Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерных технологий»

В. М. Лесин

IFTP-сервер

Проектная документация

Санкт-Петербург
2007

Оглавление

Введение

Цель проекта – разработка библиотеки промежуточного *FTP*-сервера[1] (*Intermediate FTP, IFTP*), предоставляющей доступ к структурированным сетевым ресурсам на базе протокола *FTP* [1-3]. *IFTP*-сервер позволяет задавать соответствие между рассредоточенными по сети файлами и удобной древовидной структурой. Это предоставляет пользователю возможность работать с файлами сети так, будто они расположены на одном и том же *FTP*-сервере, а при удобном *FTP*-клиенте – в одной папке. Библиотека, в первую очередь, ориентирована на работу с различными системами индексирования файлов, что позволяет создавать мощный инструмент структурирования данных в корпоративных и домашних локальных сетях.

При разработке библиотеки использовался подход с явным выделением состояний [4]. Стоит отметить, что в основном документе, описывающем *FTP*-протокол [2], также применяются диаграммы состояний, однако они носят общий характер и лишь иллюстрируют основные концепции работы с протоколом – порядок обработки последовательностей команд и кодов отклика. Использование данной технологии позволяет выделить поведение объекта управления, локализовав всю логику в одном или нескольких объектах-автоматах. Это помогло избежать многих типичных ошибок для подобных многопользовательских программ, связанных с непредвиденными состояниями системы.

Все программные компоненты были разработаны для операционных систем семейства *Windows NT*. *IFTP*-сервер представляет собой динамически подключаемую библиотеку (*Dynamic Link Library, DLL*), созданную с использованием технологии *COM* и *ATL* на языке *C++*. Вместе с библиотекой поставляется приложение, демонстрирующее ее работу.

Результаты работы опубликованы на сайте http://is.ifmo.ru в разделе «Проекты».

[1] FTP – File Transfer Protocol (протокол передачи файлов, англ.). Протокол был разработан в 1971 году. Позволяет работать со структурой файлов и папок на удаленном компьютере. До сих пор остается наиболее популярным протоколом, для этих целей.

Глава 1. Файлы общего доступа в локальных сетях

1.1. Организация данных в локальных сетях

В настоящее время большинство корпоративных и домашних компьютеров объединены в локальные сети. Такие сети имеют много достоинств, в том числе, они значительно снижают сетевой трафик к внешним серверам в интернете. Это позволяет, как уменьшить общую сетевую нагрузку на узлах связи, так и удешевить услуги интернет-провайдеров для конечных пользователей.

Многие пользователи локальных сетей предпочитают открывать доступ к своим файловым ресурсам (дистрибутивам, мультимедийным файлам) для других пользователей локальных сетей. Для этого существует несколько механизмов:

- встроенные в операционные системы средства поддержки общего доступа к файлам и папкам (например, сервис *File Sharing* в операционных системах семейства *Windows NT*);
- организация на своих компьютерах *FTP* серверов.

Второй способ оказывается предпочтительнее, поскольку обладает рядом преимуществ:

- не зависит от операционной системы, установленной у пользователей. *FTP*-клиенты существуют для всех наиболее используемых ОС и часто встроены в интернет-браузеры;
- предоставляет возможность гибкой настройки прав доступа для разных пользователей (вплоть до ограничения скорости доступа к данным);
- позволяет отслеживать историю обращений к ресурсам и т.д.

На практике в большинстве домашних локальных сетей совместный доступ к файлам осуществляется именно посредством *FTP*-серверов.

Помимо отличающихся механизмов доступа к данным существует два основных подхода к организации хранения данных в локальных сетях: *на выделенных серверах* и *на пользовательских компьютерах*.

Хранение данных на выделенных серверах. Часто такие сервера создают сами организаторы локальных сетей. При этом они обычно заботятся и об удобной системе навигации и поиска необходимых данных. К сожалению, такие системы требуют постоянной поддержки и обновлений и поэтому редко используются.

Распределенное хранение данных на пользовательских компьютерах. В этом случае каждый пользователь локальной сети сам решает, какую информацию он хочет сделать доступной для других. При этом он редко заботится об удобстве навигации и поиска по

своему ресурсу. Поиск также часто осложняется несоответствием имен файлов и их содержимого – это особенно типично для файлов, загруженных из интернета.

1.2. Индексация файлов

Работа с большим количеством файлов без возможности поиска крайне неудобна. Можно с большой долей вероятности утверждать, что в домашней локальной сети на 20000 компьютеров найдется хотя бы один компьютер, с которого можно скачать альбом группы “Queen”, но если нет службы поиска такого компьютера, то все механизмы обеспечения общего доступа к файлам оказываются бесполезными.

В решении проблемы поиска может помочь системы индексирования. При этом разные категории файлов стоит индексировать с разными критериями. Применительно к локальным сетям (особенно к домашним), наибольший интерес, по мнению автора, представляют индексаторы мультимедийных данных (такие данные часто составляют основной объем информации, доступной в локальных сетях). В настоящее время существует несколько проектов, которые строят базы данных, структурируя музыкальные файлы по артистам, альбомам и композициям (см., например, [5]).

Возникает вопрос: даже построив удобную для поиска базу данных, как получить доступ сразу к набору файлов (например, чтобы скопировать себе целиком альбом), ведь многие из них могут оказаться на разных компьютерах? Одно из решений – использовать веб-интерфейс (протокол *HTTP*), особенно учитывая то, что доступ к поисковой машине осуществляется обычно через него же. Однако у протокола HTTP нет поддержки для запроса сразу набора файлов. Использование специальных менеджеров закачки, таких как *ReGet* [6], также является неудовлетворительным решением, поскольку они есть лишь у малого числа пользователей. В данной работе для решения задачи доступа к группе файлов предлагается использовать клиент-серверное приложение на основе *FTP*-протокола.

1.3. Доступ к группе файлов

Основная идея сервиса *IFTP* заключается в отображении разрозненных музыкальных файлов в структурированный вид. Для этого можно использовать, например, иерархию, приведенную на рис. 1.

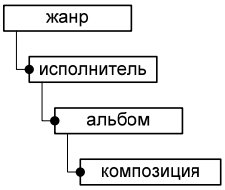

Рис. 1. Пример иерархии для хранения музыкальных файлов

Это позволяет отображать в дереве файлов и папок данные в том виде, в котором ими удобно пользоваться, а для каждого файла держать ссылку на его настоящее местонахождение (причем их может быть несколько). Это обеспечивает пользователю возможность получить доступ ко многим файлам с общим признаком (находящимся, возможно, на разных серверах) всего лишь одним *FTP* запросом с помощью своего *FTP*-клиента, который есть практически у всех пользователей компьютеров (например, в операционных системах *Windows*, *FTP*-клиент встроен в стандартный файловый менеджер *Explorer*).

Предлагаемый сервис позволяет не хранить копии проиндексированных файлов. Таким образом, система индексирования может даже и не предоставлять полную иерархию для отображения в *FTP*-клиенте, а строить «на лету» по результату поискового запроса лишь ту часть иерархии, которая требуется пользователю.

Одной из наиболее важных особенностей *IFTP*-модели является то, что при копировании файлов с удаленного сервера на компьютер пользователя, трафик направляется не через *IFTP*-сервер, а напрямую от удаленного сервера к пользователю (рис. 2).

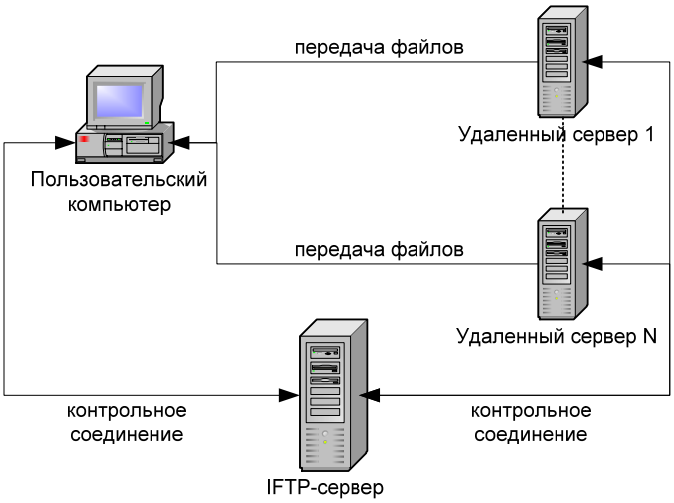

Рис. 2. Модель IFTP-сервера. Передача файлов идет напрямую от удаленного сервера к конечному пользователю, минуя IFTP- сервер

Процесс получения пользователем интересующих данных (в структурированном виде) представляет собой следующую последовательность действий.

1. Запрос поисковой системы, которая обеспечивает поиск в базе данных ресурсов интересующую пользователя информацию.
2. Получение пользователем *FTP*-ссылки на найденный ресурс.
3. Копирование с помощью своего *FTP*-клиента нужных данных по предоставленной ссылке.

Глава 2. Постановка задачи

Целью данного проекта является разработка сервера, соответствующего изложенной выше *IFTP*-модели. Этот сервер должен обеспечивать соответствие между структурированной файловой иерархией, подготовленной, например, индексатором, и реальным местонахождением файлов на различных *FTP*-серверах. *FTP*-клиенту, получившему доступ к *IFTP*-серверу, должно быть предоставлено дерево каталогов, соответствующее структурированной файловой иерархии. При обращении непосредственно к файлу клиент должен получать доступ к его настоящему местоположению.

Поскольку *IFTP*-сервер представляет собой библиотеку и не имеет собственного пользовательского интерфейса, вместе с ним должно быть разработано и демонстрационное приложение.

Глава 3. Модель FTP-протокола

Для того, чтобы описать модель *FTP*-протокола, введем ряд определений.

FTP-команды (*FTP-commands*) – последовательность команд, с помощью которой *FTP*-клиент осуществляет взаимодействие с *FTP*-сервером.

Код отклика (*reply code*) – последовательность символов, посылаемая *FTP*-сервером в ответ на команды со стороны клиента.

Интерпретатор протокола (*protocol interpreter*) – модуль, обеспечивающий интерпретацию *FTP*-команд и их параметров со стороны сервера и кодов отклика со стороны клиента.

Контрольное соединение (*control connection*) – *TCP* соединение, устанавливаемое между сервером и клиентом для обмена командами и кодами отклика.

Соединение для передачи данных (*data connection*) – *TCP* соединение, устанавливаемое между сервером и клиентом для передачи полезной информации (файлов или описания структуры файловой системы).

Процесс передачи данных (*data transfer process*) – модуль, осуществляющий передачу информации по соединению для передачи данных.

В представленной модели (рис. 3) клиентское приложение инициирует контрольное соединение с сервером.

