethernet, FDDI, Lo-opback, PPP, Raw IP, Token Ring*	*Apple Talk, DecNet, DLC, IP, IPX, NetBIOS, OSI*	Задается пользователем (*DNS, FTP, HTTP, IMAP, NFS, POP, SMTP, SNMP, Telnet, X*11 и т. д.

проанализирован. В коммутируемых сетях или сетях, сегментированных мостами, будет контролироваться только тот сегмент, в котором находится станция мониторинга.

Современные коммутаторы позволяют "копировать" трафик всей сети (или виртуальных *ЛВС*) на заданный порт. Станция мониторинга может быть подключена к этому порту. К сожалению, подобное включение не поможет в том случае, если сегменты сети соединены маршрутизаторами, как это практикуется в достаточно больших *IP*-интерсетях.

Далее, необходимо выбрать формат дистрибутива *ntop*, который может быть в виде:

- исходного кода (может быть скомпилирован практически на любой *UNIX*- или *Win*32-платформе);
- исполняемого файла или бинарного пакета для различных версий *UNIX (Linux, IRIX* 6.2, *Solaris* 2.7 *i* 386/*SPARC, HP-UX* 11.X, *FreeBSD* 3.X, *AIX* 4.1);
- приложения для *Windows* (демо-версия; количество анализируемых пакетов ограничено одной тысячей).

Как *UNIX*-, так и *Win*32-версии основываются на одном исходном коде и требуют наличия библиотеки *libcap*. На поддерживаемых *UNIX*-платформах после установки *libcap* и распаковки архива с исходным кодом *ntop* компилируется и инсталлируется:

```
# cd /ntops-directory/ntop-1.3
# sh ./configure
# make
# make install
# exit
```

Если дистрибутив *ntop* имеется в виде бинарного пакета, то процесс установки зависит от используемого менеджера пакетов. Демо-версия *ntop* для *Win*32 распространяется бесплатно в виде исполняемого приложения. Полная версия доступна на платной основе. Полные тестовые версии доступны по адресу *ftp://ftp.ntop.org/pub/local/ntop/snapshots/*.

После инсталляции *ntop* должен быть запущен пользователем с привилегиями администратора системы, после чего он производит анализ передающихся по сети пакетов. Если *ntop* запущен в режиме с *Web*-интерфейсом, то он в свою очередь запустит собственный *Web*-сервер (его порт задается в качестве параметра в командной строке при запуске). Таким образом, статистика, генерируемая *ntop*, доступна с помощью *Web*-браузера через *URL*

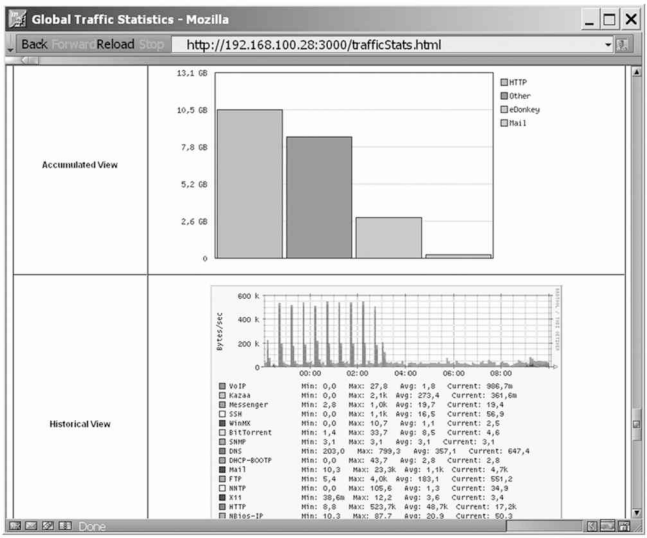


Рис. 4.41. Общее распределение трафика по *IP*-протоколам

вида *http://hostname:portnumber/*. Здесь *hostname* — имя или адрес компьютера, на котором запущен *ntop*; *portnumber* — порт, к которому привязывается встроенный *Web*-сервер *ntop* (может задаваться как параметр в командной строке).

Текущая версия *ntop* поддерживает модульные расширения (*plug-ins*). С их помощью стандартная версия *ntop* может получить дополнительные возможности. Примерами таких модулей являются средства анализа протоколов *ICMP*, *ARP/RARP* и модуль поддержки *WAP*. Инсталляция модулей опциональна, их подключение осуществляется селективно при запуске *ntop*.

Примеры использования. На рис. 4.41—4.43 представлены примеры некоторых возможностей *ntop*. Работа *ntop* отображена в режиме с *Web*-интерфейсом. На

каждом рисунке трафик изображен без привязки информации к конкретным узлам. Статистика по трафику представляет собой общую информацию о нем.

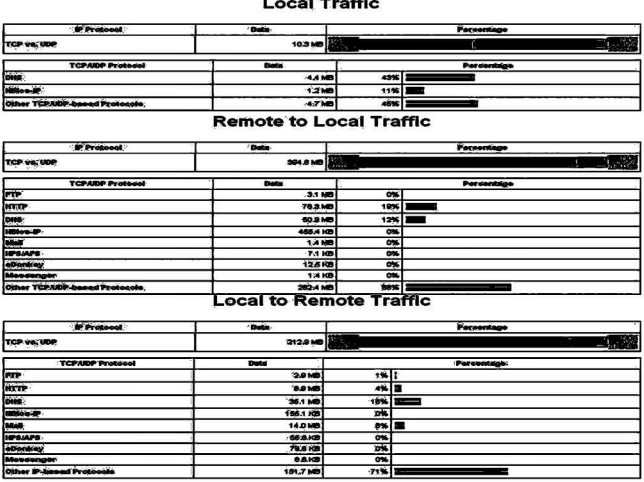


Рис. 4.42. Локальный, входящий и исходящий трафики

На рис. 4.41 приведены таблица и график, отражающие распределение трафика по *IP*-протоколам. Информация, собранная *ntop*, показывает, что наибольший трафик в сети создают *Web*-серверы и браузеры. Также видна ретроспектива потребляемого трафика, распределенного по протоколам. Данный вид статистики важен для понимания администратором сети природы трафика и порождающих его приложений. Эта информация позволяет правильно распределять и использовать пропускную способность сети.

Таблица на рис. 4.42 показывает статистику по локальному, входящему и исходящему трафику. Узел считается *локальным*, если он подключен к той же широковещательной сети, что и станция мониторинга; в противном случае он считается удаленным. Таблица "*local*

traffic" отражает обмен трафиком между локальными узлами. Из данного примера видно, что трафик *DNS* занимает 43 % от всего локального трафика. Таблица "*remote to local traffic*" представляет входящий трафик, генерируемый удаленными узлами.

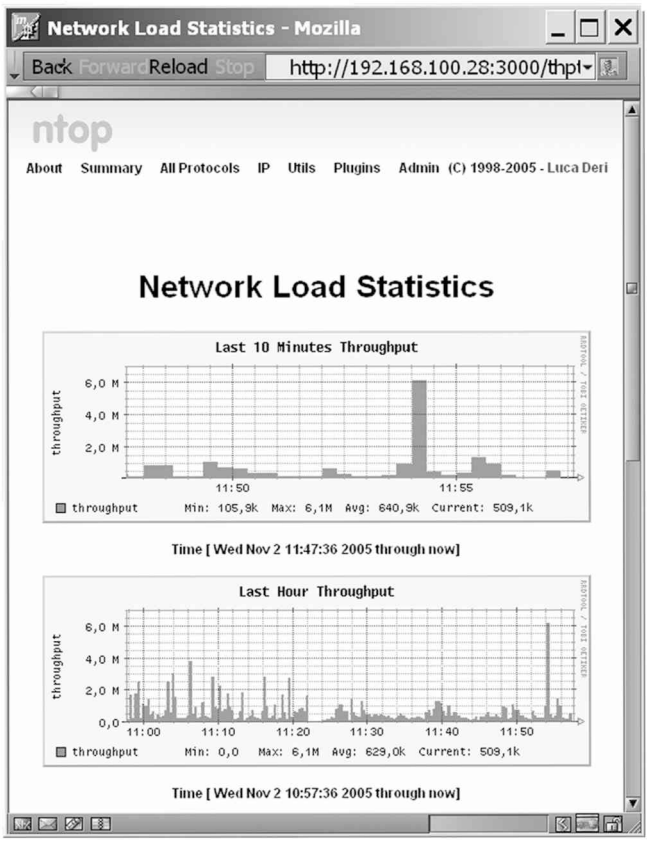


Рис. 4.43. Статистика загрузки сети

На рис. 4.43 представлен другой пример глобальной статистики по трафику — графики загрузки сети. Это

графики, показывающие во временном разрезе общую нагрузку, наблюдаемую

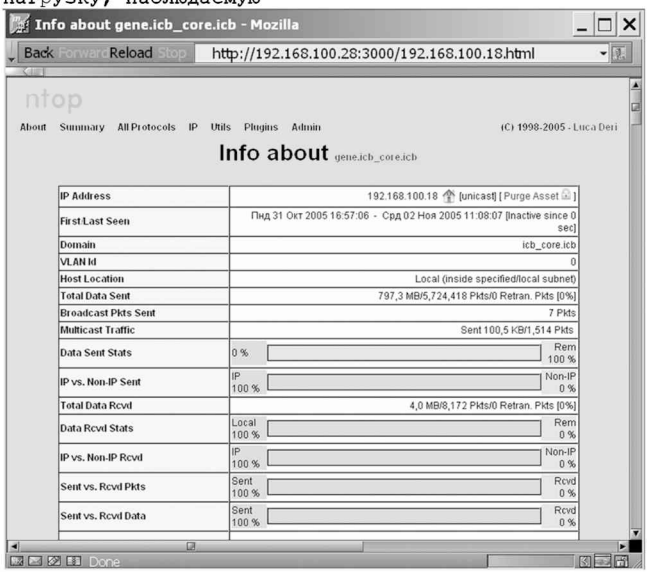


Рис. 4.44. Информация об узле

в контролируемом сегменте сети. Два графика показывают загрузку сети за разные временные интервалы — 10 минут и 1 час. Эта статистика важна для определения периодов максимальной и минимальной загрузки сети.

Таким образом, администратор сети способен оптимизировать расписание задач, требующих интенсивного использования сетевых ресурсов (резервное копирование) или влияющих на функционирование сети (конфигурация сетевого оборудования).

Представляет также интерес обнаружение непредвиденных всплесков нагрузки, свидетельствующих о чрезмерном использовании сетевых ресурсов пользователями или о других нестандартных ситуациях.

Предыдущие примеры проиллюстрировали использование *ntop* для получения обобщенной информации о трафике. На рис. 4.44
представлена часть информации, собранной об отдельном сетевом узле. Перечень сведений включает в себя *IP*-адрес, местонахождение узла (локального), общую статистику о количестве переданных и принятых данных (от локальных и удаленных узлов), количество переданных широковещательных пакетов и т.д.

Ntop может анализировать отдельные *IP*-пакеты и соотносить их с *TCP*-сессиями. На рис. 4.45 показана таблица активных *TCP*-сессий для выбранного узла, отображающая активные соединения. В таблице для каждой сессии приводятся узел-инициатор и респондент соединения, количество переданных и принятых данных, время установления соединения и продолжительность сессии.

Следуя изложенным выше принципам и механизмам построения защищенных *VPN*-соединений, описываемые средства мониторинга можно применять не только самостоятельно, но и в комбинации со специализированными маршрутизаторами. При этом можно достичь более устойчивой работы сети и повышения стойкости зашифрованных данных по сравнению со сценарием, когда для построения защищенных соединений используются только специализированные (и зачастую весьма дорогостоящие) маршрутизаторы [161].

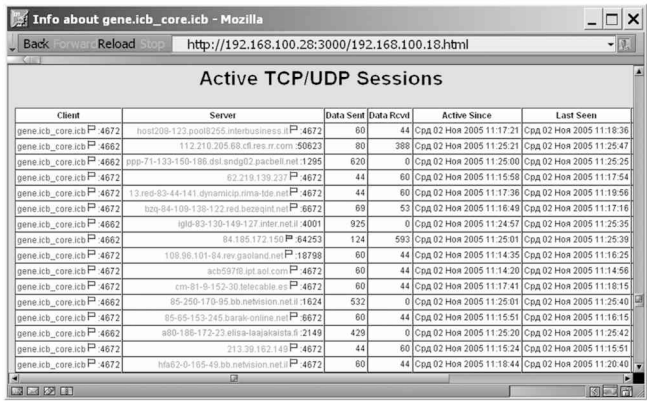


Рис. 4.45. Активные *TCP*-сеансы

ГЛАВА 5

WEB-ТЕХНОЛОГИЯ КАК СРЕДСТВО ИНТЕГРАЦИИ ИНФОРМАЦИОННОГО РЕСУРСА РАСПРЕДЕЛЕННОЙ ИС

5.1. Методы доступа к реляционным базам данных

Повсеместное использование ИС, хранящих в своих базах ресурсы различных форм и моделей представления данных (реляционные, гипертекстовые, мультимедийные и т.п.), в частности, интегрирующих реляционные и гипертекстовые БД [1], требует унификации пользовательского интерфейса для поддержания доступа ко всем используемым в ИС форматам данных [58].

Публикации [10] показывают, что и сейчас организации хранят в БД лишь около 20 % своей информации. В основном это объясняется тем, что до относительно недавнего времени реляционные СУБД были ориентированы в первую очередь на хранение и обработку сложной структурированной информации. В то же время основной объем информационных ресурсов предприятий, так называемое информационное наполнение (*content*), состоит из неструктурированных данных, хранящихся, как правило, в виде отдельных файлов (офисные документы, отсканирован-

[1] Термин *гипертекстовая база данных* используется здесь в широком смысле слова, ибо БД в узком смысле — это средство хранения структурированной, изменяемой информации, в то время как гипертекстовые БД применяются для хранения слабоструктурированной и редко изменяемой информации.

ные изображения, данные, сгенерированные компьютером, файлы *XML* и *HTML*, передаваемые по факсу сообщения, аудио- и видеоклипы, электронная почта и т. п.). Отмечается, что многие структурированные данные также зачастую хранятся в виде неупорядоченных наборов файлов, например электронных таблиц. И в этой связи создание единой информационной среды — одна из наиболее актуальных задач. Центральное место в решении этой задачи занимают СУБД. Однако доступ к реляционным БД по сетям *Internet/intranet* затруднен.

Опираясь на наиболее естественный для человека способ получения информации, *Web*-технология предоставляет универсальный, интуитивно понятный инструмент для доступа к данным. Это приемлемый и доступный подход к интеграции информационных ресурсов. Позже, проведя анализ служб (начиная с *WWW*), работающих через *Web*, выделив поколения служб и рассмотрев в контексте интеграции приложений место *Web*-служб, среди основных подходов и групп технологий для решения этой задачи, этот подход назовут "*user interface integration*" — интеграция пользовательских интерфейсов [162, 163].

Возможны несколько решений задачи доступа к данным, содержащимся в реляционных БД. Некоторые из них предполагают передачу содержимого БД на клиентскую машину и его дальнейшую обработку специально привлекаемыми для этого программными средствами. Однако их применение ограничено из-за размера базы, который может быть велик, а также из-за отсутствия на локальной машине обработчиков соответствующих баз.

Альтернативные решения поддерживают архитектуру клиент-сервер и предполагают разделение этой задачи на две составляющие:

- клиент формирует запрос к БД из *html*-документа;
- сервер обрабатывает клиентский запрос и формирует в общем случае динамическую *web*-страницу (*html*-документ).

Наиболее распространенным решением этой задачи является использование *CGI*-прикладных программ (из-за их универсальности) или *API*-модулей, расширяющих функциональность *Web*-сервера.

Технология работы таких программных расширений следующая. Необходимый пользователю запрос к базе данных выполняется из *html*-документа. Это может быть заранее сформированный статический запрос, либо запрос, формируемый пользователем путем заполнения соответствующей формы. Запрос передается *Web*-серверу, который запускает *CGI*-приложение. Последнее либо непосредственно взаимодействует с требуемой базой данных, либо транслирует запросы пользователей в операторы языка *SQL*; соединяется с базой данных и ждет результатов исполнения. Результаты выводятся *Web*-сервером пользователю в виде статической или динамически[2] создаваемой *web*-страницы.

Кратко рассмотрим каждый из методов.

5.1.1. Технологии доступа в файл-серверных системах

Стандартные обработчики файлов. Метод состоит в установке на клиентском компьютере СУБД, соответствующей просматриваемым данным. Это требует дополнительной настройки *Internet*-браузера на новый[3] тип данных (без этого он не может интерпретировать и отображать в окне просмотра все типы данных). Обычно браузер позволяет запускать другие приложения и передавать им для дальнейшей обработки принятый документ в соответствующем формате. При инсталляции программы обработки необходимо задать новый тип файлов, *MIME*-тип, подтип и выбрать приложение для их обработки [67,164].

---

[2] Динамическая генерация *html*-страниц является базовым элементом в задачах интеграции информационных ресурсов, когда содержимое страницы формируется на основе информации из БД или другого источника.

[3] Обычно для подсказки браузеру, что ему предстоит обрабатывать, сервер формирует код типа документа, основываясь на спецификации *MIME* (*multipurpose internet mail exttnsijns* — многоцелевые почтовые расширения *Internet*) [67,164]. *MIME* дает программе-обработчику возможность определить тип файла. Некоторые *Web*-серверы при передаче файла браузеру пересылают также и *MIME*-описания. *MIME*-описание состоит из двух частей: тип и подтип. Например, в *MIME*-описании *video/mpeg* тип (*type*) — *video* (видеофайл), а подтип (*mpeg*) — разновидность формата видеофайлов.

В качестве примера рассмотрим настройку браузера *Netscape Navigator* (рис. 5.1) на обработку файлов *MDB*, в которых могут храниться формы, запросы, таблицы и отчеты СУБД *Microsoft Access*. В меню ***Edit, Preferences…*** (табл. ***Helper Applications***) необходимо выполнить упомянутые выше настройки [67,164]. Табл. ***Helper Applications*** диалогового окна ***Preferences*** показывает все сконфигурированные программы обработки и типы файлов, для которых обработчик не определен. Инсталляция программы обработки для типа файла, который не представлен в списке, подробно описана в работе [67].

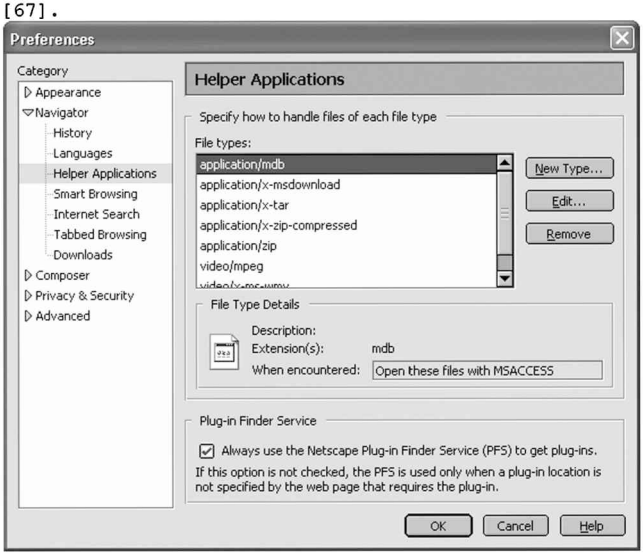


Рис. 5.1. Настройка браузера *Netscape Navigator* для обработки файлов *Microsoft Access*

Таким образом, если при просмотре *html*-документа браузер обнаруживает гиперссылку на файл с расширени-

ем *MDB*, то эта БД перекачивается по сети в ОЗУ компьютера-клиента и на этом же компьютере загружается СУБД *Access* для ее просмотра.

Использование автономных приложений в файл-серверной системе. При таком подходе для просмотра БД на компьютере-клиенте используется не *Microsoft Access*, а некоторое автономное приложение. Оно загружается, как правило, быстрее и потребляет на порядок меньше системных ресурсов, чем СУБД, в силу различия их размеров. При создании таких приложений удобно использовать, например, систему визуального программирования *Microsoft Visual Basic* [165]. Для просмотра обширной технической документации, включающей в себя текстовые документы значительного объема, чертежи, схемы, фотографии, *Visual Basic* позволяет создать в приложении встроенный *Web*-браузер (рис. 5.2).

Рис. 5.2. Просмотр технической документации *Web*-браузером, встроенным в приложение

Использование подключаемых модулей. Если в качестве браузера выбран *Netscape Navigator*, то для обработки и отображения файлов различных форматов можно также воспользоваться подключаемыми модулями (*Plug-Ins*) — небольшими утилитами, расширяющими возможности браузера и придающими ему новые, не предусмотренные разработчиками функции. Каждый подключаемый модуль поставляется с собственной инсталляционной программой, которая и выполняет интеграцию модуля с браузером. Единственно возможное препятствие к применению такого подхода — отсутствие соответствующего подключаемого модуля для определенного типа файлов.

Для просмотра перечня установленных в браузере модулей необходимо из меню ***Help*** выбрать команду ***About Plug-ins*** (о встроенных программах), после чего щелкнуть на ссылке в верхней части появившейся в окне браузера странице.

Рассмотренные подходы имеют недостатки, свойственные всем файл-серверным системам, а именно:

- БД необходимо по сети перекачивать в ОЗУ компьютера-клиента, при этом реакция на запрос пользователя последует через большой промежуток времени;
- для работы с базой используются не стандартные средства *Web*-браузера, а специально написанное (автономное) приложение, находящееся на клиентском компьютере.

Рассмотрим основные решения по доступу к базам данных, использующие архитектуру клиент-сервер.

5.1.2. Технологии, поддерживающие архитектуру клиент—сервер

Технология доступа к БД через *Web*-сервер. Для доступа к реляционным БД из *html*-документа используются программные расширения *Web*-сервера [70, 71, 166—170]. Из них наиболее распространены *CGI*-программы (сценарии). Механизм передачи данных об информационном запросе реализуется посредством включения форм ввода данных в *html*-документ на клиентской машине и с помощью работы с внешними по отношению к *Web*-серверу *CGI*-

программами, взаимодействие с которыми происходит через специфицированный протокол *CGI* (*common gateway interface*). *CGI*-программа осуществляет обработку полученной из форм информации. При этом программы браузеров не взаимодействуют с *CGI*. Обмен данными с *CGI*-приложени-ями выполняется только *Web*-серверами. Браузеры и *Web*-сер-веры взаимодействуют между собой, используя протокол *HTTP*.

На рис. 5.3 представлены основные схемы доступа пользователя к данным со стороны *Web*-сервера.

SQL Server — это программные продукты фирм *IBM, Ingres, Microsoft, Novell, Oracle, Sybase* и др. БД на основе языка *SQL* обеспечивают стабильную среду хранения данных, а СУБД имеют высокую производительность для того, чтобы организовать параллельную работу многих пользователей. Интерфейс *ODBC* (*open data base connectivity*) обеспечивает взаимодействие с произвольным сервером БД посредством языка запросов *SQL* [78, 171].

Схема 1 представляет собой почти классический вариант клиент-серверного приложения с доступом к *SQL*-серверу. Отличие заключается лишь в том, что роль приложения клиента в данном случае выполняет *CGI*-сценарий.

Выбор стандарта *ODBC* определяет библиотека функций, обеспечивающая унифицированный программный интерфейс взаимодействия с БД, а также загружаемые драйверы, преобра-

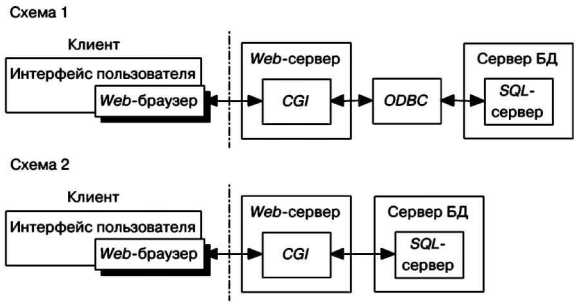


Рис. 5.3. Доступ пользователя к реляционным БД с использованием *Web*-технологий

зующие вызовы функций библиотеки *ODBC* и взаимодействующие с источником данных (транслирующие *SQL*-запросы).

Приложение, использующее интерфейс *ODBC*, поддерживает такие базовые средства взаимодействия с БД:

- соединение и отсоединение от СУБД (например, функции *SQLConnect()*, *SQLDisconnect()*);
- выполнение *SQL*-запросов и выборка результатов этих запросов (функции *SQLExecDirect()*, *SQLFetch()*, *SQLGetData()*);
- разрешение обработки транзакций в режиме он-лайн (функция *SQLTransact()*).

Это приложение поддерживает также средства, внешние по отношению к интерфейсу *ODBC*.

При разработке ИС на базе *Web*-сервиса использование схемы 1 имеет следующие преимущества:

1) возможность построения распределенных ИС при наличии соответствующих драйверов и библиотек *ODBC*;

2) независимость программного кода от используемого *SQL*-сервера, а также перенос кода на другие платформы.

Однако могут возникнуть трудности, связанные с ограниченностью набора *SQL*-запросов, поддерживаемого драйвером *ODBC*. *SOL*-запросы не позволяют воспользова-

ться в полной мере всеми функциональными возможностями конкретного *SQL*-сервера.

Схема 2 реализует взаимодействие *CGI*-сценария с сервером БД напрямую, без использования промежуточных драйверов. Ре-ализация данной схемы возможна благодаря тому, что большинство *SQL*-серверов поставляются совместно с библиотеками для различных языков программирования, реализующими *API* данного сервера. *API SQL*-сервера обычно обеспечивает как локальное, так и удаленное взаимодействие с БД. Поэтому такое решение может быть универсальным и не зависеть от программно-аппаратных платформ ИС (только при наличии библиотек *API* для различных операционных систем[4]).

Рассмотрим в качестве примера *SQL*-сервер *Postgres* [169] для *ОС UNIX*. В поставку сервера включена библиотека *LIBPQ*. Она реализует прикладной программный интерфейс с *SQL*-сервером *Postgres* для различных языков программирования. По сути, *LIBPQ* является набором процедур (функций), позволяющих клиентским программам передавать запросы серверу *Postgres* и получать на них ответы. Рассмотрим примеры базовых функций *LIBPQ* для языка Си, которые помогут понять механизм использования таких библиотек в клиентских приложениях. Для этого приведем небольшой фрагмент программы, в которой осуществляется:

- подсоединение к БД *stuff SQL*-сервера, запущенного на машине *hostname.org*. *SQL*-сервер прослушивает порт 5432;
- *SQL*-запрос к этой БД;
- вывод результатов запроса на экран;
- завершение связи с *SQL*-сервером.

```
...
PGconn * conn;
PGresult * res;
```

[4] Если протокол взаимодействия с *SQL*-сервером открыт и документирован, то возможна разработка собственных библиотек *API*.

```
...
conn=PQsetdb ("hostname.org", "5432", NULL, NULL, "stuff");
res=PQexec(conn, "SELECT * FROM people WHERE name='Иванов'");
for (i=0; i < PQntuples(res); i++) {
      for (j=0; j < PQnfields; j++)
            printf(" %-15s", PQgetvalue(res, i, j));
      printf("\n");
}
PQfinish(conn);
...
```

Преимущества схемы 2:

1) поставляемые вместе с *SQL*-сервером библиотеки *API*, как правило, максимально оптимизированы и удобны для разработки клиентских приложений, работающих с данным *SQL*-сервером;

2) использование *API SQL*-сервера позволяет реализовывать распределенные ИС, не привязанные к определенной аппаратно-программной платформе.

Недостатки схемы 2:

1) трудоемкость процесса разработки библиотеки *API* для конкретной программно-аппаратной платформы;

2) невозможность переноса программного кода. Переход к другому *SQL*-серверу требует значительных изменений ПО.

Использование схем 1 и 2 при реализации ИС широкого применения определяется возможностью построения распределенных, многопользовательских, программно- и аппаратно-независимых информационно-вычислительных комплексов.

В схемах 1 и 2 предполагается наличие *SQL*-сервера, в то же время данные можно хранить в базах с собственной структурой (например, в формате *DBF, MDB, TXT)*. Отличие такого подхода заключается в том, что *SQL*-сервер как посредник между клиентскими программами и БД не используется. Данный подход применяется при реализации "легковесных" ИС для маленьких рабочих групп, в таких случаях использование *SQL*-серверов приводит к неоправданному усложнению и удорожанию ИС.

5.1.3. Программные расширения *Web*-сервера

***CGI*-сценарии.** Программные расширения *Web*-сервера *CGI, API, Fast CGI, WebDBC* и др. [70, 71, 166–170] позволяют организовать связь между *html*-документами (*web*-страницами) и СУБД реляционного типа. Пользователю запрос к реляционной БД необходимо выполнить из *html*-документа, однако средств для формирования таких запросов в языке *HTML* нет, поскольку основное назначение *HTML* — оформление *web*-страниц и установление связей между документами. Как отмечалось, наиболее распространенный способ наращивания возможностей *HTML* — интерфейс *CGI*.

В спецификацию языка *HTML* включены специальные дескрипторы, предназначенные для создания заполняемых пользователем форм [65, 67, 172]. Браузер обеспечивает отправку введенной пользователем в форму информации на *Web*-сервер, где она

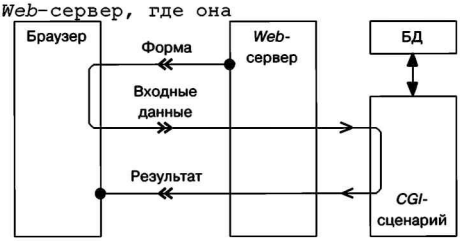


Рис. 5.4. Схема взаимодействия с *CGI*-сценарием

будет обработана некоторой внешней программой — сценарием *CGI*. Браузер непосредственно не взаимодействуют с *CGI*. Формы ввода данных обеспечивают взаимодействие с пользователем, а *CGI*-сценарий — программное расширение *Web*-сервера — обрабатывает полученную из форм информацию.

Стандарт *CGI* (общий шлюзовой интерфейс для запуска внешних программ под управлением *Web*-сервера) специфицирует взаимодействие *CGI*-сценария и *Web*-сервера. Иными словами, он описывает, как должен происходить обмен данными между *Web*-сервером и *CGI*-сценарием.

Упрощенно это выглядит так (рис. 5.4). Когда пользователь запрашивает у *Web*-сервера некоторую информацию, сервер анализирует характер запроса клиента. Если сервер определяет, что заданный *URL* указывает на *CGI*-сценарий, то он запускает этот сценарий как отдельный процесс и передает ему содержимое формы через переменные среды либо через стандартный ввод (*stdin*) в качестве входных данных. Стандартный вывод (*stdout*) процесса обычно оформляется сценарием на языке *HTML* и возвращается сервером пользователю по протоколу *HTTP*.

CGI поддерживается практически всеми *Web*-серверами и операционными системами. Сценарии *CGI* могут разрабатываться на любом языке программирования при условии соблюдения стандарта на ввод/вывод данных. Вместе с тем *CGI* является средством наиболее низкого уровня для динамического формирования гипертекстовых страниц. Этим обусловлены положительные и отрицательные стороны данного подхода при решении поставленной задачи. С одной стороны, у разработчика есть совершенно свободный выбор языка программирования и связанных с ним вспомогательных средств, что, в принципе, позволяет решать любую задачу, а с другой — реализация, отладка и сопровождение проектов при использовании *CGI* требуют слишком больших затрат на программирование.

При каждом выполнении *CGI*-сценария на сервере запускается новый процесс, поэтому ошибочный *CGI*-сценарий не может повлиять на работу *Web*-сервера. Но, поскольку создание процесса отнимает достаточно много системных ресурсов, это ведет к потере производительности сервера.

Кроме того, при использовании *CGI*-программ для доступа к базам данных из-за невозможности поддерживания непрерывного соединения *Web*-сервера и соответствующей СУБД очень сложно произвести учет пользователей БД, так как при генерации очередного запроса требуется новое подключение. Но в то же время закрытие соединения после обработки каждого запроса огра-

ничивает возможности несанкционированного доступа к БД.

У *CGI* также ограничена способность функционирования — спецификация предусматривает только простую "ответную" роль сценария при генерации результата на запрос пользователя. *CGI*-программы не имеют взаимосвязей с установлением аутентификации пользователя и проверки его входных данных.

Приведем пример *CGI*-запроса для запуска на поисковом сервере *www.altavista.com* отдельного поискового процесса.

```
FORM method=GET action="http://www.altavista.digital.com/cgi-bin/query">
<INPUT TYPE=hidden NAME=pg VALUE=q>Search
<SELECT NAME=what>
<OPTION VALUE=web SELECTED>the Web
<OPTION VALUE=news>Usenet
</SELECT>and Display the Results
<SELECT NAME=fmt>
<OPTION VALUE="." SELECTED>in Standard Form
<OPTION VALUE=c>in Compact Form
<OPTION VALUE=d>in Detailed Form
</SELECT>
<br>
<INPUT NAME=q size=45 maxlength=200 VALUE="">
<INPUT TYPE=submit VALUE=Submit><br>Search with Digital's Alta
Vista
<a href="http://www.altavista.digital.com/cgi-bin/query?pg=aq&what=
web">
Advanced Search</b></a>
</FORM>
```

В *<FORM>* определено:

- *CGI*-приложение, которое обрабатывает запрос, адресованный *http://www.altavista.digital.com/cgi-bin/query*;
- передаваемые поля — *pg, what, fmt, q*.

Кроме того, для расширенного поиска в виде ссылки выполнен *CGI*-запрос с установленными полями:

**
Advanced Search.

Пример решения задачи просмотра в *Internet*-браузере таблиц реляционной БД с помощью *Windows CGI*-приложения, использующего доступ к БД через протокол *ODBC*, рассмотрен авторами в работе [173].

API-приложения. *API*, как и *CGI*, — это программное расширение *Web*-сервера, но приложения, построенные с их *(API)* помощью, не обязательно переносятся на другие серверные платформы *Web*. Интерфейс *API* является собственностью конкретного разработчика: *ISAPI* — *Internet server API* (*Internet information server* фирмы *Microsoft*); *NSAPI* — *Netscape server API* (например, *Web*-сервера *Netscape* версии 2.0: *Enterprise Server* или *FastTrack Server* корпорации *Netscape*); *WSAPI* — сервер *WebSite* фирмы *O'Relly* и др.

В отличие от *CGI*, функции *API* зависят от типа используемого сервера, поскольку у каждого из них свой *API*. Например, интерфейс *WSAPI* полностью совместим с *ISAPI*, однако будет работать гораздо лучше, если использовать сервер *WebSite* [71, 174].

API ликвидирует один из главных недостатков *CGI* — порождение нового серверного процесса при запуске каждой *CGI*-программы. Объясняется это тем, что программой, выполняющей код *API*, является сам *Web*-сервер. Расширение *API* создается в виде библиотеки динамической компоновки *DLL*. Обращение к такой библиотеке выполняется из документа *html*, аналогично обращению к программам *CGI*, которые выполняются из форм или ссылок, созданных соответственно с помощью тегов *<FORM>* и *<A>* [66, 67, 70, 71, 172]. Когда пользователь обращается к расширению *API*, соответствующая библиотека *DLL* загружается в адресное пространство *Web*-сервера и становится его составной частью, поэтому для ее повторного выполнения не нужно создавать новый процесс. Так, если ряд пользователей одновременно обратятся к одному и тому же расширению *API*, то в память серверного процесса будет загружена одна копия библиотеки *DLL*-расширения, которая будет работать в мультизадачном режиме.

Кроме того, поскольку расширение *API* работает в рамках процесса *Web*-сервера, а не в рамках отдельного процесса (как это происходит при запуске программы

CGI), оно постоянно находится в состоянии ожидания запросов, поэтому на запуск программы и порождение нового процесса времени не требуется. *API*-приложение может пользоваться всеми доступными серверу ресурсами, т.е. иметь прямой доступ к его информационным и коммуникационным ресурсам. Это также способствует повышению производительности сервера.

Еще одно преимущество прикладных *API*-программ — способность отслеживать информацию по клиентам. Поскольку *API*-программа не высвобождает занятую ею память в промежутках между последовательными запросами от конкретного клиента, соответствующая информация может сохраняться и повторно использоваться при поступлении от него нового запроса.

Интерфейс *ISAPI* поддерживает и фильтры. Фильтры *ISAPI* также реализуются в виде библиотек динамической компоновки *DLL*. Фильтры способны контролировать весь поток данных, проходящих через сервер, на уровне протокола *HTTP*. Поэтому их можно применять для аутентификации при попытке установить соединение с защищенным портом, а также для сбора статистической информации об использовании ресурсов сервера удаленными пользователями.

Сравнивая *CGI*-сценарии и расширения *API*, отмечаем следующее. Хотя расширения *API* и превосходят по быстроте действия *CGI*-сценарии, но они работают в рамках серверного процесса, а значит, должны отлаживаться особенно тщательно. Ошибка в приложении *API* может привести к аварийному завершению всего сервера. Что же касается программы *CGI*, работающей как отдельный процесс в собственном адресном пространстве, то она практически не способна вывести из строя сервер. Если в программе *CGI* будет допущена критическая ошибка, то это приведет к аварийному завершению самой программы, но не сервера. Поскольку расширение *API* работает в мультизадачном режиме, могут возникнуть дополнительные проблемы при отладке.

Таким образом, приложениям *API* присущи следующие недостатки [169]:

• "языковая" зависимость — прикладные программы могут быть написаны только на языках, поддерживаемых в данном *API* (обычно это *C* или *C++*);

• ограниченность применения — программы, написанные для конкретного *API*, могут использоваться только на данном сервере;

• архитектурная зависимость — *API*-приложения зависимы от архитектуры сервера: если сервер поддерживает однопоточность, то многопотоковые приложения не получают преимущества в быстродействии. При изменении производителем архитектуры сервера модуль *API* обычно тоже подвергается изменениям и прикладные программы соответственно требуют переделки или даже могут быть написаны заново.

Система *FastCGI*. Она предоставляет разработчикам возможность и далее работать с *CGI*-программами, но уже с дополнительными выгодами от прямого обращения к *API*-средствам сервера. Используя интерфейс *FastCGI*, программа *CGI* загружается в память лишь один раз и остается там резидентно. Вместе с тем *FastCGI*-приложения запускаются отдельными изолированными процессами. Они являются постоянно работающими и после выполнения запроса не завершаются, а ожидают новых запросов. Благодаря этому устраняются проблемы снижения быстродействия, вызванные необходимостью постоянного повтора операций "загрузить—исполнить—выгрузить". *FastCGI* — это компромиссное решение для двух типов разработок на базе *CGI* и *API*.