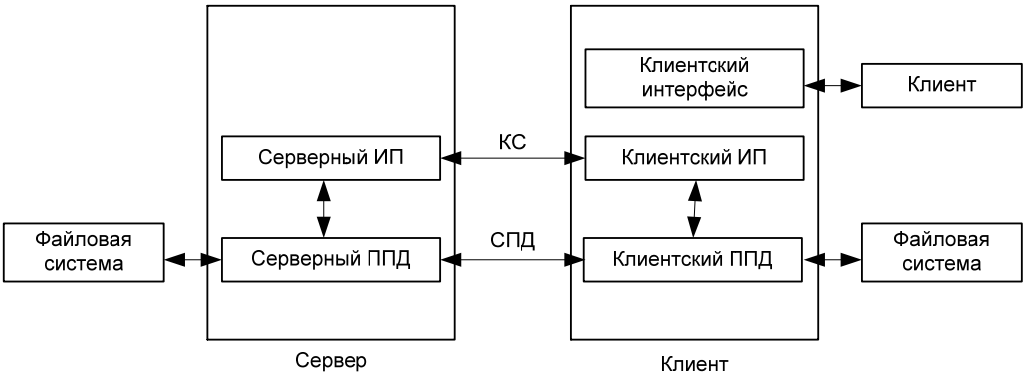

Рис. 3. Обобщенная модель FTP-протокола

Обычно сервер заранее регламентирует, с каким портом клиент должен устанавливать соединение. В противном случае, используется стандартный порт 21, заданный по умолчанию *FTP*-протоколом. Затем клиент передает информацию о пользователе (имя пользователя, пароль, при необходимости дополнительные параметры учетной записи). Если сервер принимает параметры пользователя, клиент передает настройки соединения для передачи данных (направление, адрес и режим передачи данных). После этих операций сервер и клиент готовы к обмену файлами. Соединение для передачи данных используется только для передачи самих файлов и описания текущего каталога.

Существует, также и другая модель соединения, использующая *FTP*-протокол (рис. 4).

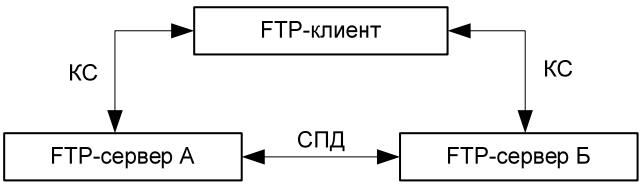

Рис. 4. Модель FXP

Такая модель называется FXP (File eXchange Protocol) [7]. Она используется для передачи файлов между двумя *FTP*-серверами напрямую, минуя компьютер клиента. В FXP-

модели, клиент устанавливает соединение с двумя разными *FTP*-серверами, запрашивая пакеты данных у первого сервера от имени второго. Преимущество этого метода в том, что клиент может перекачивать файлы по сети с большой скоростью между серверами, обладая лишь низкоскоростным доступом к самим серверам.

Глава 4. Команды FTP-протокола

Как уже упоминалось выше, все взаимодействие, связанное с управлением соединением между клиентом и сервером, осуществляется за счет обмена командами и кодами отклика. Существует большое количество *FTP*-команд, описанных как в стандарте *FTP*-протокола [3], так и во многих его расширениях. В стандарте также описан минимальный набор интерпретируемых сервером команд, достаточных для корректного функционирования сервера. *IFTP*-сервер полностью реализует минимальный набор команд, а также некоторые команды из расширенного набора.

4.1. FTP-команды

Все команды и коды отклика передаются по контрольному соединению в виде *Telnet*-строк (по протоколу *Telnet* в кодировке *ASCII*). Каждая команда начинается со своего кода (символьная последовательность не более четырех символов), затем через пробелы следуют параметры команды. Любая командная строка должна заканчиваться символом *<CLRF>* (возврат каретки, перевод строки), и интерпретатор протокола не должен начинать анализировать команду до получения этого символа[1].

Команды не чувствительны к регистру, поэтому строки *RETR*, *Retr* и *retr* будут восприниматься одинаково. Это касается и параметров команд, задающих режим передачи данных.

Далее приводятся описания команд, используемых в предлагаемом в работе *IFTP*-сервере. Команды описываются с использованием их стандартных названий, в скобках указаны символьные коды.

[1] Необходимость в символе, обозначающем конец команды, вызвана потоковой моделью (нет логического разбиения на пакеты) *TCP*-протокола, на основе которого создается контрольное соединение.

4.1.1. Команды управления доступом

USER NAME (USER). Эта команда задает имя учетной записи для доступа к файловой структуре сервера. Параметром команды является само имя учетной записи. Обычно это первая команда, которую посылает клиент серверу. В зависимости от политики управления доступом, серверу может быть достаточно имени учетной записи для аутентификации. В противном случае, он может запросить пароль.

PASSWORD (PASS). Эта команда задает пароль для текущей учетной записи, аргумент – строка с паролем. Пароль в общем случае передается без шифрования. Стандарт лишь рекомендует не отображать строку пароля при вводе в клиентском приложении. В отличие от самих команд, строка с паролем может быть чувствительна к регистру. Команда посылается сразу после команды *USER* в случае, если серверу недостаточно имени учетной записи для аутентификации.

CHANGE WORKING DIRECTORY (CWD). Эта команда изменяет рабочий каталог в дереве каталогов сервера. Параметром команды выступает полный или относительный путь к каталогу.

REINITIALIZE (REIN). Команда оканчивает работу пользователя, сбрасывая все его настройки. Если при получении команды, происходит передача пользовательских данных, сервер дожидается окончания передачи. Эта команда переводит сервер в то же состояние, которое идентично моменту сразу после создания контрольного соединения. Поэтому за этой командой обычно следует *USER*.

LOGOUT (QUIT). Эта команда оканчивает работу пользователя. Она во многом аналогично команде *REIN*, но после окончания текущей передачи данных, если таковая имеется, происходит разрыв контрольного соединения.

4.1.2. Команды, задающие параметры передачи данных

Эти команды могут следовать в любом порядке. Все параметры, используемые для организации передачи данных между сервером и клиентом, имеют значения по умолчанию. В стандарте рекомендуется использовать их только в случае отличий от умолчаний. На практике же большинство *FTP*-серверов не готовы к передаче данных, если прежде не была получена команды *PORT* или *PASV*, задающие направления и адреса для организации соединения. Аналогично многие клиенты перед любым запросом на загрузку себе файла высылают эти команды, иногда даже изменяя эти параметры.[1]

[1] Иногда клиентское приложение изменяет порт для соединения. Тем самым оно уменьшает вероятность отказа, вызванную тем, что операционная система не успела закрыть *сокет* для предыдущего порта.

DATA PORT (PORT). Команда задает адрес компьютера, с которым сервер будет организовывать соединения для передачи данных. Этот адрес является единственным параметром команды и имеет следующий формат:

```
"PORT h1,h2,h3,h4,p1,p2",
```

где *h1:h2:h3:h4* задают *IP*-адрес (*h1* – старший байт), а *p1 и p2* – соответственно старший и младший байты номера порта.

В *FTP*-протоколе предусмотрен также и так называемый пассивный режим, при котором клиент, а не сервер, устанавливает соединение для передачи данных. Он особенно удобен, если сервер не может определить адрес клиента (например, сервер находится во внешней сети по отношению к сегменту сети клиента). К сожалению, в *IFTP* это механизм не может быть использован, так как в пассивном режиме клиент получает лишь один адрес для организации соединения, а в модели *IFTP* (глава 3) серверов, на которых хранятся конечные файлы, может быть несколько.

REPRESENTATION TYPE (TYPE). Команда задает тип представления для передаваемого файла. Используется для совместимости различных операционных систем [3]. На практике большинство *FTP*-клиентов в настоящее время используют тип *Image* (*I*).

FILE STRUCTURE (STRU). Команда задает структуру пересылаемого файла. На практике обычно используется значение по умолчанию – *File* (*F*).

TRANSFER MODE (MODE). Команда задает режим передачи данных. Наиболее часто используемый – *Stream* (*S*). В таком режиме признаком окончания передачи файла является разрыв соединения для передачи данных.

4.1.3. Сервисные команды

ABORT (ABOR). Эта команда останавливает выполнение предыдущей сервисной команды. Если предыдущая команда была завершена к моменту получения команды *ABOR*, сервер не предпринимает никаких действий. Команда чаще всего используется для отмены передачи по соединению для передачи данных. В этом случае не только останавливается передача, но также и разрывается соединение для передачи данных.

Стоит отметить, что если происходит разрыв соединения для передачи данных, то выполнение соответствующей сервисной команды останавливается. Повторная передача данных начнется только по запросу клиента.

При разрыве же контрольного соединения сервер останавливает выполнение текущей команды, разрывает соединение для передачи данных (если оно установлено) и прекращает работу с данным пользователем.

LIST (LIST). Аргументом этой команды является путь к каталогу, а если он не указан, то команда применяется к текущему каталогу. Команда инициирует посылку содержимого каталога по соединению для передачи данных. К сожалению, не существует стандартного формата для передаваемого по этой команде списка. Поэтому клиентским приложениям приходится анализировать получаемый список и самостоятельно выбирать один из наиболее часто используемых.

RETRIEVE (*RETR*). Эта команда инициирует копирование файла, указанного в качестве своего аргумента, на компьютер клиента. Передача файла происходит по соединению для передачи данных.

PRINT WORKING DIRECTORY (*PWD*). Эта команда выдает пользователю имя текущего каталога. Строка с каталогом передается по контрольному соединению.

NOOP (*NOOP*). Это команда не вызывает никаких действий, кроме как ответного отклика об успешном ее выполнении. Может использоваться для проверки соединения с сервером.

4.1.4. Синтаксис команд

Приведем синтаксис используемых в *IFTP*-сервере команд в форме Бэкуса-Наура.

```
<command> ::=
      USER <SP> <username> <CRLF>         |
      PASS <SP> <password> <CRLF>         |
      CWD  <SP> <pathname> <CRLF>         |
      QUIT <CRLF>                         |
      REIN <CRLF>                         |
      PORT <SP> <host-port> <CRLF>        |
      TYPE <SP> <type-code> <CRLF>        |
      STRU <SP> <structure-code> <CRLF>   |
      MODE <SP> <mode-code> <CRLF>        |
      RETR <SP> <pathname> <CRLF>         |
      ABOR <CRLF>                         |
      PWD  <CRLF>                         |
      LIST [<SP> <pathname>] <CRLF>       |
      NOOP <CRLF>

<username>  ::= <string>
<password>  ::= <string>
<string>    ::= <char> | <char><string>
```

```
<char>      ::= любой символ таблицы ASCII кроме <CR> и <LF>
<byte-size> ::= <number>
<host-port> ::= <host-number>,<port-number>

<host-number>     ::= <number>,<number>,<number>,<number>
<port-number>     ::= <number>,<number>

<number>    ::= любое десятичное число от 1 до 255
<form-code> ::= N | T | C
<type-code> ::=
            A [<sp> <form-code>]
          | E [<sp> <form-code>]
          | I
          | L <sp> <byte-size>

<structure-code>  ::= F | R | P
<mode-code>       ::= S | B | C
<pathname>        ::= <string>
```

4.2. Коды отклика

Отправив команду серверу, клиент дожидается получения коды отклика. Механизм кодов отклика используется для подтверждения выполнения или невыполнения команды, а также для получения результата запросов в случае, если данные передаются не по соединению для передачи данных (например, команда *PWD*). Коды, как и команды, передаются *Telnet*-строкой и оканчиваются символом *<CLRF>*. Любой код отклика состоит из трех цифр и пробельного символа, за которым следует текстовое описание кода. Текстовые описания носят исключительно рекомендательный характер и могут отличаться от реализации к реализации (за исключением приведенного примера с командами вроде *PWD*, у которых текстовые описания суть результаты выполнения команд).