Протокол *FastCGI* объединяет информацию среды, стандартный ввод, вывод и сообщения об ошибках в единое дуплексное соединение. Это позволяет *FastCGI*-программам выполняться на удаленных машинах, используя *TCP*-соединения между *Web*-сервером и *FastCGI*-модулем [70, 71, 169].

Java-технология для доступа к базам данных. *Java*-технология применяется как альтернатива *CGI*-технологии [71, 168, 175, 176]. Для взаимодействия *Java*-программы с внешним сервером БД разработан специализированный протокол *JDBC* (фирма *SunSoft*). Он позволяет использовать стандартный набор процедур вы-

сокого уровня для доступа к различным БД. Прикладной программный интерфейс *JDBC* реализуется поверх других протоколов, включая *ODBC*. Это значит, что все БД, допускающие работу с *ODBC*, будут взаимодействовать с *JDBC* без изменений.

Модель *Java* основана на том, что апплеты *Java* загружаются на клиентскую станцию с сервера и выполняются на клиенте с помощью браузера, который интерпретирует *Java*-программы. Так, при использовании *JDBC Internet/intranet*-пользователи подключаются к *Web*-серверу и загружают *HTML*-документ с *Java*-апплетом. Апплет выполняется на клиентском компьютере в среде *Web*-браузера и устанавливает связь с сервером БД с помощью протокола *JDBC*.

JDBC можно применять такими способами: когда *Web*-сервер вызывает *Java*-апплет (традиционный подход) и когда к СУБД обращается *Java*-апплет, выполняющийся на клиентской стороне, с аргументами, представленными *CGI*-строкой клиента.

Преимущество *JDBC*-подхода состоит в том, что он распределяет приложение и заставляет клиента использовать собственные ресурсы для выполнения необходимых ему функций (проблема "утолщения" клиента). При этом клиентский компьютер в приемлемое время справляется с интерпретацией всех апплетов [168, 169]. Недостаток же заключается в том, что ответственность за управление доступом к БД перекладывается на сервер, который должен отслеживать запросы от неизвестных клиентов.

Таким образом, *Java*-технология предоставляет возможность доступа к БД на стороне *Web*-клиента, в то время как *CGI*-программы осуществляют это на стороне *Web*-сервера.

Программы *WebDBC*. Аналогично работают и *WebDBC*, которые являются отдельным видом *CGI*-сценариев. Технология *WebDBC* [71] разработана для систем, ориентированных на обработку сложных многоступенчатых транзакций. Она предполагает формирование клиентского *CGI*-запроса и его дальнейшую обработку на *Web*-сервере специальной программой *WebDBC*. Программа *WebDBC* выполняет предписанные ей инструкции: транслирует запросы пользователя

в операторы языка *SQL* и взаимодействует с требуемой БД. После этого она ждет результатов исполнения. Когда результаты готовы, *WebDBC* преобразует их в формат пакетов *http* и передает через *Web*-сервер пользователю (рис. 5.5).

WebDBC имеет характерные особенности, направленные на поддержание сложных коммерческих услуг по сети *Internet*, такие, как возможность проверки подлинности клиента и его прав, ограничение доступа к различным БД и др. [71].

Инструментальные средства разработки приложений, использующих API для доступа к базам данных

Существует ряд программных разработок, которые служат для наращивания возможностей сервера и при этом избавляют от необходимости составлять сложные программы для доступа к *API* [71, 174].

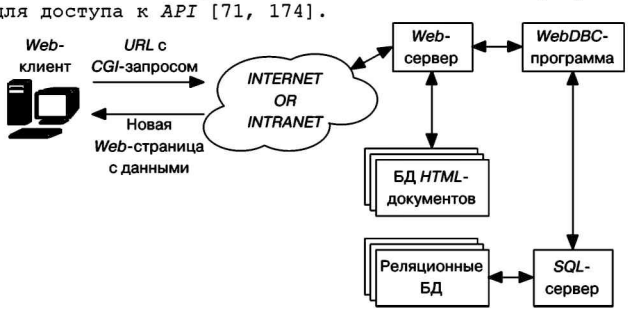


Рис. 5.5. Схема взаимодействия клиент—сервер при использовании *WebDBC*

Программный пакет *LifeWare* корпорации *Netscape*. Он позволяет разрабатывать интерактивные *Web*-приложения. Взаимодействие между *Web*-сервером и *LifeWare* осуществляется через интерфейс *NSAPI*. Библиотеки расширения *Web*-сервера обеспечивают функции доступа к реляционным СУБД *(Informix, Oracle, Sybase, ODBC)*, что

делает возможным создание с их помощью клиент-серверных приложений [71].

Для программирования в *LifeWare* используется язык *Java-Script*, операторы которого встраиваются непосредственно в *html*-документы с помощью специальных тегов *<server>* и *<client>*. Операторы, которые находятся внутри тега *<server>*, исполняют-ся на *Web*-сервере во время передачи *html*-документа клиенту. Операторы тега *<client>* исполняются клиентским браузером при получении *html*-страницы, что позволяет перенести часть логики приложения с сервера на клиентскую машину.

Инструментальная среда *Borland IntraBuilder*. Она разработана исключительно для доступа к БД с использованием *Web*-браузеров в качестве клиентских мест [177]. Преимущества *Intra-Builder* перед другими программными продуктами следующие:

- *rapid application development (RAD)* — быстрая разработка приложений на базе механизмов "*two-way tools*" и открытой компонентной архитектуры;
- производительность 32-разрядных систем — *IntraBuilder* работает под *Windows 95* и *Windows NT*, используя широкие возможности этих операционных систем;
- поддержка основных стандартов для *Web*-платформ — *Microsoft* и *Netscape;*
- поддержка большинства технологических стандартов — *Java applets, JavaScript, Microsoft Windows ActiveX, ODBC, CGI, ISAPI, NSAPI, HTML, OLE* и др. Дополняется этот набор новыми стандартами благодаря собственной открытой архитектуре;
- быстрый и эффективный механизм доступа к БД — *Borland DataBase Engine (BDE);*
- масштабируемость — доступ как к настольным СУБД, так и к серверам БД;
- расширяемость и наращиваемость — возможность объединения многих серверов в единую сеть на основе технологии *OLE Enterprise*.

IntraBuilder поддерживает *API*-интерфейсы *Web*-серверов под *Windows NT* и *Windows* 95: *NSAPI* и *ISAPI*, а также стандарт *CGI*. Клиентом *IntraBuilder* может быть любой *Web*-браузер, поддерживающий *HTML* 2.0. Для загрузки на клиентские места *Java*-аплетов необходимо использовать браузеры, которые их поддерживают.

Возможности *IntraBuilder* могут быть расширены в двух направлениях: на клиентской части (включением в приложение *Java*-аплетов[5] и элементов *ActiveX*) и на серверной части (созданием собственных объектов для среды *IntraBuilder* и подключением динамических библиотек *(DLL)*, разработанных в *Delphi* или на *C++* [174]).

Пакет инструментальных средств *Denali*. *Denali* — это условное наименование пакета инструментальных средств для обслуживания сценариев управления элементами *ActiveX* [70, 71]. Он разрабатывался *Microsoft* для фирменного сервера *Internet Infor-mation Server*. *Denali* выполняет компиляцию сценариев на сервере для достижения более высокого быстродействия. С помощью встраиваемых модулей *Active Scripting* будет обеспечиваться совместимость *Denali* с *VBScript* и *JavaScript*, а также с другими языками сценариев, например *Perl*, *Python* и *REXX*. Предусматривается обслуживание *ActiveX*-элементов. *Denali* будет содержать встроенные средства для сопровождения запросов клиентов, доступа к БД и проверки согласований.

Встраиваемые языки

Развитию технологии *CGI* способствовало появление встроенных языков (*embedded languages*). Встроенный язык — это технология, предусматривающая встраивание программы, написанной на некотором языке программирования, в текст *html*-страницы. Перед отправкой на компьютер клиента такая страница проходит предварительную обработку соответствующим препроцессором

[5] Создавать *Java*-аплеты с помощью *IntraBuilder* нельзя. *IntraBuilder* позволяет только включать *Java*-аплеты в приложения, поскольку он использует механизм *Open Connection* и имеет соответствующие средства.

(интерпретатором), который на основе *html*-кода и встроенного кода формирует результирующую *html*-страницу и передает ее *Web*-серверу. Препроцессором могут быть как внешние по отношению к *Web*-серверу программы, так и интегрированные в его систему модули. В последнем случае встроенный язык получает дополнительные преимущества в виде, например, возможности доступа к внутренним структурам данных *Web*-сервера. Кроме того, технологии доступа, базирующиеся на использовании встроенных языков, могут рассматриваться как первый этап развития технологии мобильных агентов [178]. Среди множества встроенных языков (*ASP*, *JSP*, *ColdFusion*, *Perl*, *PHP*, *Python* и др.) в последнее время, судя по всему, наиболее распространены *PHP* [179, 180] и *ASP* [181, 182]. Они и будут рассмотрены далее.

Технология *PHP*. На самом общем уровне *PHP* выполняет те же функции, что и *CGI* (обработка данных в *HTML*-формах, динамическое создание гипертекстовых страниц, взаимодействие с браузером клиента и т.п.) [173]. Синтаксически этот язык программирования напоминает *C*, *Perl*, *JavaScript*. Функциональные возможности *PHP* делают его очень удобным средством для решения таких задач, как интеграция БД и *Web*, синтаксический анализ, математические вычисления, сетевые взаимодействия (низкого уровня и с использованием сетевых протоколов высокого уровня).

Приведем пример, раскрывающий принцип работы с базами данных. Предположим, у нас есть БД *firma*, в которой имеются таблицы *prices* и *products*. В первой хранятся цены товаров (поле *price*), во второй — количество товаров (поле *num*). Ключевым полем в обеих таблицах является код товара (*kod*). Допустим, требуется вывести на *web*-страницу общую стоимость имеющихся в наличии товаров. На листинге 5.1 приведен текст программы на *PHP* и результирующая *html*-страница.

Листинг 5.1

HTML-файл, содержащий программу на *PHP*	Результирующая *HTML*-страница

```
<html>
<head>
<title>Пример 1</title>
</head>
<body>
<?php
$con=pg_connect("dbname=firma");

$query=pg_exec($con,
"SELECT
sum(prices.price*products.num)
FROM products, prices
WHERE products.kod=prices.kod");
$col=pg Fetch Row($query,0);
echo "<h3>Общая стоимость товаров на
складе: $col[0] грн.</h3><hr>\n";
pg close($con);
?>
</body>
</html>
```

```
<html>
<head>
<title>Пример 1
  </title>
</head>
<body>
<h3>Общая стоимость товаров на скла-
де:
14000 грн.</h3>
<hr>

</body>
</html>
```

Поясним ключевые, на наш взгляд, моменты. Во-первых, *Web*-сервер настраивается таким образом, что все файлы с расширением *.php3* проходят предварительную обработку соответствующим препроцессором. Во-вторых, препроцессор *PHP*, принимая на вход файл с программой, выделяет из него содержимое тэгов *<?php … ?>*. Таким образом он получает текст программы, затем интерпретирует ее, как и любой другой язык программирования. В-третьих, в результате прохождения файла через препроцессор формируется *html*-страница, которая и передается браузеру для визуализации.

В приведенном примере использовался *SQL*-сервер *PostgreSQL* и, соответственно, встроенные функции *PHP* для взаимодействия с *PostgreSQL* (*pg_**). Вначале было открыто соединение с БД *firma* на локальной машине (*$con= pg_connect ("dbname = firma")*). Затем выполнился *SQL*-оператор (*$query = pg_exec ($con, "SELECT sum (prices.price* products.num) FROM products, prices WHERE products.kod = prices.kod")*). Далее были произведены выборка результата в массив *$col* (*$col = pg_Fetch_Row ($query,0)*), вывод (*echo "<h3>Общая сто-*

имость товаров на складе: $*col*[0] грн.</*h*3><*hr*>*n*") и закрытие соединения (*pg_close* ($*con*)).

Наиболее важным свойством *PHP* для поставленной задачи является поддержка значительного количества различных баз данных. В настоящее время поддерживаются *Oracle*, *Informix*, *MS SQL Server*, *Adabas*, *Interbase*, *Solid*, *dBase*, *mSQL*, *Sybase*, *Empress*, *mySQL*, *Unix dbm* и др., а также стандарт *ODBC*. Имеется возможность интегрирования внешних библиотек для решения специфических задач, например анализа текстов на *XML*.

Одна из положительных черт *PHP* — открытость кода и кроссплатформенность, что позволяет применять его в неоднородных вычислительных средах. Наиболее распространенным является использование *PHP* с *Web*-сервером *Apache* под управлением операционной системы семейства *UNIX*.

В *PHP* реализовано следующее: поддержка объектно-ориентированного программирования; пользовательских сессий; взаимодействие с *Java*; обработка регулярных выражений; поддержка таких *Internet*-протоколов, как *LDAP*, *SNMP*, *IMAP*, *COM* (для *Windows*), а также стандарта *WDDX* для комплексного обмена данными с другими приложениями. В общей сложности в *PHP* имеется более 1000 функций для решения задач обработки информации.

Использование *PHP* для доступа к БД территориально удаленных ИС обуславливается следующим. Широкий спектр высокоуровневых средств программного интерфейса сокращает время на разработку—отладку—сопровождение сложных систем. Кроссплатформенность делает возможным объединение различных программно-аппаратных платформ. Количество поддерживаемых серверов баз данных значительно упрощает задачу интеграции неоднородных данных. Открытость исходных текстов программ обеспечивает расширение системы новыми модулями, требуемыми для решения задачи.

Чтобы получить пример *PHP*-сценария, интегрирующего распределенные информационные ресурсы, нужно несколько изменить начальные условия предыдущего примера. Предположим, что таблица *prices* находится в БД *office*

на машине с адресом *office.kiev.ua* под управлением сервера баз данных *PostgreSQL* и операционной системы *Linux*. А таблица *products* — в БД *sklad* на машине *sklad.lviv.ua* под управлением сервера баз данных *MS SQLServer* и операционной системы *Windows NT*. Задача остается такой же — найти общую стоимость имеющихся на складе товаров (листинг 5.2).

Листинг 5.2

HTML-файл, содержащий программу на *PHP*

```
<html>
<head>
<title>Пример 2</title>
</head>
<body>
<?php
// Получить данные о ценах

$con_office=pg_connect("dbname=office
user=dbuser host=office.kiev.ua");
$query_prices=pg_exec($con_office, "SELECT
* FROM prices");
for ($i=0; $i<pg_numrows($query_prices); $i++)
{
     $row=pg Fetch Row($query prices,$i);
     $prices[$row[0]]=$row[1];
}
pg close($con office);
// Получить данные о количестве товаров
$con sklad=mssql connect("sklad.lviv.ua",
"dbuser","passwd");
mssql_select_db("sklad",$con_sklad);
$query_products=("SELECT * FROM products",
$con sklad);
for ($i=0; $i<mssql_num_rows($query_products);
$i++)
{
$row=mssql_fetch_row($query_products,$i);
$products[$row[0]]=$row[1];
}
mssql_close($con_sklad);
// Вычислить общую стоимость товаров
while ( $b=each($prices) ) $sum = $sum +
$prices[$b[0]] * $products[$b[0]];
echo "<h3>Общая сумма товаров на складе:
```

Результирующая *HTML*-страница

```
<html>
<head>
<title>Пример 2
  </title>
</head>
<body>
<h3>Общая стоимость товаров
на
складе: 14000 грн.</h3>
<hr>
</body>
</html>
```

```
$sum грн.</h3><hr>\n";
?>
</body>
</html>
```

Результат выполнения, как видим, аналогичен программе, приведенной в листинге 5.1. Только в данном случае *PHP*-сцена-рий в ходе выполнения связался с двумя разными серверами баз данных, расположенными на разных машинах, которые связаны между собой через *Internet*.

В заключение перечислим новшества, появившиеся в очередной версии *PHP* [180]. Препроцессор *PHP* базируется на новом механизме — *Zend Engine* — и использует технологию предварительной компиляции и последующего выполнения кода (в отличие от построчной интерпретации и выполнения в предыдущих версиях). По некоторым оценкам это повышает скорость выполнения сценариев в 5—200 раз.

Среди новых возможностей *PHP* есть такие, как встроенные функции поддержки пользовательских *HTTP*-сессий, технологий *COM/DCOM* под *Windows*, буферизация вывода, улучшенный механизм сбора мусора. Кроме сервера *Apache*, *PHP* теперь может встраиваться в качестве модуля в другие *Web*-серверы (*MS Internet Information Server*, *AOL Server* и др.).

Технология *ASP*. Технология *Active Server Pages*[6] *(ASP)* [181—189] позволяет разрабатывать динамические *Web*-приложения на базе *html*-документа и сценариев на языках *Visual Basic* или *JavaScript*, *Perl*, *Python*. *ASP* работает с такими операционными системами, как *OS/2*, *UNIX*, *Mac OS*. К числу *Web*-серверов, поддерживающих данную технологию, можно отнести *Netscape Enterprise*, *Netscape FastTrack*, *IBM ICSS* 4.2, *ISAPI*-совместимый

[6] *ASP* — бесплатное приложение из пакета *IIS* 3.0 или старших версий *Internet Information Server*.

сервер и серверы под *UNIX*-платформу [181], среди которых наиболее известен *Web*-сервер *Apache*.

Интеграция интерпретатора языка программирования в ПО *Web*-серверов является более эффективным подходом по сравнению с его вызовом как отдельного процесса ОС. В отличие от *CGI*, где для каждого пользователя запускается процесс (рис. 5.6), в *ASP* создается поток (рис. 5.7). Для переключения процессов необходимо сохранение регистров процессора в стеке, а для переключения потоков этого не требуется. Представим, что к серверу обратились *N* пользователей одновременно. При использовании *CGI* имеем *N* процессов, а в случае *ASP* — один *OLE*-объект и *N* потоков к нему (см. рис. 5.7).

Основу *ASP*-файлов составляет язык программирования *Visual Basic*, широко используемый в продуктах *Microsoft Office*, поэтому программные скрипты стали называть *Visual Basic Script* (*VBScript*) [182]. *VBScript* может выполняться как на стороне кли-

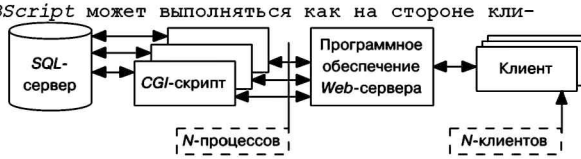


Рис. 5.6. Схема формирования документов с использованием *CGI*-скриптов

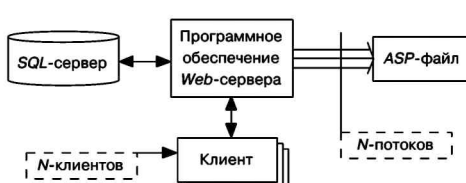


Рис. 5.7. Схема формирования документов с использованием *ASP*-технологии

ента, так и на стороне *Web*-сервера. При обработке *ASP*-файла на стороне *Web*-сервера последний интерпретирует программный код, создавая обыкновенный *html*-документ. Для лучшего понимания сути разработок приложений на основе *ASP*-техно-логий обратимся к листингу 5.3.

Файл, содержащий *ASP*-код, размещается на *Web*-сервере в подкаталоге для исполняющихся скриптов и в данном случае служит для приветствия пользователя, вошедшего в систему. *VBScript* выделяется с двух сторон знаком % и заключается в угловые скобки. Интерпретация браузером полученного *HTML*-кода приведена в правой колонке листинга 5.3. Этот *HTML*-код автоматически генерируется *Web*-сервером при обращении к *ASP*-файлу. Результат зависит от параметра, переданного в *ASP*-файл из внешнего источника. Внешним источником может быть ссылка из другого *html*-документа, вызов документа из *ASP*-файла, сгенерированного скриптом, либо из *CGI*-программы. Параметр, передаваемый в *ASP*-файл, может быть как статичный, так и генерируемый программным путем. Приведем пример жестко фиксированного параметра: *<a href="/CGI-bin/ ErrorCode.ASP?ID=*3">Вход

Листинг 5.3

HTML-файл, содержащий программу на *ASP*	Результирующая *HTML*-страница
<html> <head> <title>Welcome</title> </head> <body > < % ID = Request.QueryString("ID") Select Case ID Case 1 NameOut = "Незарегистрированный пользователь! Доступ запрещен."	<html> <head> <title>Welcome</title> </head> <body> Добро пожаловать в систему! </body> </html>

```
    Case 2
     NameOut = "До свидания!"
    Case 3
     NameOut = "Добро пожаловать в систе-
     му!"
 End Select
%>
<h1><i>< %=NameOut %></i></h1>
</body>
</html>
```

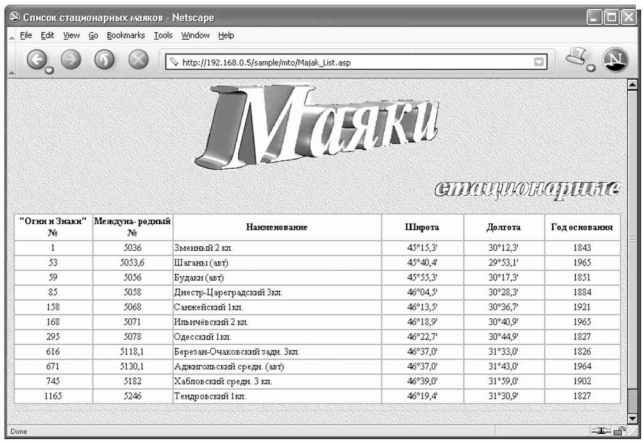
в систему пользователя №1</*a*>. Знак вопроса сообщает *Web*-серверу, что дальнейший текст определяет имена и значения параметров, передаваемых в документ *Error Code.ASP*. Параметр *ID*=3 загружается в окружение *Web*-сервера перед интерпретацией *ASP*-кода.

Стандартные функции любого языка программирования позволяют решать довольно большой объем задач, но при необ-ходимости выхода за рамки стандартных функций реализация поставленной задачи значительно усложняется. Язык *VBScript* является масштабируемым, его функциональные возможности расширяются за счет использования объектов операционной системы [183]. Объектно-ориентированный подход [184], применяемый в *ASP*-технологии, хорошо иллюстрируется примером *ASP*-кода, реализующего выборку данных из БД под управлением *MSSQL Server* (листинг 5.4).

Листинг 5.4

```
<html><head><title>Состав сооружений</title></head><body>
…
< %
Param = Request.QueryString("Param")
Data = Request.QueryString("Data")
Session.timeout = 5
If IsObject(Session("MTO_conn")) Then
  Set conn = Session("MTO_conn")
Else
```

```
  Set conn = Server.CreateObject("ADODB.Connection")
  conn.open "MTO","user","passwprd"
  Set Session("MTO_conn") = conn
End If
  SQL = "SELECT * FROM [Bashn] "
  If cstr(Param) <> "" And cstr(Data) <> "" Then
    SQL = SQL & " WHERE [" & cstr(Param) & "] = " & cstr(Data)
  End If
  Set rs = Server.CreateObject("ADODB.Recordset")
  rs.Open SQL, conn, 3, 3
  On Error Resume Next
  rs.MoveFirst
  do while Not rs.eof
%>
…
    <td width="45 %"><font COLOR="#000000" size="3"><u><strong>
< %=Server.HTMLEncode(rs.Fields("Tip").Value) %></strong></u></font></td
></tr>
…
< %= rs.Fields("God").Value %></font></td></tr>
…
 < %= rs.Fields("Visota").Value %></font></td></tr>
…
< %= rs.Fields("Sostojanie").Value %></font></td></tr></table><hr>
< %rs.MoveNext
loop %>
</body>
</html>
```



"Огни и Знаки" №	Междуна- родный №	Наименование	Широта	Долгота	Год основания
1	5036	Змеиный 2 кл.	45°15,3'	30°12,3'	1843
53	5053,6	Шаганы (авт)	45°40,4'	29°53,1'	1965
59	5056	Будаки (авт)	45°55,3'	30°17,3'	1851
85	5058	Днестр-Цареградский 3кл.	46°04,5'	30°28,3'	1884
158	5068	Сынжейский 1кл.	46°13,5'	30°36,7'	1921
168	5071	Ильичёвский 2 кл.	46°18,9'	30°40,9'	1965
295	5078	Одесский 1кл.	46°22,7'	30°44,9'	1827
616	5118,1	Березан-Очаковский задн. 3кл.	46°37,0'	31°33,0'	1826
671	5130,1	Аджигольский средн. (авт)	46°37,0'	31°43,0'	1964
745	5182	Хабловский средн. 3 кл.	46°39,0'	31°59,0'	1902
1165	5246	Тендровский 1кл.	46°19,4'	31°30,9'	1827

Рис. 5.8. Виртуальная *HTML*-страница из реляционной БД ГосНГИС

Результат работы данного сценария приведен на рис. 5.8.

Внедренные программные участки образуют линейную схему: вначале идет инициализация всех переменных, а затем цикл основной программы. Для доступа к БД *VBScript* использует подключаемую библиотеку *DAO* (объекты доступа к данным) [185].

DAO предоставляет разработчику режим подключения клиент—сервер, называемый *"ODBCDirect"*, где устанавливается прямое подключение к источнику данных *ODBC*, без необходимости загрузки ядра БД *Microsoft Jet* в память. Это полезно в ситуациях, требующих использования специфических характеристик *ODBC*, таких, как структура построения, архитектура организации *ODBC*-драйвера.

Пользователь имеет возможность с помощью средств *ODBCDirect* отправлять запросы *DAO* на различные серверы баз данных. При этом следует учитывать, что разные серверы могут использовать отличающиеся диалекты языка *SQL* и вся ответственность по синтаксису возлагает-

ся на разработчика. В рассмотренном примере это диалект *Microsoft SQL Server*.

Инициализация доступа к БД происходит при передаче *SQL*-запроса в качестве параметра объекту *DAO* (листинг 5.5). Общая

Листинг 5.5

```
< %
SQL = "SELECT * FROM [Bashn]"
   If cstr(Param) <> "" And cstr(Data) <> "" Then
     SQL = SQL & "WHERE [" & cstr(Param) & "] = " & cstr(Data)
   End If
   Set rs = Server.CreateObject("ADODB.Recordset")
   rs.Open SQL, conn, 3, 3
   On Error Resume Next
   rs.MoveFirst
%>
```

схема взаимодействия пользователя с базами данных в рамках *ASP*-технологии подобна приведенной на рис. 5.7.

В случае реализации тех же действий на основе *CGI*-скрипта программный код займет больший объем и, следовательно, потребует повышенных трудозатрат. При необходимости оперативной корректировки кода в *ASP*-приложениях это решается намного быстрее, чем при использовании *CGI*-скриптов.

Преимущества технологии *ASP* состоят в следующем:

- наличие большого числа средств визуальной разработки *ASP*-файлов (например, *Front Pages, Access*) позволяет значительно сократить время разработки *Internet*-приложения;
- применение технологии потоков повышает эффективность функционирования *Web*-сервера;
- возможность использования любого объекта, существующего в данный момент в ОС, значительно расширяет функциональность *Internet*-приложений, причем, принимая во внимание современные средства разработки объектов (*Delphi, C, C++* и др.), расширяет практически неограниченно.

Недостатком *ASP*-технологии можно считать зависимость от платформы. Хотя в настоящее время и делаются попытки переноса этой технологии на *UNIX*-серверы, но весь ее смысл состоит в тесной интеграции с такими технологиями *Microsoft*, как *COM*, *ActiveX*, *DHTML*, *OLEDB*, которые отсутствуют в других операционных системах.

Выводы

Анализ технологий и подходов к интеграции информационного ресурса ИС, приведенный в обзорной части главы, а также опыт использования отдельных технологий при создании конкретных разработок [4, 60, 62] позволяют сделать два важных вывода. Во-первых, практически все технологии, направленные на автоматизацию процесса создания интегрированных решений, не являются универсальными по отношению к платформам серверов и клиентов. Во-вторых, средства, позволяющие создавать переносимые (в идеале — платформонезависимые) решения, еще достаточно сложны в реализации.

Доступ к структурированным данным, хранящимся в базах данных ИС, с предоставлением пользователю информации в виде динамически создаваемых документов через интерфейс среды *WWW* — *Web*-браузер осуществляется посредством программных расширений *Web*-сервера. Из их числа наиболее распространены, в силу их универсальности, сценарии *CGI* (*CGI-scripts*). Они поддерживаются практически всеми *Web*-серверами, ОС и могут разрабатываться на любом языке программирования. Однако реализация, отладка и сопровождение проектов с использованием *CGI* требуют слишком больших затрат на программирование. Поэтому сегодня применение встроенных языков является наиболее приемлемым подходом к использованию *Web*-технологии при объединении информационного ресурса распределенных ИС. Этому способствуют среды разработки, которые постоянно совершенствуются, простота построения пользовательского интерфейса, широкий спектр прикладного программного интерфейса, малое время на разработку, отладку, сопровождение и др.

ГЛАВА

ПРИМЕРЫ ПОСТРОЕНИЯ ЭКОНОМИЧНЫХ РАСПРЕДЕЛЕННЫХ ИС

6.1. Государственная навигационно-гидрографическая ИС

При решении вопросов обеспечения безопасности мореплавания необходимо использовать значительное число источников информации, регламентирующей различные аспекты судоходства. Этот трудоемкий процесс может быть ускорен путем применения современных информационных технологий, поз-воляющих организовать эффективный поиск необходимой пользователю информации в специально организованных и соответствующим образом структурированных БД, касающихся той или иной предметной области.

Автоматизировать процессы накопления, хранения и поиска информации по вопросам мореплавания призвана созданная в соответствии с Государственной программой "Гидрография" ГосНГИС [61—63], приведенная на рис. 6.1.

Учитывая территориальную разобщенность служб навигационно-гидрографиче-ского обеспечения мореплавания, ГосНГИС создавали как территориально распределенную ИС. ГосНГИС включает в себя базовый узел и периферийные [1] узлы, размещенные на государственных гидрографических предприятиях. Архитектура ИС (рис. 6.2) может

[1] Деление на базовый и периферийные узлы отражает всего лишь их территориальную обособленность и ни в коем случае не затраги-

включать в себя и другие периферийные узлы, которые содержат прикладные (тематические) БД, например, в картографическом производстве, службах оповещения и т.п. (рис. 6.3). Периферийные узлы функционируют как автономно, обеспечивая наполнение и корректировку баз данных, так и в составе ГосНГИС, предоставляя доступ удаленным пользователям к содержимому баз данных и возможность поиска в них информации с заданными свойствами.

Таким образом, информационный ресурс системы сосредоточен в базовом и периферийных узлах ИС. Он представляет собой, с точки зрения пользователей, единую информационную среду. При необходимости она может быть расширена за счет виртуально присоединяемых к ГосНГИС через *Internet* ресурсов глобального информационного пространства из заданной предметной области.

Центральным звеном ГосНГИС является ее информационная база. Информация в ней структурирована по тематическим разделам, отвечающим наиболее актуальным вопросам деятельности Гидрографической службы Украины (рис. 6.4). Каждый тематический раздел включает в себя ряд подразделов.

Тематический раздел **МОРСКОЕ ЗАКОНОДАТЕЛЬСТВО** имеет такие подразделы: ***Международное (Мировое); Региональное (Государственное и Межгосударственное); Национальное***. В подразделе ***Государственное законодательство*** для каждого из государств мира приводятся законодательные акты, правительственные постановления, заявления, ноты и т.п., регламентирующие режимы экономической зоны, территориального моря, портов, проливов и каналов, континентального шельфа, определяющие правила судоходства, регулирование рыболовства и т.д.

Подраздел ***Межгосударственное законодательство*** для каждого из морских регионов (Азовское море, Атлантический океан, Баффинов пролив и т.д.) имеет информационные блоки: *Договора, Декларации, Конвенции, Протоколы и Соглашения*.

В подразделе ***Национальное законодательство*** приведена структурированная законодательная база, регламентирующая морехозяйственную деятельность в Украине по направлениям: судоходство, добыча (использование) морских ресурсов, экономическая зона, континентальный шельф, охрана окружающей природной среды, соглашения по Черноморскому флоту, навигационно-гидрографическое обеспечение морей, научные исследования и т.п.

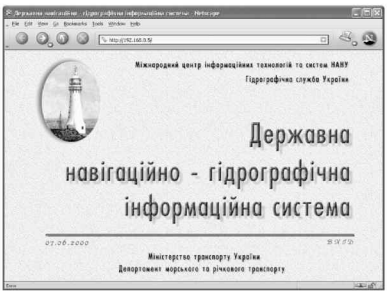


Рис. 6.1. Домашняя страница *Web*-сайта ГосНГИС

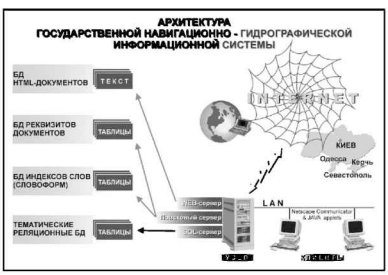


Рис. 6.2. Архитектура ГосНГИС

Рис. 6.3. Фрагменты цифрового видеоклипа о ГосНГИС

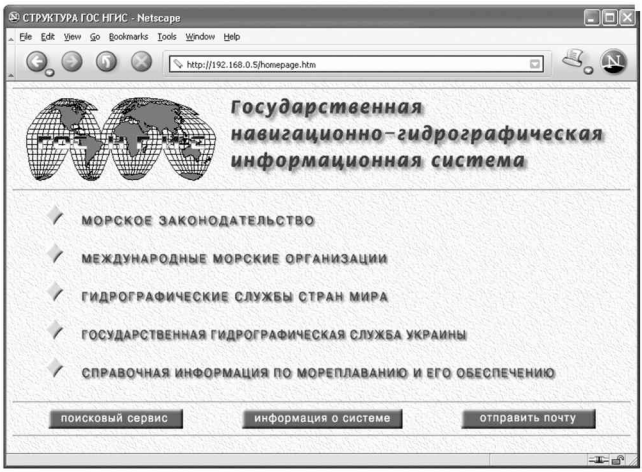


Рис. 6.4. Главная страница ГосНГИС

Тематический раздел **МЕЖДУНАРОДНЫЕ МОРСКИЕ ОРГАНИЗАЦИИ** содержит такие информационные блоки: *Международные организации по изучению Мирового океана; Международные организации по морскому судоходству; Международные организации по рыболовству; Другие международные организации*.

В каждом информационном блоке содержатся перечень организаций на русском и английском языках (рис. 6.5), а также содержатся ссылки на реквизиты конкретных организаций, в том числе *Internet*-адреса *Web*-сайтов и электронной почты (*E-mail*). Пользователь может ограничиться имеющейся в ГосНГИС информацией достаточно общего характера о каждой из этих организаций, либо задать *Internet*-адрес интересующей организации и получить доступ к мировым информационным ресурсам.

Тематический раздел **ГИДРОГРАФИЧЕСКИЕ СЛУЖБЫ СТРАН МИРА** содержит сведения о национальных гидрофизических службах стран мира (рис. 6.6).

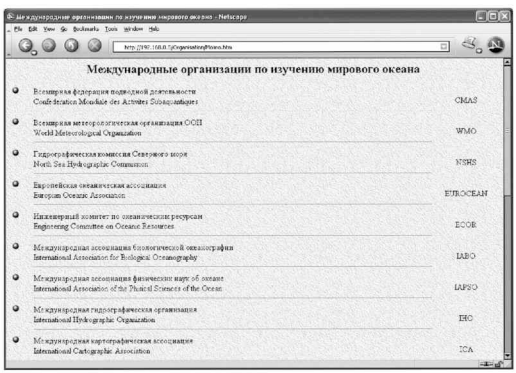


Рис. 6.5. Страница ГосНГИС международных организаций по изучению Мирового океана

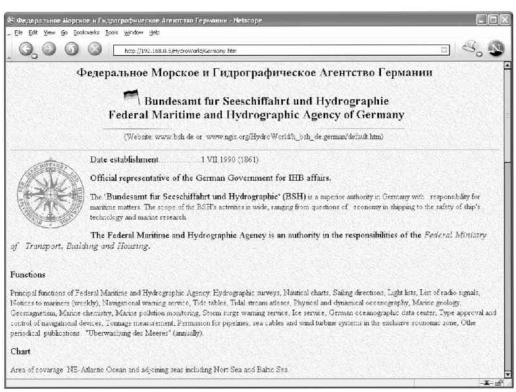


Рис. 6.6. Страница ГосНГИС федерального морского и гидрографического агентства Германии

В тематический раздел **ГОСУДАРСТВЕННАЯ ГИДРОГРАФИЧЕСКАЯ СЛУЖБА УКРАИНЫ** входят такие информационные блоки: *Статус, структура, задачи; Планы работ (текущие, перспективные); Гидрографические экспедиции, работы; Навигационное оборудование театров; Техническое и материальное обеспечение; Пресса; Почта.*

Блок *Техническое и материальное обеспечение* представлен банком данных БД_ТМО (рис. 6.7). Он содержит структурированные периодически обновляемые данные об эксплуатационных характеристиках навигационно-гидрографического оборудования; предназначен для хранения, поиска, анализа, обработки и передачи пользователям информации в электронном виде о технических ресурсах гидрографических предприятий.

В блоке *Пресса* содержатся публикации, имеющие непосредственное отношение к проблемам Государственной гидрографической службы Украины и ее деятельности.

Тематический раздел **СПРАВОЧНАЯ ИНФОРМАЦИЯ ПО МОРЕПЛАВАНИЮ И ЕГО ОБЕСПЕЧЕНИЮ** состоит из следующих блоков: *Глобальная морская система связи при бедствии и для обеспечения безопасности* (ГМССББ); *Навигационно-гидрографические издания* с каталогами на воды Украины и воды Мирового океана (Лоции, Описания, Атласы, Каталоги, Руководства, Инструкции, Методики, Правила, Положения, Пособия, Справочники, Таблицы, Ежегодники, Периодические издания, Обзоры, Извещения мореплавателям, Электронные навигационные карты); *Навигационно-гидрографические приборы; Судоходство; Порты; Судостроение и судоремонт.*

В этом разделе пользователь найдет: руководства по Всемирной службе навигационных предупреждений (ВСНП); международной автоматизированной службе НАВТЕКС; материалы кни-ги "Радиотехнические средства навигационного оборудования Черного и Средиземного морей"; инструкции по промеру и маршрутному промеру; правила гидрографической службы; наблюдения за гидрометеорологической обстановкой, информацию об издаваемых в Украине журналах "Порты Украины", "Судо-ходство" и

многое другое. Пример представления в ГосНГИС книги Я.Е. Шпигельмана и В.М. Воронцова "Справочник дунайского капитана" приведен на рис. 6.8.

Механизм навигации в информационном пространстве ГосНГИС аналогичен навигации в *Internet*. Браузер обеспечивает удобную навигацию между документами, переходы по гиперссылкам, поиск необходимой информации в просматриваемом документе, сохранение частей документов на локальном диске, вывод их на

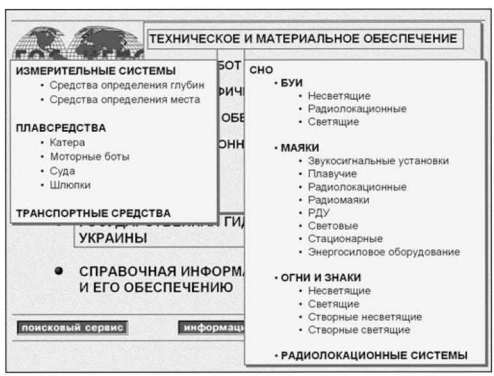


Рис. 6.7. Фрагмент из цифрового видеоклипа о ГосНГИС

Рис. 6.8. Фрагмент представления книги "Справочник дунайского капитана"

печать и др. Глубина поиска в тематических разделах, переход по ссылкам в другие разделы и возвращение в предыдущие формировались определенным образом при создании гипертекстовой БД. Термин "гипертекстовая база данных", имеющий отношение к слабоструктурированной и редко изменяемой информации, используется здесь в широком смысле слова.

Для работы с морскими электронными навигационными картами (ЭНК) — информационный блок *Навигационно-гидрографические издания* — используется специально написанная программа *dkexpand*, обеспечивающая совместно со свободно распространяемой программой-просмотрщиком карт *DKLOOK* (*dKart Explorer*) [190] компании Моринтех (С.-Петербург) просмотр браузером картографических данных в формате *dKart*, принятом в РФ и некоторых странах СНГ. *DKLOOK* служит для визуализации наборов картографической информации в международном формате *DX*-90 или *DDF* и предоставляет пользователю такие возможности работы с картой, как включение/выключение логических слоев, изменение масштаба изображения карты, получение информации из локальной базы данных об атрибутах какого-либо объекта карты и др. (рис. 6.9).

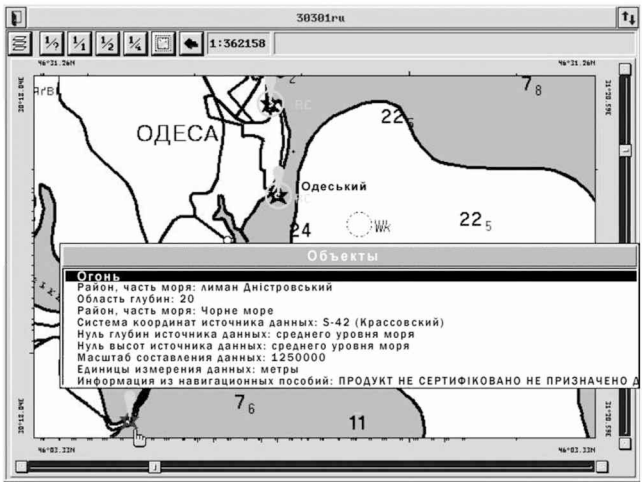


Рис. 6.9. Фрагмент карты

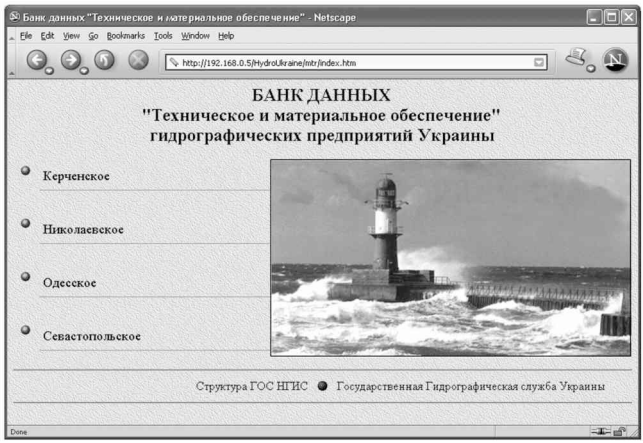


Рис. 6.10. Фрагмент банка данных "Техническое и материальное обеспечение"

Доступ к структурированной информации, хранящейся в реляционных БД ГосНГИС, например в блоке *Техническое и материальное обеспечение* — БД_ТМО — тематического раздела **ГОСУДАРСТВЕННАЯ ГИДРОГРАФИЧЕСКАЯ СЛУЖБА УКРАИ-НЫ** (см. рис. 6.4), осуществляется из браузера с использованием *ASP*-технологии [182]. В своей основе *ASP* — это встраиваемый в *html*-документ сценарий, который исполняется на сервере подобно *CGI*-приложению (см. гл. 5). Код этого сценария динамически выполняется при запросе требуемых данных, а получившаяся виртуальная *html*-страница отправляется программе просмотра.

БД_ТМО работает под управлением *Microsoft SQL*-сервера. Использование интерактивно настраиваемых *html*-страниц позволяет организовать быстрый доступ к требуемой информации. Так, установив курсор на строку с названием информационного блока *Техническое и материальное обеспечение* и щелкнув "мышью", пользователь увидит на экране перечень территориальных гидрографи-

ческих предприятий, в которых в соответствии с зонами их ответственности сосредоточены ресурсы по техническому и материальному обеспечению Госгидрографии Украины (рис. 6.10).

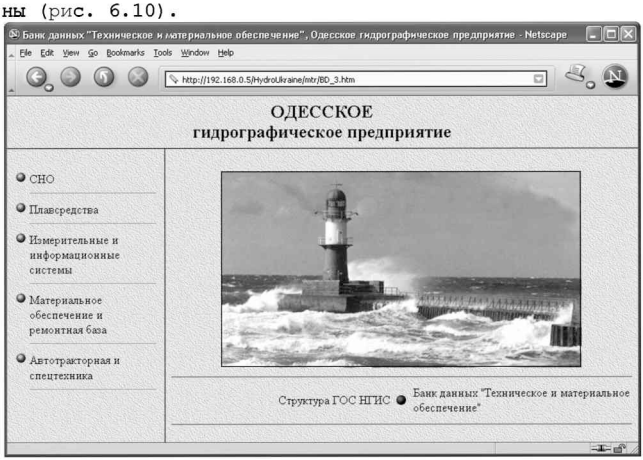


Рис. 6.11. Фрагмент банка данных "Техническое и материальное обеспечение"

Установка курсора на поле с надписью "Одесское гидрографическое предприятие" (см. рис. 6.10) и нажатие левой кнопки "мыши" вызовет в левой части экрана появление фрейма, в котором приведена описанная выше структура БнД технического и материального обеспечения.

В свою очередь выбор СНО (рис. 6.11) приводит к смене изображения маяка в правом фрейме на описание детальной структуры в части СНО — средства навигационного оборудования морей (рис. 6.12). При подводе указателя к строке "Маяки стационарные" выдается список маяков, находящихся в ведении Одесского гидрографического предприятия, с указанием их основных атрибутов: номер по книге № 2217 "Огни и знаки", междуна-

родный номер, наименование и др. (см. рис. 5.8). На рис. 6.13 показана дальнейшая работа пользователя со стационарными маяками.

Несмотря на сложность структуры БД_ТМО, пользователь достаточно легко получает необходимую информацию, по существу, не замечая, с чем работает — с реляционной или гипертекстовой базой данных.

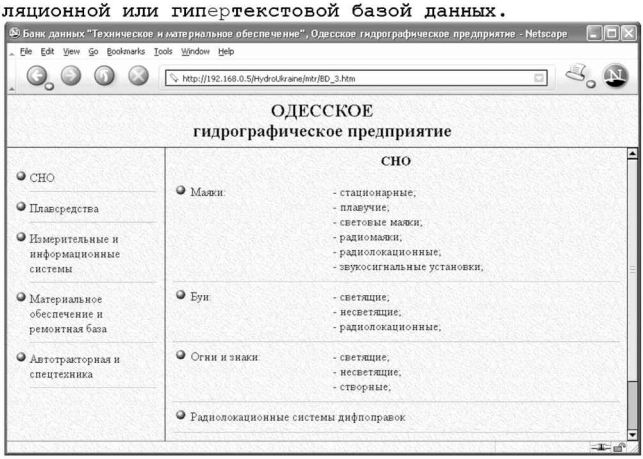


Рис. 6.12. Фрагмент банка данных "Техническое и материальное обеспечение"

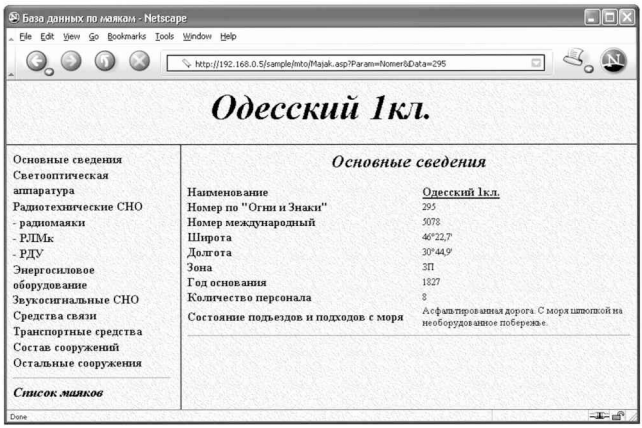


Рис. 6.13. Фрагмент банка данных "Техническое и материальное обеспечение"

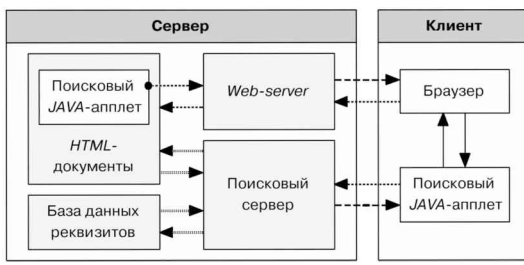


Рис. 6.14. Схема функционирования поискового сервиса

Поисковый сервис [191, 192]. Он служит для нахождения необходимых пользователю документов и (или) отдельных слов, фраз, выражений, содержащихся в них, и базируется на использовании поискового сервера, реля-

ционной БД реквизитов документов и клиентских *Java*-апплетов. Вспомогательные звенья в цепочке сервер—клиент — *Web*-сервер и *Web*-браузер.

Логику функционирования поискового сервиса можно условно разделить на четыре фазы (рис. 6.14).

Фаза 1 ( )

На компьютер пользователя системы (Клиент) при обращении *браузера* на корневую *html*-страницу *Web-сервера* передается *поисковый Java-апплет*. Запуск апплета осуществляется с помощью активизации соответствующей ссылки.

Фаза 2 ( )

Апплет, выполняя запросы клиента на поиск и получая результаты, непосредственно взаимодействует с *поисковым сервером* системы с помощью механизма *Berkley Sockets* протокола *TCP/IP* и специально разработанного протокола высокого уровня.

Фаза 3 ( )

Поисковый сервер, используя *базу данных реквизитов*, осуществляет контекстный поиск в среде *html*-документов и формирует результаты для поискового апплета. Результаты представляют собой *URL (Uniform Resource Locators)* документов, удовлетворяющих критериям поиска.

Фаза 4 ( )

Пользователь с помощью диалогового окна апплета инициирует загрузку найденных *html*-страниц в браузер.

Такова общая схема функционирования поискового сервиса. Перейдем к детальному рассмотрению компонентов системы поиска.

База данных реквизитов документов (БДРД) предназначена для хранения ссылок на все гипертекстовые документы ИС вместе с их классификационными характеристиками (реквизитами). Основной причиной включения такой базы данных в систему поиска явилась необходимость уменьшения количества документов, просматриваемых в процессе контекстного поиска.

В соответствии со спецификой предметной области БДРД включает следующий перечень атрибутов:

- идентификационный номер, который представляет собой фактически номер последовательной нумерации документов и используется в качестве первичного ключа для связи со справочниками и вспомогательными таблицами;
- наименование документа;
- регистрационный номер;
- дата принятия документа;
- год опубликования;
- дата вступления в силу данного документа;
- язык, на котором опубликован документ;
- тип документа (закон, постановление, кодекс, резолюция, лоция и т.п.);
- источник, указывающий, где впервые опубликован документ;
- населенный пункт, где впервые был опубликован или принят документ;
- организация (орган, ведомство, департамент), издавшая, опубликовавшая или принявшая к исполнению документ;
- страна, ратифицировавшая, принявшая, подписавшая или опубликовавшая данный документ;
- область классификации, к которой может быть отнесен документ;
- универсальный локатор ресурса (*URL*), по которому документ можно найти в банке *html*-документов.

Атрибуты при необходимости могут быть изменены или дополнены.

С учетом принятой классификации БДРД содержит следующие таблицы:

Documents — основная таблица записей реквизитов документов;

Urls — таблица электронных адресов месторасположения документов;

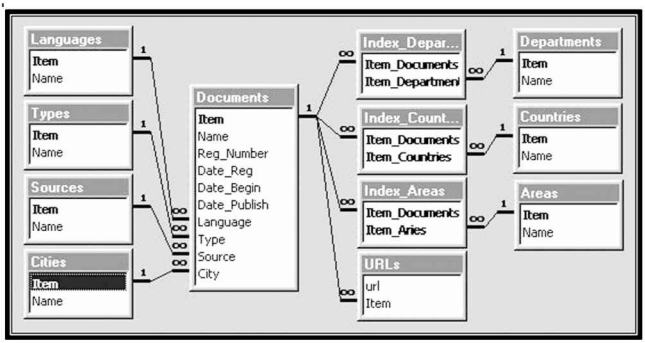


Рис. 6.15. Схема взаимосвязей атрибутов таблиц БДРД

Areas — справочная таблица областей классификации документов;

Countries — справочная таблица стран;

Departments — справочная таблица ведомств/департаментов;

Index_Areas — связующая таблица областей классификации и документов;

Index_Countries — связующая таблица стран и документов;

Index_Departments — связующая таблица ведомств и документов;

Languages — справочная таблица языков;

Types — справочная таблица типов документов;

Cities — справочная таблица населенных пунктов, где впервые был опубликован или принят документ;

Sources — справочная таблица источников, в котором впервые был опубликован документ.

Все перечисленные таблицы представлены в нормализованной форме, что обеспечивает поиск и произвольную выборку информации по любому возможному критерию поиска. Схема взаимосвязей атрибутов БДРД представлена на рис. 6.15.

Серверная часть поискового сервиса — *поисковый сервер* — предназначена для обработки запросов на кон-

текстный поиск и поиск по реквизитам документов, поступающих от удаленных пользователей системы. Поисковый сервер является приложением, которое выполняется на аппаратном сервере системы и имеет доступ к реляционной БДРД и банку гипертекстовых документов ИС.

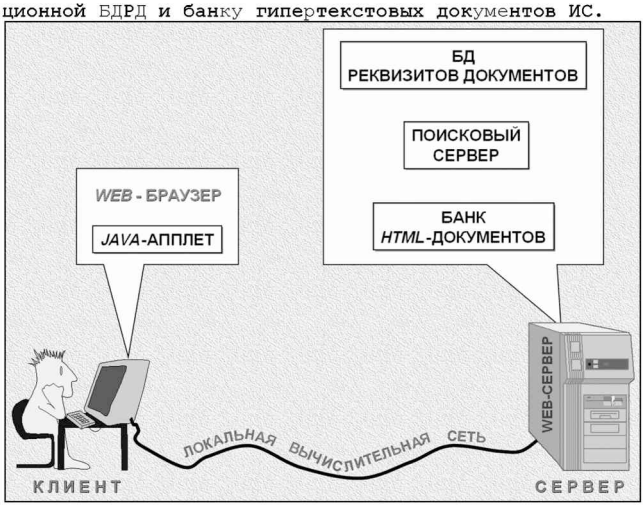


Рис. 6.16. Схема поиска в ГосНГИС

Клиентами по отношению к поисковому серверу выступают *Java*-апплеты (рис. 6.16) [74, 75], загружаемые пользователем с гипертекстовых страниц *Web*-сервера. *Java*-апплет, по существу, выполняет функции интерфейса между удаленными пользователями и поисковым сервером: пользователь в специальном диалоговом окне *Java*-апплета (рис. 6.17) формирует *SQL*-запрос для поиска по БДРД и запрос для контекстного поиска.

Работа клиентских апплетов осуществляется в соответствии со специальным протоколом взаимодействия с поисковым сервером (ПВПС) [191], реализованным на базе сетевого протокола *TCP/IP* по стандарту *Windows Sockets* 1.1 [193]. Контекстный поиск выполняется на основе эф-

фективных алгоритмов библиотеки регулярных выражений *(regular expression library)*. При контекстном поиске используется информационное пространство *Web*-сервера системы — множество документов в формате *HTML*. Поиск документов по известным реквизитам осуществляется в одноименной БД, зарегистрированной в качестве источника данных на сервере с помощью соответствующего драйвера. Связь с базой

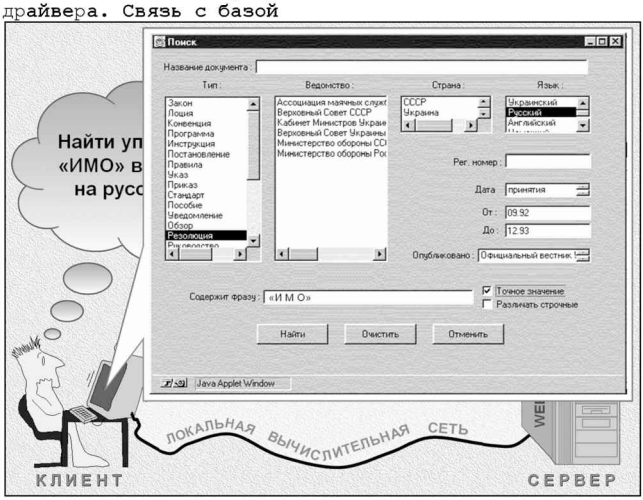


Рис. 6.17. Диалоговое окно для формирования поискового запроса

данных осуществляется в соответствии со стандартом *ODBC* (*Open Data Base Connectivity*) [78]. *ODBC* — открытый интерфейс доступа к базам данных — представляет собой набор драйверов и библиотеку функций, которые позволяют прикладной программе осуществлять унифицированный доступ к различным СУБД, используя структурированный язык запросов *SQL*.

Поисковый сервер является многопоточным (*multithreaded*) 32-разрядным *Windows*-приложением. Использование многопоточности и 32-разрядного программно-

го интерфейса в программе повышает ее эффективность и надежность, исключает отказы клиентам на соединение и соответствует общим традициям написания серверных приложений для *Internet*. Хорошо зарекомендовавшие себя открытые стандарты *ODBC* и *Windows Sockets* 1.1 обеспечивают аппаратную и программную независимость поискового сервера, простоту переноса этого приложения на другие операционные платформы, а алгоритмы библиотеки регулярных выражений — возможность гибкого и быстрого контекстного поиска информации среди больших объемов данных. Для настройки поискового сервиса системы используется специальная программа — панель управления поисковым сервером. Основными параметрами настройки являются: *URL Web*-сервера ИС; номер порта *TCP/IP*, на котором осуществляется прослушивание запросов от клиентов сервера; путь к корневому каталогу гипертекстовых документов *Web*-сервера; имя зарегистрированного в операционной системе источника данных *ODBC*, связанного с базой данных реквизитов *html*-документов. С помощью панели управления осуществляется также запуск и останов поискового сервиса (рис. 6.18).

Java-апплет, используя механизм *Socket*, устанавливает связь с поисковым сервером. Обмен информацией с поисковым сервером осуществляется с помощью протокола ПВПС, состоящего из набора команд с параметрами. При рассмотрении ПВПС необходимо выделить две разновидности взаимодействия клиент—сервер: инициализацию соединения и отработку запросов на поиск.

После инициализации апплет устанавливает связь с поисковым сервером, обновляет собственные справочники (типы документов; ведомства, издавшие документ; страны, язык и др.) и сохраняет их содержимое до следующего сеанса. Если в процессе инициализации поисковый сервер не отвечает на запрос, то считается, что он не найден и дальнейшая работа с ним невозможна.

Инициализация соединения между клиентом и сервером осуществляется по команде с параметрами *CONNECT*. Запрос клиента *CONNECT* устанавливает связь с поисковым сервером и передает размеры содержимого справоч-

ников БДРД. В параметрах запроса *CONNECT* передаются размеры справочников. Если

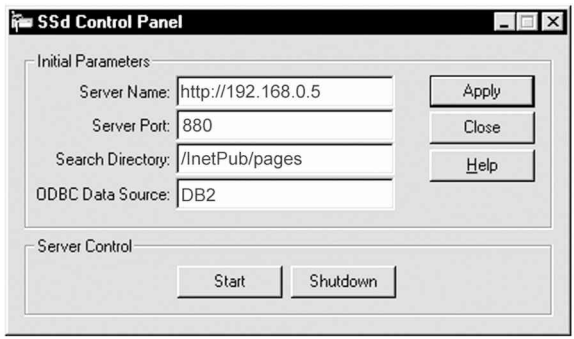


Рис. 6.18. Панель управления поисковым сервером

переданный размер меньше размера аналогичного справочника на сервере, то сервер включает в свой ответ данные для обновления содержимого этого справочника на стороне клиента. Система инициализации разработана для уменьшения последующего трафика между клиентскими местами и сервером.

Запрос клиента:

Команда	**Параметры**
CONNECT	*TypeSize DepartmentSize CountrySize LanguageSize SourceSize*

Ответ сервера:

Данные
Type1 … TypeN Department1 … DepartmentN Country1 … CountryN Language1 … LanguageN Source1 … SourceN

Отработка запросов клиента на поиск осуществляется по команде *SEARCH*. Запрос *SEARCH* имеет такие параметры: область поиска, представленная в виде части *SQL*-запроса к базе реквизитов, описывающей поле *WHERE*;

шаблон для контекстного поиска; флаг "точный поиск/поиск подстроки"; флаг "игнори-рование/учет регистра шаблона".

В ответ на этот запрос сервер передает *URL* документов, соответствующих критериям поиска. Ответ состоит из трех компонентов: команда *START* (начало передачи сервером результирующего пакета); команда с параметрами *FIND URL Title* (передача результата поиска в виде *URL* документа и его заголовка); команда *STOP* (конец передачи сервером результатов поиска). Пара *"START FIND"* передается столько раз, сколько документов было найдено. Имеется также дополнительная команда *ERROR*, сигнализирующая о возникновении ошибочной ситуации на сервере.

Запрос клиента:

Команда	**Параметры**
SEARCH	*Request Pattern ExactSearch IgnoreCase*

Ответ сервера:

Команда	**Параметры**
START	*--без параметров--*
FIND	*URL Title*
…	…
STOP	*--без параметров--*
ERROR	*--без параметров--*

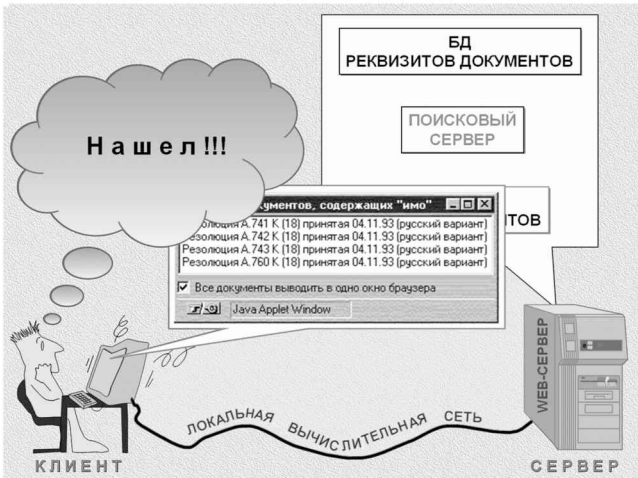


Рис. 6.19. Иллюстрация завершения поискового процесса

В реализации протокола все команды и параметры представляют собой строковые переменные, разделяемые символом "переход на новую строку" ("\n" в нотации языка Си).

В результате успешного поиска на экран будет выведено дополнительное окно (рис. 6.19), содержащее заголовки соответствующих документов. Выбор одного из них приведет к появлению еще одного окна браузера, содержащего указанный документ.

6.1.1. Некоторые особенности представления документов в ГосНГИС

Фреймы. Рассмотрим материал (текст, графика и др.), поместившийся в физические размеры экрана компьютера, и на примере продемонстрируем простоту и удобство представле-

ния различного рода линейного[2] текста в виде *web*-страниц (*html*-документов). Характерный пример этого положения — оглавление книги. Окно *Web*-браузера разбивается на несколько областей (кадров, или фреймов, — *frames*), в которых одновременно отображаются разные документы. Оглавление загружается в кадр, который занимает узкий столбец в левой части окна браузера. Оно содержит перечень ссылок на все главы книги. Каждая такая ссылка обеспечивает связь с кадром, который занимает остальную часть окна. Пользователь может просматривать главы, сохраняя в левой части экрана оглавление. Ему не нужно постоянно возвращаться к предыдущей странице.

Поскольку оглавление, помещенное во фрейме, всегда будет в поле зрения, пользователь сможет выбрать другой его пункт и сразу же получить требуемую информацию (рис. 6.20, 6.21).

Приведем еще один вариант использования фреймов в представлении текстовых и графических материалов. В Приложении 6 к книге "Правила наблюдения на кораблях и судах ВМФ за гидрометеорологической обстановкой" (Издание Гидрографического управления МО СССР, 1967. — 271 с.) приведена таблица "Шкалы силы ветра". В Приложении 7 помещены фотографии поверхности моря при различной силе ветра. Гипертекстовая технология позволяет объединить этот материал и учесть особенности его компьютерного представления (рис. 6.22 и 6.23). Пользователю неудобно просматривать таблицы, размеры которых превышают физические размеры экрана компьютера, так как заголовки могут не попасть в поле зрения. Поэтому таблицы, занимающие в документе-оригинале несколько страниц, представляют в виде фреймов.

С помощью фреймов делят страницу на части, каждая из которых обновляется отдельно. Фреймы позволяют так сконструировать *web*-страницу, что заголовок таблицы (верхний фрейм) будет закреплен в определенном месте экрана, а содержимое таблицы (нижний фрейм) с помощью

---

[2] Форма организации текстового материала, при которой его единицы представлены не в линейной последовательности, а как система явно указанных возможных переходов и связей между ними (гипертекст). Следуя этим связям, можно читать материал в любом порядке, образуя разные *линейные* тексты [30].

линейки прокрутки будет перемещаться относительно ее заголовка (см. рис. 6.22).

Пример отображения текстов на русском и английском языках в окне *Web*-браузера, разбитого на вертикальные окна (кадры), приведен на рис. 6.24. Пользователю предоставлена возможность просмотра этих документов в многооконном интерфейсе на основе фреймов. Это позволяет видеть на экране компьютера одновременно несколько документов, причем пользователь при просмотре может вручную изменять размеры столбцов кадров, делая видимым тот или иной документ.

Кадры, содержимое которых не умещается в границах отведенного для них пространства, при отображении браузером снабжаются линейками прокрутки.

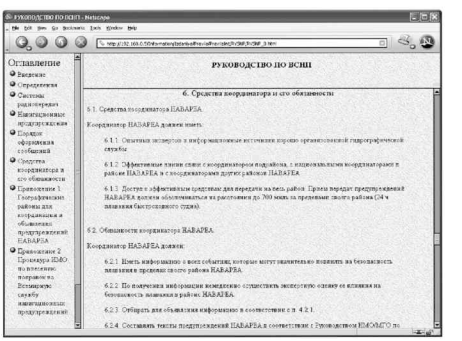


Рис. 6.20. Пример представления документа во фреймах

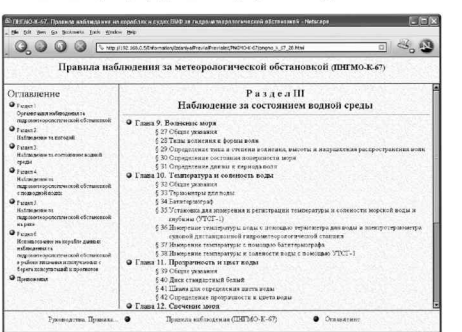


Рис. 6.21. Пример представления документа во фреймах

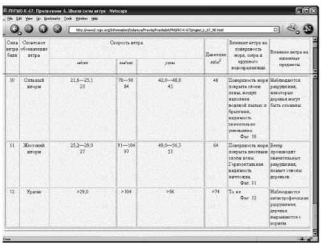


Рис. 6.22. **Пример представления таблицы во фреймах**

Рис. 6.23. **Размещение рисунка, представленного во фрейме**

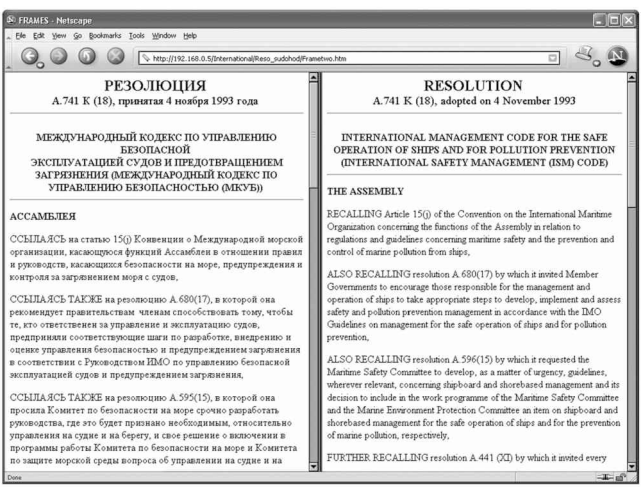


Рис. 6.24. Отображение текстов на русском и английском языках в окне *Web*-браузера, представленного вертикальными окнами (кадрами)

Расширения *HTML*. В ряде случаев громоздкие таблицы с рассчитанными по тем или иным формулам данными, ко-

торые уместны в печатном издании, совершенно неприемлемы для просмотра в окне браузера. Приведем фрагмент книги "Инструкция по гидрографическому тралению (ИТ-74)" (Издание ГУНиО МО СССР, 1974. — 102 с.):

"ПРИЛОЖЕНИЕ 8
к стр. 67

ПРИВЕДЕНИЕ УГЛОВ, ИЗМЕРЕННЫХ СЕКСТАНОМ, К ГОРИЗОНТУ

Углы, измеренные секстаном, приводят к горизонту введением поправок.

Поправки вычисляют по формуле

$$(x - x_н) = 1{,}05\left(\frac{a_1 + a_2}{2}\right)^2 \operatorname{tg}\frac{x_н}{2} - 1{,}05\left(\frac{a_1 - a_2}{2}\right)^2 \operatorname{ctg}\frac{x_н}{2} = I - II,$$

где (x-xн) — поправка в измеренный угол, в минутах;

xн — угол, измеренный секстаном;

• •

$$I = 1{,}05\left(\frac{a_1 + a_2}{2}\right)^2 \operatorname{tg}\frac{x_н}{2}, \quad II = 1{,}05\left(\frac{a_1 + a_2}{2}\right)^2 \operatorname{ctg}\frac{x_н}{2}.$$

Значения первого и второго членов поправки выбирают из таблицы величин I и II ***(табл. 7).***

Вычисление поправки выполняется в следующем порядке…"

Далее на трех страницах "Инструкции…" следует таблица 7, выдержки из которой приведены на рис. 6.25.

При ссылке на таблицу 7 вместо однообразного мелькания строк и столбцов цифр пользователю предоставляется специальная форма для ввода и получения затребованных данных (рис. 6.26). Форма создана с помощью средств языка *HTML* [65]; расчет величин I и II по формулам, приведенным в верхнем окне формы, производится в соответствии с созданным сценарием на языке *JavaScript* [65—67]. Текст программы сценария внедрен непосредственно в *html*-документ, так называемый *JavaScript* для клиента.

Применение форм совместно со встроенными *JavaScript*-сце-нариями существенно уменьшает объем *html*-документа, в результате чего достигается увеличение эффективности *html*-кода, снижается трафик сети и уменьшается время загрузки страницы.

Таблица 7

Таблица величин I и II

$$I = 1{,}05\left(\frac{a_1 + a_2}{2}\right)^2 \operatorname{tg}\frac{x_н}{2}; \quad II = 1{,}05\left(\frac{a_1 - a_2}{2}\right)^2 \operatorname{ctg}\frac{x_н}{2}$$

$\frac{a_1+a_2}{2}$ / $x_н$	1°	2°	3°	4°	5°	6°	7°	8°	9°	10°	$\frac{a_1-a_2}{2}$ / $x_н$
20°	0′	1′	2′	3′	5′	7′	9′	12′	15′	18′	160°
21	0	1	2	3	5	7	10	12	16	19	159
22	0	1	2	3	5	7	10	13	17	20	158
23	0	1	2	3	5	8	10	14	17	21	157
24	0	1	2	4	6	8	11	14	18	22	156
25	0	1	2	4	6	8	11	15	19	23	155
156	5	20	44	1 19							24
157	5	21	46	1 23							23
158	5	22	49	1 26							22
159	6	24	51	1 31							21
160	6	25	54	1 35							20

Рис. 6.25. Фрагменты таблицы 7 из "Инструкции по гидрографическому тралению (ИТ-74)"

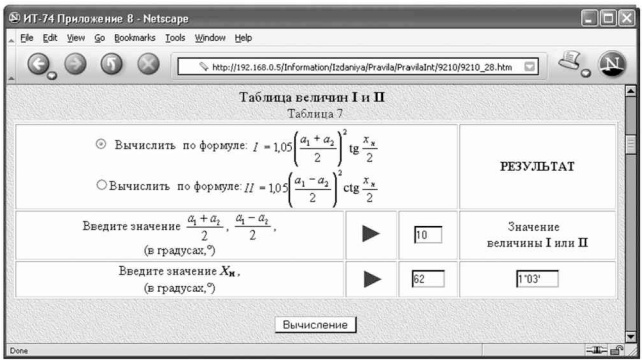
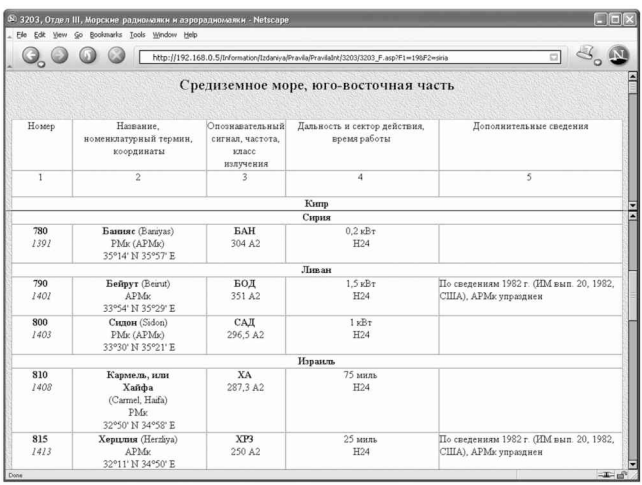


Рис. 6.26. Окно вычислений поправки



Номер	Название, номенклатурный термин, координаты	Опознавательный сигнал, частота, класс излучения	Дальность и сектор действия, время работы	Дополнительные сведения
1	2	3	4	5
		Кипр		
		Сирия		
780 *1391*	**Баниас** (Baniyas) РМк (АРМк) 35°14' N 35°57' E	**БАН** 304 А2	0,2 кВт H24	
		Ливан		
790 *1401*	**Бейрут** (Beirut) АРМк 33°54' N 35°29' E	**БОД** 351 А2	1,5 кВт H24	По сведениям 1982 г. (ИМ вып. 20, 1982, США), АРМк упразднен
800 *1403*	**Сидон** (Sidon) РМк (АРМк) 33°30' N 35°21' E	**САД** 296,5 А2	1 кВт H24	
		Израиль		
810 *1408*	**Кармель, или Хайфа** (Carmel, Haifa) РМк 32°50' N 34°58' E	**ХА** 287,3 А2	75 миль H24	
815 *1413*	**Херцлия** (Herzliya) АРМк 32°11' N 34°50' E	**ХРЗ** 250 А2	25 миль H24	По сведениям 1982 г. (ИМ вып. 20, 1982, США), АРМк упразднен

Рис. 6.27. Фрагмент таблицы описания радиомаяков

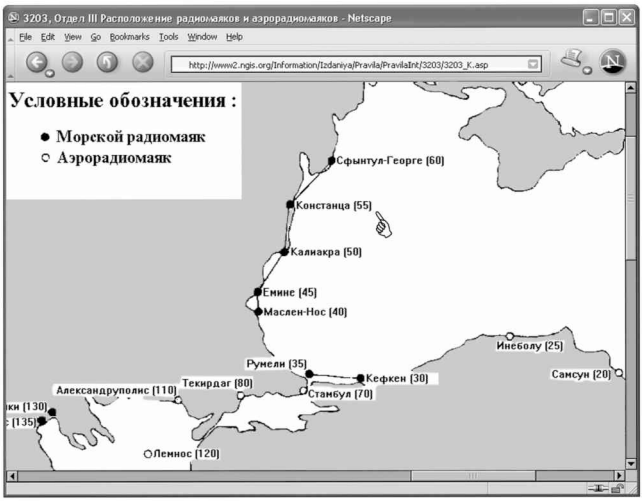
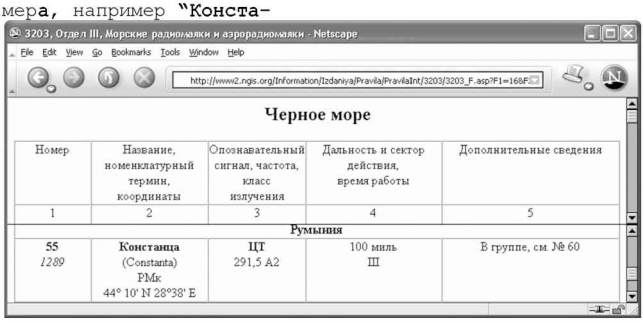


Рис. 6.28. Фрагмент схемы расположения радиомаяков и аэрорадиомаяков

Графическая ссылка (карты ссылок). Карты ссылок *(image-maps)* позволяют создавать гиперссылки, связанные с различными "горячими" областями изображения. Их иначе называют мультиадресными ссылками. Это значит, что графический элемент содержит не одну, а ряд ссылок, при активизации которых открывается документ, связанный с этой зоной. В качестве изображения используют любое графическое представление информации — "изображение-карты". Для иллюстрации этого положения обратимся к руководству "Радиотехнические средства навигационного оборудования Черного и Средиземного морей" (Издание ГУНиО МО СССР, 1983. — 87 с.). В этой книге приведено описание радиомаяков в виде таблицы (рис. 6.27), дана схема их расположения, а в отдельных таблицах — алфавитные указатели названий и опознавательных сигналов радиотехнических средств навигационного оборудования (СНО).