Также как и команд, существует большое количество кодов отклика, используемых в протоколе *FTP*. Всех их можно разделить по назначению на пять типов (отличаются по первой цифре).

1 – положительный промежуточный отклик.** Используется для того, чтобы сообщить, что сервер приступил к выполнению команды, но это процесс может занять некоторое время (например, передача файла). После такого кода всегда посылается код с результатом выполнения команды, но до этого момента посылать серверу другие команды запрещено.

2 – отклик о положительном завершении.** Сообщает, что операция была успешно завершена и можно посылать новую команду.

3 – положительный предварительный отклик.** Сообщает о том, что сервер приступил к выполнению команды и требуется дополнительная информация для продолжения работы. Используется обычно при обработке последовательности команд (например, если для регистрации пользователя требуется пароль, сервер ответит кодом 331 на команду *USER*, в противном же случае, кодом 230).

4 – отклик о временном отрицательном завершении.** Используется в случае, когда команда не может быть выполнена, но препятствующая причина лишь временна. При получении такого кода, повтор текущей последовательности команд с начала с теми же параметрами может привести к успеху.

5 – отклик об отрицательном завершении.** Свидетельствует о том, что команда не была успешно выполнена. Причина может быть в нарушении последовательности команд, синтаксической ошибке в команде или параметре и т.д.

Глава 5. Проектирование

Реализация *IFTP* использует подход с явным выделением состояний. В программе отделяется поведение объекта управления от входных воздействий, которые он может получить, и выходных действий, которые он может совершить. Поведение выделяется в один или несколько вложенных или одноуровневых автоматов, которые обычно содержатся в объекте управления.

5.1. Асинхронная обработка событий в автоматах

Основную работу по организации взаимодействия между объектами управления и автоматами в данной работе выполняют события. Они также используются для двустороннего взаимодействия вложенного и родительского автоматов. Использование запроса состояния вложенного автомата не отвечает парадигме инкапсуляции, поэтому применение механизма событий для этих целей оказывается предпочтительнее.

Стоит отметить, что использование механизма, в котором событие превращается в непосредственный вызов автомата в том же месте, где оно активируется, может привести

к рекурсии. В этом случае приходится обычно внимательно следить за тем, чтобы действия, вызванные обработкой события, сами не активировали другие события.

Существует также модель событий, использующая очередь для хранения событий, и цикл *"while-true"* для их извлечения и обработки. Обычно, такая модель негативно сказывается на быстродействии программы, поскольку цикл работает даже в случае, когда нет событий, ожидающих обработку.

В данной работе используется модель событий, основанная на оконных сообщениях (применяются пользовательские оконные сообщения ОС *Windows NT*). Такая модель также организует очередь необработанных событий, однако проверяет ее только в случае, если эта очередь не пуста (это обеспечивает ОС), и тем самым экономит процессорное время.

Описанный механизм асинхронных событий реализован в базовом автомате, поэтому его применение в производном автомате крайне просто. Единственное, что требует дополнительных действий – реализация диспетчера событий в родительском автомате для перенаправления событий вложенным автоматам в случае использования иерархии автоматов.

5.2. Базовый автомат

Для упрощения разработки произвольных автоматов в работе был реализован базовый автомат, в задачи которого входят:

- реализация механизма асинхронных сообщений;
- реализация механизма проверки условий перехода и выполнения выходных воздействий при смене состояния и обработке событий.

Любой автомат, использованный в настоящей работе, унаследован от базового и переопределяет две его функции:

- проверка условий перехода в новое состояние;
- выполнение выходных воздействий при переходе в новое состояние.

В случае если автоматы образуют иерархию, то родительские автоматы также перекрывают функцию диспетчеризации событий.

Листинг кода базового автомата на языке *C++* приведен в приложении 1, листинг всех остальных разработанных автоматов приведен в приложении 2.

5.2. Объекты управления, используемые в IFTP

В проекте используется около 40 классов, три из них являются объектами управления, а пять – автоматами, описывающими их поведение. С точки зрения организации кода, автоматы, отвечающие за поведение объектов управления, являются подобъектами этих объектов управления. В иерархии автоматов, вложенные автоматы являются подобъектами родительских автоматов.

Реплика удаленного сервера (*RemoteServer Object*). Этот объект отвечает за взаимодействие с удаленным сервером, на котором хранятся конечные файлы. Такая реплика при необходимости создается для каждого удаленного сервера на каждого активного пользователя. Таким образом, если пользователю потребовалось получить доступ к файлу, находящемуся на некотором удаленном сервере, то *объект управления активным пользователем* (см. ниже) создает для этого удаленного сервера соответствующую реплику, которая обеспечивает контрольное соединения с этим сервером. Реплика удаленного сервера представляет собой полноценный *FTP*-клиент с расширенными функциями ретрансляции кодов отклика соответствующим активным пользователям. За поведение этого объекта отвечает автомат *A5***.**

Объект управления активным пользователем (*UserSite Object*). Этот объект обеспечивает взаимодействие с *FTP*-клиентом активного пользователя и соответственно хранит в себе все параметры соединения, текущий каталог, обрабатывает пользовательские команды и формирует коды отклика. Он также обеспечивает хранение списка *реплик удаленных серверов*, необходимых пользователю. Логикой этого объекта управляет основной автомат *A2* и вложенные в него автоматы *A3* и *A4*.

Объект учета пользователей (*CentralPoint Object*). Основной объект программы. Именно этот объект создается при запросе пользователя на инициализацию *FTP*-сервера. Обеспечивает механизм подсоединения и удаления пользователей и хранит в себе список *объектов управления активными пользователями*. Логикой объекта управляет автомат *A1*.

5.4. Интерфейс библиотеки

IFTP-сервер является библиотекой, и для его функционирования нужна программа, предоставляющая соответствие между структурированной файловой иерархией и реальными адресами на удаленных *FTP*-серверах (описания такой программы приведено в главе 7). Для взаимодействия с таким приложением в библиотеке разработан набор *COM*-интерфейсов, описание которого приводится ниже.

5.4.1. Интерфейсы управления *FTP*-сервером

Листинг 1. Основной интерфейс IFTP-сервера

```
MIDL_INTERFACE( "86856E0A-2F98-4897-B1B8-4C4DE3E3F803" )
IFTPServer : IUnknown
{
   virtual void __stdcall StopServer   ( BOOL force = FALSE ) = 0 ;

   virtual BOOL __stdcall StartServer
      ( ILinksProvider* provider, ITracerManager* tracermng,
        short port = 21, DWORD ip = 0x0100007f ) = 0  ;
};
```

Интерфейс `IFTPServer` является основным интерфейсом для взаимодействия с *IFTP*-сервером и позволяет инициировать сервер и останавливать его работу. При инициализации серверу указываются адрес и порт для соединения, а также передаются интерфейсы на объект, предоставляющие информацию о местоположении файлов (`ILinksProvider`), и объект, обеспечивающий ведение лога (`ITracerManager`).

Листинг 2. Интерфейс IAuthentication

```
MIDL_INTERFACE( "B9818B2C-E975-43db-A510-7A11DA2A6494" )
IAuthentication : IUnknown
{
   enum USER_PRIVILEGE
   {
      UP_NO_USER     = 0,
      UP_NEED_PASW   = 1,
      UP_NO_PASW     = 2
   };

   virtual USER_PRIVILEGE __stdcall CheckUser   ( string const& user )  = 0 ;
   virtual BOOL           __stdcall CorrectPassw( string const& user,
   string const& passw ) = 0 ;
};
```

Интерфейс `IAuthentication` обеспечивает доступ к информации об учетных записях пользователей *IFTP*-сервера.

5.4.2. Интерфейсы для ведения логов

Далее приводятся интерфейсы, для организации системы логов.

Листинг 3. Интерфейсы для ведения логов

```
MIDL_INTERFACE( "4CA77ACA-5A6D-4482-9E36-6462D70947DC" )
ITracerManager : IUnknown
```

```
{
   virtual ITracerPtr __stdcall CreateTracer( LPCWSTR name ) = 0 ;
};

MIDL_INTERFACE( "8DE0520E-9545-4947-9ADD-09E0FAEF66EF" )
ITracer : IUnknown
{
   virtual void __stdcall Trace( LPCWSTR msg ) = 0 ;
};
```

Интерфейс `ITracerManager` позволяет создавать произвольный именованный поток для ведения лога, а интерфейс `ITracer` предоставляет доступ к этому потоку.

5.4.3. Интерфейсы для работы со структурой файловой иерархии

Листинг 4. Интерфейсы для работы со структурой файлов

```
MIDL_INTERFACE( "660408A1-4FC2-472f-9127-27F07A700348" )
ILink : IUnknown
{
   // имя каталога или файла без префикса родительского каталога
   virtual string const& __stdcall GetImageName  () = 0 ;
   // имя FTP-сервера в формате адрес:порт
   virtual string const& __stdcall GetFTPServer  () = 0 ;
   // полный путь к файлу на сервере
   virtual string const& __stdcall GetFTPLink    () = 0 ;
   virtual BOOL          __stdcall IsFolder      () = 0 ;
   virtual long          __stdcall GetSize       () = 0 ;
   virtual void          __stdcall GetTime
            ( short& year, short& month, short& date,
              short& hour, short& minute ) = 0 ;
};
_COM_SMARTPTR_TYPEDEF( ILink, __uuidof( ILink )) ;

MIDL_INTERFACE( "D268E288-E7EF-45a4-8EA4-DD803F2CA473" )
ILinksIterator : IUnknown
{
   // позволяет возобновить перебор элементов в каталоге
   virtual void      __stdcall Reset() = 0 ;
   // возвращает NULL если достигнут конец списка
   virtual ILinkPtr  __stdcall Next () = 0 ;
};
_COM_SMARTPTR_TYPEDEF( ILinksIterator, __uuidof( ILinksIterator )) ;

MIDL_INTERFACE( "411CCEA6-BCFC-4703-A0AE-6C31DB4DB738" )
ILinksProvider : IUnknown
{
   typedef string string ;

   // имя сервера без префикса "FTP:\\"
```

```
   virtual BOOL __stdcall GeIFTPAccount
    ( string const& FTPServer, LPWSTR user, LPWSTR pass, size_t& size ) = 0 ;

   // предоставляет итератор по указанному каталогу, возвращает NULL, если
   // каталог не найден
   virtual ILinksIteratorPtr __stdcall CreateItemIterator
                                        ( string const& folderName ) = 0 ;
   virtual ILinkPtr          __stdcall Search
                                        ( string const& fileName   ) = 0 ;
};
```

Интерфейс `ILinksProvider` предоставляет доступ к структурированной файловой иерархии, построенной, например, индексатором. В нем присутствуют методы для создания итераторов по содержимому каталога и возможность поиск элемента в иерархии. Он также предоставляет информацию об именах учетных записей и паролей для соединения с удаленными серверами.

Интерфейс `ILinksIterator` позволяет перебрать содержимое папки, а интерфейс `ILink` предоставляет полную информацию о каждом элементе в структурированной файловой иерархии, включая его положение в иерархии и реальный адрес на удаленном *FTP*-сервере.

Глава 6. Описание автоматов

6.1. Автомат A1. Управление объектом учета пользователей

6.1.1. Словесное описание

Автомат предназначен для управления поведением объекта учета пользователей (*CentralPoint object*). Он реализует логику присоединения новых и удаление отсоединившихся пользователей. Также предусмотрено остановка сервера с ожиданием окончания всех операций по пересылки файлов.

6.1.2. Схема связей

Схема связей автомата *A1* приведена на рис. 5.

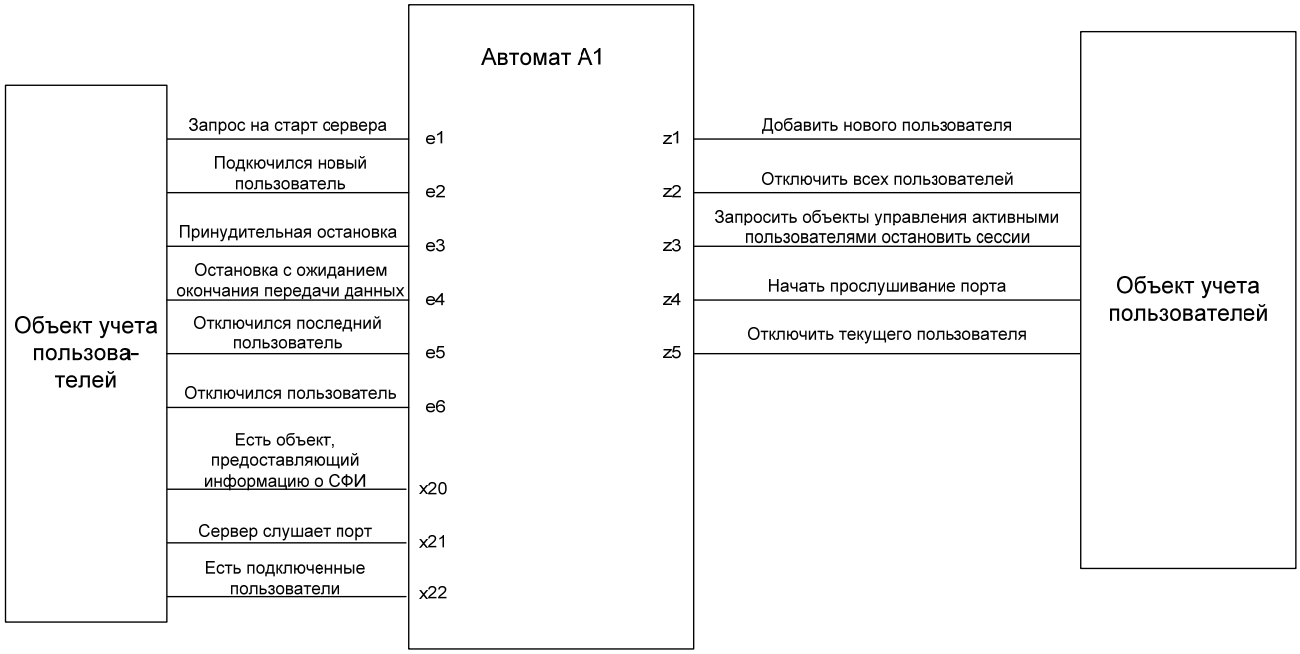

Рис 5. Схема связей автомата А1

6.1.3. Граф переходов

Граф переходов автомата *A1* приведена на рис. 6.

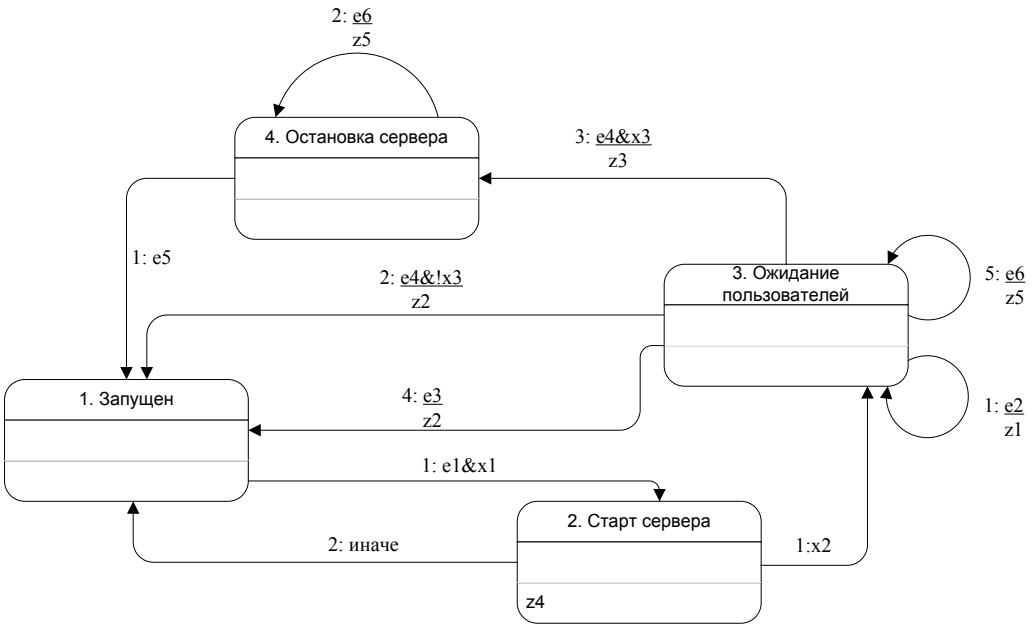

Рис. 6. Граф переходов автомата А1

6.2. Автомат A2. Управление активным пользователем

6.2.1. Словесное описание

Это основной автомат, определяющий поведение *объекта управления активным пользователем*. В его задачи входят:

- обработка пользовательских команд;
- организация передачи данных по соединению для передачи данных между *IFTP*-сервером и клиентом;
- организация передачи данных между удаленным *FTP*-сервером и клиентом.

Автомат содержит два вложенных автомата: *A3*, который отвечает за инициализацию пользователя и *A4*, который организует передачу данных по соединению для передачи данных.

6.2.2. Схема связей

Схема связей автомата *A2* приведена на рис. 7.

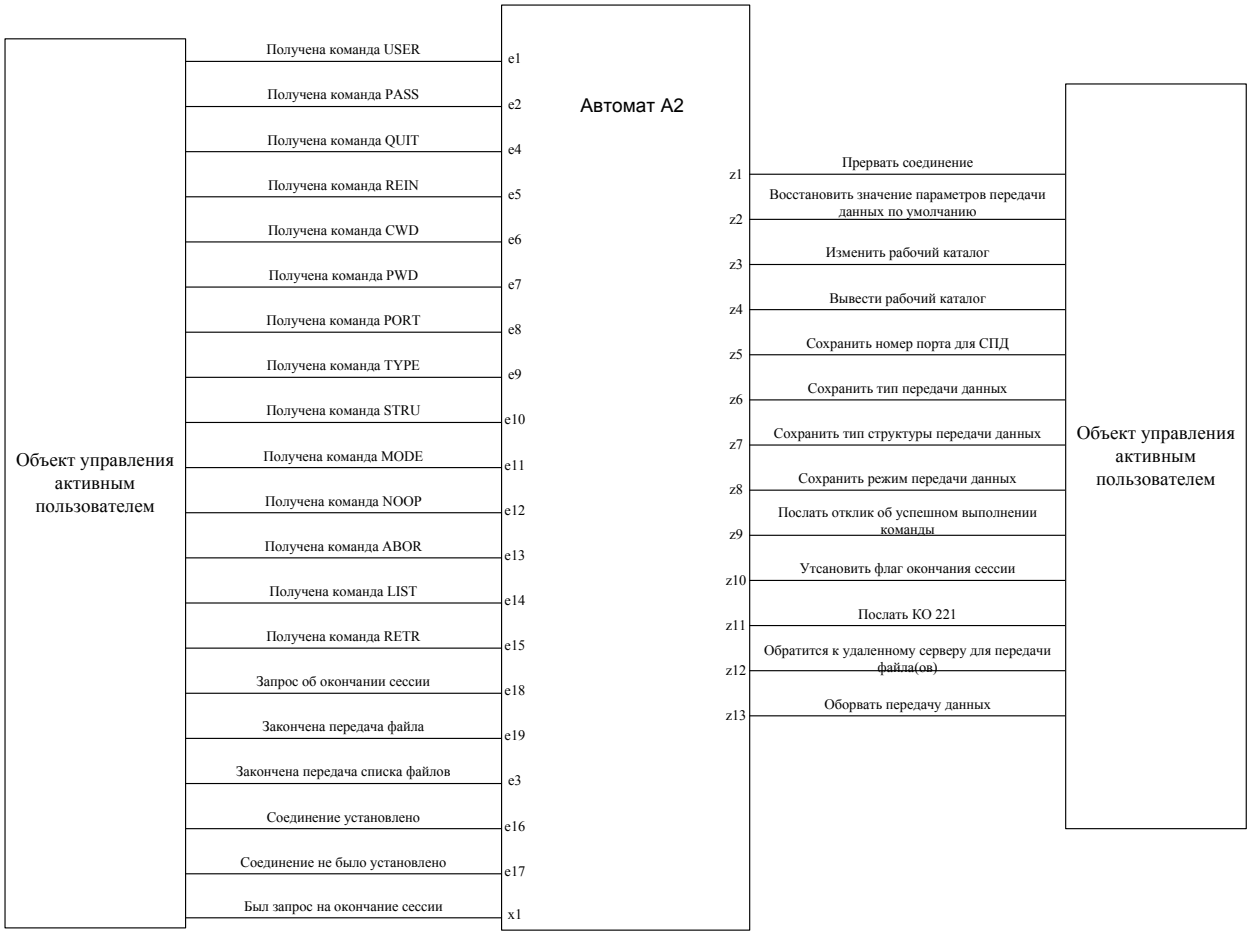

Рис. 7. Схема связей автомата A2

6.2.3. Граф переходов

Граф переходов автомата *A2* приведена на рис. 8.