На схеме расположения радиомаяков (рис. 6.28) с помощью специального графического редактора обозначены активные зоны карты ссылок. Ими служат названия и номера, например "Конста-



Черное море

Номер	Название, номенклатурный термин, координаты	Опознавательный сигнал, частота, класс излучения	Дальность и сектор действия, время работы	Дополнительные сведения
1	2	3	4	5
		Румыния		
55 *1289*	**Констанца** (Constanta) РМк 44° 10' N 28°38' E	**ЦТ** 291,5 А2	100 миль Ш	В группе, см. № 60

Рис. 6.29. Фрагмент таблицы с описанием радиомаяка "Констанца"

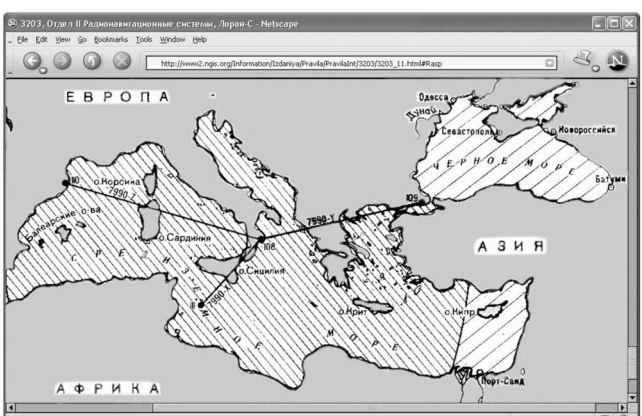


Рис. 6.30. Схема расположения станций радионавигационной системы Средиземного моря

нца" **[55]**, которые присвоены каждому радиомаяку в соответствии с принятой классификацией. Пользователь, перемещая курсор по схеме, видит, как он меняет свой вид и превращается в маленький указующий перст, попав на гиперссылку **[55]**. При ее активизации осуществляется доступ к той части таблицы, где содержится описание радиомаяка "Констанца" (рис. 6.29).

Средиземноморская цепь

Пары станций

Название и номер станции		Частотный параметр	Назначение и обозначение станции	Длина базы,м	Азимут базы с ведущей	Базовая задержка, *единицы РНП*	Кодовая задержка, *единицы РНП*	Эквивалент единицы РНП, м	Координаты	
									широта N	долгота E
Лампедуза (Lampedusa)	111	7990	ВМ—X	526011,4	226°19'26,02"	1755,970	11 000,000 *10 999,848*	299,5300	35°31'20,88"	12°31'29,96"
Катандзаро (Catanzaro)	108		ВЩ						38 52 20,61	16 43 05,96
Карга (Karga)	109	7990	ВМ—Y	980457,7	72 42 37,21	3273,300	29 000,000 *28 999,979*	299,5300	40 58 20,93	27 52 01,52
Катандзаро (Catanzaro)	108		ВЩ						38 52 20,61	16 43 05,96
Эль-Эстартит (El Estartit)	110	7990	ВМ—Z	1198012,1	291 29 06,53	3999,690	47 000,000 *47 000,50*	299,5300	42 03 36,49	3 12 15,90
Катандзаро (Catanzaro)	108		ВЩ						38 52 20,61	16 43 05,96

Рис. 6.31. Фрагмент таблицы с описанием станций радионавигационной системы Средиземного моря

Однако представлять, например, схему расположения станций радионавигационной системы Средиземного моря (рис. 6.30) в виде трех (по числу пар станций) активных зон нецелесообразно, так как информация о них компактна и содержится в таблице (рис. 6.31), занимающей неполный экран. Иное дело — мно-гостраничная таблица с описанием радиомаяков.

Если пользователь начнет знакомство с радиотехническими средствами навигационного оборудования (РТСНО) Черного и Средиземного морей непосредственно с морских радиомаяков, то, активизировав гиперссылку "Отдел III" оглавления книги, увидит в левой части экрана содержание, а в правой — описание этого раздела (рис. 6.32). Дальнейший выбор страны, предположим Италии, приведет к смене информации в правом кадре экрана: появится перечень морей, омывающих Италию, и ее островов с расположенными на них и побережье РТСНО (рис. 6.33). Используя гиперссылку "Тирренское море", пользователь

вызовет на экран уже известную ему таблицу с описанием радиомаяков на побережье Италии у берегов Тирренского моря (рис. 6.34).

Таким образом, есть возможность перейти непосредственно из таблицы (см. рис. 6.34) в рисунок с изображением схемы расположения радиомаяков и аэрорадиомаяков. Так, выбрав в этой таблице гиперссылку **[450]**, увидим уже знакомую карту-схему с маяком "Карена" в центре экрана (рис. 6.35).

Центровка карты-схемы относительно маяка "Карена" достигается благодаря использованию программы по определению

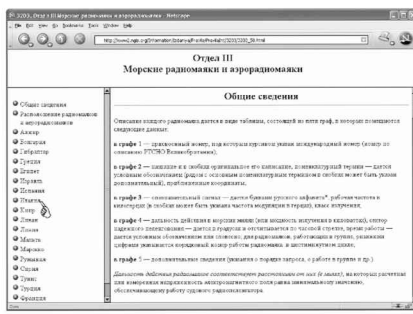


Рис. 6.32. Фрагмент книги "Радиотехнические средства навигационного оборудования Черного и Средиземного морей"

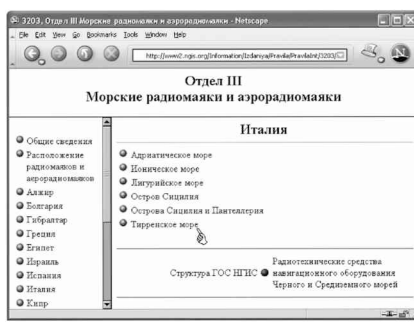
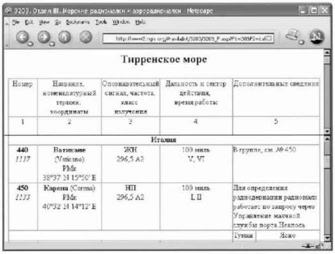


Рис. 6.33. **Фрагмент книги "РТСНО Черного и Средиземного морей"**



Тирренское море

Номер	Название, номенклатурный термин, координаты	Опознавательный сигнал, частота, класс излучения	Дальность и сектор действия, время работы	Дополнительные сведения
1	2	3	4	5
		Италия		
440 *1137*	**Ватикане** (Vaticano) PMk 38°37' N 15°50' E	ЖН 296,5 A2	100 миль V, VI	В группе, см. № 450
450 *1133*	**Карена** (Carena) PMk 40°32' N 14°12' E	НП 296,5 A2	100 миль I, II	Для определения радиодевиации радиомаяк работает по запросу через Управление маячной службы порта Неаполь
				Туман / Ясно

Рис. 6.34. Фрагмент книги "РТСНО Черного и Средиземного морей"

разрешения экрана и процедуры позиционирования карты. Прием параметров для позиционирования карты-схемы происходит с помощью функций *ASP*-файлов [182].

Приведем фрагмент программы (листинг 6.1), определяющей разрешение, установленное на мониторе пользователя.

Листинг 6.1

```
<SCRIPT Language=JavaScript>
var height=0;
var width=0;
if (self.screen) { // for NN4 and IE4
      width = screen.width
      height = screen.height
}
else if (self.java) { // for NN3 with enabled Java
     var jkit = java.awt.Toolkit.getDefaultToolkit();
     var scrsize = jkit.getScreenSize();
     width = scrsize.width;
     height = scrsize.height;
}
</SCRIPT>
```

Процедура позиционирования карты my_scroll() выполняется после загрузки карты-схемы:

```
<SCRIPT Language=JavaScript>
function my_scroll()
     {
      if (width > 0 && height > 0) {
        window.scroll(<%=Hor%>-width/2,<%=Ver%>-height/2);
```

```
        } else {
            window.scroll(<%=Hor%>-340,<%=Ver%>-240);          }
    }
</SCRIPT>
```

Координаты выбранного по гиперссылке маяка передаются из документа (см. рис. 6.34) на сервер путем вызова *ASP*-файла **3203_K.asp?Hor=1263&Ver=350** с указанными параметрами. Эти параметры подставляются в функцию **my_scroll()**. Переменным ***Hor*** и ***Ver*** присваиваются значения 1263 и 350 соответственно. В результате пользователь видит изображение карты-схемы, позиционированное относительно маяка "Карена" **[450]**.

В случае обращения к алфавитному указателю названий радиотехнических СНО пользователь выбирает из предложенной

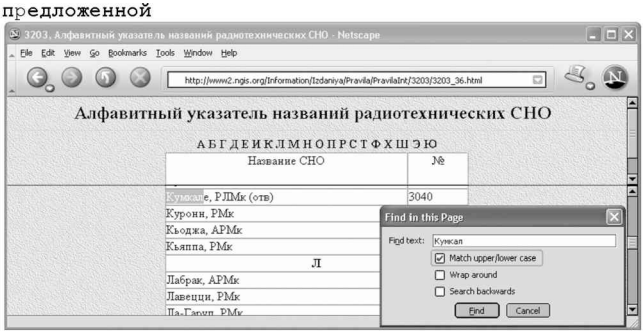


Рис. 6.36. Поиск радиотехнических средств навигационного оборудования

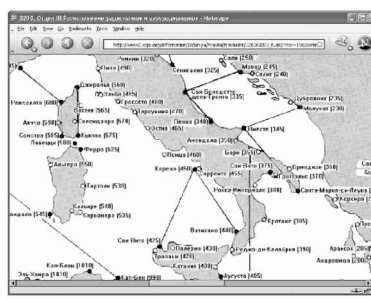
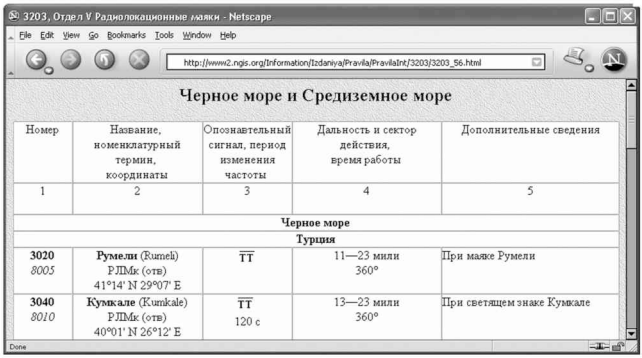


Рис. 6.35. Карта-схема маяков Тирренского моря



Черное море и Средиземное море

Номер	Название, номенклатурный термин, координаты	Опознавательный сигнал, период изменения частоты	Дальность и сектор действия, время работы	Дополнительные сведения
1	2	3	4	5
Черное море				
Турция				
3020 *8005*	**Румели** (Rumeli) РЛМк (отв) 41°14' N 29°07' E	TT	11—23 мили 360°	При маяке Румели
3040 *8010*	**Кумкале** (Kumkale) РЛМк (отв) 40°01' N 26°12' E	TT 120 с	13—23 мили 360°	При светящем знаке Кумкале

Рис. 6.37. Результат поиска

ему строки алфавита (рис. 6.36) нужную букву. Если развернувшийся перед ним список достаточно велик, организует поиск требуемого элемента стандартными средствами браузера, вызвав окно "*Find in This Page…*" из меню *Edit*. Далее, по присвоенному СНО номеру пользователь может перейти в таблицу с описанием радиолокационных маяков (рис. 6.37).

Отметим, что приведенные здесь примеры, безусловно, не исчерпывают всех особенностей и, главное, возможностей формирования гипертекстовой базы ГосНГИС. Существующая ныне и постоянно совершенствуемая программная среда позволяет создавать содержательно емкие, высокофункциональные и вместе с тем быстро загружаемые *web*-страницы с ясно воспринимаемой информацией. Необходимо только соизмерять желаемую форму представления информации с необходимостью этих требований, а также с возможностью и средствами, затрачиваемыми на ее достижение. Другими словами, надо найти разумную середину в представлении материала.

Так, применение в ГосНГИС каскадных таблиц стилей (*CSS*) в значительной мере упростило бы подготовку большинства однородных текстовых документов. *CSS* предназначены для отображения в браузере различных *html*-страниц или всего *Web*-сайта в едином стилевом

исполнении. С помощью *CSS* можно задать определенный стиль отображения практически любому элементу *html*-страницы, а также некоторым элементам браузера, например, полосе прокрутки. Применение *CSS* позволяет избежать ряда отрицательных моментов.

1. Создавая новый *html*-документ, нет необходимости "держать в голове" эталонную *html*-страницу — файл *CSS* создается один раз на основе одного *html*-документа (эталона) и при создании нового *html*-документа прописывается в нем в виде ссылки.

2. От нового разработчика *html*-документов (дизайнера) не потребуется углубленного изучения *html*-кода с соответствующей затратой времени (чтобы разобраться к какому элементу применено какое форматирование) — достаточно определить какие стили *CSS* назначены для каждого элемента.

3. При изменении дизайна *html*-документа или какой-либо его части не требуется вносить эти изменения в каждую *html*-страницу — все изменения производятся в файле *CSS* и динамично отображаются в каждом *html*-документе, имеющем ссылку на файл стилей.

Ниже приведены два фрагмента *html*-кода, отформатированные с помощью *CSS* [194, 195] и обычных *html*-тегов соответственно [67].

```
<link href="style.css" rel="stylesheet" media="screen">

<p class="text1">
Эти материалы содержатся в отдельных изданиях Дунайской Комиссии.
</p>

<div align="center">
<p> <font size="1" color="black" face="Arial,Helvetica,Geneva, Swis
s,SunSans-Regular">
  <b> Эти материалы содержатся в отдельных изданиях Дунайской Комиссии.
</b></font>
 </div>
```

К сожалению, несмотря на все преимущества каскадных таблиц стилей перед обычным форматированием, не все браузеры поддерживают *CSS* в полном объеме их свойств и возможностей.

Еще одним технологическим решением, которым можно воспользоваться при построении *Web*-сайтов, является

технология *Flash* [196–199]. *Flash* – мощное, но простое в использовании средство создания анимированных проектов (меню, клипы, заставки) на основе векторной графики со встроенной поддержкой интерактивности. Фактически *Flash* объединяет в себе три программы, предоставляющие возможность создания графики, анимации и *Flash*-фильмов. *Flash* является рабочим инструментом художников и дизайнеров, дополняющим создаваемые ими *Web*-проекты анимацией и звуком. Для работы с *Flash* не обязательно иметь какой-либо опыт в программировании – пакет позволяет создавать *Web*-узлы с элементами интерактивности без необходимости написания исходных кодов *JavaScript*, *Java* или *HTML*, кроме того, *Flash* обладает собственным языком программирования *ActionScript*, который и привносит в проекты элемент интерактивности.

В *Flash*-клипах используются звуковые файлы в формате *WAV* и *MP3*. Звуки можно привязать к моментам отображения отдельных сцен либо к действиям с активными элементами дизайна (нажатие кнопок и пунктов меню), а также к действиям с активными зонами произвольной формы (движение курсора над каким-либо участком и выход из него, нажатие или освобождение кнопки мыши и т.п.). Приемлемым для *Internet* является также и незначительный размер получаемых в итоге анимационных роликов.

Для просмотра *Flash*-приложений в *Internet* достаточно иметь на компьютере установленный модуль, который свободно распространяется в сети (при использовании свежих версий браузеров необходимые средства имеются). Можно сохранить *Flash*-приложение и в формате, который воспроизводится без дополнительных модулей, – он встраивается непосредственно в ролик. Такой способ очень удобен для разработки мультимедийных презентаций.

Flash можно использовать в ГосНГИС для построения различных сцен навигации по морским картам (на момент создания

системы при построении карт были задействованы формат изображений *GIF*, технологии *Imagemap*, *JavaSript* и *CSS*). В этом случае *Flash* незаменим, так как позволяет избежать большого количества файлов со сложным и длинным *HTML*-кодом, контекстного меню и различных графических элементов. Однако при больших объемах работ по созданию сайта применение *Flash*, пожалуй, экономически невыгодно из-за высоких трудозатрат.

6.2. Сеть распределенных гидрометеорологических банков данных

Примером построения ИС "природного" профиля (системы, содержащей данные о состоянии среды) является работа, проведенная в рамках заданий Национальной программы информатизации Украины, в которых предполагалось создать систему баз данных и знаний о национальных богатствах и природных ресурсах страны. В частности, было предусмотрено создание сети распределенных гидрометеорологических баз данных, ориентированных на широкое использование упомянутой информации для изучения, охраны и воспроизведения природного потенциала Украины.

Государственная гидрометеорологическая служба Минэкоресурсов Украины хранит уникальную информацию, накапливаемую с 1800 г. С тех времен начались регулярные наблюдения на 80-ти гидрометеорологических станциях. Разветвленная на территории Украины сеть стационарных и передвижных пунктов наблюдений по гидрометеорологическим показателям осуществляет почти весь контроль за состоянием окружающей среды (рис. 6.38) [200]. Но, кроме сохранения материалов наблюдений, следует приложить немало усилий, чтобы осуществить автоматизированный доступ к архивным данным, использовать их для расчетов гидрометеорологических характеристик и распространить информацию.

Гидрометеорологическая служба представляет собой многоуровневую систему, которая для решения задач и выполнения функций, возлагаемых на нее, использует в своей повседневной деятельности большие объемы разнообразной гидрометеорологической информации. Эффектив-

ное использование этой информации с целью обеспечения потребителей данными о текущих и ожидаемых гидрометеорологических условиях невозможно без применения современных программно-технических средств ведения баз данных, обработки и анализа информации.

Рис. 6.38. Карта контроля за состоянием окружающей среды Украины

Особое место при использовании гидрометеорологической информации занимают потребности организаций и учреждений в интегрированных, специально вычисляемых гидрометеорологических показателях в пространственно-временном аспекте. Спрос на такую информацию постоянно возрастает. Между тем доступ к архивным данным наблюдений, сосредоточенным в организациях гидрометеорологической службы, осуществляется, как правило, не в автоматизированном режиме. Поэтому подготовка ответа на такие трудоемкие запросы требует много времени и усилий высококвалифицированных специалистов. Это снижает потенциал использования информации, полученной в результате долгосрочных наблюдений.

В организациях гидрометеорологической службы за последние годы осуществлен переход к автоматизации всех звеньев деятельности, связанных с накоплением, обработкой и распространением гидрометеорологической информации. Однако технологические возможности применяемых ИС крайне ограничены, а существующие специализированные БД не охватывают весь спектр гидрометеорологической информации. Эффективное использование информационных ресурсов гидрометеорологической службы предусматривает объединение баз данных наблюдений в такой ИС, которая бы интегрировала информацию, имела унифицированную систему доступа к ее ресурсам и осуществляла быстрый поиск необходимых данных [60].

Решение этой задачи имеет два этапа. Первый — создание и внедрение в эксплуатацию сети[3] распределенных гидрометеорологических БД, т.е. создание на основе современных технологий экономной распределенной ИС, которая бы интегрировала информацию, связанную с автономными, локально сопровождаемыми БД наблюдений по видам гидрометеорологической информации[4]; обеспечивала непосредственный доступ к национальным и международным банкам природного профиля, работающим под эгидой Всемирной метеорологической организации (ВМО);

---

[3] Под сетью распределенных гидрометеорологических баз данных понимаем объединение в единое проблемно ориентированное информационное пространство множества специализированных тематических баз гидрометеорологических данных (СТБГМД) с возможностью доступа к их ресурсам. Эти БД, в общем случае, географически или территориально распределены, локально формируются и сопровождаются. В них сосредоточены режимно-справочные архивные материалы наблюдений гидрометеорологической службы, информация в которых во временном отношении делится на срочную (с разным количеством сроков наблюдений за сутки) и обобщенную за суточные и месячные интервалы. СТБГМД в свою очередь группируются в БнД по видам гидрометеорологической информации. Так образуется сеть распределенных гидрометеорологических БнД — СРГМБнД.

[4] Существует агрометеорологическая, аэрологическая, актинометрическая, гидрологическая (озерная и речная), метеорологическая (приземная), морская гидрометеорологическая (прибрежная), морская судовая, озонометрическая, радиометеорологическая, снеголавинная и другая гидрометеорологическая информация.

была способна довести до пользователей в удобном для восприятия виде упомянутую информацию. При этом необходимо разработать типовые программно-технические решения для создания сети распределенных гидрометеорологических банков данных (СРГМБнД) и построить ее компоненты на единой методологической основе с целью проведения консолидированной технической политики в этой области.

Следующий этап — наполнение БД архивными данными в первую очередь с непрерывным рядом наблюдений (чем длиннее ряд, тем ценнее информация). Особое внимание следует обратить на наполнение БД архивными данными, поскольку это требует определенных кропотливых и длительных усилий, вызванных тем, что режимные данные сохраняются на бумажных, магнитных и других носителях и приведены в разных видах и форматах.

Исходя из изложенного Центр информационных технологий и систем предложил использовать разработанную и апробированную им оригинальную прозрачную технологию построения и поддержки функционирования ИС, которые объединяют в проблемно ориентированную информационную среду территориально или географически распределенные, локально сопровождаемые банки (базы) данных и знаний с разными моделями представления. В основе технологии – концептуальная модель (см. гл. 2) распределенных ИС широкого применения на типовых серверных платформах.

С учетом требований к сети гидрометеорологических БнД создана действующая модель распределенной ИС "Банк гидрометеорологических данных": промоделирован БнД по метеорологии и создан БнД по агрометеорологии как составные части сети распределенных гидрометеорологических банков данных. Первая очередь распределенной ИС – БнД по агрометеорологии — внедрена в опытную эксплуатацию в УкрГМЦ, а планом работ на последующие годы предусмотрено введение в действие второй очереди ИС в ЦГО (рис. 6.39).

В основе построения ИС лежит клиент-серверная архитектура, реализованная с использованием *Internet/intranet*-техноло-гий. Серверная часть пред-

назначена для хранения, визуализации и поиска структурированной (таблицы) и неструктурированной (текст) информации. Она состоит из *Web*-, *SQL*-серверов и унифицирована для всех банков сети. *Web*-сервер предоставляет доступ клиентам к информационной базе гипертекстовых документов. Вместе с дополнительными программными компонентами он обеспечивает визуализацию структурированной информации, которая хранится в реляционных БД. Таким образом, гипертекстовые страницы являются средством представления информации разных форматов. Тенденция к объединению разнородной информации, которая базируется на наиболее естественной для человека возможности ее получения (видеть, читать), не нова и широко применяется в ИС, использующих *Web*-технологии.

Web-сервер, кроме того, вместе с компонентами ОС обеспечивает аутентификацию клиентов и реализацию разнообразных механизмов защиты информации.

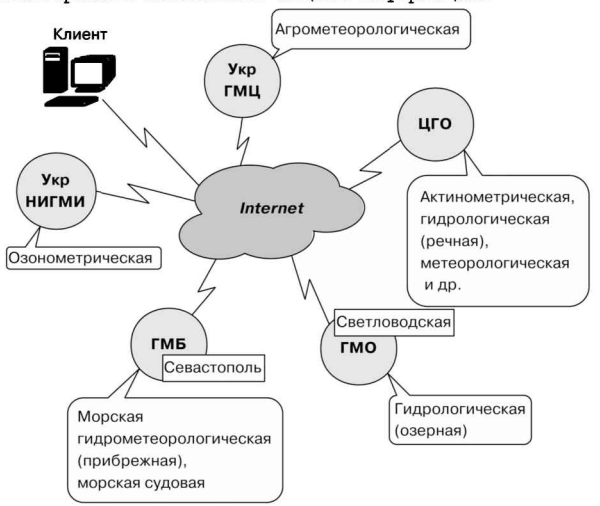


Рис. 6.39. Структура сети распределенных гидрометеорологических БнД:

УкрГМЦ — Украинский гидрометереорологический центр, Киев; УкрНИГМИ — Украинский научно-исследовательский гидрометеорологический институт, Киев; ГМБ — Гидрометеорлогическое бюро Севастополь; ГМО — Светловодская гидрометеорологическая обсерватия; ЦГО — Центральная геофизическая обсерватория, Киев

Клиентская часть представляет собой *Web*-браузер и реализует унифицированный пользовательский интерфейс для доступа к данным разных форматов.

Коммуникационная инфраструктура СРГМБнД. Данные, которые собираются Государственной системой гидрометеорологических наблюдений, после тщательной проверки, всестороннего контроля качества, анализа и редактирования, обобщений первичных измерений и наблюдений сосредоточиваются в архивах. Перечень основных организаций, где сохраняются режимно-справочные гидрометеорологические данные, приведен на рис. 6.39.

Серверы, размещенные на одной территориальной площадке, представляют собой узел. Коммуникационной инфраструктурой, которая связывает узлы, служит *Internet* или любая другая сеть (каналы передачи данных)[5], поддерживающая протоколы *TCP/IP*. Клиенты получают доступ к узлам через *Internet* (см. рис. 6.39.) или, при наличии аппаратуры удаленного доступа на узле, по выделенным или коммутируемым телефонным линиям. Такое решение дает возможность осуществить доступ к информации практически для неограниченного количества пользователей независимо от их географического положения и обеспечивает интеграцию СРГМБнД в мировое информационное пространство.

Защита информации от несанкционированного доступа. Защита осуществляется с учетом того, что в сети БнД сохраняется информация разных степеней значимости, доступ к которой должен регламентироваться нормативными актами. Защита обеспечивается комплексом программных средств, назначающим пользователям права доступа к информационным ресурсам СРГМБнД, и

[5] На узлах в этом случае должны быть предусмотрены технические средства для организации доступа клиентов, например по телефонным линиям связи.

делится на защиту от несанкционированного доступа к данным и защиту информационного обмена.

Под защитой от несанкционированного доступа подразумевается деление пользователей на группы и иерархическая структуризация информационных ресурсов по уровням доступа для этих групп пользователей. Реализация механизма деления прав доступа возлагается на *Web*-сервер (как на основное звено, формирующее клиентский интерфейс) и файловую систему. Элементами, доступ к которым контролируется со стороны *Web*-сервера, являются виртуальные каталоги. Файловая система может назначать права доступа как к каталогам, так и к отдельным файлам.

Защита информационного обмена в первую очередь направлена на шифрование трафика между отдельным узлом и клиентами СРГМБнД. Поскольку в информационном обмене принимают участие только *Web*-сервер и *Web*-браузер, использующие для взаимодействия единый протокол — протокол обмена гипертекстом *HTTP*, нет необходимости применять избыточные комплексные решения, например виртуальные частные сети. Последние применяют в целях организации прозрачного защищенного взаимодействия *сетей*, в которых одновременно используются разные протоколы. Для реализации защищенного взаимодействия на уровне приложений необходима поддержка механизма такого взаимодействия самими же приложениями: *Web*-сервером и *Web*-браузером.

В СРГМБнД можно реализовать криптографическую защиту трафика на транспортном уровне стека протоколов *TCP/IP* путем использования протокола *SSL*. Поддержка протокола *SSL* встроена в большинство современных *Web*-серверов и *Web*-браузеров[6]. Хотя теоретически *SSL* может применяться для защиты информационного обмена любых приложений, использующих транспортный протокол *TCP/IP*, в настоящее время он применяется исключительно для защиты протокола *HTTP* и служит фактическим стандартом для *Web*-серверов и *Web*-браузеров в случае

[6] Протокол *SSL*, встроенный в *Web*-браузер, бесплатно распространяется в виде дополнительных программных модулей с открытым исходным ко-дом. Пример подобного подхода — бесплатно распространяемый *Web*-сер-вер *Apache*.

защищенного обмена данными. Преимущество этого метода состоит в том, что он требует поддержки только на уровне приложений (клиента и сервера), принимающих участие в информационном обмене, и не накладывает никаких дополнительных ограничений на оборудование, обеспечивающее передачу трафика.

Информационный ресурс[7]. Информационный ресурс банка ги-дрометеорологических данных (рис. 6.40) сосредоточен в тематических БД (см. рис. 6.39) и может быть расширен благодаря виртуально присоединенным к нему через *Internet* ресурсов национальных и международных банков природной информации, которые работают под эгидой ВМО.

Полностью сформированная сеть распределенных гидрометеорологических банков данных будет состоять примерно из 15 специализированных банков, структурированных по видам гидрометеорологической информации. Каталог (рис. 6.41) должен помочь пользователям как можно быстрее найти необходимую гидрометеорологическую информацию, ответить, как она делится во временных рамках и в каком виде (формат и модель) представлены данные наблюдений, получить краткое их описание, узнать, в чьей собственности находятся данные, ограничение на доступ к БнД и т.п.

БнД по метеорологии. В качестве тематического БнД на начальном этапе реализации выбран режимно-справочный БнД по

[7] Авторы признательны сотрудникам Департамента гидрометеорологической службы и мониторинга Минэкоресурсов Украины, Центральной геофизической обсерватории Украинского гидрометеорологического центра за содействие в выполнении этой задачи.

Рис. 6.40. Составные информационного ресурса СРГМБнД

метеорологии, который содержит метеорологические архивы ос-новных срочных и обобщенных за суточные и месячные интервалы результатов метеорологических наблюдений на станциях Украины за разные годы их проведения, начиная с 1881 г. На основе данных метеорологических архивов "ВОСХОД" и "СУТКИ-76" (данные в архивах представлены в символьном виде в формате ЯОД), переданных Украине Всесоюзным научно-исследовательским гидрометеорологическим институтом (ВНИИГМИ — МЦД, Обнинск), разработана интегрированная база режимно-справочных данных "МЕТЕО-1-ИС". Интегрированная БД содержит результаты основных восьми срочных (архив "ВОСХОД") и обобщенных за суточные интервалы времени

(архив "СУТКИ-76") метеорологических наблюдений по 195 метеостанциям за 1971—1975 гг.

Приведем основные характеристики интегрированной БД:

Платформа — *IBM/i* 86 класса *Pentium* III, ОС *Windows* 2000 *Advanced Server;*

Web-сервер — *Internet Information Server* 5.0 *(IIS);*

СУБД — *Microsoft SQL Server* 2000;

общий объем БД — более 350 Мб;

таблица срочных данных содержит 554398 записей (40 метеостанций);

таблица обобщенных за суточные интервалы времени данных содержит 332627 записей.

Microsoft SQL-сервер *(MS SQL)* обеспечивает лучшую интеграцию с *Web*-сервером *Internet Information Server,* он прост в эксплуатации и сопровождении. *MS SQL Server* 7.0 также значительно проще в управлении по сравнению с другими СУБД, что позволяет возложить эти функции *SQL Server* на администраторов *Windows*. Совокупная стоимость владения *MS SQL Server 7.0,* по публикуемым данным, невысокая. Однако при увеличении объема реляционной БД (РБД) возможна ситуация, когда резко

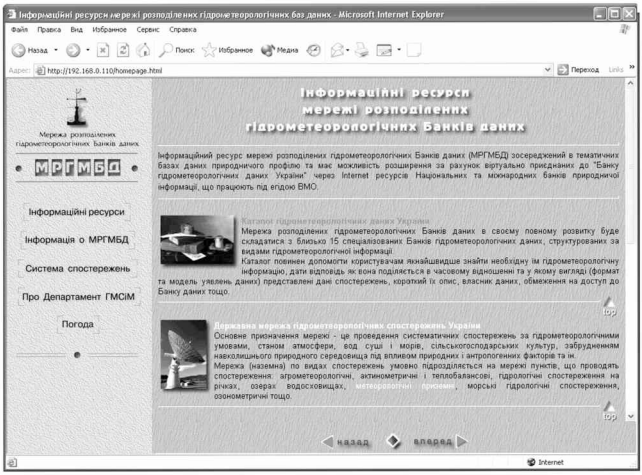


Рис. 6.41. Домашняя страница *Web*-сайта "Банк гидрометеорологических данных"

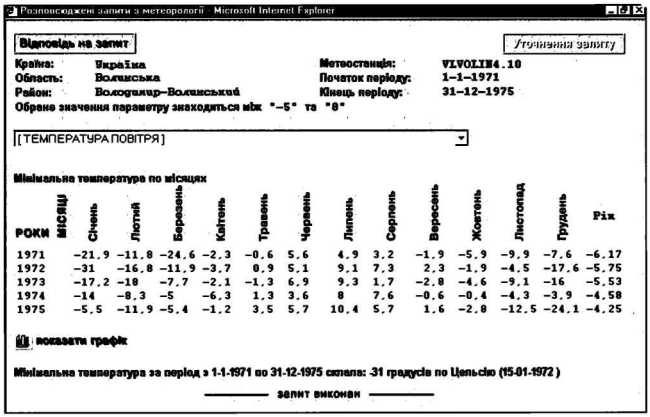
возрастает загрузка *SQL*-сервера и его производительность оказывается недостаточной для нормального функционирования ИС. В этом случае решением может быть приобретение более мощного и, следовательно, дорогостоящего оборудования, а также создание системы, обеспечивающей балансирование нагрузки между несколькими серверами, содержащими идентичные копии РБД ИС. При этом достигается максимально эффективное масштабирование системы: при увеличении нагрузки включается в работу дополнительный *SQL*-сервер (или серверы), сохраняющие производительность системы на должном уровне. Другое важное преимущество подобного подхода — повышение надежности функционирования ИС путем использования избыточных компонентов.

Программное обеспечение (ПО) по обработке метеорологической информации реализует наиболее распро-

страненные запросы[8] и отображает их результаты в символьном и графическом виде (рис. 6.42, 6.43). Его функциональность обеспечивается *SQL*-процедурами (*Stored Procedure* — хранимые процедуры[9]). ПО поддерживает *единый* клиентский интерфейс и дает пользователю возможность доступа к хранимой в ИС информации любого формата. В основе ПО заложены средства динамической генерации *web*-страниц — программные расширения, непосредственно интегрированные в *Web*-сервер (встроенный язык *ASP*). Использование сложных запросов в *ASP*-технологии связано с риском возникновения ошибок, появление которых зависит от продолжительности выполнения сложного запроса. Если время выполнения запроса или процедуры достаточно продолжительно, то в *ASP*-скрипте может возникнуть ошибка. Решается эта проблема увеличением значения параметра, ответственного за ожидание выполнения. Величина, которую необходимо задать параметру, определяется экспертным путем, исходя из времени выполнения сохраненной процедуры или

[8] Запросы выполняются по 140 элементам таблиц интегрированной базы режимно-справочных данных "МЕТЕО-1-ИС".

[9] Хранимые процедуры — программы на языке *SQL*, которые сохраняются в базе данных и выполняются средствами самой СУБД. Выполнение операторов *SQL* средствами самой СУБД осуществляется значительно быстрее, чем обращение к *SQL* из программы пользователя и возвращение ей результатов. Хранимые процедуры обеспечивают также централизованный источник программного кода, который потом может быть использован другими клиентскими приложениями.



Розпоширювані запити з метеорології - Microsoft Internet Explorer

Відповідь на запит | Уточнення запиту

Країна: Україна | Метеостанція: VLYOLIN4.10
Область: Волинська | Початок періоду: 1-1-1971
Район: Володимир-Волинський | Кінець періоду: 31-12-1975

Обране значення параметру знаходиться між "-5" та "8"

[ТЕМПЕРАТУРА ПОВІТРЯ]

Мінімальна температура по місяцях

РОКИ / МІСЯЦЬ	Січень	Лютий	Березень	Квітень	Травень	Червень	Липень	Серпень	Вересень	Жовтень	Листопад	Грудень	Рік
1971	-21,9	-11,8	-24,6	-2,3	-0,6	5,6	4,9	3,2	-1,9	-5,9	-9,9	-7,6	-6,17
1972	-31	-16,8	-11,9	-3,7	0,9	5,1	9,1	7,3	2,3	-1,9	-4,5	-17,6	-5,75
1973	-17,2	-18	-7,7	-2,1	-1,3	6,9	9,3	1,7	-2,8	-4,6	-9,1	-16	-5,53
1974	-14	-8,3	-5	-6,3	1,3	3,6	8	7,6	-0,6	-0,4	-4,3	-3,9	-4,58
1975	-5,5	-11,9	-5,4	-1,2	3,5	5,7	10,4	5,7	1,6	-2,8	-12,5	-24,1	-4,25

показати графік

Мінімальна температура за період з 1-1-1971 по 31-12-1975 склала: -31 градусів по Цельсію (15-01-1972)

запит виконан

Рис. 6.42. Результаты запроса информации

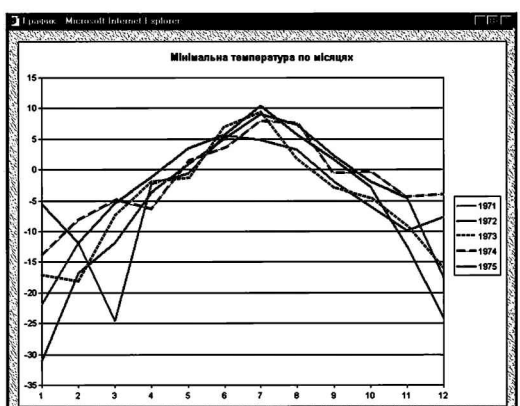


Рис. 6.43. Графическая форма результатов запроса информации

запроса. На возникновение ошибки влияет также параметр, который определяет время ожидания выполнения инструкции в скрипте. Величина этого параметра должна быть меньше времени, отведенного на ожидание выполнения всего скрипта. Это нужно для резервирования времени, необходимого для обработки ошибки, возникающей в процессе работы скрипта. Дополнительно используется инструкция < % response.buffer = True % >, что предотвращает возникновение подобной ошибки в браузере [201–203].

Создание и отображение графиков построено на основе расширения *Microsoft Office 2000* с помощью компонента *Microsoft Web Office Component*. Этот компонент представляет собою библиотеку классов для создания разнообразных графиков. Компонент может быть использован как локально (непосредственно на месте клиента), так и на стороне *Web*-сервера для отображения графиков, создаваемых программно в виде файлов в графическом формате *GIF*. Имя файла уникально и создается на основе идентификатора сессии и текущего времени. Графические файлы, созданные во время сессии, уничтожаются по ее окончании (*Session_OnEnd*). Это реализуется соответствующей процедурой в *ASP*-файле. Для функционирования графической подсистемы требуется лишь зарегистрировать компонент *Microsoft Web Office Component* на *Web*-сервере. Им реализуются такие функции, как создание файлов с графиками во временном каталоге с произвольными аргументами или с развертыванием аргументов вдоль оси абсцисс по месяцам года; разворачиваются горизонтально результаты запроса для одного, двух или трех параметров.

Для придания всему *Web*-сайту единого дизайна используются каскадные таблицы стилей — *CSS* [194, 195], обеспечивающие необходимое восприятие страниц. Применяются также *Java*-скрипты для создания определенных эффектов при отображении отдельных *html*-документов, а именно: открытие документов в новых окнах браузера с предоставлением необходимого размера и атрибутов окна; воспроизведение эффекта нажатия на кнопки с

надписью, по которым предполагается переход к новому разделу или отдельному документу; навигация по документам с учетом предыстории перемещения по ним и т.п. На рис. 6.41 приведена одна из *web*-страниц с использованными таблицами стилей и *Java*-скриптами.

БнД по агрометеорологии. В этот БнД входят режимно-спра-вочные архивные данные агрометеорологических наблюдений о росте и развитии сельскохозяйственных культур, метеоусловиях их произрастания, наблюдения опорных воднобалансовых станций и почвенно-испарительных пунктов. В нем также сохраняется нормативно-справочная информация о климатических и агроклиматических стандартных нормах по метеостанциям и средним нормам по области, что позволяет сравнивать выбранные текущие значения элементов с нормой.

Общая структура БД, составленная на основе анализа информации по агрометеорологии [204—207] и запросов к БнД, содержит:

- справочную информацию;
- информацию о постах наблюдений;
- архивные данные агрометеорологических наблюдений, которые, в свою очередь, включают в себя:

— агрометеорологические элементы — фенологические характеристики (запасы продуктивной влаги, элементы производительности сельскохозяйственных культур и т.п.);

— метеорологические элементы (условия выращивания сельскохозяйственных культур).

Концептуальная модель БД (рис. А1—А5, Приложение А) базируется на общей модели БД, перечне агро- и метеорологических элементов, структуре информации справочников и запросов к БД. Начальная структура БД имела 473 таблицы, связанные 745 отношениями (Агрометеорологический ежегодник за сельскохозяйственный год по территории Украины содержит 138 таблиц формата А3 со слабоструктурированными данными). В результате оптимизации концептуальной модели согласно правилам нормали-

зации реляционных отношений физическая[10] модель БД (рис. А6—А10) имеет 21 таблицу и 25 отношений между ними.

Особенностью спроектированной БД является возможность дополнения существующей информации новыми агрометеорологическими элементами с определенной структурой[11]. Это не требует изменения физической структуры БД, поскольку вся информация о данных классифицируется и сводится в три основные группы (декадные показатели, среднемесячные показатели и "события года", происходящие один раз в год в определенную дату). В этих группах сохраняются как все архивные данные наблюдений по фенологии, так и сопутствующая метеорологическая информация. Разделение данных осуществляется по уникальным идентификаторам, описание которых хранится в справочнике. Итак, для того чтобы внести новый элемент в БД, необходимо только расширить справочник с описанием кодов агрометеорологических элементов.

Разработанная структура БД запрограммирована в *SQL*-скрипт, который воссоздает ее в *SQL*-сервере. С помощью встроенных средств в *MS SQL Server* структура физической базы данных отображается в виде *Database Diogram* (см. Приложение А). Это позволяет, в случае необходимости, на этапе эксплуатации спроектированной БД изменять отношения между таблицами, дополнять поля или изменять их свойства.

Интерфейс пользователя спроектирован с учетом психологической ориентации на специалиста в предметной области, которому должно быть понятно, как вводить информацию, и было бы удобно это делать. Диалог ко-

---

[10] Модели изготовлены в пакете *Power Designer*, который предназначен для разработки реляционных БД. *Power Designer* имеет описание синтаксиса *SQL-языка* для большого числа СУБД. В процессе проектирования БД предоставляется возможность синхронизации концептуальной и физической моделей. По окончании проектирования пакетом генерируется отчет о структуре БД с предоставлением диаграммы взаимоотношений между таблицами, описанием объектов системы, таблиц и полей в этих таблицах.

[11] Отдельные элементы агрометеорологических наблюдений, которые должны сохраняться в БД по агрометеорологии, могут в дальнейшем изменяться.

нечного пользователя (специалиста в предметной области) с банком данных происходит на естественном профессиональном языке с использованием, где необходимо, графических изображений. Ориентация на конечного пользователя также означает, что представление результатов запросов должно происходить в виде, общепринятом для его предметной области. Учитывается распределение запросов и культур по группам и их обобщение.

Так, запросы классифицированы по группам АГРО и МЕТЕО, в каждой из которых сформулированы *обобщенные* запросы, инвариантные сельхозкультурам. Некоторые из них по группе АГРО приведены ниже:

- температура грунта и высота снежного покрова;
- запасы продуктивной влаги в грунте;
- результаты снегосъемки на полях на последний день декады;
- глубина промерзания, оттаивания грунта и высота снежного покрова;
- даты сева, массового наступления фаз развития, сбора урожая;
- элементы продуктивности и структура урожая в различных фазах развития сельскохозяйственных культур и трав (плодовых культур);
- структура урожая в массовой фазе созревания и полной зрелости;
- высота и густота стояния растений в различных фазах развития;
- результаты определения жизнеспособности плодовых культур зимнего и весеннего обследования садов;
- повреждения сельскохозяйственных культур и трав опасными гидрометеорологическими явлениями, вредителями и болезнями. В этом запросе уточняются данные о повреждении культуры:
  - по одной станции;
  - в границах области;
  - по всем метеостанциям Украины за конкретный год;
  - по станциям за отдельный период.

В соответствии с этим построен интерфейс пользователя. Последний задает культуру, выбирает интересую-

щую его группу АГРО- или МЕТЕО-запросов и затем конкретизирует запрос.

Перечень запросов и их характеристики хранятся непосредственно в БД. Это, во-первых, позволяет модифицировать (переносить их из одной группы в другую, образовывать новые группы) и добавлять запросы "на лету" — без остановки серверов (*Web, SQL*) ИС, точно так же, как при редактировании базы *html*-документов. Во-вторых, пользователь имеет полное представление о данных и реализуемых запросах, хранящихся в текущей копии БД.

Первое окно интерфейса пользователя, выполненное в виде карты Украины, предоставляет возможность наглядно осуществить выбор отдельной, нескольких или всех областей Украины (рис. 6.44). Произведя выбор, пользователь получает доступ к просмотру и селекции гидрометеостанций, принадлежащих этим областям[12] (рис. 6.45). Затем он задает период наблюдения (год, месяц, декада); выбирает интересующую его сельскохозяйственную культуру, группу АГРО- или МЕТЕО-запросов, и в конце процедуры — конкретный запрос (рис. 6.46—6.48). В случае необ-

[12] При расчете среднеобластных показателей не принимается во внимание произведенная ранее селекция гидрометеостанций, т. е. расчет производится по всем станциям области независимо от воли пользователя.

Рис. 6.44. Окно интерфейса пользователя

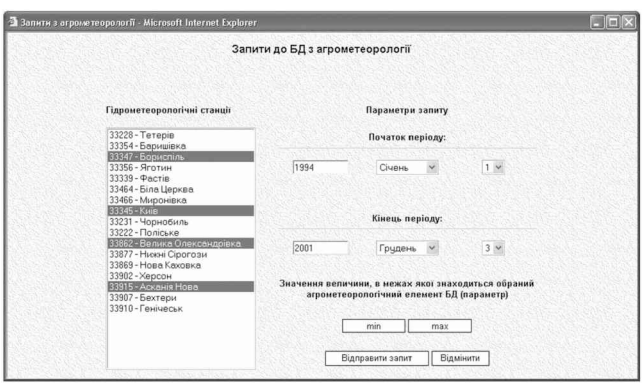


Рис. 6.45. Селекция гидрометеостанций

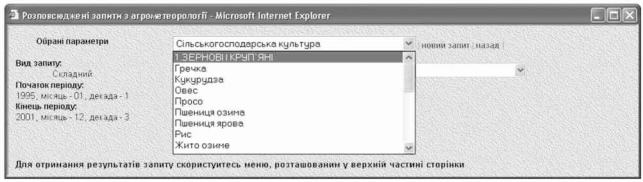


Рис. 6.46. Фрагмент окна интерфейса пользователя – выбор культуры

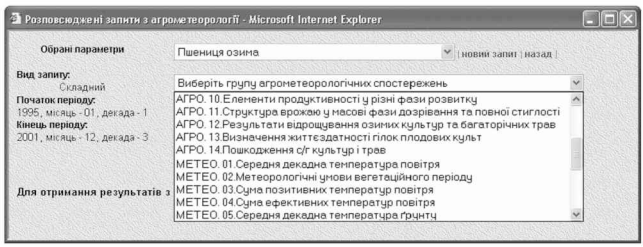


Рис. 6.47. Выбор группы АГРО- или МЕТЕО-запросов

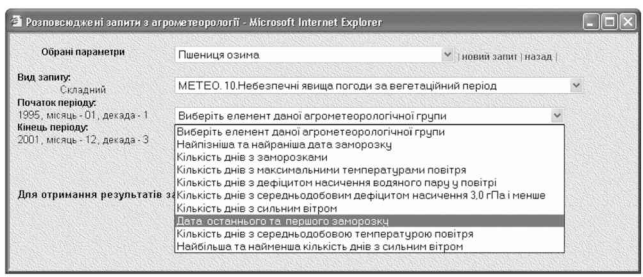


Рис. 6.48. Выбор запроса из группы

ходимости ведется уточнение запроса (рис. 6.49–6.56). Примененные решения по уточнению запросов делают интерфейс более гибким, что дает возможность пользователю, не возвращаясь к началу диалога, выбирать

условия запроса, т.е. определять последовательность действий по доступу к информации: вначале

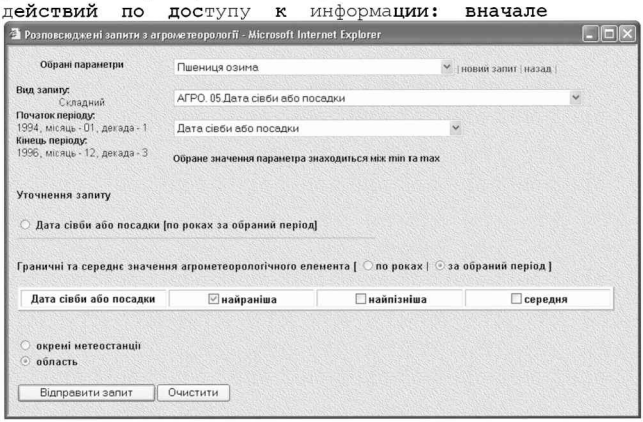


Рис. 6.49. Уточнение запроса по дате сева или посадки

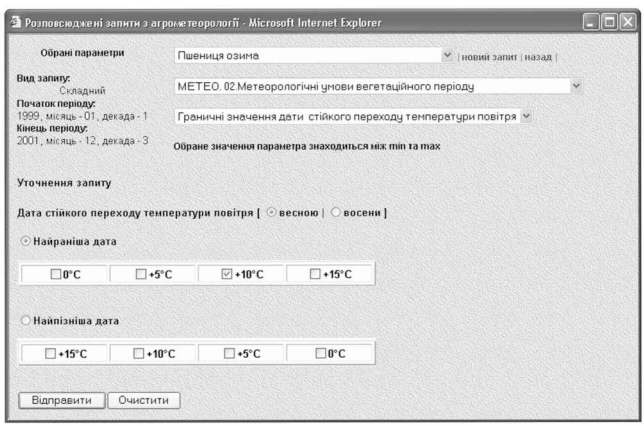


Рис. 6.50. Уточнение запроса по граничным значениям даты устойчивого перехода температуры воздуха

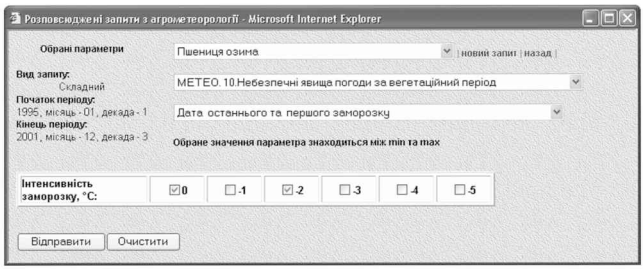


Рис. 6.51. Уточнение запроса по дате последнего и первого заморозков

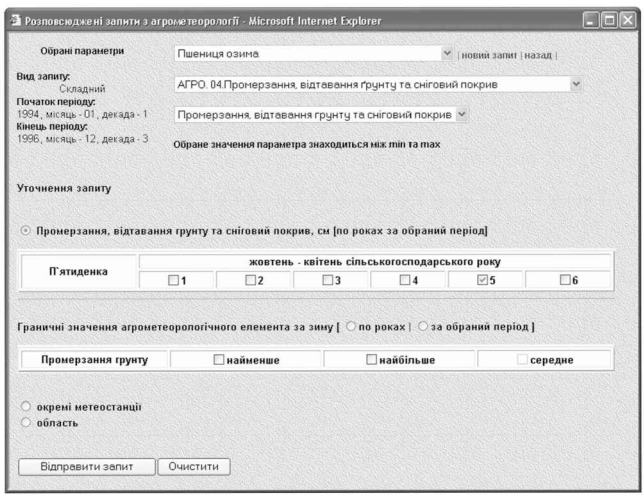


Рис. 6.52. Уточнение запроса по промерзанию, оттаиванию грунта и снежному покрову

получить архивные данные за выбранный период наблюдений; потом затребовать предельные и среднее (по выбору) значения агрометеорологического элемента (например, наиболее раннюю или наиболее позднюю дату сева) по отдельной станции или по

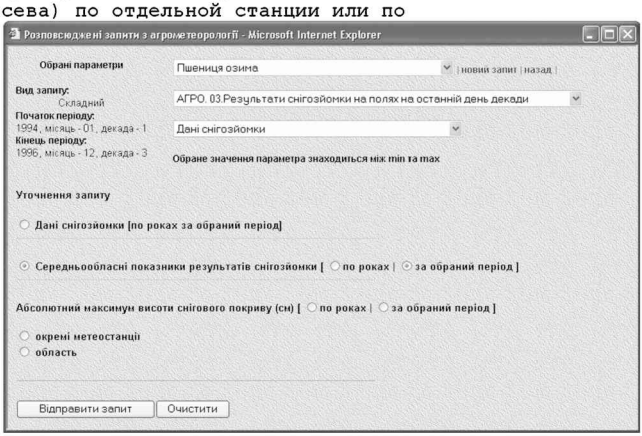


Рис. 6.53. Уточнение запроса по данным снегосъемки

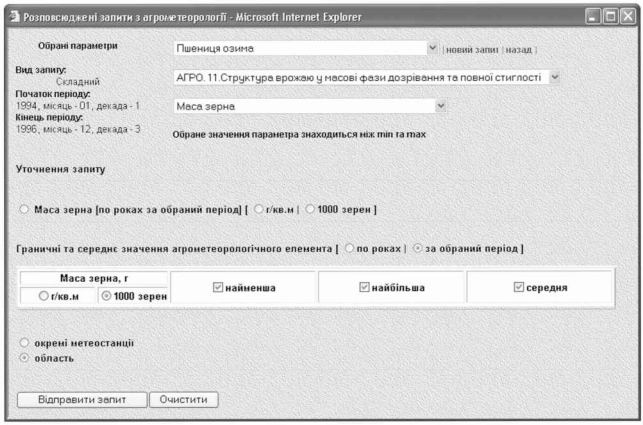


Рис. 6.54. Уточнение запроса по массе зерна

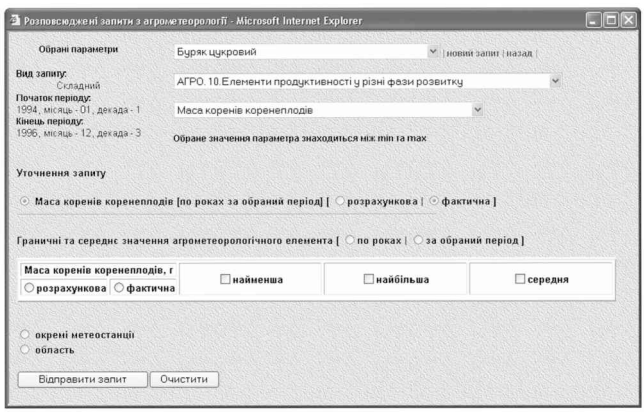


Рис. 6.55. Уточнение запроса по массе корней корнеплодов

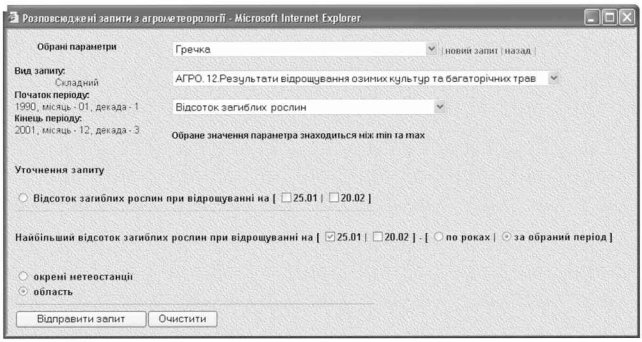


Рис. 6.56. Уточнение запроса по проценту погибших растений

области в целом; выбрать массу зерна в г/м^2 или массу 1000 зе-рен, расчетную или фактическую массу корнеплодов и т.п.

Заметим, что, задавая временной период, пользователь может произвольно задать год, месяц и декаду начала и конца наблюдений. Если же пользователь хочет проанализировать данные,

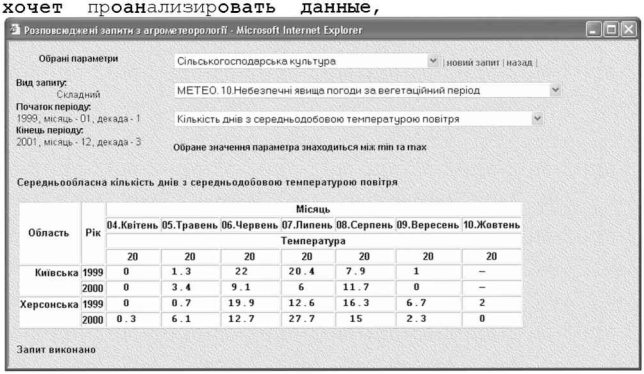


Рис. 6.57. Результат запроса по группе METEO

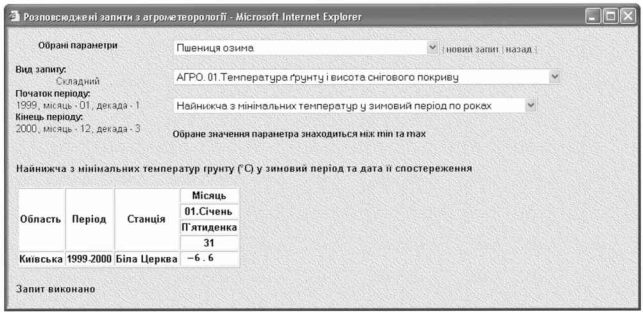


Рис. 6.58. Абсолютный минимум температуры грунта под озимой пшеницей за зимний период 1999/2000 гг. по Белой Церкви

например, в разрезе 21-й декады, т.е. 3-й декады июля, то он должен указать: месяц — 07 (июль), декада — 3; месяц — 07 (июль), декада — 3 соответственно начало и конец периода.

Ниже в качестве примера приведены *ASP*-файл и соответствующая *SQL*-процедура, использованные для выполнения и отображения результата запроса "Дата последнего и первого заморозка" (листинг 6.2, 6.3). На рис. 6.57–6.64 показаны результаты некоторых запросов к БнД по агрометеорологии.

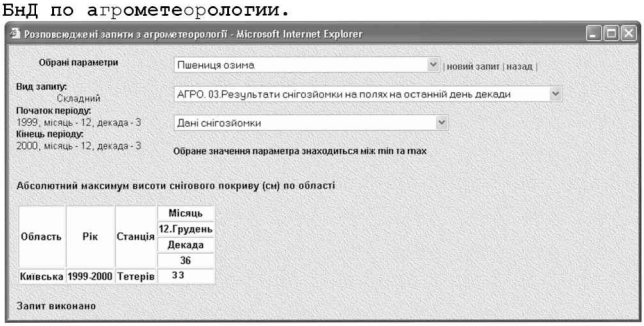


Рис. 6.59. Абсолютный максимум высоты снега на первую декаду декабря по Киевской области в 1999/2000 сельскохозяйственном году

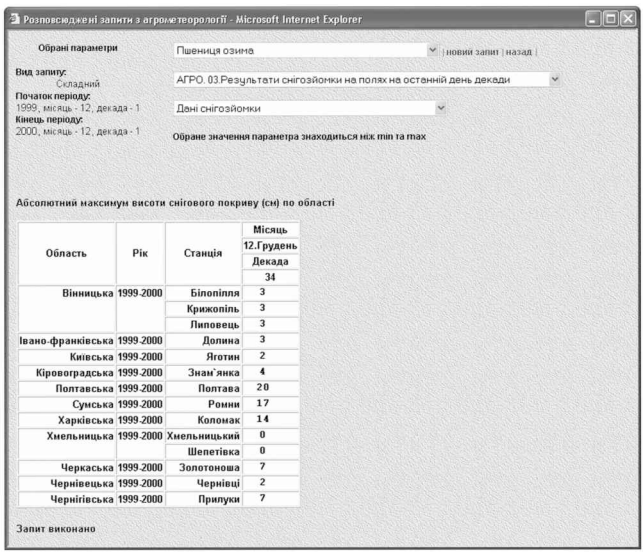
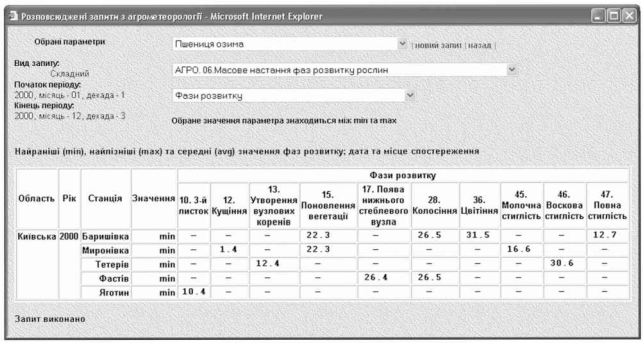


Рис. 6.60. Абсолютный максимум высоты снега на первую декаду декабря по областям Украины в 1999/2000 сельскохозяйственном году



Розповсюджені запити з агрометеорології - Microsoft Internet Explorer

Обрані параметри: Пшениця озима | новий запит | назад |

Вид запиту: Складний

Початок періоду: 2000, місяць - 01, декада - 1

Кінець періоду: 2000, місяць - 12, декада - 3

АГРО. 06.Масове настання фаз розвитку рослин

Фази розвитку

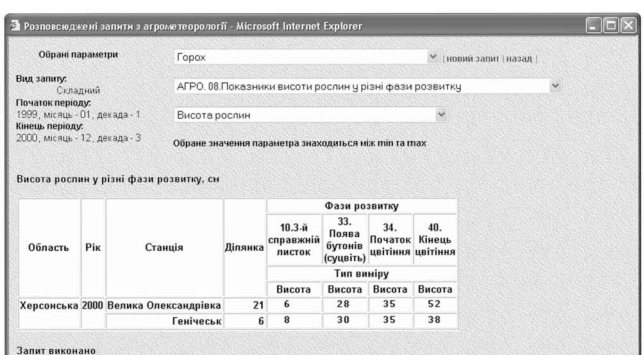
Обране значення параметра знаходиться між min та max

Найраніші (min), найпізніші (max) та середні (avg) значення фаз розвитку; дата та місце спостереження

Область	Рік	Станція	Значення	Фази розвитку									
				10. 3-й листок	12. Кущіння	13. Утворення вузлових коренів	15. Поновлення вегетації	17. Поява нижнього стеблевого вузла	28. Колосіння	36. Цвітіння	45. Молочна стиглість	46. Воскова стиглість	47. Повна стиглість
Київська	2000	Баришівка	min	–	–	–	22.3	–	26.5	31.5	–	–	12.7
		Миронівка	min	–	1.4	–	22.3	–	–	–	16.6	–	–
		Тетерів	min	–	–	12.4	–	–	–	–	–	30.6	–
		Фастів	min	–	–	–	–	26.4	26.5	–	–	–	–
		Яготин	min	10.4	–	–	–	–	–	–	–	–	–

Запит виконано

Рис. 6.61. Показатели ранних и поздних фаз развития кукурузы по Херсонской области



Розповсюджені запити з агрометеорології - Microsoft Internet Explorer

Обрані параметри: Горох | новий запит | назад |

Вид запиту: Складний

Початок періоду: 1999, місяць - 01, декада - 1

Кінець періоду: 2000, місяць - 12, декада - 3

АГРО. 08.Показники висоти рослин у різні фази розвитку

Висота рослин

Обране значення параметра знаходиться між min та max

Висота рослин у різні фази розвитку, см

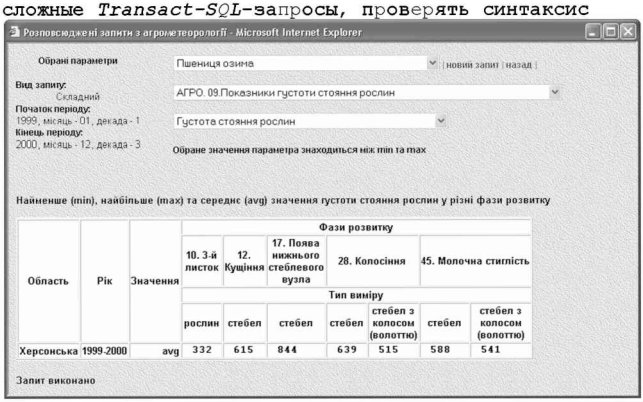
Область	Рік	Станція	Ділянка	Фази розвитку			
				10.3-й справжній листок	33. Поява бутонів (суцвіть)	34. Початок цвітіння	40. Кінець цвітіння
				Тип виміру			
				Висота	Висота	Висота	Висота
Херсонська	2000	Велика Олександрівка	21	6	28	35	52
		Генічеськ	6	8	30	35	38

Запит виконано

Рис. 6.62. Высота растений гороха в разных фазах развития по Херсонской области

Создаются *Stored Procedure* стандартными средствами, которые входят в состав *SQL Server – SQL Server Query Analyzer*. Он позволяет, кроме создания и редактирования *Stored Procedures*, выполнять запросы к ба-

зам данных *Microsoft® SQL Server™*, в том числе и сложные *Transact-SQL*-запросы, проверять синтаксис



Розповсюджені запити з агрометеорології - Microsoft Internet Explorer

Обрані параметри: Пшениця озима | новий запит | назад |

Вид запиту: Складний

Початок періоду: 1999, місяць - 01, декада - 1

Кінець періоду: 2000, місяць - 12, декада - 3

АГРО. 09.Показники густоти стояння рослин

Густота стояння рослин

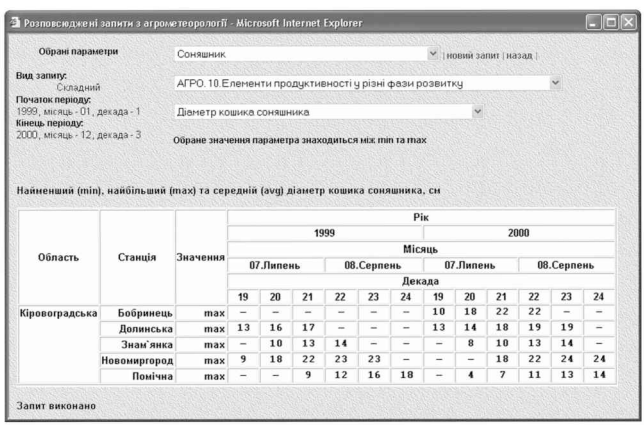
Обране значення параметра знаходиться між min та max

Найменше (min), найбільше (max) та середнє (avg) значення густоти стояння рослин у різні фази розвитку

Область	Рік	Значення	Фази розвитку						
			10. 3-й листок	12. Кущіння	17. Поява нижнього стеблевого вузла	28. Колосіння		45. Молочна стиглість	
			Тип виміру						
			рослин	стебел	стебел	стебел	стебел з колосом (волоттю)	стебел	стебел з колосом (волоттю)
Херсонська	1999-2000	avg	332	615	844	639	515	588	541

Запит виконано

Рис. 6.63. Показатели густоты стояния озимой пшеницы в разных фазах развития по Херсонской области



Розповсюджені запити з агрометеорології - Microsoft Internet Explorer

Обрані параметри: Соняшник | новий запит | назад |

Вид запиту: Складний

Початок періоду: 1999, місяць - 01, декада - 1

Кінець періоду: 2000, місяць - 12, декада - 3

АГРО. 10.Елементи продуктивності у різні фази розвитку

Діаметр кошика соняшника

Обране значення параметра знаходиться між min та max

Найменший (min), найбільший (max) та середній (avg) діаметр кошика соняшника, см

Область	Станція	Значення	Рік											
			1999						2000					
			Місяць											
			07.Липень			08.Серпень			07.Липень			08.Серпень		
			Декада											
			19	20	21	22	23	24	19	20	21	22	23	24
Кіровоградська	Бобринець	max	–	–	–	–	–	–	10	18	22	22	–	–
	Долинська	max	13	16	17	–	–	–	13	14	18	19	19	–
	Знам'янка	max	–	10	13	14	–	–	–	8	10	13	14	–
	Новомиргород	max	9	18	22	23	23	–	–	–	18	22	24	24
	Помічна	max	–	–	9	12	16	18	–	4	7	11	13	14

Запит виконано

Рис. 6.64. Максимальный диаметр корзины подсолнечника по Кировоградской области

команд для диалекта *Transact-SQL*, просматривать в графическом виде последовательность (план) выполнения запроса, возвращать результаты запроса в виде таблицы или текста и др.

Листинг 6.2

```
<!-- #include file = "MakeReport.asp" -->
<html>
<head>
        <meta    http-equiv="content-type"    content="text/html;charset=windows-
        1251">
        <title>Дата останнього та першого заморозку</title>
<LINK rel="Stylesheet" type="text/css" href="css/tablestyle1.css">
</head>
<body background="../Images/bground.jpg">

<%
sta=Request.QueryString("sta")
change=Request.QueryString("rq")
beginyear=Request.QueryString("by")
beginmonth=Request.QueryString("bm")
begindekada = Request.QueryString("bd")
endyear=Request.QueryString("ey")
endmonth=Request.QueryString("em")
enddekada = Request.QueryString("ed")
tmin = Request.QueryString("tmin")
tmax = Request.QueryString("tmax")
c0 = Request.Form("C0")
c1 = Request.Form("C1")
c2 = Request.Form("C2")
c3 = Request.Form("C3")
c4 = Request.Form("C4")
c5 = Request.Form("C5")
if cstr(c0)<>"1" then
  c0 = "0"
end if
if cstr(c1)<>"1" then
  c1 = "0"
end if
if cstr(c2)<>"1" then
  c2 = "0"
end if
if cstr(c3)<>"1" then
  c3 = "0"
end if
if cstr(c4)<>"1" then
```

```
      c4 = "0"
     end if
     if cstr(c5)<>"1" then
       c5 = "0"
     end if
     %>
    <%
    Session.timeout = 30
    If IsObject(Session("Meteo_conn")) Then
      Set conn = Session("Meteo_conn")
                                    Else
      Set conn = Server.CreateObject("ADODB.Connection")
      conn.open "Agro","sa","sql"
      Set Session("Meteo_conn") = conn
    End If
    conn.CommandTimeout=360
    Server.ScriptTimeout=360
    %>
    <%
    sql = "execute sp_me_frostall" & """ & cstr(sta) & """ & "," &
    cstr(beginyear) & "," & cstr(beginmonth) & "," & cstr(begindekada) & "," &
    cstr(endyear)                                &                          ","
    & cstr(endmonth) & "," & cstr(enddekada) & "," & cstr(c0) & "," & cstr(c1)
    & "," & cstr(c2) & "," & cstr(c3) & "," & cstr(c4) & "," & cstr(c5)
    %>
    <SCRIPT LANGUAGE=JScript RUNAT=SERVER>
    function PrintReport(rs)
    {
    all= new Array("reg_name", "sta_name", "eventname",
"te_prop", "cte_year", "cte_date", "timeyear");
    alltypes= new Array("varchar(50)", "varchar(50)", "varchar(100)", "int",
"int", "varchar(20)", "varchar(50)");
    Cases = new Array("eventname", "timeyear", "te_prop");
    Groups = new Array("cte_year", "reg_name", "sta_name");
    Groupnames = new Array("Рік", "Область", "<nobr>Інтенсивність->/nobr>
<br>Станція");
    Casenames = new Array("Параметер->", "Період->");
    Values = new Array("cte_date");
    OutReport(rs,all,alltypes,Groups,Groupnames,Cases,Casenames,Values);
    }
    </SCRIPT>
    <font size="2" color="red"><b>Дати останнього та першого
заморозків</b></font><br><br>
```

```
<%
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn, 3, 3
PrintReport rs
%>
<p><font size="2" color="red"><b>Запит виконан</b></font>
</body>
</html>
```

Листинг 6.3

```
/* Дата заморозку */
CREATE PROCedure sp_me_frostall
 (  @station varchar (1024),
   @byear int, @bmonth int, @bdek int,
   @eyear int, @emonth int, @edek int,
   @int0 int, @int1 int, @int2 int, @int3 int, @int4 int, @int5 int )
AS
Begin
declare @lcount int
declare @position int
declare @maxlen int
declare @oldpos int
declare @bdate datetime
declare @edate datetime

create table #sta(lsta_id int)

set @bdate = convert(datetime,convert(char(2),(@bdek-1)*10+1) + "-" +
convert(char(2),@bmonth) + "-" + convert(char(4),@byear))
  set @edate = convert(datetime,convert(char(2),(@edek-1)*10+1) + "-" +
convert(char(2),@emonth) + "-" + convert(char(4),@eyear))
  set @maxlen = len(@station)
  set @position = 0
  set @oldpos = 0

While @position<=@maxlen
  begin
    if   (SUBSTRING(@station,   @position,   1)=',')   and   ((@position -
  @oldpos)>1)
      begin
       insert into #sta (lsta_id) values(convert(int,SUBSTRING(@station, @oldpos+1,
@position - @oldpos - 1)))
```

```
            set @oldpos = @position
           end
          set @position = @position + 1
         end
        if (@position - @oldpos - 1)>0
         begin
          insert into #sta (lsta_id) values(convert(int,SUBSTRING(@station, @oldpos+1,
@position - @oldpos - 1)))
         end
   SELECT reg_name = (SELECT dbo.region.reg_name FROM dbo.region
        WHERE dbo.region.reg_code = (
              SELECT dbo.station_information.reg_code
              FROM dbo.station_information
               WHERE          (dbo.station_information.sta_number          =
    dbo.agro_event.sta_number ))
                                                                  ),
    sta_name = (
    SELECT sta_name FROM dbo.station_information
        WHERE              (dbo.station_information.sta_number              =
        dbo.agro_event.sta_number)
             ),
    eventname = (SELECT pd_fname FROM dbo.line_dictionary
          WHERE dbo.agro_event.pd_id = dbo.line_dictionary.pd_id
           ),
   timeyear = (case when month(te_date)>=6 then 'Осень' ELSE 'Весна' END),
   YEAR(te_date) as cte_year,
   convert(varchar(2),DAY(te_date))+"."+convert(varchar(2),   MONTH(te_date))   as
cte_date,
   te_prop
FROM dbo.agro_event
WHERE (dbo.agro_event.pd_id in (8200,8201,8202)) and
(@bdate <= te_date) and (te_date <= @edate ) and
((te_prop = 0 and @int0 = 1) or (te_prop = -1 and @int1 = 1) or
(te_prop = -2 and @int2 = 1) or (te_prop = -3 and @int3 = 1) or
(te_prop = -4 and @int4 = 1) or (te_prop = -5 and @int5 = 1))
Order by 1,2,3,4,5

 drop table #sta
End
```

6.2.1. Проблема "грязных данных"

Дальнейшее наполнение архивными данными за предыдущий период регулярных наблюдений позволит использовать БнД для эффективного агрометеорологического обеспечения сельскохозяйственного комплекса страны. Поскольку режимные данные сохраняются на бумажных, магнитных и других носителях, приведены в разных видах и форматах, им свойственна так называемая проблема "грязных данных" (данные в силу тех или иных причин неоднозначно идентифицируются), которая в последнее время тревожит Мировое сообщество [208]. Это затрудняет автоматизацию процесса ввода архивных данных в БД. Основная

причина – отсутствие единообразия, должной строгости в представлении архивных данных. Приведем несколько наиболее типичных примеров, подтверждающих это. Так, в таблицах агрометеорологического ежегодника (*Ежегодник*) используется временной диапазон аграрного года, т.е. сентябрь–декабрь предыдущего, январь–август текущего года. Например, таблица 10.12 (рис. 6.65) взята из *Ежегодника* за 1998–1999 гг., следовательно, можно предположить, что даты с сентября по декабрь относятся к 1998 г., а с января по август – к 1999 г. Если принять во внимание хронологию следования чисел в таблице, логичнее отнести даты к 1999 г., но ограничивающее правило периода аграрного года требует отнести эти даты к 1998 г.

На рис. 6.66 показано засорение данных всевозможными текстовыми сообщениями ("выпас скота" и др.), не подлежащими классификации. В колонке "фаза развития" (табл. 39.8 на рис. 6.66) отмечены фазы развития культуры на дату обследования. При заполнении, вероятнее всего, наблюдателю предлагалось ввести идентификатор фазы, что привело к множеству допустимых вариантов написания одной и той же фазы развития. В результате – типичная ошибка, допускаемая при сок-ращении наименования фазы. В одном случае сокращение заканчивается точкой, в другом – без точки. В приведенных фрагментах табл. 15.1 и 45.11 подобные ошибки очевидны. Что-

ТАБЛИЦА 10.12
ЗАПАСЫ ПРОДУКТИВНОЙ ВЛАГИ В ПОЧВЕ (мм

Дата	Запасы продуктивной влаги в слое, см				
	0 -5	0-10	0-20	0-50	0-100
*		(001)Гречиха			
Киевская область					
33466 Мироновка, уч-к: 32, разрез: 4					
07.04	0	19	41	99	181
17.04	0	14	33	85	162
08.07	0	17	36	71	138
		...			
17.07	0	16	36	82	149
22.07	0	18	40	93	166
03.09	0	19	42	94	187
06.09	0	17	37	84	167

Рис. 6.65. Фрагмент таблицы *Ежегодника*

бы использовать различные наименования, обозначающие одно и то же понятие, необходимо создать словарь сокращений или пояснений, но это не дает полной гарантии опознавания рассматриваемого участка текста в строке с необходимым идентификатором, поскольку наличие орфографических ошибок, опечаток и прочих неточностей усложняют сопоставление значения строк со справочниками. При этом также следует учитывать, что период сбора агрометеорологической информации достаточно велик и за это время ряд терминов претерпевает изменения.