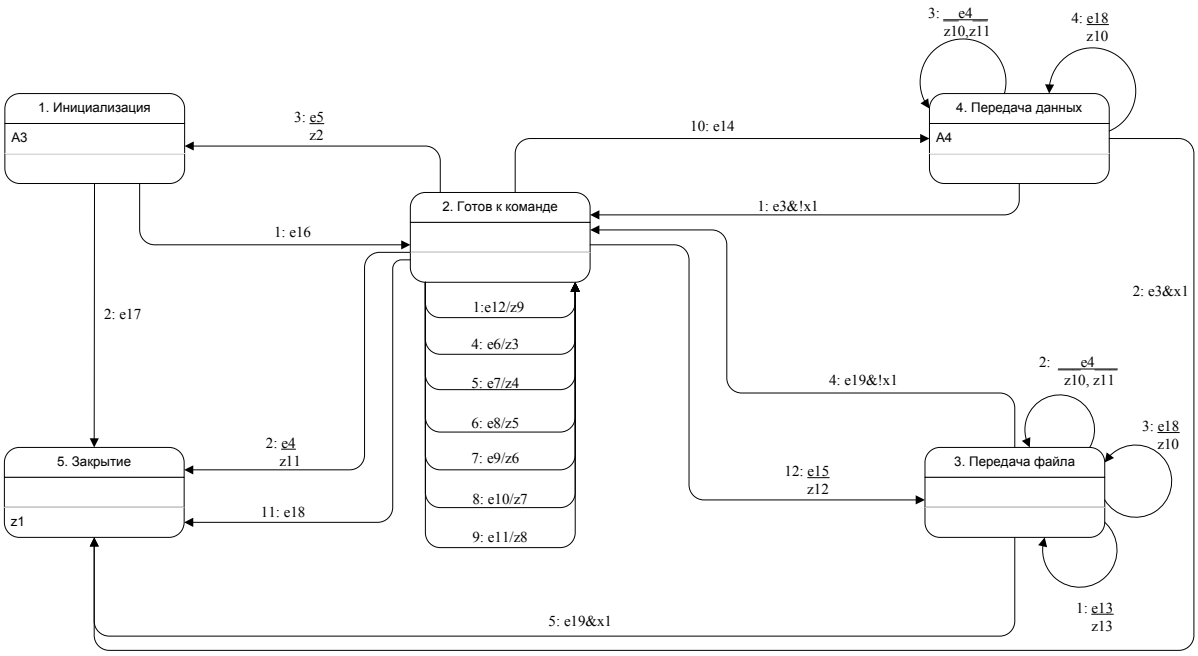

Рис. 8. Граф переходов автомата A2

6.3. Автомат A3. Инициализация пользователя

6.3.1. Словесное описание

Автомат используется для проверки подлинности учетной записи: имени и пароля пользователя, присоединяющегося к *IFTP*-серверу. Активируется в первом же состоянии своего родительского автомата *A2*. Параметры учетных записей, допустимых в *IFTP*-сервере предоставляются через интерфейс `IAuthentication`. Некоторые имена пользователей могут не предполагать наличие паролей (например, для получения доступа к серверу пользователю «anonymous» обычно не требуется пароль). В таких случаях пароль запрашиваться не будет, и аутентификация пользователя будет успешно завершена сразу после ввода соответствующего имени.

6.3.2. Схема связей

Схема связей автомата *A3* приведена на рис. 9.

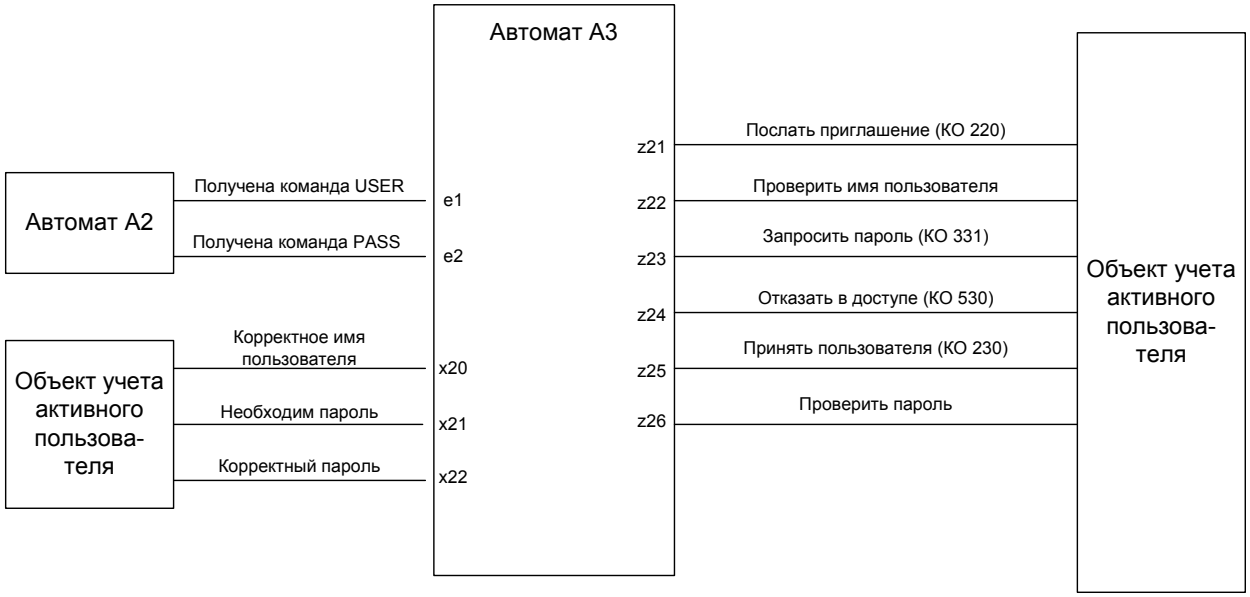

Рис. 9. Схема связей автомата А3

6.3.3. Граф переходов

Граф переходов автомата *A3* приведена на рис. 10.

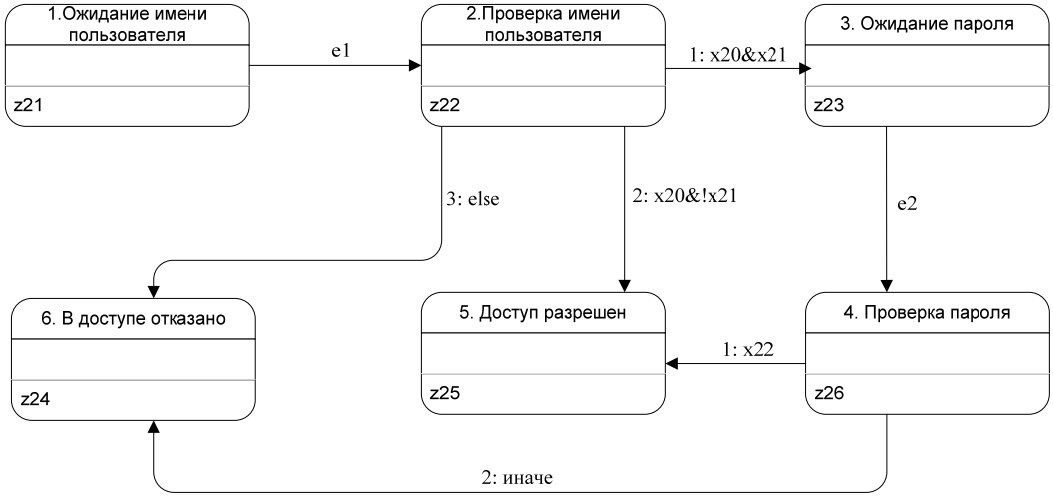

Рис. 10. Граф переходов А3

6.4. Автомат А4. Передача данных от *IFTP* сервера по СПД

6.4.1. Словесное описание

Автомат управляет процессом передачи данных по соединению для передачи данных между *IFTP*-сервером и клиентом. По соединению для передачи данных, как уже упоминалось ранее, согласно протоколу *FTP* передаются либо файлы, либо результаты запросов некоторых *FTP*-команд (для таких команд передавать результаты по контрольному соединению нецелесообразно в силу их большого размера). Передача файлов в *IFTP*-модели происходит между удаленным *FTP*-сервером и клиентом, минуя

IFTP-сервер, поэтому *IFTP*-серверу остается передавать по соединению для передачи данных лишь ответы на соответствующие команды (например, команды *LIST*). Автомат реализует все этапы передачи данных по соединению для передачи данных, предусмотренные протоколом: создание соединения, ожидание окончания передачи, корректное завершение соединения.

6.4.2. Схема связей

Схема связей автомата *A4* приведена на рис. 11.

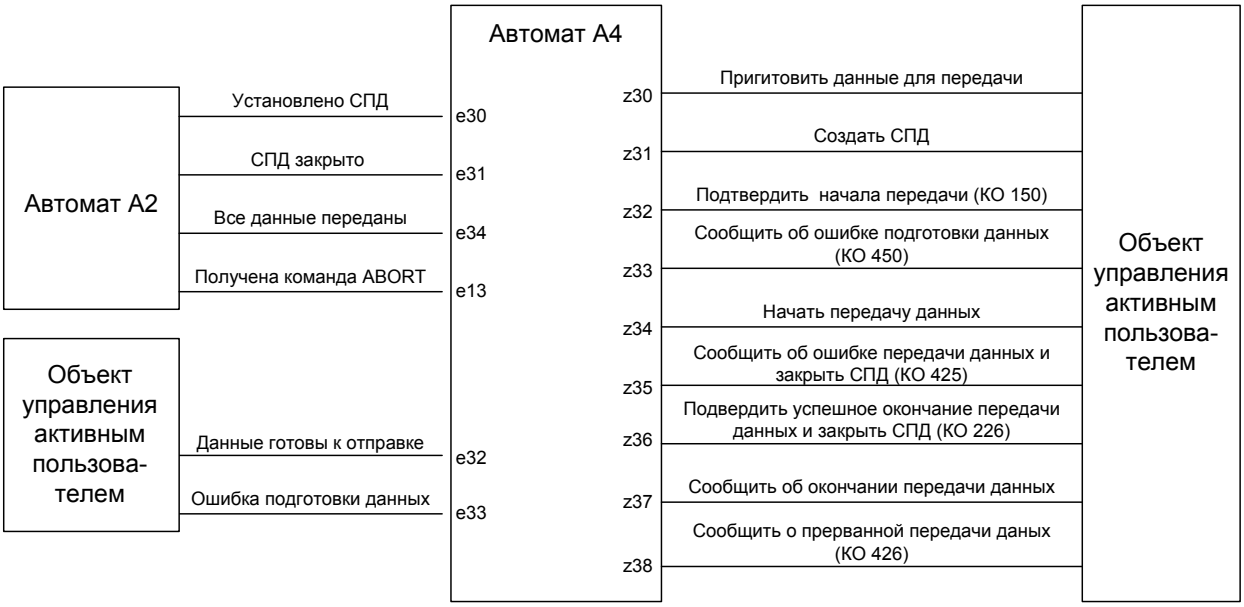

Рис. 11. Схема связей автомата A4

6.4.3. Граф переходов

Граф переходов автомата *A4* приведена на рис. 12.