Кроме того, наблюдается смещение данных в колонках таблицы, а в ряде случаев данные и вовсе отсутствуют (табл. 15.1 на рис. 6.66). Значит ли это, что наблюдения не проводились или значение агроэлемента равно нулю, — определить зачастую невозможно.

В последние годы УкрГМЦ подготавливает данные в *Excel*. Типичной ошибкой является написание даты: в колонках таблицы "месяц" вместо цифры 1 использовался символ "I", а в колонке "запас, мм" вместо числа 60 использована комбинация из цифры "6" и буквы "о". (рис. 6.67). Визуально эти ошибки определить достаточно сложно, так как символы внешне похожи, но при задании некоторых условий форматирования *Excel* обнаружить их легко. Описанный случай типичен при вводе данных со сканера с последующим их распознаванием.

При оформлении документов *Excel* и подобных им часто используются различные способы форматирования и

специальные символы. Например, в *Excel* есть возможность представить написание чисел двумя способами: традиционно арабскими или римскими цифрами. В файле *Excel* сохраняется реальное значение числа вне зависимости от формы его представления, но при переводе страницы в другой формат, например *.txt*, сохраняется лишь отформатированное представление числа (рис. 6.68). При этом можно использовать комбинацию различных алфавитов и специализированных шрифтов (например, *Symbol*).

Более сложная ситуация возникает при интерпретации комбинированных чисел (см. правую выделенную колонку на рис. 6.68). В этом случае для распознавания даты необходимо четко описать свойства представления чисел в колонке и ввести ряд правил, контролирующих процесс интерпретации данных.

Приведенные примеры не исчерпывают всего многообразия некорректности представления данных в архивах. Эти и другие
(не рассмотренные здесь) упущения в большинстве случаев не приводят к тому, что архивные документы становятся малопонятными для человека, но при их компьютерной обработке возникает масса проблем, требующих привлечения высококвалифицированных специалистов в предметной области.

Многообразие текстовых таблиц и неоднородность представления информации в них также затрудняет автоматизацию процесса внесения архивных данных в БД.

Рис. 6.66. Фрагменты таблиц *Ежегодника*

ТАБЛИЦА 39.8
РЕЗУЛЬТАТЫ ВЕСЕННЕГО ОБСЛЕДОВАНИЯ СЕЯНЫХ ТРАВ ВТОРОГО И ПОСЛЕДУЮЩИХ ЛЕТ ЖИЗНИ

Номер и название станции (поста)	Номер участка	Состояние трав весной 19...(n+1) г. дата обследования	фаза развития	высота растений, см	число стеблей на 1м²	общая оценка состояния, балл
Полтавская область						
Веселый Подол	807	20.04	возобн.вегетац	9	296	4
Полтава	18	18.04	возобн.вегетац	4	233	4
*		(357)Люцерна 2-го и более лет жизни				
Харьковская область						
Золочев	1	22.04	отраст(отращ)	5	312	3
*		(357)Люцерна 2-го и более лет жизни				
Черкасская область						
Смела	835	16.04	отраст(отращ.)	2	390	4
Жашков	838	18.04	отраст(отращ.)	5	180	4

ТАБЛИЦА 45.11
ТЕМПЕРАТУРА ПОЧВЫ И ВЫСОТА СНЕЖНОГО ПОКРОВА НА УЧАСТКЕ С ПЛОДОВЫМИ КУЛЬТУРАМИ НА ПОСЛЕДНИЙ ДЕНЬ ПЯТИДНЕВКИ

Номер и название станции (поста)	Номер участка	Наименование горизонта	Показатель	Глубина, см	1995 г. X: 5	10	15	20	25	31
*							(235)Яблоня			
Волынская область										
Любешов	28	АМ-29А	Срочная,гр.С	20						
			Срочная,град.С	40						
			Выс.снега,см							
*							(235)Яблоня			
Полтавская область										
Веселый Подол	83	АМ-29А	Срочная,гр.С	20						
			Срочная,гр.С	40						
			Выс.снега,см							

Номер и название станции (поста)	Номер участка	...	дата	...	дата	...	дата	...	дата	...	дата	...	дата	...
*			(815)Ячмень яровой											
Кировоградская область														
Новомиргород	24		14.06	4			08.07	4	16.07	4				
Знаменка	44		14.06	4			04.07	4	12.07	4				
Кировоград	67		12.06	3			26.06	3	16.07	3				
Гайворон	18		14.06	4			26.06	3	12.07	4				
Бобринец	43						стравливание животными							
Долинская	45		06.06	2			30.06	2	14.07	2				

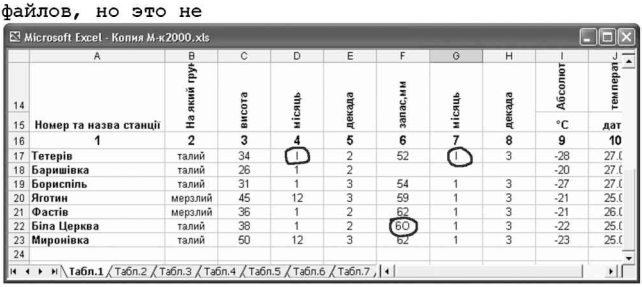
ТАБЛИЦА 15.1
ДАТА ПОСЕВА, МАССОВОГО НАСТУПЛЕНИЯ ФАЗ РАЗВИТИЯ, УБОРКИ УРОЖАЯ И ОБЩАЯ ОЦЕНКА СОСТОЯНИЯ (БАЛЛЫ) КУКУРУЗЫ

Номер и название станции (поста)	Номер участка	...	9-й лист		15-й лист		Выметывание метелки		Цветение метелки		Цветение початка		Молочная спелость	
			дата	б	дата	б	дата	б	дата	б	дата	б	дата	балл
*														
Одесская область														
Сербка	9		24.06	3			выпас скота							
Раздельная	6		20.06	4	10.07	4	10.07	3						
Сарата	22		08.06	4	30.06	4	10.07	3			20.07			
Болград	1		12.06	4	30.06	4	14.07	3	20.07	3	20.07		04.08	3

Номер и название станции (поста)	Номер участка	...	Выметывание метелки		Цветение метелки		Цветение початка		Молочная спелость		Дата уборки на силос	Дата уборки на зерновой к
			дата	б	дата	б	дата	б	дата	балл		
*												
Херсонская область												
Новая Каховка	47		14.07	3	18.07	3	28.07		31.07	3	02.08	
Аскания Нова	52		20.07	4	24.07	4	31.07		14.08	4	18.08	
Херсон	58		24.07	3	26.07	3	-		наблюдения прекращены			

Продолжение рис. 6.66

Итак, данные результатов наблюдений агрометеорологов Украины по метеорологическим факторам и состоянию сельскохозяйственных культур на производственных посевах после критического контроля сведены в архивы. В каком бы виде ни хранилась эта информация, привычным рабочим документом для агрометеорологов являются тома *Ежегодника* (два тома формата А3 за сельскохозяйственный год). Макет таблиц *Ежегодника* за все время претерпел не менее трех редакций. Принятая классификация таблиц *Ежегодника* (например, по виду культур: озимые зерновые, яровые и др.) не снижает разновидности их типов (возможность объединения таблиц без нарушения целостности их содержимого и непротиворечивости данных). Говорить о ручном вводе данных из неэлектронных источников не приходится, так как, кроме появления новых ошибок, это чревато необозримыми затратами времени из-за обширности материала (более 400 агроэлементов по 184 гидрометеостанциям). Правда, некоторая часть материалов *Ежегодника* может быть представлена в виде *DOS *.txt*-файлов, но это не


Microsoft Excel - Копия М-к2000.xls

	A	B	C	D	E	F	G	H	I	
14		На який груп	висота	місяць	декада	запас,мм	місяць	декада	Абсолют	температ
15	Номер та назва станції								°C	дат
16	1	2	3	4	5	6	7	8	9	10
17	Тетерів	талий	34	1	2	52	1	3	-28	27.(
18	Баришівка	талий	26	1	2				-20	27.(
19	Бориспіль	талий	31	1	3	54	1	3	-27	27.(
20	Яготин	мерзлий	45	12	3	59	1	3	-21	25.(
21	Фастів	мерзлий	36	1	2	62	1	3	-21	26.(
22	Біла Церква	талий	38	1	2	60	1	3	-22	25.(
23	Миронівка	талий	50	12	3	62	1	3	-23	25.(
24										

Табл.1 / Табл.2 / Табл.3 / Табл.4 / Табл.5 / Табл.6 / Табл.7 /

Рис. 6.67. Фрагмент таблицы с данными

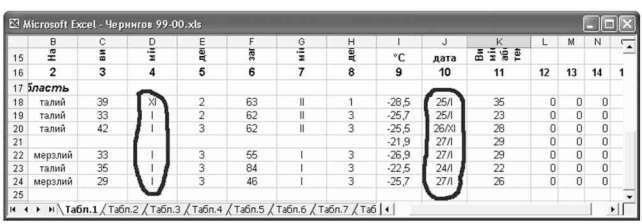
упрощает задачу, так как этим файлам присущи те же недостатки, что и текстовому материалу.

Для решения этой задачи необходимы:

• механизм приведения текстовых табличных данных, представленных в различном виде (печатные таблицы *Ежегодника*, *DOS* *.*txt*-файлы, файлы *MS Excel*-таблиц), к единому формату, пригодному для анализа, распознавания и автоматического ввода в БД;

• средства анализа табличных данных, распознавания и коррекции наиболее часто встречающихся в тексте ошибок, поддающихся формализации, и непосредственного ввода табличных данных в базы ИС как с печатных, так и с электронных носителей информации.

Решение задачи предполагает создание временного хранилища данных — некоего документа, используемого в качестве



	B	C	D	E	F	G	H	I	J	K	L	M	N
15	На	ви	ні	де	за	ні	де	°C	дата	Ви ні об ти			
16	2	3	4	5	6	7	8	9	10	11	12	13	14	1
17	*Власть*													
18	талий	39	XI	2	63	II	1	-28,5	25/I	35	0	0	0	
19	талий	33	I	2	62	II	3	-25,7	25/I	23	0	0	0	
20	талий	42	I	3	62	II	3	-25,5	26/XI	28	0	0	0	
21								-21,9	27/I	29	0	0	0	
22	мерзлий	33	I	3	55	I	3	-26,9	27/I	29	0	0	0	
23	талий	35	I	3	84	I	3	-22,5	24/I	22	0	0	0	
24	мерзлий	29	I	3	46	I	3	-25,7	27/I	26	0	0	0	
25														

Рис. 6.68. Фрагмент таблицы с данными

промежуточного формата структурированных данных. Разметка структуры хранимой в нем информации должна производиться, скажем, по типу таблиц *Ежегодника* либо любого другого вида данных. Описываемые правила анализа, схема выборки, кодирования и внесения данных в базу не должны накладывать жестких ограничений на содержимое документа, т.е. они могут быть едины для однотипных документов в любом временном промежутке. Иными словами, для каждого типа текстовых таблиц должен быть создан файл метаданных, который определял бы как правила анализа (структуры) таблиц, так и схему внесения данных в базу ИС. Тогда процедура внесения архивных данных сведется к сопоставлению файла текстовых таблиц (например, *TXT*-файл) с файлом метаданных, который отвечает обрабатываемому типу табличных

данных; при совпадении информация *TXT*-файла кодируется соответствующими кодификаторами из справочников и записывается в базу согласно описанию распоз-нанного блока.

Применение прогрессивных *XML*-технологий [194,209] для решения этой задачи, *XML*-представление данных при импорте и экспорте во внешнюю среду, например при подготовке отчетов для региональных подразделений, и прочее позволят связать ИС "Банк гидрометеорологических данных" информационным потоком с поставщиками данных для архивов, используя *XML*-документы в качестве универсального формата для обмена информацией между отдельными компонентами с различной структурой.

Выводы

Как видим, еще на одном примере подтверждена эффективность применения *Web*-технологий для построения распределенной ИС, в которой информационные ресурсы состоят из фактографических (структурированных) и слабоструктурированных данных. В рассмотренном примере развиты механизмы:

- формирования запросов к БД ИС и предоставления пользователю информации через единый интерфейс *Web*-браузер. Разработанное программное обеспечение по обработке агро- и метеорологической информации поддерживает *единый* клиентский интерфейс и дает пользователю возможность доступа к информации любого формата, которая содержится в ИС, а типовые программные решения — построить узлы сети на единых методологических основах и в необходимом количестве;
- надежной защиты информационного обмена между узлами и клиентами СРГМБнД, которая происходит с шифрованием трафика криптографическими методами с использованием протокола *SSL*.

Намечены пути решения проблемы автоматизации процесса внесения неоднородных архивных данных в реляционные БД.

Кроме изложенного, представляется дальнейшее развитие работ по интеллектуализации интерфейса пользователя, наподобие геоинформационных систем, которое позволит выводить разнообразную информацию по запросам к БД на карту Украины с привязкой к географическим координатам о запрашиваемых событиях и возможностью детализации отображаемых районов.

Исследования и разработки осуществлялись в рамках Национальной программы информатизации. Привязка основных научно-технических решений к структуре Гидрометеорологической службы Украины отражает не ограниченную специализацию, а универсальность и возможность широкого их использования непосредственно для других областей человеческой деятельности и социальной практики. Такое направление работ, как свидетельствуют материалы международных конференций, получает развитие за границей. Отечественный и зарубежный опыт подтверждает его эффективность. Так, XXI конференция Дунайских стран по гидрологическому прогнозированию и гидрологическим основам управления водными ресурсами [210] показала, что в европейских странах прослеживается четкая тенденция перехода к полной автоматизации деятельности национальных гидрометеорологических служб: охватываются все звенья обработки данных — от их сбора с помощью автоматических станций до прогнозирования гидрометеорологических характеристик.

К этим работам привлекаются большие компании, в частности, работами по созданию румынской национальной объединенной метеорологической системы занимается американская аэрокосмическая компания *Lockheed Martin (Thomas F. Pratelo* (2002) *SIMIN — Romanian National Integrated Meteorological System. 21st Conference of The Danubian Countries, report № 1, National Institute of Meteorology and Hydrology, Bucharest, Romania*) [210].

В то же время следует отметить, что архитектура сети гидрометеорологической информации ведущего европейского производителя систем сбора и управления гид-

рометеорологическими данными — финской компании *Vaisala* (*Hannu Kokko* (2002) *Inegrated Meteorological and Hydrological Networks — from Sensors to databases. 21st Conference of The Danubian Countries, report № 2, National Institute of Meteorology and Hydrology, Bucharest, Romania*) [210] подобна архитектуре предложенной распределенной сети банков гидрометеорологических данных. Это свидетельствует, что работа сделана на мировом уровне и при окончательном завершении может быть предложена странам СНГ.

ИМПОРТ/ЭКСПОРТ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

7.1. Технология импорта табличной информации в реляционные БД

Информационные системы, действующие на основе архивных данных, в период своего становления требуют наполнения присущих им тематических БД, объединяемых в БнД, значительными объемами информации. Так, распределенная гидрометеорологическая сеть БнД в полном развитии должна содержать примерно 15 специализированных БнД, структурированных по видам гидрометеорологической информации [см. параграф 6.2]. Если принять во внимание только агрометеорологическую информацию, то за сельскохозяйственный год по территории Украины она представлена 138 таблицами формата А3 (свыше 800 стра-ниц) со слабоструктурированными данными, сведенными в тома агрометеорологического *Ежегодника*. Разветвленная сеть стационарных и передвижных пунктов наблюдений по гидрометеорологическим

представления информации в таблицах вследствие таких факторов, как:

- большое количество видов наблюдений, отображающих многообразие информации;
- существенно различные структуры таблиц при значительном их числе;
- неустойчивость структуры таблицы, поскольку на протяжении всего периода наблюдений она претерпевала существенные изменения.

Эта проблема характерна не только для гидрометеорологической службы. В любых других прикладных областях, где для хранения используются, в общем случае, разнородные реляционные БД, должна быть решена задача автоматизации импорта неоднородных массивов архивных данных в БД.

Анализ общепринятой схемы импорта данных

Классическая схема импорта данных в произвольной системе клиент—сервер, направленная на актуализацию реляционных БД, приведена на рис. 7.1. Для различных архитектур клиент—сервер по-

строение аналогичной схемы очевидно и выполняется перераспределением компонентов системы.

Классическая схема импорта данных имеет следующие характерные особенности:

1) формирование исходных данных (выбор типа данных) производится оператором в ходе диалога с клиентской частью системы или предопределяется тем или иным процессом, подобным тому, что выполняет оператор;

2) наличие промежуточного звена, которое отвечает за логику обработки информации применительно к указанному типу данных и выбор соответствующего алгоритма внесения;

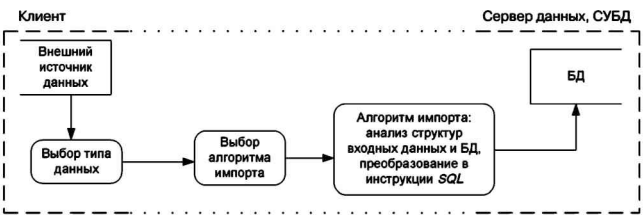


Рис. 7.1. Классическая схема импорта данных

3) алгоритм внесения указанного типа данных, роль которого часто играет хранимая процедура, осуществляет преобразование информации в элементарные структуры — инструкции *SQL* с учетом требований СУБД [211].

Хранимые процедуры, выполняемые на сервере БД, обеспечивают целостность реляционной БД по ссылкам при внесении данных, к тому же используемые при их написании знания о структуре БД и формате вносимой информации дают выигрыш в быстродействии. Такое решение часто является единственно верным и экономически оправданным в случае однородных структур вносимых данных и проектов с небольшими БД. Однако при значительном разнообразии в структуре входной информации затраты на разработку такого механизма актуализации БД недопустимо велики — по сути, под каждый типовой набор вносимых данных создается отдельная хранимая процедура. Кроме того, указанный подход не обладает достаточной гибкостью (возможность быстрой и легкой модификации ПО) и мобильностью.

В зависимости от архитектуры системы клиент—сервер (например, при широко распространенной на сегодня трехуровневой архитектуре) упомянутая выше логика обработки и выбора алгоритма внесения будет реализована в промежуточном слое, называемом сервером приложений.

Несмотря на очевидные преимущества многоуровневой архитектуры [212], снимающие остроту некоторых из перечисленных недостатков (в частности, при определенных условиях допускается переносимость ПО на различные вычислительные платформы и операционные среды), в целом классический подход требует реализации отдельного алгоритма обработки[1] для каждого фиксированного типа вносимых данных. Вследствие чего адаптивность и гибкость относительно изменений внешних условий (типы входных данных, структура БД и др.) становятся нереализуемыми.

С увеличением количества типов информации теряется простота эксплуатации таких систем, что делает их непривлекательными для конечного пользователя, а разработка новых на основе классической схемы импорта крайне затруднена. Далее будет предложена технология внесения табличных данных, лишенная большинства из рассмотренных недостатков.

Общая схема решения

При разработке технологии автоматизации процесса внесения архивных данных в реляционные БД необходимо следующее:

- независимость от структуры входной информации;
- независимость от структуры БД[2].

Прежде всего введем следующие некоторые определения.

Правило импорта — произвольная информация, которая описывает тип данных и достаточна для внесения информации в БД из

[1] Алгоритмом обработки может быть хранимая процедура, компонент сервера приложений, отдельное клиентское приложение и др.

[2] Имеется в виду наименьшая зависимость от структуры БД, допускающая автоматизацию этого процесса.

любого представителя этого типа. Задание совокупности правил импорта определяет типы вносимой информации.

ПО импорта — компонент ИС, реализующий преобразование информации в соответствии с указанным правилом импорта из некоего произвольного представления, задаваемого типом входных данных, в реляционное для заданной БД.

Управляющее импортом ПО — программная реализация процесса соотнесения элементу данных его правила импорта, если таковое существует, и выполнения некоторого первичного, одинакового для всех входных данных, преобразования, т.е. приведения данных к требованиям ПО импорта.

С учетом введенных определений обобщим классическую схему импорта данных (рис. 7.2).

Здесь набор хранимых процедур представляет собой правила импорта, а СУБД — ПО импорта; роль управляющего импортом играет промежуточное ПО (прикладное ПО, *Web*-интерфейс, оператор и т.п.).

По определению, управляющее импортом ПО не зависит от изменения набора типов входной информации, т.е. устойчиво по отношению к ее неоднородности. Следовательно, управляющее

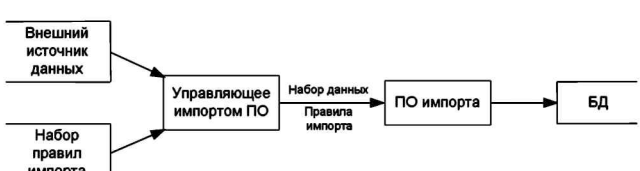


Рис. 7.2. Обобщенная схема импорта данных

импортом ПО независимо от его сложности не является существенным компонентом в рассматриваемой задаче и должно быть учтено лишь в программной реализации технологии.

Решение будем строить, разделив работу ПО импорта по внесению данных с указанным правилом импорта на два этапа: сначала выполним приведение входной информации к некоторой промежуточной структуре данных, а затем осуществим перенос этой преобразованной информации в БД. Таким образом, для удовлетворения требований независимости от структуры входной информации и структуры БД достаточно:

1) выбрать такую промежуточную структуру данных — *наборы импорта*, которая не зависит от формы представления входной информации и допускает автоматизацию процессов внесения для широкого класса БД;

2) задать формализм описания многообразия типов входной информации (правил импорта), допускающий автоматизацию преобразования данных к выбранным промежуточным структурам в рамках конкретной прикладной задачи.

Предлагаемая технология импорта данных в реляционные БД (рис. 7.3) состоит из таких этапов:

1) формирование, а впоследствии (при автоматическом внесении данных) выбор для входной табличной информации описания структуры и правил импорта;

2) совместный анализ структур входных данных на основе описания их типов и структуры БД, в результате которого генерируется алгоритм преобразования данных в последовательность наборов импорта;

3) выполнение построенного алгоритма преобразования и внесение информации из полученных структур в БД с поддержанием ее целостности, чем достигается независимость от структуры БД.

Заметим, что второй и третий этапы автоматизированы и не требуют вмешательства оператора.

Дальнейшее решение заключается в построении наборов импорта, т.е. искомой промежуточной структуры данных, пригодной для хранения произвольной реляционной информации; приведении формализма правил импорта — *метаданных о таблицах* (*МТ-данные*), достаточного для описания широкого класса табличных

данных; создании алгоритмов, автоматизирующих процессы их преобразования в последовательности наборов импорта и последующего переноса информации в БД.

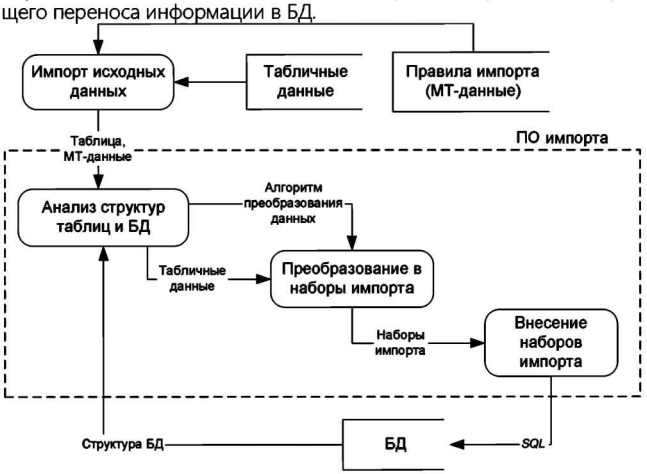


Рис. 7.3. Схема предлагаемой технологии импорта данных в реляционную БД

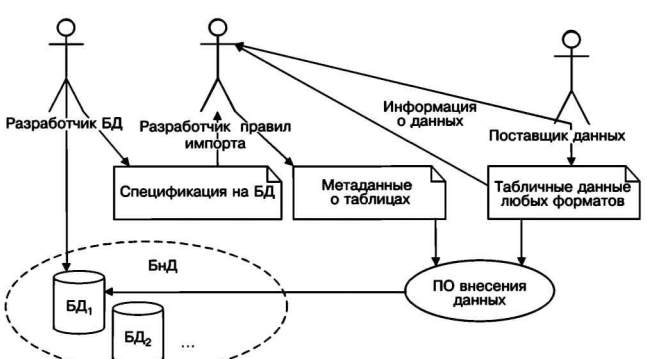


Рис. 7.4. Механизм импорта данных

Механизм импорта (рис. 7.4) в качестве составного элемента процесса создания и ведения БнД ИС должен действенно разделять роли участников процесса подготовки данных к внесению, разработки МТ-данных и эксплуатации механизма в целом в рамках ИС и обладать инвариантностью к используемым СУБД и программным платформам.

Необходимость в разделении ролей вызвана в немалой степени различной специализацией сторон. Это требование удовлетворяется стандартизацией интерфейсов и форматов обмена данными, построением последних на базе *XML* [194, 209, 213, 214]. Инвариантность к программной платформе и используемым СУБД достигается применением *Java* [215] в качестве основного языка программирования и стандарта *JDBC* [216] доступа к реляционным БД.

Наборы импорта и автоматическая поддержка целостности БД

Согласно определению наборы импорта не должны зависеть от формы представления входной информации. Проведем их построение, взяв в качестве входа некоторый простейший вид табличной информации.

Набор данных назовем *простым*, если каждый из его элементов вносится в одно и то же поле БД, а это поле — *целевым по отношению к набору*.

Представим *простую табличную информацию* (*простую таблицу*) в виде табличных данных, таких, что наборы данных, образованные семействами содержимого клеток из произвольно взятого столбца, являются простыми, при этом разным столбцам соответствуют разные целевые поля БД.

Очевидно, что произвольная таблица БД представляет собой простую табличную информацию. Более того, если рассматривать данные из нее в качестве входных, то их внесение[3] будет осуществляться построчно без какого-либо изменения их структуры. При этом правилом импорта отдельной строки служит набор целевых полей БД, отвечающих столбцам таблицы. Это наводит на мысль, что в случае простой таблицы ее строка, взятая вместе со своим прави-

[3] Внесению эквивалентно преобразование в инструкции *SQL*.

лом импорта, является набором импорта. Таким образом, можно ввести следующее определение.

Для заданной БД множество ее таблиц[4] обозначим T; поле БД — как пару (*имя_таблицы*, *имя_поля*), опустив приставку "имя"; множество полей таблицы $t \in T$ — $R(t)$. Тогда *набором импорта* назовем такое непустое семейство $\boldsymbol{IC} = \{(t_i; f_i; v_i)\}_{i=1}^{w}$, где $t_i \in T$, $f_i \in R(t_i)$, v_i — допустимое значение поля f_i, при котором соответствующая совокупность $\{(t_i; f_i)\}_{i=1}^{w}$ полей БД образует множество.

Заметим, что построение из заданного ряда входных данных набора импорта является необходимым условием для внесения этих данных в БД. Исследуем, в каких случаях эта процедура является достаточной для импорта информации.

Некоторому множеству таблиц T поставим в соответствие граф отношений таблиц как ориентированный граф $\boldsymbol{TRG} = \langle \boldsymbol{T}; \boldsymbol{E} \rangle$, где T — множество вершин, а множество ребер E задается следующим образом. Упорядоченная пара таблиц $(t, t') \in T \times T$ принадлежит E тогда и только тогда, когда некоторое поле таблицы t является экспортируемым ключом таблицы t'.

Утверждение. Достаточным условием для внесения данных из набора импорта в реляционную БД, если таковое принципиально возможно, является отсутствие циклов[5] в графе отношений таблиц этого набора.

---

[4] В параграфе 7.1 термин "таблица" употребляется только по отношению к таблицам реляционной БД. Для обозначения различных структур данных используются такие уточнения, как "табличная информация", "таб-личные данные", "простая таблица" и т.д.

[5] Петли считаем частным случаем циклов.

Доказательство. Опишем процедуру импорта данных. Отсутствие циклов в графе отношений таблиц $\boldsymbol{TRG} = \langle \boldsymbol{T}; \boldsymbol{E} \rangle$ обусловливает существование вершины (таблицы) t_0, не являющейся концом ни одного ребра. Это означает, что для t_0 уже известны значения всех полей, необходимых для внесения данных, поскольку они содержатся в исходном наборе импорта. Внесем данные в таблицу t_0. В результате этого будут зафиксированы, т.е. станут известными, значения всех экспортируемых ею ключей. Далее модифицируем набор импорта, изъяв внесенные элементы и добавив указанные значения экспортируемых ключей вместе с полями, которые их используют. Очевидно, что полученный объект есть набор импорта, уже не содержащий ссылок на таблицу t_0. Удалив из графа отношений вершину t_0 вместе с выходящими из нее ребрами, мы сведем задачу к исходной, но с меньшим числом таблиц. Повторяя описанные действия еще $|T| - 1$ раз, внесем все данные в БД.

Замечание. Наличие в утверждении оговорки о принципиальной возможности внесения данных вызвано тем, что в наборе импорта должны быть представлены значения всех необходимых для импорта полей БД с ограничением целостности *NOT NULL*, которые не являются первичными или экспортируемыми ключами.

Далее будем рассматривать только такие БД, граф отношений таблиц которых не содержит циклов.

В большинстве случаев при проектировании реляционных БД удается избежать использования циклических конструкций как средства хранения древовидных структур произвольной глубины.

Таким образом, при использовании наборов импорта в качестве промежуточной структуры данных возможна автоматизация под-

держки целостности БД при импорте данных. Это и обуславливает независимость от структуры БД.

Независимость от структуры входной информации

Выполним построение формализма описания входных данных, позволяющего осуществить преобразование информации из таблиц *Ежегодника* в последовательности наборов импорта. Определим алгоритм этого преобразования, ориентируясь на "естест-венное" восприятие структуры информации в таблице. Рассмотрим пример типичной таблицы *Ежегодника* (рис. 7.5).

Чтобы описать структуру этой таблицы, как и структуру остальных таблиц *Ежегодника*, условно разделим ее данные на *заголовок* и *клеточную часть*, или *содержимое*. *Простые строки* определим как строки из клеточной части, которые представляют собой ряды данных, соответствующие столбцам таблицы. Остальные непустые строки содержимого таблицы назовем *строками-секциями*. Строки-секции служат для более детального описания определенных последовательностей простых строк таблицы. Заметим, что рассматриваемую таблицу можно без потери информации преобразовать в таблицу, клеточная часть которой состоит лишь из простых строк (рис. 7.6).

Аналогичное преобразование возможно для любой из таблиц *Ежегодника*. Это позволяет, не ограничивая общности рассуждений, в дальнейшем пренебречь наличием строк-секций в табличных данных.

Роль заголовка таблицы при внесении из нее данных в БД. Часто заголовок содержит дополнительную информацию, являющуюся *характеристикой* данных как в отдельных столбцах таблицы (диапазоны "0—5", "0—10" и т.п.), так и в группах столбцов

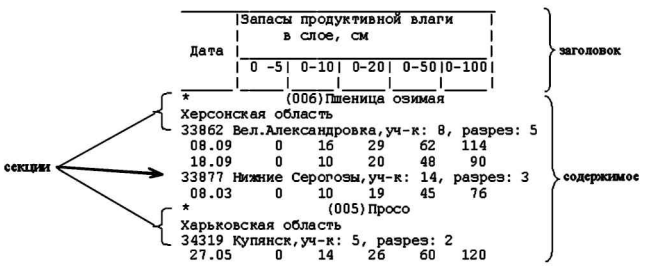


Рис. 7.5. Основные структурные элементы таблиц *Ежегодника*

(вид измерения "Запасы продуктивной влаги в слое, см"). В отдельных случаях, это зависит от структуры целевой БД, заголовок задает лишь поля БД. Чаще характеристиками служат значения-идентификаторы, вносимые в БД одновременно с данными столбцов, которые они характеризуют. Если какой-либо характеристике отвечает несколько столбцов, то такие столбцы назовем *зависимыми* по этой характеристике.

Главная особенность заголовка состоит в способности описывать отношения между различными характеристиками. Для формализации этого свойства введем следующее определение.

Деревом характеристик таблицы назовем неориентированный граф, являющийся деревом, при этом концевые вершины (листья) дерева соответствуют столбцам таблицы и содержат пары (*таблица*, *поле*); корень — фиктивная вершина (введена для удобства построения алгоритмов преобразования), а каждая из внутренних вершин содержит набор (*таблица*, *поле*, *значение*), отвечающий характеристике столбцов-листьев соответствующего поддерева.

Наглядным примером, иллюстрирующим взаимосвязь между заголовком таблицы и ее деревом характеристик, является

Дата	Запасы продуктивной влаги в слое, см					Культура	Код к-ры	Область	Станция	Код станц.	Уч-к	Р-н
	0 -5	0-10	0-20	0-50	0-100							
08.09	0	16	29	62	114	Пшеница озимая	006	Херсонская	Вел.Александровка	33862	8	5
18.09	0	10	20	48	90	Пшеница озимая	006	Херсонская	Вел.Александровка	33862	8	5
08.03	0	10	19	45	76	Пшеница озимая	006	Херсонская	Нижние Серогозы	33877	14	3
27.05	0	14	26	60	120	Просо	005	Харьковская	Куплянск	34319	5	2

Рис. 7.6. Результат преобразования, не содержащий строк-секций

Рис. 7.7. Пример дерева характеристик

рис. 7.7. Если отбросить информацию из строк-секций, то дерево характеристик отвечает рассмотренной ранее таблице *Ежегодника*. Значения характеристик на рисунке взяты в квадратные скобки, а символы C_i задают отображение в столбцы таблицы.

Алгоритм преобразования табличных данных в наборы импорта. Пусть задана таблица, ее дерево характеристик *CT*, а произвольная строка *r*, подлежащая внесению, фиксирована. Удобно считать, что в

листьях *CT* заданы значения, взятые из клеток зафиксированной строки, т. е. листья структурно ничем не отличаются от внутренних вершин дерева. Тогда можно говорить о задаче преобразования в наборы импорта полученного графа — *CT(r)*. Пусть *x* — произвольная вершина графа *CT(r)*, *L(x)* — совокупность тех ее дочерних вершин, которые являются концевыми, а остальные дочерние вершины обозначим *O(x)*. Также представим группу наборов импорта *G(x)*, которую порождает поддерево *CT(r)* с корнем в *x*. Заметим, что исходная задача состоит в нахождении *G* = *G(корень)*.

Необходимо отметить, что в отличие от простой таблицы, где строка порождает один набор импорта, произвольная таблица *Ежегодника* преобразуется в группу наборов импорта. Условимся, что разбиение исходной строки на группы клеток, соответствующие наборам импорта, определяется лишь теми характеристиками, которые являются прямыми предками концевых вершин. Этого вполне достаточно для решения поставленной задачи.

В основу алгоритма положена определенная интерпретация отношений "родства" и "соседства" между вершинами *CT(r)*. Так, в случае "родства" вершина-родитель просто группирует наборы импорта, взяв их от дочерних, что отражает понятие "более общая зависимость". Под отношением "соседство" будем понимать множество *S* = {(v; *L*))}, где v ∈ *O(x)*, а *x* пробегает вершины *CT(r)*, т.е. "соседство" описывает ситуации, когда в заголовке таблицы характеристики "соседствуют" с ее столбцами (см. клетку "Дата" на рис. 7.5). В этом случае характеристики *L(x)* должны делегировать себя в каждый из наборов импорта, порождаемых "соседями", т.е. вершинами v, такими, что (v; *L(x)*) ∈ *S*. Сформулированное правило является следствием положения об одновременности внесения данных из зависимых столбцов.

Учитывая изложенное, задаем алгоритм вычисления *G(x)*:

1) при $O(x) = \varnothing$: $G(x) := \{L(x) \cup \{x\}$, если x не является корнем исходного дерева, и $G(x) := \{L(x)\}$ в противном случае;

2) при $O(x) \neq \varnothing$:

а) $G(x) := \bigcup_{y \in O(x)} G(y)$;

б) для каждого $g \in G(x)$

$$g := \begin{cases} g \cup L(x) \cup \{x\}, & \text{если } x \text{ — не корень,} \\ g \cup L(x) & \text{в противном случае.} \end{cases}$$

Таким образом, вычисление G участвует в рекуррентной процедуре конструирования групп $G(x)$, начиная с листьев дерева характеристик.

Выполненное построение МТ-данных, или формализма описания табличных данных, посредством которого можно осуществить их преобразование в последовательность наборов импорта, завершает конструирование технологии импорта табличных данных в реляционные БД. Класс таблиц, для которых применимо такое преобразование, определяется принципиальной возможностью описания зависимостей данных в виде дерева характеристик.

Отметим, что описанный алгоритм ответствен за внесение данных по определенным правилам. Задача составления этих правил полностью возложена на человека (см. рис. 7.4). Цель нашей работы — лишь разработка средства, позволяющего автоматизировать процесс импорта для широкого класса табличных данных.

Получение информации из БД по МТ-данным

Построенная технология предполагает, что внесение данных осуществляется с помощью формализма описания структур входных таблиц — деревьев характеристик (МТ-данных). Естествен вопрос о постановке обратной задачи — экспорта информации из БД в табличное представление, задаваемое указанным формализмом.

Итак, пусть задано дерево характеристик. Концевые вершины указывают исходные поля БД — источники данных для столбцов таблицы. Для экспортируемых данных, определяемых этими полями, существуют два вида ограничений: налагаемые структурой БД (целостность по ссылкам) и обусловленные наличием внутренних вершин в дереве характеристик (ограничения в виде равенств для некоторых полей БД).

Известно [211], что запрос на получение из БД информации, отвечающей этим видам ограничений, можно выразить одной *SQL*-инструкцией вида:

SELECT **список_полей_выборки** ***FROM*** **список_таблиц** ***WHERE***
(ограничения на целостность по ссылкам: поле$_k$ = поле$_m$)
AND **(ограничения в виде равенств: поле$_i$ = значение$_j$).**

Таким образом, задача экспорта данных из БД по дереву характеристик свелась к тривиальной — формированию и выполнению одной *SQL*-инструкции известного вида. Следует отметить, что полученные данные еще необходимо преобразовать в таблицу с исходной структурой секций.

7.2. Программная реализация

Требования к механизму импорта табличных данных, приведенные на рис. 7.4 — разделение ролей, инвариантность к СУБД и программной платформе — были положены в основу разработанного программного комплекса (ПК) "Анализатор текстовых табличных данных (АТТД)". ПК АТТД предназначен для ввода архивов *Ежегодника*, представленных в текстовых файлах символами псевдографики (текстовые таблицы), в ИС "Банк гидрометеорологических данных".

ПК АТТД включает в себя:

- *XMLTable* (*XML*-таблица) — подъязык *XML* для представления входной табличной информации;
- *MetaTable* (*MT*-таблица) — подъязык *XML* для представления правил импорта — метаданных о таблицах;
- библиотеку классов языка *Java*, реализующую технологический процесс импорта/экспорта данных, а также поддержку форматов данных (*MT*-, *XML*-таблиц и текстовых таблиц);
- клиентское приложение для работы с библиотекой импорта/экспорта данных;

• *XSL*-средства [217, 218] преобразования файлов *MT*-таблиц и *XML*-таблиц в *html*-документ для содержательного просмотра;

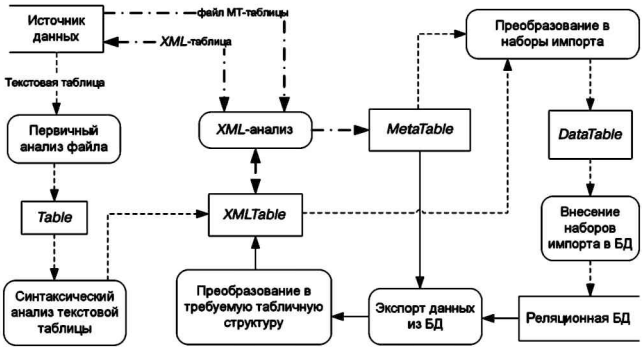


Рис. 7.8. Диаграмма информационных потоков:
\- - - - - - - - - - потоки при импорте данных; ———— — потоки при экспорте данных; - · - · - · - · - потоки при обоих режимах

средства конфигурирования[6] АТТД для импорта/экспорта архивной информации *Ежегодника*.

На рис. 7.8 показан процесс импорта/экспорта данных ПК АТТД.

***XMLTable* (*XML*-таблица) — универсальный формат представления табличной информации.** *XML*[7]-таблица представляет собой определенный подъязык *XML*, выражающий структуру *XML*-файлов, предназначенных для хранения информации, аналогичной текстовой таблице.

Рассмотрим текстовую таблицу (рис. 7.9).

Соответствующая ей *XML*-таблица может быть представлена следующим образом:

```
<?xml version = '1.0' encoding = 'windows-1251'?>
<!DOCTYPE data-root SYSTEM "./путь/agromet.dtd">
```

[6] Файлы МТ-таблиц, настройки для целевой СУБД — *MS SQL Server* 2000 и т.д.

[7] Справочные сведения по всем *XML*-элементам и атрибутам подъязыков *XML*, используемых в ПК, приведены в Приложениях Б и В.

```
<?xml-stylesheet                                              type="text/xsl"
href="./путь/шаблон_форматирования.xsl"?>
<data-root head-ref="./путь/МТ-файл.xml" tableNo="">
 <s>(006)Пшеница озимая</s>
 <s>Киевская область</s>
 <s>33466 Мироновка,уч-к: 25, разрез: 5</s>
 <r><i>07.08</i><i>0</i><i>16</i><i>34</i><i></i><i></i></r>
 <r><i>17.08</i><i>0</i><i>13</i><i>28</i><i></i><i></i></r>
 <r><i>08.09</i><i>0</i><i>24</i><i>46</i><i>84</i><i>125</i></r>
 <r><i>28.09</i><i>10</i><i>22</i><i>44</i><i>97</i><i>160</i></r>
</data-root>
```

ТАБЛИЦА 10.12

ЗАПАСЫ ПРОДУКТИВНОЙ ВЛАГИ В ПОЧВЕ (мм)

Дата	Запасы продуктивной влаги в слое, см				
	0 -5	0-10	0-20	0-50	0-100
*	(006)Пшеница озимая				
Киевская область					
33466 Мироновка,уч-к: 25, разрез: 5					
07.08	0	16	34		
17.08	0	13	28		
08.09	0	24	46	84	125
28.09	10	22	44	97	160

Рис. 7.9. Пример текстовой таблицы

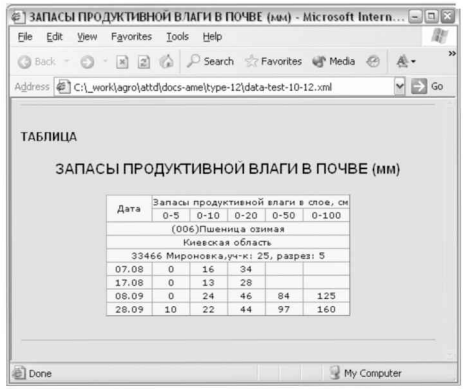


Рис. 7.10. Пример *HTML*-представления *XML*-таблицы

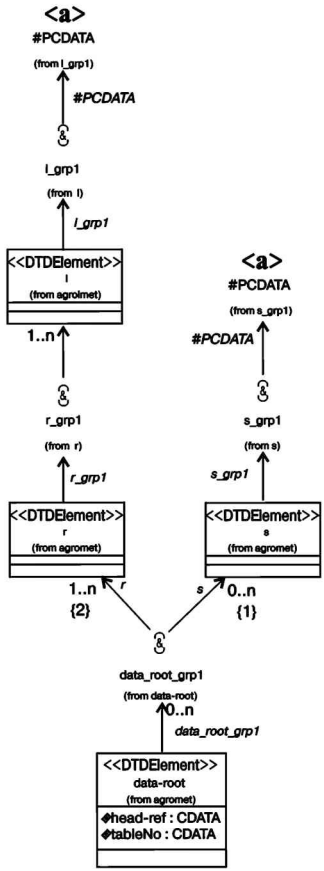


Рис. 7.11. Общий вид структуры *XML*- таблицы

Для удобного просмотра *XML*-таблицы были разработаны *XSL*-шаблоны, осуществляющие ее преобразование в *html*-документ. Преобразование выполняется автоматически при открытии *XML*-таблицы в *Internet Explorer* версии 6.0 или выше (рис. 7.10).

XML-таблица определена в файле ***agromet.dtd*** с корневым элементом ***data-root*** (см. Приложение Б).

XML-таблица с использованием *UML*-нотации[8] [219] схематически изображена на рис. 7.11.

***MetaTable* (МТ-таблица) — формализм описания табличных данных.** Файл МТ-дан-ных — файл описания текстовых таблиц или файл метаданных о текстовых таблицах (МТ-файл) — подъязык *XML*, выражающий структуру *XML*-файлов, предназначенных для хранения деревьев характеристик — правил импорта табличной информации в реляционные БД.

Назначение МТ-файла — предоставить ПК АТТД ин-

[8] Здесь и далее *DTD*-диаграммы и структуры *Java*-классов используют систему обозначений языка моделирования *UML*.

формацию о структуре анализируемой текстовой таблицы и правилах внесения в БД содержащихся в таблице данных.

Рассмотрим конструкции МТ-файлов в качестве руководства к их написанию. Приводимые ниже примеры не претендуют на полноту описания, однако они, в той или иной интерпретации часто встречаются в реальных ситуациях в качестве составных элементов конструкции МТ-файлов. С одной стороны, эти примеры проиллюстрируют общность подхода к описанию табличных данных, с другой — позволят в достаточной степени подготовить читателя к работе с МТ-файлами. Ведь одной из основных задач разработчика правил импорта, занимающегося написанием МТ-файлов, и есть умение выделить в реальной системе простые структуры и применить к ним приемы, которые будут рассмотрены далее.

Перед созданием МТ-файлов приведем необходимые сведения.

1. Синтаксис МТ-файла.

1) допустимые элементы:

- ***metaTable, head, section, child;***

2) допустимые атрибуты:

- ***metaTable:*** отсутствуют;
- ***head:*** отсутствуют;
- ***section: fmt, beg;***
- ***child: type, required, value, date, tableTo, fieldTo, dTable, dFieldC, dFieldV, dFieldT, dTypeV;***

3) корневой элемент:

- ***metaTable.***

2. Структура МТ-файла.

Здесь (рис. 7.12) принято следующее. Простому набору табличных данных, т.е. когда каждый из его элементов вносится в одно и то же поле БД, которое является целевым по отношению к этому набору, соответствуют *xml*-элементы *child*. При этом наборы данных могут образовываться семействами содержимого клеток из произвольно взятого столбца, а разным столбцам соответствуют разные целевые поля БД[9]. Для описания секций, которые, в свою очередь, служат для детального описания определенных последовательностей простых

[9] См. параграф 7.1 "Наборы импорта и автоматическая поддержка целостности БД".

строк таблицы, принят элемент *section*. Совокупность элементов *section* является списком "по вложенности" ожидаемых в документе секций.

Заголовку текстовых таблиц соответствует элемент *head* (играет роль фиктивной вершины дерева характеристик таблицы — "пустого" корня дерева).

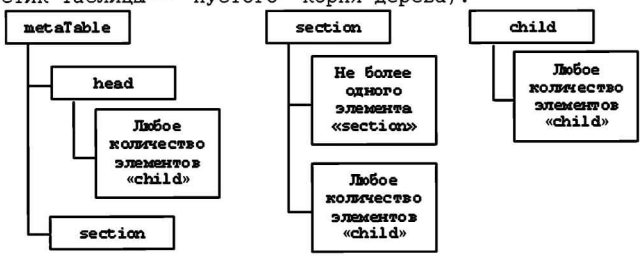


Рис. 7.12. Структура МТ-файла

Структуре заголовка текстовых таблиц отвечает дерево элементов *child* в *head*. Колонкам таблицы — листья этого дерева (см. рис. 7.7).

Итак, пусть поставлена задача: дана БД учета повреждений сельскохозяйственных культур со структурой, приведенной на рис. 7.13, требуется сконструировать МТ-файл, обеспечивающий ввод данных в эту БД из текстовой таблицы.

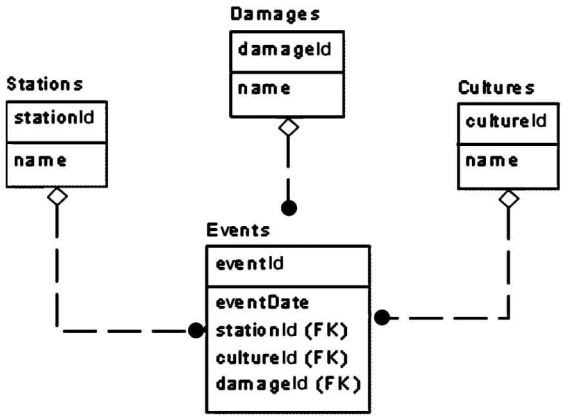


Рис. 7.13. Пример БД учета повреждений сельскохозяйственных культур

Пусть: ***Damages*** — элемент данных, хранящий информацию о возможных повреждениях; ***Cultures*** — список сельскохозяйственных культур; ***Stations*** — список станций наблюдения; ***Events*** — даты выявлений повреждений ***damageId*** культур ***cultureId*** на станциях ***stationId***. Таблицы станций и культур (таблицы, которые хранят статичные данные, будем называть *словарными* или *словарями*) считаются заполненными на момент внесения данных.

Вначале создадим простой МТ-файл, продемонстрировав смысл основных элементов и атрибутов.

Пример 1

Данные содержатся в текстовой таблице, приведенной на рис. 7.14. В ней культуры и станции представлены номерами записей в БД.

На первом этапе построения МТ-файла следует выяснить, куда в БД (имя таблицы, имя поля) будет вноситься информация из столбцов текстовой таблицы. Совокупности изменяющихся данных, имеющих одно целевое поле в БД, в МТ-файле соответствует *XML*-элемент

child. Для указания целевой таблицы и поля в БД используются его атрибуты ***tableTo*** и ***fieldTo*** соответственно.

Анализируя приведенный на рис. 7.14 фрагмент таблицы, следует обратить внимание на два момента.

Первый заключается в том, что в начальных двух столбцах некоторые значения в клетках таблицы опущены. Подразумевается, что они соответствуют значениям первой непустой клетки этого столбца, расположеной выше текущей. Ситуация с пустой клеткой в столбце **"Дата"**, вероятнее всего, может отличаться от описанной (дата наблюдения может быть и иной). В БД в обоих случаях должно внестись значение ***NULL***. Для того чтобы программа не допускала внесения в БД значения ***NULL***, а попыталась проставить его таким же значением, как и предыдущее, следует указать атрибут ***required***. Вторым — является тип данных последнего столбца (дата). Дело в том, что при интерпретации текстовой таблицы возможны два варианта поведения ПК АТТД. По умолчанию все трактуется как строка символов, за одним исключением, — если в МТ-файле для *XML*-элемента ***child*** указан атрибут ***date***, то данные--строки, соответствующие этому элементу, будут интерпретироваться как даты в опреде-

Культура	Станция	Повреждение	Дата
2	12	заморожены листья полегание	02.05.2002
	14	засыхание листьев	27.04.2002
1	50	порваны листья	03.06.2002
	23	поврежд. болезнями	14.05.2002

Рис. 7.14. Фрагмент 1 текстовой таблицы *Ежегодника*

ленном формате записи, который считается неизменным внутри одной текстовой таблицы и указывается в глобальном файле опций.

Итак, с учетом сделанных замечаний искомый МТ-файл принимает вид:

```
<?xml version='1.0'?>
<metaTable>
  <head>
    <child type="metaname" required="" tableTo="Events" fieldTo= "cul-
    tureId" />
    <child type="metaname" required="" tableTo="Events" fieldTo= "sta-
    tionId" />
    <child type="metaname" tableTo="Damages" fieldTo="name" />
    <child type="metaname" date="" tableTo=" Events" fieldTo= "eventDate"
    />
  </head>
</metaTable>
```

Схематически МТ-файл приведен на рис. 7.15.

В рассматриваемом примере поле ***damageId*** является "конф-ликтным", т.е. его значение отсутствует во вносимом наборе, но, вместе с тем, целостность БД требует его наличия. Данные в этом случае поступают в поле ***name*** таблицы ***Damages***, затем, используя идентификатор вставленной в таблицу ***Damages*** записи, производится их внесение в таблицу ***Events***.

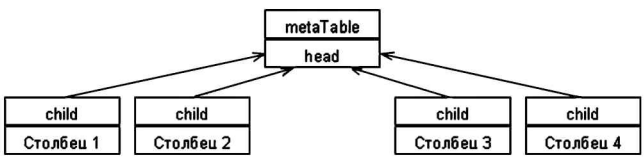


Рис. 7.15. Схематическое представление структуры МТ-файла

Культура	Станция	Повреждение	Дата
Кукуруза	Овруч	заморожены листья полегание	02.05.2002
	Луцк	засыхание листьев	27.04.2002
Ячмень	Новопсков	порваны листья	03.06.2002
	Дружба	поврежд. болезнями	14.05.2002

Рис. 7.16. Фрагмент 2 текстовой таблицы *Ежегодника*

В случае, если по каким-либо причинам (избыточность информации, "разнотипность") требуется проигнорировать данные, соответствующие какому-нибудь ***child***, то следует использовать такой формат его записи в МТ-файле:

«<child type="metaname" />».

Таким образом, после сопоставления МТ-файла и текстовой таблицы в соответствии с приведеными выше правилами (напомним, что если клетка не содержит текста, то, когда не указан флаг ***"required"***, проставляется ***NULL;*** в противном случае берется значение с предыдущей строки, соответствующее этому же ***child***[10]; в случае непустой клетки присутствующее там значение вносится без изменений, либо, если указано ***"date"***, происходит преобразование даты в некий внутренний формат и т.д.) решается задача преобразования результатов анализа клеточной части таблицы в последовательность наборов импорта и внесения данных в БД. Внесение данных осуществляется одной транзакцией, в случае какой-либо ошибки производится откат.

Пример 2

Словарные значения. Пусть в БД таблицы культур ***Cultures*** и названий ***Stations*** заполнены как словарные; так же, как и в текстовой таблице, присутствуют названия культур и станций, а не их числовые идентификаторы (рис. 7.16). Тогда, чтобы свести задачу о внесении данных из примера 2 к задаче из примера 1, нужно воспользоваться следующими атрибутами *XML*-элемента ***child***: ***dTable*** — таблица-словарь, ***dFieldC*** и ***dFieldV*** — поля в ***dTable***. Значение первого поля нужно получить, значение второго — указано в текстовой таб-

[10] Мы отождествили клетку и *XML*-элемент ***child*** из МТ-файла, описывающий данные в ней. Мы и далее будем употреблять в тексте соответствия такого рода, т.е. не различать совокупности элементов и отдельных представителей этих совокупностей. На самом деле набор ***child*** может полностью состоять из одного элемента-значения (это будет продемонстрировано далее), в этом случае такое отождествление является естественным.

лице. Когда известны значения этих атрибутов, клетки заполняются по правилу:

новое значение:= первый элемент отношения «*SELECT dFieldC FROM dTable WHERE dFieldV='старое значение'*».

Если в результате выполнения указанного выше запроса возвращается пустое отношение, то генерируется сообщение об ошибке и программа прекращает работу.

Теперь уместно вспомнить об атрибуте ***type*** из примера 1, присутствие которого обязательно. Он может принимать два значения: ***"dictionary"*** – для указания ПК АТТД, что ***child*** будет содержать словарные значения; ***"metaname"*** – во всех остальных случаях.

Итак, МТ-файл для таблицы, представленной на рис. 7.16, выглядит следующим образом:

```
<?xml version='1.0'?>
<metaTable>
  <head>
    <child type="dictionary" required="" tableTo="Events" fieldTo= "cul-
    tureId"
      dTable="Cultures" dFieldC="cultureId" dFieldV="name" />
    <child type="dictionary" required="" tableTo="Events" fieldTo= "sta-
    tionId"
      dTable="Stations" dFieldC="stationId" dFieldV="name" />
<child type="metaname" tableto="damages" fieldto="name" />
    <child type="metaname" date="" tableTo="Events" fieldTo= "eventDate"
    />
  </head>
</metaTable>
```

При необходимости можно указать дополнительное ограничение по значению некоторого поля в таблице-словаре (***dTable***). Это делается с помощью атрибутов: ***dFieldT*** – поле и его значение ***dTypeV***. При указании этих атрибутов формируются запросы вида «***SELECT dFieldC FROM dTable WHERE dFieldV='старое значение' AND dFieldT='dTypeV'***», в остальном процесс аналогичен описанному выше.

Пример 3

Секции. Основные структурные элементы текстовых таблиц, в том числе строки-секции, приведены на рис. 7.5. Напомним определение. Простые строки — это строки из клеточной части таблиц, которые представляют собой ряды данных, соответствующие столбцам таблицы. Строки-секции (остальные непустые строки содержимого таблицы) служат для более детального описания определенных последовательностей простых строк таблицы и не изменяются, в отличие от содержимого простых строк, во всех таблицах одного типа.

Повреждение	Дата
Культура: Кукуруза	
Станция: Овруч	
замороженые листья	02.05.2002
полегание	
Станция: Луцк	
засыхание листьев	27.04.2002
Культура: Ячмень	
Станция: Новопсков	
порваны листья	03.06.2002
Станция: Дружба	
поврежд. болезнями	14.05.2002

Рис. 7.17. Фрагмент 3 текстовой таблицы *Ежегодника*

Рассмотрим следующий фрагмент текстовой таблицы (рис. 7.17). На рисунке строки-секции выделены *жирным* шрифтом.

Строки-секции **"Культура"** назовем внешними по отношению к строкам-секциям **"Станция"**, и наоборот, строки-секции **"Станция"** являются вложенными по отношению к строкам-секциям **"Культура"**. Будем считать, что всегда типы строк-секций в текстовой таблице можно упорядочить по вложенности.

Для описания совокупности строк-секций в МТ-файле предусмотрен *XML*-элемент ***section***. Его атрибут ***fmt*** (*формат секции*) содержит запись, включающую в себя строку-секцию словарного типа и символ **"@"**, которым заменены все ее меняющиеся части. Например, запись для строк-секций из приведеной таблицы будет выглядеть так: **"Культура: @"** и **"Станция: @"**.

После задания формата секции каждому неконстантному фрагменту секции (т.е. каждому символу **"@"**) ставится в соответствие некоторый *XML*-элемент типа ***child***, который и определяет правила внесения в БД информации из этого фрагмента.

Следует отметить, что при написании МТ-файла *XML*-элементы секции "вкладываются" соответственно той упорядоченности по вложенности, которая наблюдается в текстовой таблице.

Приведем текст получившегося МТ-файла:

```
<?xml version='1.0'?>
<metaTable>
<head>
 <child type="metaname" tableTo="Damages" fieldTo="name" />
 <child type="metaname" date="" tableTo="Events" fieldTo= "eventDate"
 />
 </head>
 <section fmt=" Культура: @" >
  <child type="dictionary" required="" tableTo="Events" fieldTo=
  "cultureId"
    dTable="Cultures" dFieldC="cultureId" dFieldV="name" />
  <section fmt=" Станция: @" >
    <child type="dictionary" required="" tableTo="Events" fieldTo=
    "stationId"
      dTable="Stations" dFieldC="stationId" dFieldV="name" />
  </section>
 </section>
</metaTable>
```

Проверка ПК АТТД, является ли текущая строка строкой-секцией, производится в самом начале, и в случае ответа "нет" происходит разбиение простых строк на клетки. Примерами "плохих" форматов секций являются: «***fmt*="@"»**, «***fmt*="@ @"»**.

Дополнительным средством обнаружения строк-секций может служить атрибут ***beg***, значением которого является один символ, например **"*"**. Задание значения атрибута таково: на первой позиции у всех строк-секций, описываемых этим *XML*-элементом, должен стоять именно этот символ, а вся остальная часть строки (начиная со второй позиции) должна соответствовать формату.

Например, описанию секции верхнего уровня (рис. 7.18) отвечает запись: ***beg="*", fmt="(@) @"***. Первая строка секции начинается с символа "*", а оставшаяся часть строки имеет следующий формат: вначале сле-

дует символ "(", за ним — данные, и закрывает их символ ")". Завершают запись меняющиеся словарные данные. И те, и другие данные следует поместить со свойствами

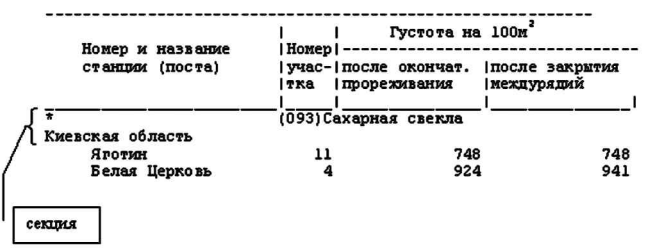


Рис. 7.18. Фрагмент 4 текстовой таблицы *Ежегодника*

первого и второго подэлементов ***child*** соответственно. Если какие-либо данные следует исключить из списков внесения в БД, то надо проставить тип ***metaname*** без указания остальных атрибутов.

Представим фрагмент МТ-файла для рассмотренного примера:

```
<section fmt="(@)@" beg="*">
<child type="metaname" />
<child type="dictionary" tableTo="tenphenology" fieldTo="pd_id"
      dTable="line_dictionary" dFieldC="pd_id" dFieldV="pd_fname"
      dFieldT="gd_id" dTypeV="2" />

<section fmt="@ область">
      <child        type="dictionary"        tableTo="tenphenology"
      fieldTo="pd_id"
            dTable="line_dictionary"                  dFieldC="pd_id"
            dFieldV="pd_fname"
            dFieldT="gd_id" dTypeV="3" />
</section>
</section>
```

Пример 4

Общий случай (дерево характеристик). На этом этапе построения МТ-файла будем считать, что столбцы и секции текстовой таблицы уже описаны, т.е. он имеет структуру, представленную на рис. 7.19.

В результате проведенной работы над МТ-файлом должно получиться уже описанное ранее дерево характеристик таблицы (на данный момент известны только его корень — ***metaTable*** и листья — ***child***-столбцы, а также некоторая часть, представляемая ***child*** в секциях). Неизвестная область изображена пунктирными стрелками.

Следует теперь выявить все характеристики и зависимости так, как они были определены ранее. Рассмотрим это на реальном примере.

В таблице, приведенной на рис. 7.20, есть шесть столбцов и три строки-секции, имеющие следующие форматы: **«(@)@»**, **«@ область»**, **«@ @, уч-к: @, разрез: @»**, при этом для первой из них задано ***beg="*"***.

Все столбцы *являются зависимыми* по характеристикам, взятым из секций. Столбцы, кроме первого, характеризуются диапазоном "0—5", "0—10" и т.д., при этом все они зависимы по виду измерения "Запасы продуктивной влаги в слое, см". Таким образом, к каждому результату измерения, который находится в клетках столбцов со второго по шестой, присоединяется характеристика (диапазон измерения) и соответственно иденти-

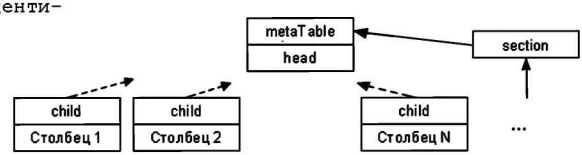


Рис. 7.19. Структура произвольного МТ-файла

фикатор вида измерения. Далее необходимо, чтобы клеткам одной простой строки отвечали даты, находящиеся в первом столбце текстовой таблицы.

Заметим, что характеристики являются константами в пределах одного типа текстовой таблицы. Из этого следует, что они постоянны для одного МТ-файла. Средство описания их внесения в БД остается тем же самым — *XML*-элемент ***child***. При этом используется атрибут ***value*** для задания значения характеристики. Поскольку эти элементы ***child*** ничем не отличаются по функционально-

сти от рассмотренных ранее элементов ***child***, то возможно их применение для получения словарных значений, форматирования дат и т.п. При этом их *XML*-иерархия практически повторяет структуру заголовка текстовой таблицы.

Рис. 7.20. Фрагмент 4 текстовой таблицы *Ежегодника*

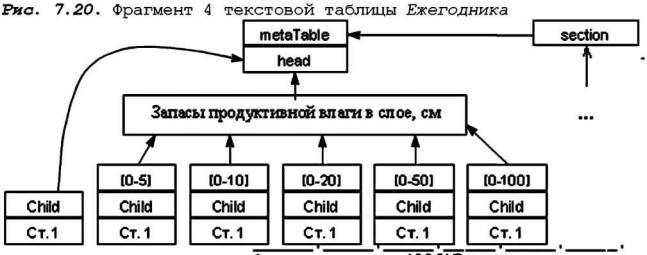


Рис. 7.21. Вариант структуры МТ-файла

Таким образом, анализируя текстовую таблицу на рис. 7.5, приходим к следующей

```
*                    (206)Виноград
Херсонская область
33869 Новая Каховка,уч-к: 1, разрез: 2
 27.03     0     10     20     56    109
 27.04     0     12     24     67    134
 27.05     0      8     15     47    102
 27.06     0      0      1     13     49
 27.07     0      0      0      5     28
 27.08     0      1      3     12     28
 27.09     0      6     13     21     40
 27.10     0      2      4      9     11
```

структуре МТ-файла (рис. 7.21). Значения характеристик указаны в квадратных скобках.

Соответствующий МТ-файл имеет следующий вид:

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE metaTable SYSTEM "..\meta-table.dtd">
<metaTable>
    <head>
      <child type="metaname" required="" date=""
                        tableTo="estimation" fieldTo="est_date" />
      <child type="metaname" value="7100"
                        tableTo="estimation" fieldTo="pd_id" >
      <child type="metaname" date="" value="01.01"
                        tableTo="use_lot" fieldTo="ul_year">
      <child type="metaname" required="" value="5"
                        tableTo="estimation" fieldTo="est_property" >
            <child type="metaname" tableTo="estimation"
            fieldTo="est_values"/>
      </child>
      <child type="metaname" required="" value="10"
                        tableTo="estimation" fieldTo="est_property" >
            <child type="metaname" tableTo="estimation"
            fieldTo="est_values"/>
      </child>
      <child type="metaname" required="" value="20"
                        tableTo="estimation" fieldTo="est_property" >
            <child type="metaname" tableTo="estimation"
            fieldTo="est_values"/>
      </child>
      <child type="metaname" required="" value="50"
                        tableTo="estimation" fieldTo="est_property" >
            <child type="metaname" tableTo="estimation"
            fieldTo="est_values"/>
      </child>
      <child type="metaname" required="" value="100"
                        tableTo="estimation" fieldTo="est_property" >
            <child type="metaname" tableTo="estimation"
            fieldTo="est_values"/>
```

```
      </child>
  </child>
  </child>
</head>

<section fmt="(@)@" beg="*">
   <child type="metaname" />
   <child type="dictionary" tableTo="use_lot" fieldTo="pd_id"
         dTable="line_dictionary" dFieldC="pd_id" dFieldV="pd_fname"
         dFieldT="gd_id" dTypeV="2" />

   <section fmt="@ область">
     <child type="metaname" />

     <section fmt="@ @,уч-к: @, разрез: @" >
         <child type="dictionary" required=""
               tableTo="estimation" fieldTo="sta_number"
               dTable="station_information" dFieldC="sta_number"
                                        dFieldV="sta_code" />