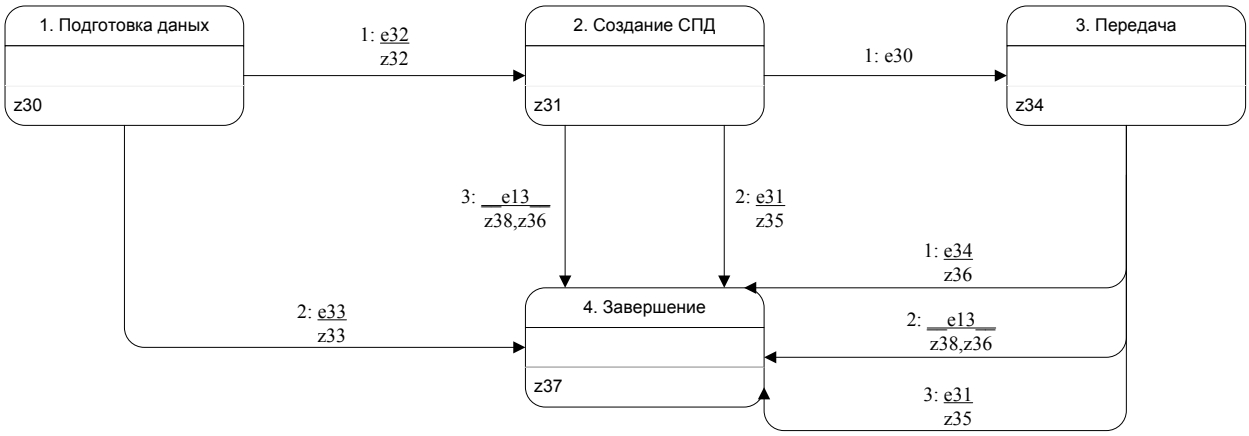

Рис. 12. Граф переходов автомата A4

6.5. Автомат A5. Соединение с удаленным *FTP*-сервером

6.5.1. Словесное описание

Автомат управляет объектом *реплика удаленного сервера*. Этот объект представляет собой *FTP*-клиент. Поэтому, в отличие от автомата *A2*, автомат *A5* обрабатывает не пользовательские команды, а коды откликов удаленного сервера.

В автомате предусмотрена возможность аутентификации на удаленном сервере. Если реплика уже была создана, то перед пересылкой файла аутентификации проводиться не будет, а посылаются лишь отличающиеся от текущих параметры организации соединения для передачи данных (например, номер *TCP*-порта).

6.5.2. Схема связей

Схема связей автомата *A5* приведена на рис. 13.

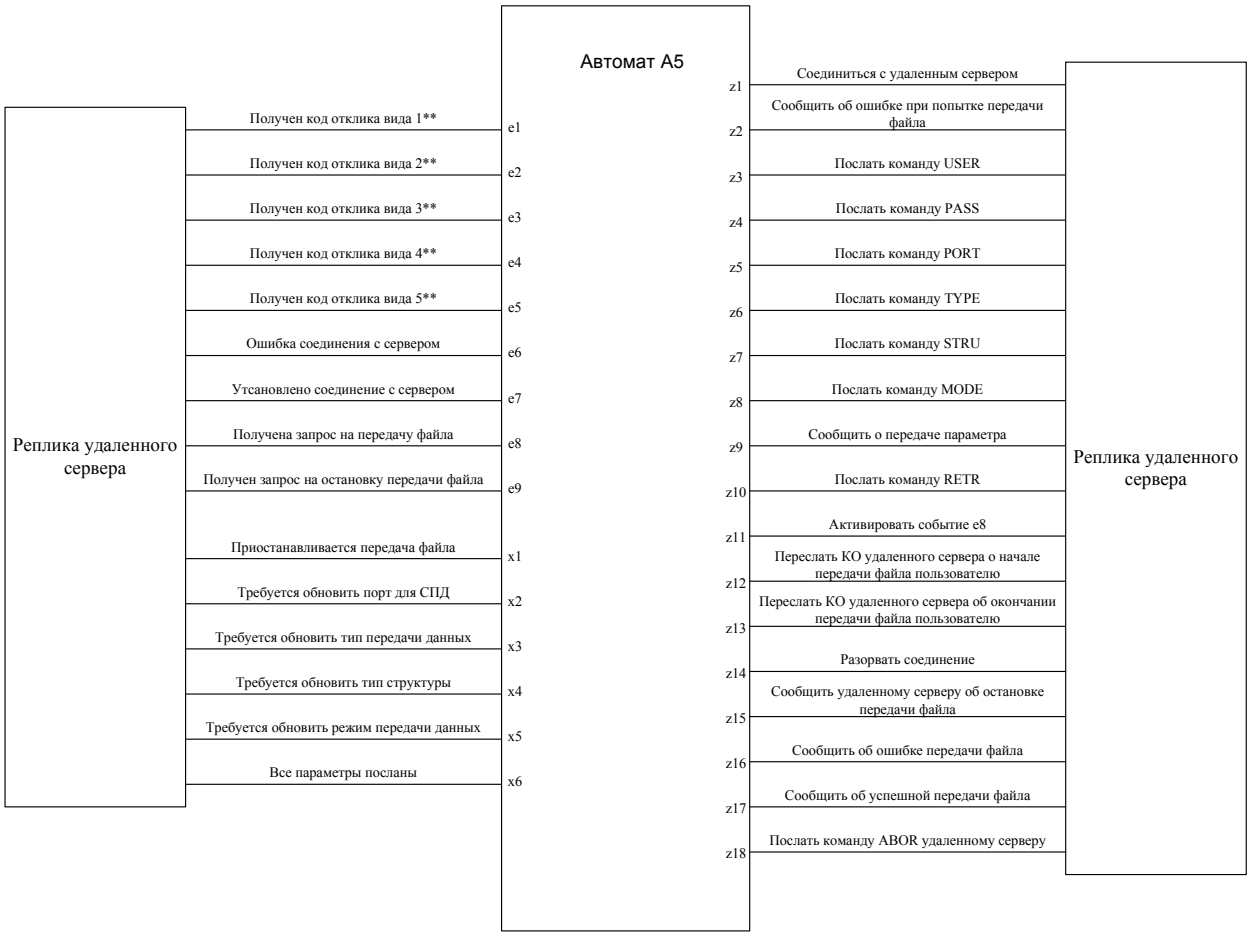

Рис. 13. Схема связей автомата A5

6.5.3. Граф переходов

Граф переходов автомата *A5* приведена на рис. 14.

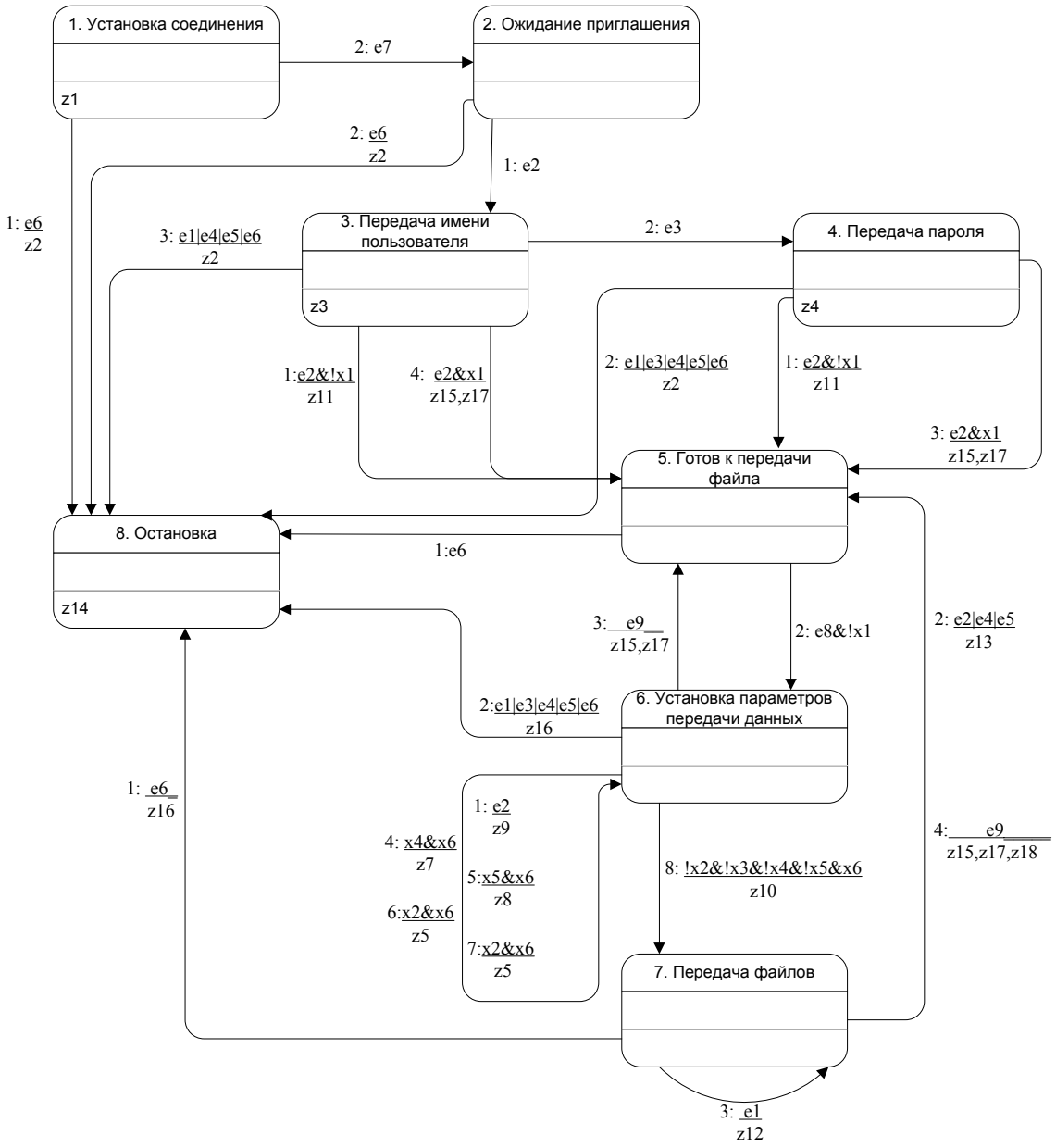

Рис. 14. Граф переходов автомата A5

Глава 7. Демонстрационное приложение

В рамках данного проекта была разработана еще одна программа – *LinksProvider*. Это приложение демонстрирует всю функциональность, предоставляемую библиотекой, и служит примером реализации интерфейсов, описанных в разделе 5.4. Программа представляет собой Windows-приложение с простым пользовательским интерфейсом, изображенным на рис.15.

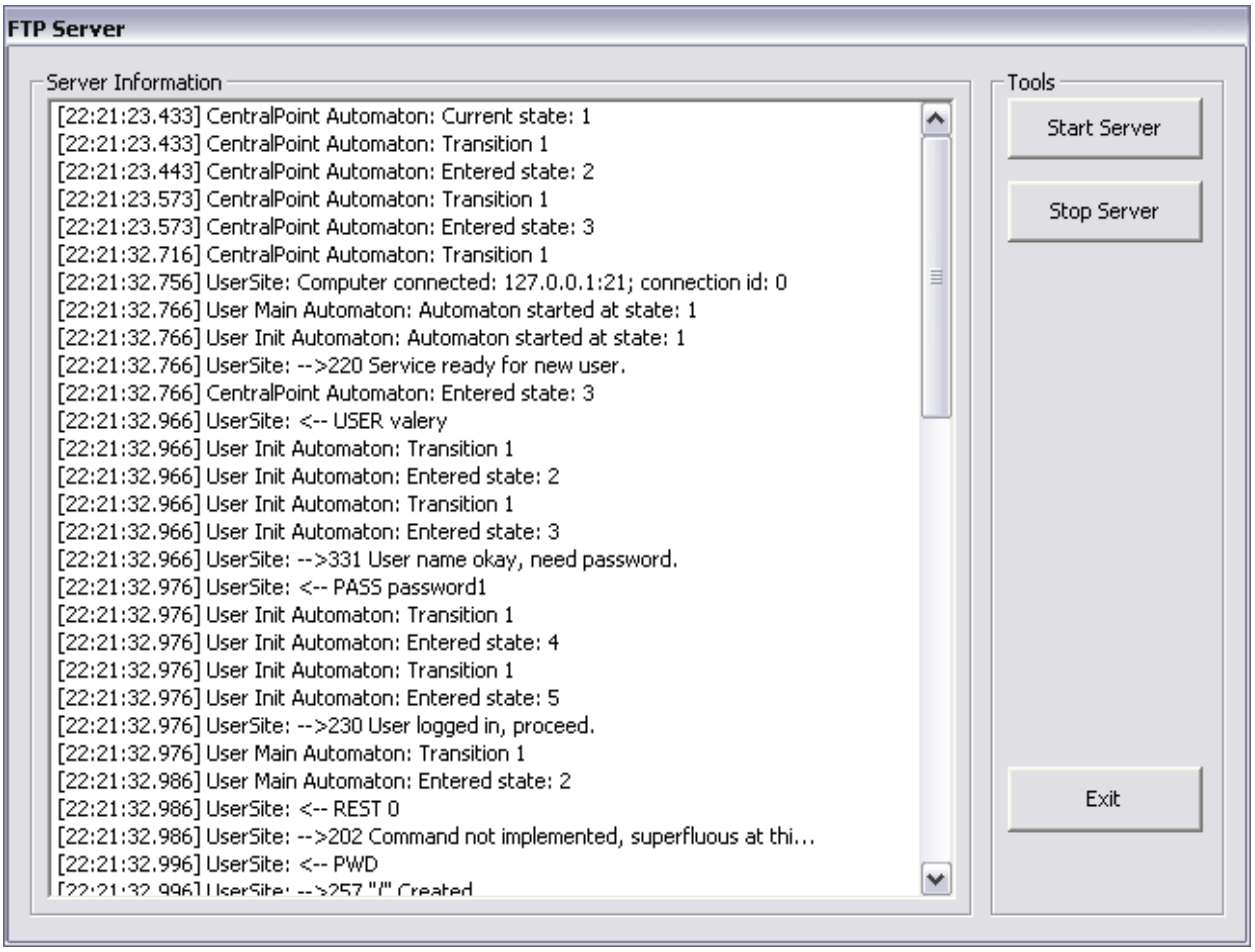

Рис. 15. Пользовательский интерфейс программы LinksProvider

Программа позволяет активировать и деактивировать сервер, а также отображает весь лог работы сервера.

Для самого *IFTP*-сервера *LinksProvider* предоставляет два сервиса.

1. Информация о структурированной файловой иерархии, используемая сервером.
2. Механизм протоколирования.

6.1. Информация о структурированной файловой иерархии

Информация о структуре файловой иерархии предоставляется через интерфейсы, описанные в пункте 5.4.3., и, как уже указывалось ранее, описывает биекцию между структурированной файловой иерархией и реальными адресами файлов на удаленных *FTP*-серверах. Эта информация содержит:

- имя файла или каталога, которое предоставляется *IFTP*-сервером пользователю;
- полный путь (с указанием удаленного *FTP*-сервера) к файлу;
- информацию о дате и размере реального файла;
- параметры учетных записей для удаленных серверов.

Вся информация о структуре файловой иерархии задается с помощью файлов конфигурации `Config\files.cfg, Config\servers.cfg`. Дополнительно указывается конфигурационный файл `Config\users.cfg` для задания учетных записей самого *IFTP*-сервера. Все пути к файлам указаны относительно положения программы *LinksProvider*.

6.2. Механизм протоколирования

В разделе 5.4.2. описаны интерфейсы, с помощью которых серверу предоставляется механизм ведения логов. Для того чтобы вести лог, вначале необходимо получить указатель на объект, управляющий потоком логов (указатель на `ITracer`). Такой объект можно создать с помощью функции `CreateTracer` интерфейса `ITracerManager` с указанием имени потока или получить указатель на уже существующий объект. Затем достаточно лишь записать строку лога в поток с помощью функции `Trace` интерфейса `ITracer`.

Для каждого потока программа *LinksProvider* создает отдельный файл в каталоге *Log* (имя потока логов становится именем файла, поэтому для него действительны все ограничения, накладываемые на имена файлов в ОС *Windows*). Это позволяет разграничивать протоколы от разных сессий *IFTP*-сервера. Пример протокола работы сервера приведен в приложении 3.

6.3. Запуск демонстрационного приложения

Для запуска демонстрационного приложения необходимо распаковать архив с бинарными файлами и запустить исполняемый файл `run.cmd`. Этот файл зарегистрирует *COM*-библиотеку сервера (для этого необходимо иметь права администратора на

компьютере) и запустит тестовое приложение *LinksProvider*. Файлы конфигурации настроены на соединение с серверами, находящимися в глобальной сети, поэтому, как уже упоминалось выше в разделе 4.1.2, компьютер пользователя должен иметь глобальный *IP*-адрес. После запуска программы и необходимо нажать на кнопку «StartServer» и тем самым активировать сервер на локальном компьютере (*IP*-адрес: 127.0.0.1, порт 21). После этих действий можно, используя любой удобный *FTP*-клиент (например, *FAR*), обратиться к локальному *IFTP*-серверу.

Заключение

Данная проектная документация демонстрирует детали разработки библиотеки *IFTP*-сервера. Она также может быть использована как руководство для написания программ, использующих эту библиотеку. Стоит отметить, что использование подхода с явным выделением состояний позволило во многом упростить разработку программы, избавив ее от непредсказуемых состояний, не предполагаемых ни автором, ни протоколом *FTP*.

Разработанный проект является частью существующей системы индексирования и поиска музыкальных файлов, которая в ближайшее время будет запущена в домашней сети Matrix города Санкт-Петербург.

Источники

1. Berstein D. J. *FTP*: File Transfer Protocol. http://cr.yp.to/*FTP*.html
2. Postel J., Reynolds J. RFC 959 – File Transfer Protocol. October 1985. http://rfc.net/rfc959.html
3. Postel J. RFC 640 - Revised *FTP* Reply Codes. June 1974. http://rfc.net/rfc640.html
4. Шалыто А. А., Туккель Н. И. SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5. http://is.ifmo.ru/download/switch.pdf
5. http://musicbrainz.org/
6. http://www.reget.com/
7. FXP. Википедия. http://ru.wikipedia.org/wiki/FXP

Приложение 1. Исходный код базового автомата

```
////////////////////////////////////////////////////////////////////////
//
// BaseAutomaton.h
// Реализация класса BaseAutomaton
//
////////////////////////////////////////////////////////////////////////

#pragma once
#include <boost/optional.hpp>
#include "trace.h"

// коды сообщений для механизма асинхронных событий и вызова
// выходных действий при смене автомата
#define WM_EVENT        WM_USER + 1
#define WM_CALCSTATE    WM_USER + 2

////////////////////////////////////////////////////////////////////////
//
// класс BaseAutomaton предоставляет механизм асинхронных событий,
// механизм вызова выходных действий при смене состояния автомата, а
// также механизм протоколирования
//

template< class State, class Event >
   struct BaseAutomaton
      : public    CWindowImpl< BaseAutomaton >
      , protected TracerImpl
{
public:
   void Init( ITracer* tracer )
   {
      TracerImpl::Init( tracer ) ;

      // создание окна
      if ( IsWindow())
         DestroyWindow() ;

      Create( NULL, rcDefault, NULL, WS_POPUP ) ;
      ShowWindow( FALSE ) ;

      // инициализация автомата
      state_.reset( initialState_ ) ;
      Trace( L"Automaton started at state: %d", initialState_ ) ;
      OnStateChanged() ;
   }

   void DumpState()
```

```
      {
         Trace( L"Current state: %d", initialState_ ) ;
      }

      BOOL ProcessEvent( Event event )
      {
         if ( IsInputEvent( event ))
         {
            PostMessage( WM_EVENT, event, 0 ) ;
            return TRUE ;
         }

         return FALSE ;
      }

   protected:
      State GetState() const { return *state_  ; }

   // оконные функции
   private:
      BEGIN_MSG_MAP( BaseAutomaton )
         MESSAGE_HANDLER( WM_EVENT     , OnEventReceived )
         MESSAGE_HANDLER( WM_CALCSTATE , OnCalcState     )
      END_MSG_MAP  ()

      LRESULT OnEventReceived( UINT, WPARAM wParam, LPARAM, BOOL& )
      {
         OnEvent( static_cast< Event > ( wParam )) ;
         return 0 ;
      }

      LRESULT OnCalcState( UINT, WPARAM, LPARAM, BOOL& )
      {
         CalcState() ;
         return 0 ;
      }


   // автоматные функции
   protected:

      typedef boost::optional< State > OptState ;
      typedef boost::optional< Event > OptEvent ;

      // переопределяются в наследнике
      virtual void CalcState     ( OptState& state ) = 0 ;
      virtual void OnStateChanged()                  = 0 ;
      virtual BOOL IsInputEvent  ( Event event ) { return TRUE ; }

   protected:
      BaseAutomaton( State initialState )
```

```
            : initialState_( initialState )
        {
        }

        ~BaseAutomaton()
        {
            if( IsWindow())
                DestroyWindow() ;
        }

        // функция позволяет проверить событие по номеру
        BOOL e( long eventNumber ) const
        {
            return event_ && ( *event_ == eventNumber ) ;
        }

        void CalcState()
        {
            OptState state ;

            CalcState( state ) ;
            event_.reset() ;

            if ( state )
            {
                state_.reset( *state ) ;
                Trace( L"Entered state: %d", *state_ ) ;
                OnStateChanged() ;

                PostMessage( WM_CALCSTATE, 0, 0 ) ; // to avoid recursion
            }
        }


    protected:
        // сообщение о событии
        virtual BOOL OnEvent( Event event )
        {
            event_.reset( event ) ;
            CalcState() ;

            return TRUE ;
        }

    protected:
        OptEvent event_        ;

    private:
        State    initialState_ ;
        OptState state_        ;
    };
```