         <child type="metaname" />
         <child type="metaname" required="" tableTo="lot"
         fieldTo="lot_number" />
         <child type="metaname" required="" tableTo="cut" fieldTo="cut_num"
         />
     </section>
   </section>
 </section>
</metaTable>
```

МТ-файл определен в файле ***agromet.dtd*** с корневым элементом ***metaTable*** (Приложение В) и схематически изображен на рис. 7.22 с использованием *UML*-нотации.

◀*Рис. 7.22.* Общий вид структуры МТ-файла

Структурное описание библиотеки классов АТТД. Комплекс представляет собой библиотеку классов *Java* со следующей структурой модулей:

- ***attd*** — исполняемые классы: ***agromet;***

1) ***parser*** — представление и анализ текстовых таблиц;

2) ***metatable*** — представление МТ-файлов, а также средства их построения;

3) ***xmltdata*** — представление *XML*-таблиц;

4) ***datatable*** — представление "простых таблиц";

• ***dbi*** — классы, ответственные за внесение наборов импорта в БД;

• ***xmldb*** — классы, ответственные за экспорт информации из БД;

• ***xmldbtable*** — классы, являющиеся промежуточным результатом экспорта данных из БД;

• ***util*** — классы общего назначения.

Большинство модулей содержит также классы-исключения для контроля ошибок процессов, которые они реализуют.

Модуль «attd.agromet.parser»

Диаграмма классов приведена на рис. 7.23 с использованием UML-нотации[11] [219].

Функциональность некоторых классов:

• ***StringArray***[12] — вспомогательный, представляет массив строк;

• ***TableHead*** — заголовок текстовой таблицы, определяющей разметку столбцов;

• ***Table*** — текстовая таблица как пара: "заголовок" и "содер-жимое";

• ***TableParser*** — позволяет получать объекты ***Table*** из текстового файла, а также извлекать из "содержимого" отдельные строки таблицы и клетки;

• ***TextTablesCorrector*** — вспомогательный, предоставляет набор функций по синтаксическому анализу текстовых таблиц.

Модуль "attd.agromet.metatable"

Диаграмма классов приведена на рис. 7.24.

---

[11] При описании структуры *Java*-классов используется система обозначений языка моделирования *UML*.

[12] Здесь и далее приводятся локальные имена классов соответствующего модуля.

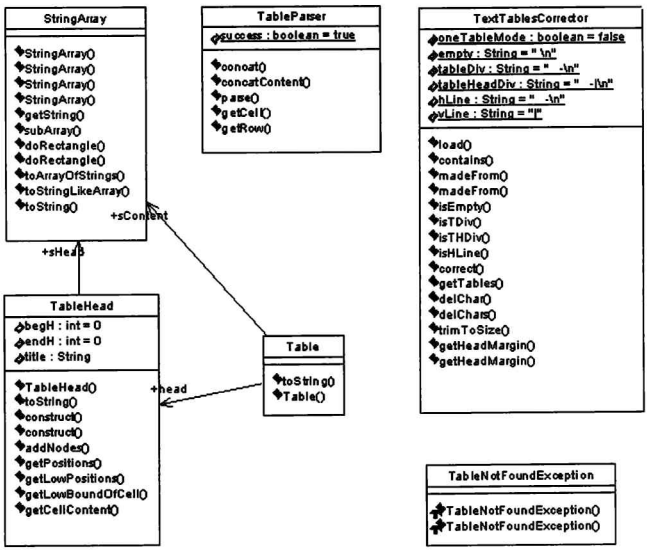


Рис. 7.23. Диаграмма классов модуля ***"attd.agromet.parser"***

Функциональность некоторых классов:

• ***SectionParser***, ***SectionInstance***, ***SectionInstances*** — используются для синтаксического анализа строк-секций текстовых таблиц;

• ***ChildGroups*** — классы-шаблоны для наборов импорта;

• остальные классы — представление МТ-файлов.

Модуль "attd.agromet.xmltdata"

Диаграмма классов приведена на рис. 7.25.

Функциональность некоторых классов:

• ***XmlTable*** — *XML*-таблица как последовательность строк таблицы и секций.

Модуль "attd.agromet.datatable"

Диаграмма классов приведена на рис. 7.26.

Функциональность некоторых классов:

• ***DataTable*** — простая таблица, т.е. строки таблицы — суть набора импорта;

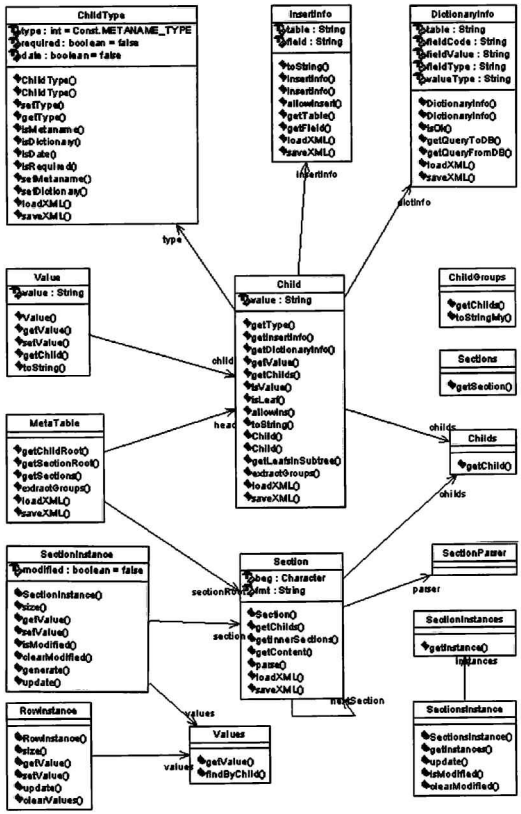


Рис. 7.24. Диаграмма классов модуля ***"attd.agromet.metatable"***

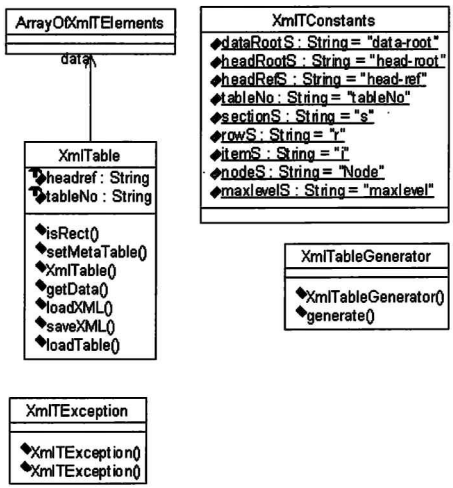


Рис. 7.25. Диаграмма классов модуля ***"attd.agromet.xmltdata"***

• ***DBFieldInfo*** — правило внесения столбца простой таблицы;

• ***DBField*** — элемент набора импорта.

Модуль "attd.agromet.dbi"

Диаграмма классов приведена на рис. 7.27.

Функциональность некоторых классов:

• ***KDatabase*** — представляет БД-приемник наборов импорта;

• остальные классы — обеспечивают необходимую функциональность для ***KDatabase***.

Модуль "attd.agromet.xmldb"

Диаграмма классов приведена на рис. 7.28.

Функциональность некоторых классов:

• ***DataExtractor*** — обеспечивает функциональность экспорта данных из БД;

• ***DDatabase*** — представляет БД-источник данных.

Модуль "attd.agromet.xmldbtable"

Диаграмма классов приведена на рис. 7.29.

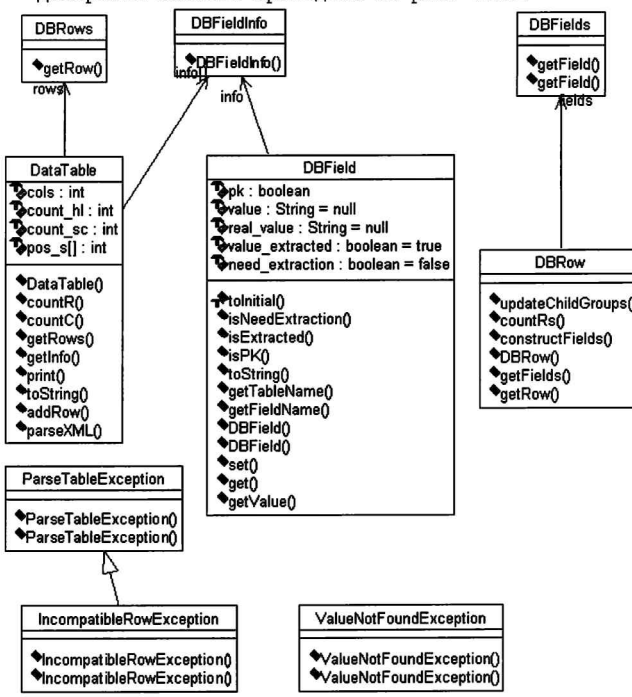


Рис. 7.26. Диаграмма классов модуля ***"attd.agromet.datatable"***

Функциональность некоторых классов:

• все классы используются модулем ***xmldb*** при экспорте информации из БД как промежуточная структура данных.

Модуль "attd.agromet.util"

Диаграмма классов приведена на рис. 7.30.

Функциональность некоторых классов:

• ***DBConnector*** – управляет установлением соединений с БД;

• ***Convertor*** – обеспечивает изменение кодировки текста;

• ***AgroConvertor*** – изменения кодировки текста в ПК АТТД;

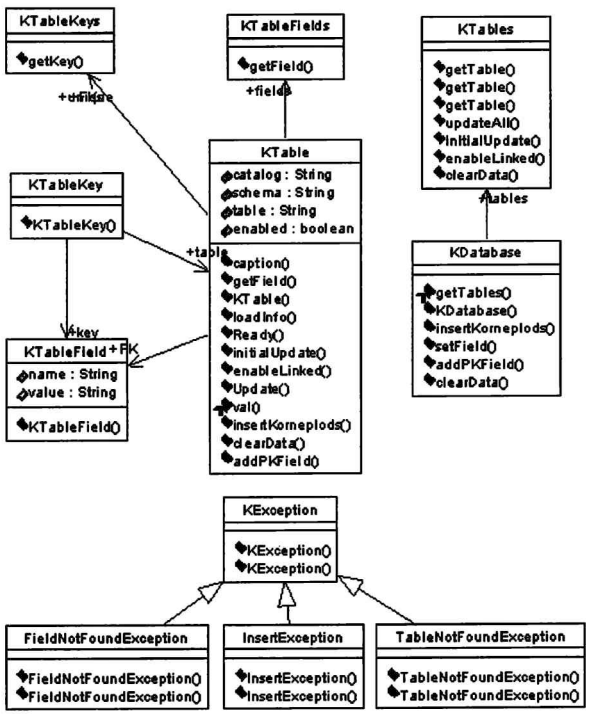


Рис. 7.27. Диаграмма классов модуля ***"attd.agromet.dbi"***

• ***Properties*** — файл свойств, или глобальных опций ПК АТТД, а именно ***ConsoleUtil*** — абстрактный класс для вспомогательных функций консольных приложений;

• ***Formatter*** — объединение функций вывода в ***dump***-файл, преобразования дат, форматирования сообщений пользователю.

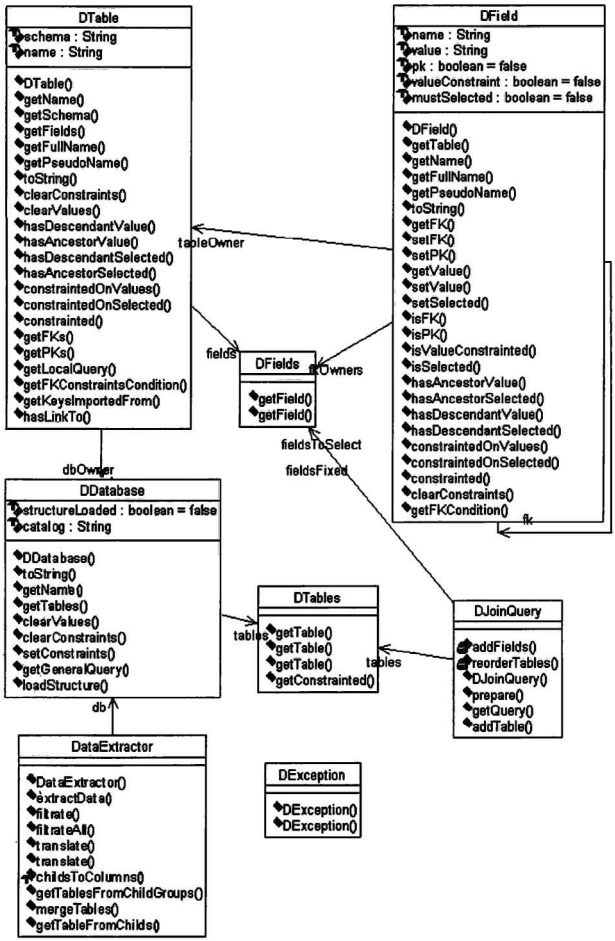


Рис. 7.28. Диаграмма классов модуля ***"attd.agromet.xmldb"***

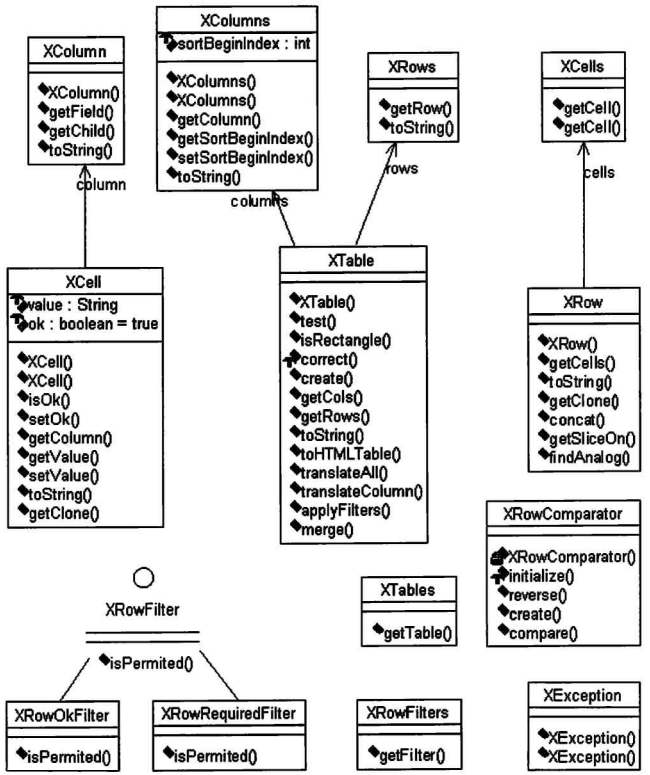


Рис. 7.29. Диаграмма классов модуля ***"attd.agromet.xmldbtable"***

Работа ПК АТТД. Для функционирования ПК совместно с СУБД *MS SQL Server* 2000 устанавливаются следующие специальные программные компоненты:

- *Java™ 2 Runtime Environment (JRE), Standard Edition, version* 1.4.0;

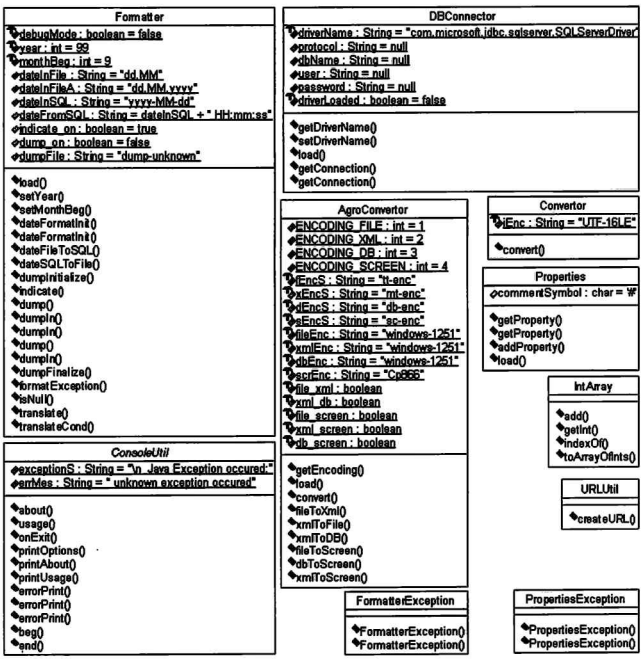


Рис. 7.30. Диаграмма классов модуля ***"attd.agromet.util"***

- *Java Virtual Machine* (*JVM*), программные библиотеки классов, необходимые для выполнения программ на языке программирования *Java*;
- *Oracle XML Parser for JAVA* 9.0.1.1.0*A Production* (файл *xmlparserv2.jar*);
- программные библиотеки классов для синтаксического анализа *XML*-файлов;
- *Microsoft JDBC Driver for MS SQL Server* (файлы *msbase.jar*, *mssqlserver.jar*, *msutil.jar*);
- программные библиотеки классов, реализующие интерфейс *JDBC* доступа к СУБД *MS SQL Server*;

- ядро ПК АТТД (файл *attd.jar*);
- программная библиотека классов, реализующая функции ПК АТТД, и некоторые другие программы (например, программа коррекции[13] текстовых таблиц) или файлы (например, файл глобальных опций[14] ПК (*attd.properties*)).

Кратко представим работу с ПК. Предварительно для каждого типа текстовых таблиц создается файл описания, или файл метаданных о текстовых таблицах (МТ-файл), который определяет как правила анализа, так и схему внесения данных из таблицы в БД. На вход ПК подаются файл текстовых таблиц (*TXT*-файл) и МТ-файл, отвечающий обрабатываемому типу табличных данных. После инициализации ПК выполняются действия в определенной последовательности, их условно можно разделить на следующие этапы[15].

1. Анализируется текстовая таблица[16] и конструируется ее *XML*-представление. На этом этапе осуществляется синтаксический разбор символьных последовательностей входного *TXT*-файла, т.е. анализируется заголовок таблицы, выделяется и анализируется клеточная часть: производится подсчет количества столбцов, выявляются секции, происходит разбиение строк на клетки и т.п.

2. Считывается МТ-файл описания содержащегося в таблице блока информации, проводится его анализ на корректность[17] и активизируется создание шаблона будущей структуры данных. На основе зависимостей, содержащихся в МТ-файле, определяется структура данных,

---

[13] Программа коррекции предназначена для "слияния" нескольких частей текстовой таблицы, расположенных в отдельных файлах.

[14] Файл глобальных опций предназначен для общего конфигурирования ПК АТТД; он предоставляет средства для управления текущим этапом работы ПК.

[15] Наличие того или иного этапа зависит от параметров в файле глобальных опций.

[16] В случае необходимости производится "слияние" нескольких частей текстовой таблицы, расположенных в отдельных файлах.

[17] Ошибки некорректности синтаксиса *XML*-файлов генерируются процессором *XML* и, как правило, имя соответствующих классов-исключений начинается с *oracle.xml.parser.v2.XML*.

используемая при внесении в БД, и конструируется таблица данных.

3. Осуществляется вывод промежуточных результатов таблицы данных в *DUMP*-файл.

4. Происходит соединение с сервером БД и запрашивается информация о структуре БД.

5. Производится попытка внесения данных в указанную БД.

6. Завершающая фаза. Если в ходе работы не возникло ошибок, то ПК АТТД сохраняет внесенную в БД информацию, в противном случае — делается отмена модификации БД и осуществляется вывод информации об ошибке. На этом ПК АТТД завершает работу.

Сообщения об ошибках ПК АТТД представляют собой исключительные ситуации языка *Java*. Будем называть их исключениями (*exceptions*).

При возникновении исключения работа программы прерывается и на экран выводится сообщение об ошибке, которое имеет формат:

Java Exception occured: <сообщение об ошибке (глобальное)>
class = <имя класса-исключения>
message = <сообщение об ошибке (локальное, определяемое классом-исключением)>

Приведем пример сообщения об ошибке.

Java exception occured: file not found
class = java.io.FileNotFoundException
message = docs\type_02\1111\r\j22-2pp.txt (Не удается найти указанный файл)

Здесь:

- имя класса-исключения — **java.io.FileNotFoundException**;
- сообщения о возникшей ошибке — **file not found, docs\type_02\1111\r\j22-2pp.txt** (не удается найти указанный файл).

Таким образом, ошибки, возникающие при работе с ПК АТТД, характеризуются их типами (классами-исключениями) и возможными для них сообщениями.

Большинство ошибок возникает при:

- неправильной установке и конфигурации ПК АТТД;
- неправильном указании пути к файлам с входными данными;
- несоответствии входных данных требованиям корректности;
- невозможности нахождения кода по строковому параметру во входных таблицах и др.

Ниже приведен перечень "собственных" исключений ПК АТТД, который в данном контексте интересен с точки зрения анализа возможных отступлений от правил описания блоков информации, содержащихся в текстовых таблицах *Ежегодника*. Приведем некоторые из них:

- **attd.agromet.datatable.IncompatibleRowException**

"Number of columns not match to assumed" — количество колонок в одном из рядов *XML*-файла данных не совместимо со структурой, описанной в файле метаданных (МТ-файле);

- **attd.agromet.datatable.ParseTableException**

"number of leaf nodes = N_1 differs from assumed = N_2" — количество колонок N_1 в файле с текстовой таблицей, определенное по последней строке заголовка, не совпадает с ожидаемым N_2, которое указано МТ-файлом;

"cannot parse row # N = 'S'" — рядок ***S*** под номером ***N*** в клеточной части текстовой таблицы не удается проанализировать — разбиение на колонки произведено неудачно и для него не найдено подходящей секции;

- **attd.agromet.datatable.ValueNotFoundException**

"Searching value [S] failed, query:Q" — не удается получить код по строковому параметру ***S*** в *XML*-файле данных из БД ***dictDBName***, где вместо ***Q*** присутствует *SQL*-запрос, возвращающий пустое отношение, т.е. значение ***S*** не найдено в справочной БД;

"Searching code [C] failed, query: Q" — по коду ***C*** не удалось получить из БД ***dictDBName*** строковый параметр (на этапе получения данных из БД);

- **attd.agromet.dbi.InsertException**

"Problems with inserting data to DB SQL = S" — возникает на этапе внесения данных в БД в случае наличия ошибки в исходных данных, которая не была вы-

явлена на этапе тестирования; необходимо проверить файл с исходными данными;

- **attd.agromet.dbi.FieldNotFoundException**

"table: TName; field: FName" — поле под именем *FName* в таблице ***TName***, указанное в файле метаданных, не найдено в БД;

- **attd.agromet.dbi.TableNotFoundException**

"table: TName" — таблица под именем ***TName***, указанная в файле метаданных, не найдена в БД;

- **attd.agromet.metatable.DestinationException**

"без сообщения" — в файле метаданных найдено пустое значение одного из атрибутов ***tableTo, fieldTo***;

- **attd.agromet.metatable.SectionFormatException**

"без сообщения" — в файле метаданных найдена хотя бы одна секция с пустыми значениями атрибутов ***fmt, beg***;

- **attd.agromet.metatable.SectionParsingException**

"sections heading not found" — в файле с текстовой таблицей количество строк в клеточной части меньше количества секций в файле метаданных;

"parsing header section failed: 'S' — строку ***S***, не удалось распознать как одну из открывающих клеточную часть секций;

"row #N: section parsing failed: 'S'" — строку ***S***, являющуюся значением элемента "секция" в *XML*-файле данных, не удалось распознать, использовав шаблоны (***N*** — порядковый номер рядка в клеточной части текстовой таблицы, пустые рядки игнорируются);

- **attd.agromet.parser.TableNotFoundException**

"cannot find text tables" — в анализируемом файле не удалось обнаружить текстовую таблицу;

- **attd.agromet.util.FormatterException**

"row #N: date conversion failed: DS" — не удалось определить дату из строки ***DS*** с помощью указанного в файле глобальных опций шаблона (***N*** — порядковый номер рядка в клеточной части текстовой таблицы, пустые рядки игнорируются);

"row #N: cannot parse date by any pattern: DS" — ни один из шаблонов даты не подходит к строке ***DS*** (***N*** — порядковый номер ряда в клеточной части текстовой таблицы, пустые рядки игнорируются);

"cannot parse date-time from sql: DATE" — на этапе получения данных из БД не удалось осуществить преобразование даты ***DATE*** из формата СУБД в выходной формат;

- **attd.agromet.util.PropertiesException**

"Unrecognized syntax while parsing properties file 'FN' at line N" — ошибка в синтаксисе файла глобальных опций ***FN*** в строке ***N***;

"some required values not filled in settings file" — в файле глобальных опций не задано значение параметра, для которого это недопустимо;

- **attd.agromet.xmldb.DException**

"Unable to build sql statement: non-conflicting order of JOINS not exists" — для указанного файла метаданных и текущей структуры БД не найдено возможности построить запрос для получения данных из БД;

- **attd.agromet.xmltdata.XmlTException**

"number of items in rows differs in XML dataTable" — количество колонок таблицы в *XML*-файле данных разное для некоторых строк таблицы (таблица не является прямоугольной).

Выводы

Таким образом, в основу построенной технологии импорта табличных данных в реляционные БД положена независимость от структуры входной информации. Это достигается построением ее метаописания, которое используется для автоматического преобразования входа в промежуточную структуру данных с последующим переносом информации в БД.

Разработанная технология реализована в виде библиотеки классов *Java*. Базовым форматом обмена информацией является *XML*. Это наряду с открытостью программных интерфейсов обеспечивает эффективное распределение ролей в процессах создания и эксплуатации механизма внесения данных. Можно также говорить о независимости кода от СУБД, поскольку все обращения к серверу БД осуществляются через интерфейсы стандарта *JDBC*.

Потребность в такой технологии была обусловлена высокими затратами на разработку механизма актуализа-

ции БД архивными данными при использовании традиционных подходов. Эти затраты обусловлены значительной неоднородностью в структурах входной информации, что и является основным критерием применения разработанной технологии.

Общность подхода к разделению данных, описанию их структур и правил импорта делает возможным использование разработки в качестве базовой технологии построения механизмов внесения информации в БД. Эффективное применение технологии с помощью разработанной библиотеки было продемонстрировано на примере автоматизации ввода архивных данных из гидрометеорологических *Ежегодников* в ИС "Банк гидрометеорологических данных".

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Концептуальная и физическая модели БД по агрометеорологии

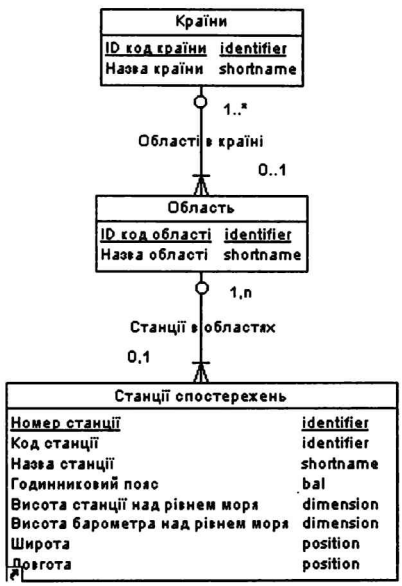


Рис. А.1. Сеть наблюдений. Географическое расположение метеостанций и пунктов наблюдений

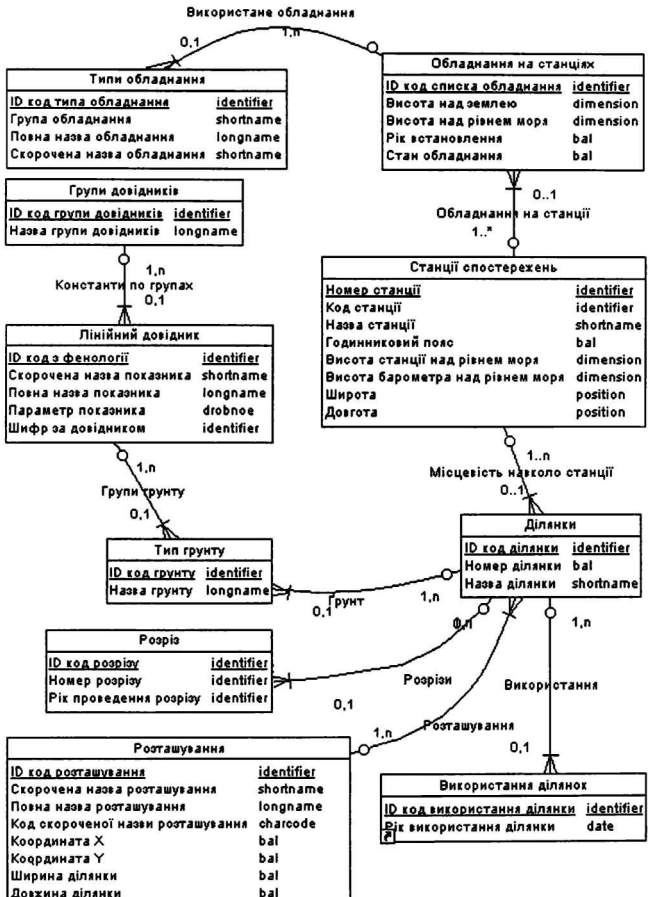


Рис. А.2. Информация о метеостанциях и постах наблюдения

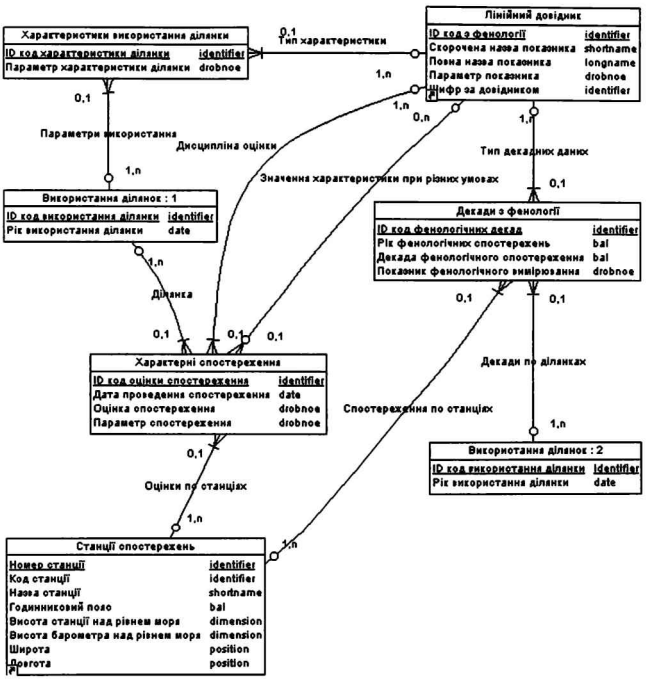


Рис. А.3. Агрометеорологические наблюдения за произростанием сельскохозяйственных культур

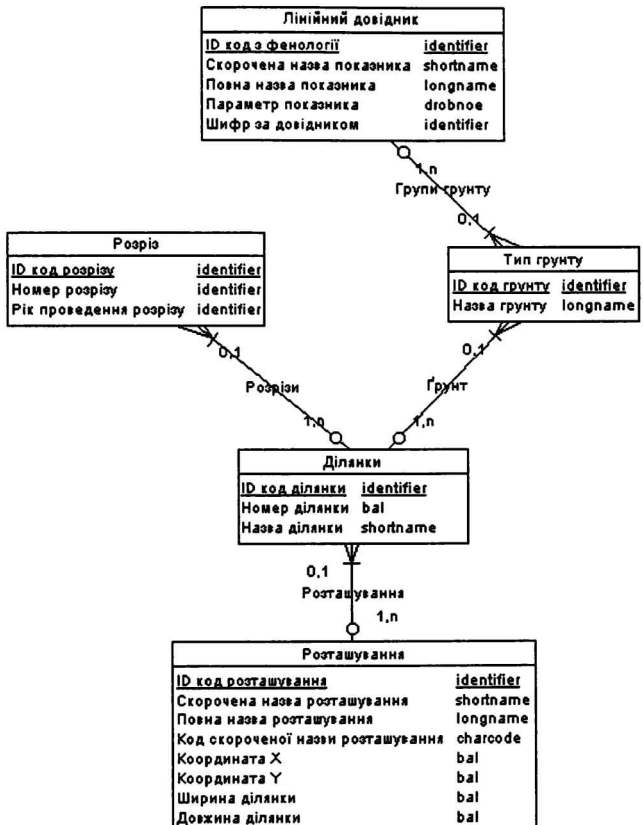


Рис. А.4. Характеристика грунта по участкам

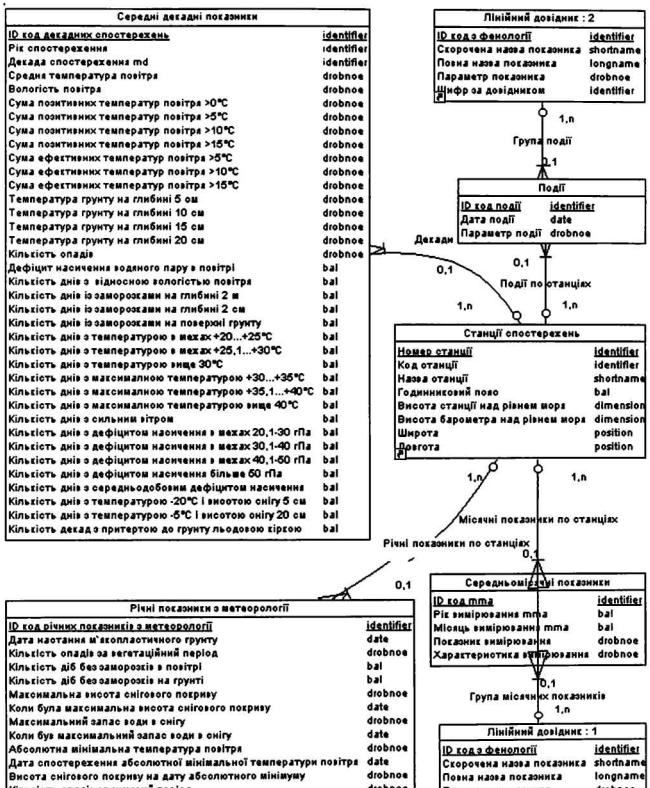
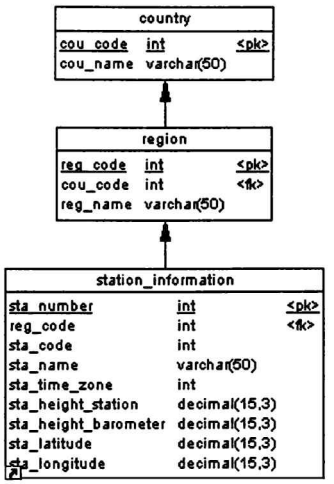


Рис. А.5. Сопутствующие метеорологические наблюдения

Physical Data Model

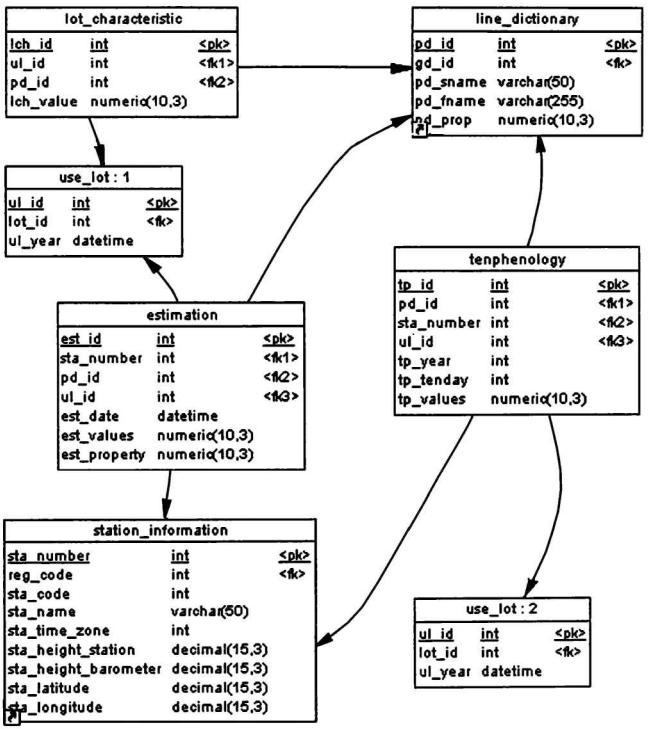
Package diagrams geographics
Diagram: Сеть наблюдений.



PowerDesigner

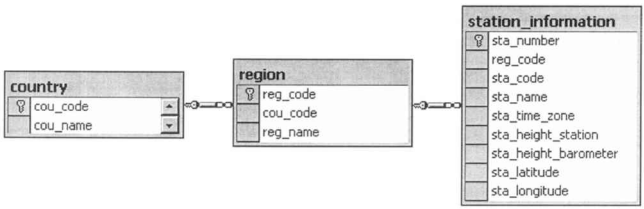
Physical Data Model

Package diagrams agro_directory
Diagram: Агрометеорологические наблюдения за произрастанием сельскохозяйственных культур



PowerDesigner

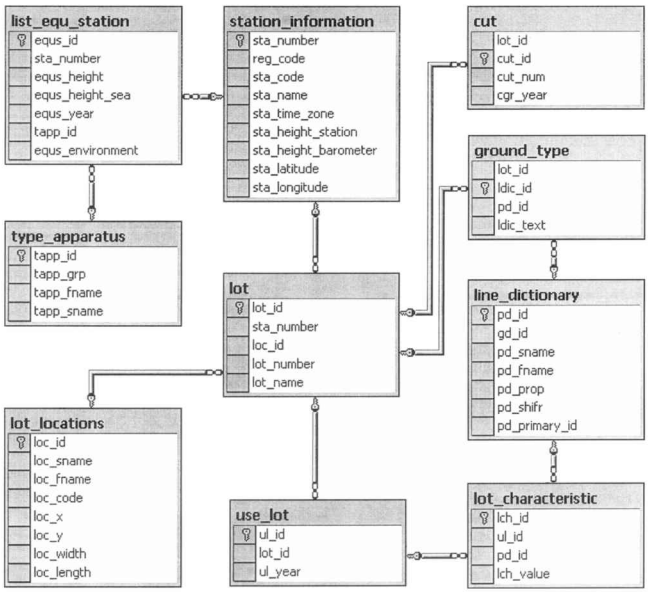
Package diagrams geographics
Diagram: Сеть наблюдений.
Географическое расположение метеостанций и пунктов наблюдений



PowerDesigner

Physical Data Model

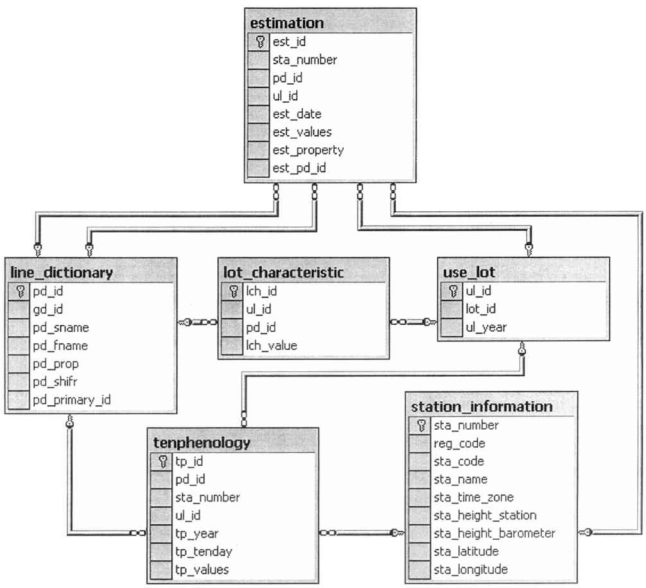
Diagram: Информация о метеостанциях и постах наблюдения



PowerDesigner

Physical Data Model

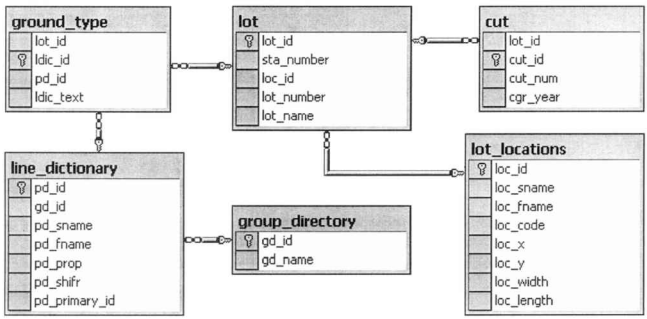
Package diagrams agro_directory
Diagram: Агрометеорологические наблюдения за произрастанием сельскохозяйственных культур



PowerDesigner

Physical Data Model

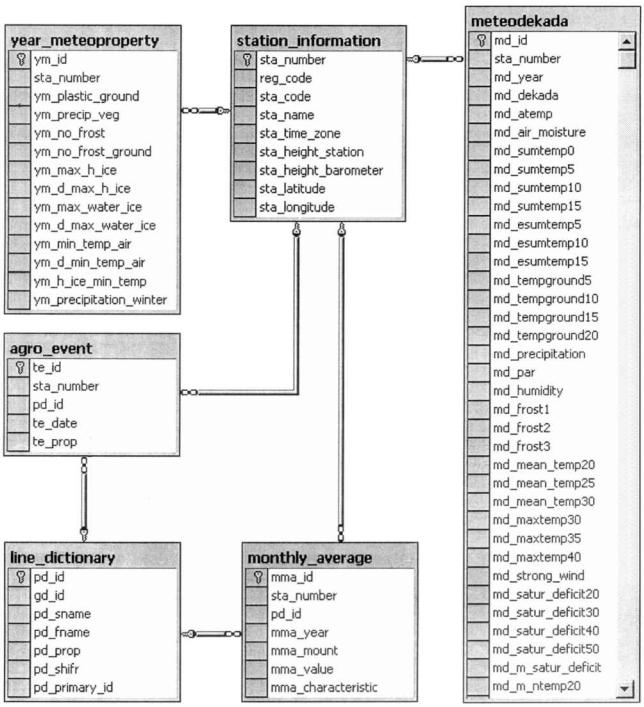
Package diagrams crop_control_group
Diagram: Характеристика грунта по участкам



PowerDesigner

Physical Data Model

Package diagrams meteo_group
Diagram: Сопутствующие метеорологические наблюдения



PowerDesigner

ПРИЛОЖЕНИЕ Б

XML-таблицы

Рассмотрим формат "*XML*-таблица" как подъязык *XML* и приведем описание элементов, атрибутов.

DTD-описание *XML*-таблицы (корневой элемент — ***data-root***):

```
<!ELEMENT data-root (s*, r+)*>
<!ATTLIST data-root
      head-ref CDATA #REQUIRED
      tableNo CDATA  #IMPLIED
>
<!ELEMENT r (i+)>
<!ELEMENT i (#PCDATA)>
<!ELEMENT s (#PCDATA)>
```

Элементы ***r*** соответствуют строкам таблицы, ***i*** — ячейкам, а ***s*** — секциям.

Обязательный атрибут ***head-ref*** содержит полный или относительный путь к MT-файлу, описывающему этот тип табличных данных.

Необязательный атрибут ***tableNo*** является номером таблицы. Он зарезервирован для таблиц агрометеорологических *Ежегодников* и имеет значение лишь при форматировании *XSL*-шаблоном.

ПРИЛОЖЕНИЕ В

MT-файлы

Приводим *DTD*-описание MT-файла.

```
<!ENTITY % typeAttT " (dictionary | metaname ) #REQUIRED ">
<!ENTITY % boolean " (true | false) #IMPLIED ">
<!ENTITY % string " CDATA #IMPLIED ">

<!ELEMENT metaTable (head-root?, head, section?)>
<!ELEMENT head-root (heading?, Node+)>
<!ATTLIST head-root
       maxlevel CDATA #REQUIRED
>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT Node (Node*)>
```

```
<!ATTLIST Node
        text CDATA #REQUIRED
>
<!ELEMENT head (child)+>
<!ELEMENT child (child)*>
<!ATTLIST child
      type %typeAttT;
      required %string;
      value %string;
      date %string;
      tableTo %string;
      fieldTo %string;
      dTable %string;
      dFieldC %string;
      dFieldV %string;
      dFieldT %string;
      dTypeV %string;
>
<!ELEMENT section (child+, section*)>
<!ATTLIST section
      fmt %string;
      beg %string;
>
```

Здесь принято следующее.
Допустимые элементы:

1) ***head;***
2) ***metaTable;***
3) ***child;***
4) ***section.***

Допустимые атрибуты:

1) ***head*** — отсутствуют;
2) ***metaTable*** — отсутствуют;
3) ***child: type, required, value, date, tableTo, fieldTo, dTable, dFieldC, dFieldV, dFieldT, dTypeV;***
4) ***section: fmt, beg.***

Заголовку таблицы отвечает элемент ***head.***
Корневой элемент: ***metaTable.***
Атрибуты элемента ***child*** можно представить в виде трех групп.
Первая группа (1) — *атрибуты общего назначения.*

1. ***type***: допустимы два значения данного атрибута:

а) ***metaname***: используется для указания того, что данный элемент ***child*** или соответствующий ему элемент таблицы используется исключительно в целях разметки структуры таблицы. Это указание сигнализирует программе, что полученные из таблицы данные должны вноситься в БД без изменения, т. е. "как есть";

б) ***dictionary***: индикатор словарного значения; в этом случае считываются атрибуты группы (3) и при внесении информации в БД делается попытка получить требуемый "код", тогда значение **::= код**.

2. ***required***: флаг (имеет значение лишь наличие или отсутствие данного атрибута, его значение игнорируется). Наличие атрибута тестируется только в случае, если данный элемент является *листком* на дереве ***child***. Если элементу соответствует колонка таблицы, то наличие атрибута влияет на заполнение пустых клеток: если он присутствует, то, встречая пустую клетку, программа проставит ее значением ближайшую "по колонке вверх" непустую, в противном случае (непустая клетка найдена не будет) реакция программы будет аналогичной отсутствию данного атрибута, т. е. значение **::= *NULL***.

3. ***value***: игнорируется, если элемент является *листком* на дереве ***child***, в противном случае его значение используется в качестве "значения" данного элемента, т. е. оно может быть внесено в базу данных, может быть словарным значением, и т.д. — к нему соответственно применяются все остальные атрибуты.

Вторая группа (2) — *атрибуты вставки в БД.*

1. ***tableTo*** — имя таблицы, в которую должно быть внесено "значение".

2. ***fieldTo*** — поле таблицы, в которую должно быть внесено "значение".

Эти атрибуты служат для формирования инструкций такого вида[1]:

INSERT INTO tableTo (fieldTo) VALUES ("значение").

---

[1] *INSERT* здесь приведен только для лучшего понимания назначения описываемых атрибутов.

Третья группа (3) — атрибуты получения "словарных" значений.

1. ***dTable***: таблица "словаря".
2. ***dFieldC***: поле — "код", ожидаемое значение.
3. ***dFieldV***: поле — "значение", содержит текущее значение.
4. ***dFieldT***: поле — "тип" словарного значения.
5. ***dTypeV***: значение поля "тип".

Атрибуты группы (3) служат для формирования запроса такого вида:

SELECT dFieldC FROM dTable WHERE (dFieldV = значение**) AND (dFieldT = dTypeV).**

Тогда в качестве "значения" для текущего элемента принимается значение ***dFieldC*** первого рядка результирующего отношения.

Атрибуты элемента ***section***:

- ***beg***: начальный символ секции; используется для выявления секций в анализируемом документе;
- ***fmt***: строка-шаблон секции; задает отображение меняющихся частей секций в элементы типа ***child***.

ОГЛАВЛЕНИЕ