Приложение 2. Исходные коды разработанных автоматов

```
////////////////////////////////////////////////////////////////////////
//
// CPAutomaton.h
// Объявление класса A1_Main, определение см. CPAutomaton.cpp
//
////////////////////////////////////////////////////////////////////////

#pragma once
#include "BaseAutomaton.h"

////////////////////////////////////////////////////////////////////////
//
// Central Point Automaton
//

class CCenrtalPoint ;

namespace central_point
{

namespace main_automaton
{

enum State
{
   S_1_STANDBY        = 1 , // начальное
   S_2_STARTING           ,
   S_3_READYFORUSERS      ,
   S_4_STOPPING           ,
} ;

} // main_automaton

enum Event
{
   E_1_START         = 1,
   E_2_NEWUSER          ,
   E_3_STOPFORCE        ,
   E_4_STOPNONFORCE     ,
   E_5_LASTDROPPED      ,
   E_6_USERDISCONNECTED
} ;

struct A1_Main
   : public BaseAutomaton< main_automaton::State, Event >
{
```

```
      typedef BaseAutomaton< main_automaton::State, Event > Base ;

      A1_Main ( CCentralPoint& cp ) ;
      LPCWSTR GetPrefix() const { return L"CentralPoint Automaton: " ; }

      void    SetTracer( ITracer* tracer ) { TracerImpl::Init( tracer ) ; }

   private:
      void CalcState     ( OptState& state ) ;
      void OnStateChanged()                  ;

   private:
      CCentralPoint& object_ ;
   };

   } // central_point



   ////////////////////////////////////////////////////////////////////////
   //
   // CPAutomaton.cpp
   // Определение A1_Main definition
   //
   ////////////////////////////////////////////////////////////////////////

   #include "stdafx.h"

   #include "CentralPoint.h"
   #include "CPAutomaton.h"

   namespace central_point
   {

   A1_Main::A1_Main( CCentralPoint& cp )
      : Base   ( main_automaton::S_1_STANDBY )
      , object_( cp )
   {
   }

   void A1_Main::CalcState( OptState& state )
   {
      using namespace main_automaton ;

      switch( GetState())
      {
      case S_1_STANDBY:

         if( e(1) && object_.x1())
         {
```

```
            Trace( L"Transition 1" ) ;
            state.reset( S_2_STARTING ) ;
         }

         break ;
      case S_2_STARTING:

         if ( object_.x2())
         {
            Trace( L"Transition 1" ) ;
            state.reset( S_3_READYFORUSERS ) ;
         }
         else
         {
            Trace( L"Transition 2" ) ;
            state.reset( S_1_STANDBY ) ;
         }

         break ;
      case S_3_READYFORUSERS:

         if( e( 2 ))
         {
            Trace( L"Transition 1" ) ;
            object_.z1() ;
            state.reset( S_3_READYFORUSERS ) ;
         }
         else if( e( 4 ) && !object_.x3())
         {
            Trace( L"Transition 2" ) ;
            object_.z2() ;
            state.reset( S_1_STANDBY ) ;
         }
         else if( e( 4 ) && object_.x3())
         {
            Trace( L"Transition 3" ) ;
            object_.z3() ;
            state.reset( S_4_STOPPING ) ;
         }
         else if( e( 3 ))
         {
            Trace( L"Transition 4" ) ;
            object_.z2() ;
            state.reset( S_1_STANDBY ) ;
         }
         else if ( e( 6 ))
         {
            Trace( L"Transition 5" ) ;
            object_.z5() ;
            state.reset( S_3_READYFORUSERS ) ;
         }
```

```
            break ;
         case S_4_STOPPING:

            if ( e( 5 ))
            {
               state.reset( S_1_STANDBY ) ;
            }
            else if ( e( 6 ))
            {
               Trace( L"Transition 2" ) ;
               object_.z5() ;
               state.reset( S_4_STOPPING ) ;
            }

            break ;
         }
      }

      void A1_Main::OnStateChanged()
      {
         using namespace main_automaton ;

         switch( GetState())
         {
         case S_2_STARTING:

            object_.z4() ;

            break ;
         }
      }

      } // namespace central point




      ////////////////////////////////////////////////////////////////////////
      //
      // UserAutomata.h
      // Объявление A2_MainAutomaton, A3_InitAutomaton и A4_DataTransfer,
      // определение см. UserAutomata.cpp
      //
      ////////////////////////////////////////////////////////////////////////

      #pragma once
      #include <boost/scoped_ptr.hpp>
      #include "BaseAutomaton.h"
```

```
//////////////////////////////////////////////////////////////////////////
//
// User site automata
//

class UserSite ;

namespace user_site
{

namespace main_automaton
{
   enum State
   {
      S_1_INITIALIZATION = 1,
      S_2_READY             ,
      S_3_FILETRANSFERING   ,
      S_4_DATATRANSFERING   ,
      S_5_CLOSING
   } ;
} // main_automaton

namespace init_automaton
{
   enum State
   {
      S_1_WAIT4USER = 1 ,
      S_2_CHECKUSER     ,
      S_3_WAIT4PASW     ,
      S_4_CHECKPASW     ,
      S_5_SUCCEEDED     ,
      S_6_FAILED
   };
} // init_automaton

namespace datatransf_automaton
{
   enum State
   {
      S_1_PREPAREDATA   = 1,
      S_2_CREATING_DC      ,
      S_3_SENDING          ,
      S_4_COMPLETED
   };
}

   enum Event
   {
      E_1_CMD_USER      = 1,
      E_2_CMD_PASS         ,
```

```
            E_3_TRANSFCOMPLETED  ,
            E_4_CMD_QUIT         ,
            E_5_CMD_REIN         ,
            E_6_CMD_CWD          ,
            E_7_CMD_PWD          ,
            E_8_CMD_PORT         ,
            E_9_CMD_TYPE         ,
            E_10_CMD_STRU        ,
            E_11_CMD_MODE        ,
            E_12_CMD_NOOP        ,
            E_13_CMD_ABOR        ,
            E_14_CMD_LIST        ,
            E_15_CMD_RETR        ,

            E_16_INITSUCC        ,
            E_17_INITFAILED      ,

            E_18_QUITREQUEST     ,
            E_19_RETRCOMPLETED   ,

            E_30_DC_OPENED  = 30 ,
            E_31_DC_ERROR        ,
            E_32_DATAREADY       ,
            E_33_DATAERROR       ,
            E_34_ALLTRANSFERED   ,
            E_35_CMD_ABORT
         } ;

      struct A2_MainAutomaton ;

      struct A3_InitAutomaton
         : public BaseAutomaton< init_automaton::State, Event >
         {
            typedef BaseAutomaton< init_automaton::State, Event > Base ;

            A3_InitAutomaton( UserSite& us ) ;
            LPCWSTR GetPrefix() const { return L"User Init Automaton: " ; }

         private:
            void CalcState     ( OptState& state ) ;
            void OnStateChanged() ;
            BOOL IsInputEvent  (Event event ) ;

         private:
            UserSite&         object_   ;
         };

      struct A4_DataTransfer
         : public BaseAutomaton< datatransf_automaton::State, Event >
      {
         typedef BaseAutomaton< datatransf_automaton::State, Event > Base ;
```

```
         A4_DataTransfer( UserSite& us ) ;
         LPCWSTR GetPrefix() const { return L"User Data Transfer Automaton: "
; }

      private:
         void CalcState     ( OptState& state ) ;
         void OnStateChanged() ;
         BOOL IsInputEvent  (Event event ) ;

      private:
         UserSite&         object_   ;
      };

      struct A2_MainAutomaton
         : public BaseAutomaton< main_automaton::State, Event >
      {
         typedef BaseAutomaton< main_automaton::State, Event > Base ;

         A2_MainAutomaton( UserSite& us ) ;
         LPCWSTR GetPrefix() const { return L"User Main Automaton: " ; }

         BOOL OnEvent    ( Event event )      ; // перенаправление событий
                                                  вложенных автоматов

      private:
         using Base::CalcState ;

      private:
         void CalcState     ( OptState& state ) ;
         void OnStateChanged() ;

      private:
         UserSite&        object_   ;

         // вложенные автоматы
         A3_InitAutomaton initAuto_ ;
         A4_DataTransfer  tranfAuto_;
      };

      } // namespace user_site
```

```
//////////////////////////////////////////////////////////////////////////
//
// UserAutomata.cpp
// Определение A2_MainAutomaton, A3_InitAutomaton и A4_DataTransfer
//
//////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "UserSite.h"
#include "UserAutomata.h"

namespace user_site
{

//////////////////////////////////////////////////////////////////////////
//
// Init Automaton
//

A3_InitAutomaton::A3_InitAutomaton( UserSite& us )
   : Base     ( init_automaton::S_1_WAIT4USER )
   , object_  ( us )
{
}

BOOL A3_InitAutomaton::IsInputEvent( Event event )
{
   return event >= E_1_CMD_USER && event <= E_2_CMD_PASS ;
}

void A3_InitAutomaton::CalcState( OptState& state )
{
   using namespace init_automaton ;

   switch( GetState())
   {
   case S_1_WAIT4USER:

      if( e(1))
      {
         state.reset( S_2_CHECKUSER ) ;
         Trace( L"Transition 1" ) ;
      }

      break ;
   case S_2_CHECKUSER:

      if( object_.x20() && object_.x21())
      {
         state.reset( S_3_WAIT4PASW ) ;
         Trace( L"Transition 1" ) ;
```

```
            }
            else if ( object_.x20() && !object_.x21() )
            {
               state.reset( S_5_SUCCEEDED ) ;
               Trace( L"Transition 2" ) ;
            }
            else
            {
               state.reset( S_6_FAILED ) ;
               Trace( L"Transition 3" ) ;
            }

            break ;
         case S_3_WAIT4PASW:

            if( e( 2 ))
            {
               state.reset( S_4_CHECKPASW ) ;
               Trace( L"Transition 1" ) ;
            }

            break ;
         case S_4_CHECKPASW:

            if( object_.x22())
            {
               state.reset( S_5_SUCCEEDED ) ;
               Trace( L"Transition 1" ) ;
            }
            else
            {
               state.reset( S_6_FAILED ) ;
               Trace( L"Transition 2" ) ;
            }

            break ;
         case S_5_SUCCEEDED:
            break ;
         case S_6_FAILED:
            break ;
         }
      }

      void A3_InitAutomaton::OnStateChanged()
      {
         using namespace init_automaton ;
         switch( GetState())
         {
         case S_1_WAIT4USER:

            object_.z21() ;
```

```
            break ;
         case S_2_CHECKUSER:

            object_.z22() ;

            break ;
         case S_3_WAIT4PASW:

            object_.z23() ;

            break ;
         case S_4_CHECKPASW:

            object_.z26() ;

            break ;
         case S_5_SUCCEEDED:

            object_.z25() ;

            break ;
         case S_6_FAILED:

            object_.z24() ;

            break ;
         }
      }

      ////////////////////////////////////////////////////////////////////////
      //
      // Data Transfer Automaton
      //

      A4_DataTransfer::A4_DataTransfer( UserSite& us  )
         : Base      ( datatransf_automaton::S_1_PREPAREDATA )
         , object_   ( us )
      {
      }

      BOOL A4_DataTransfer::IsInputEvent( Event event )
      {
         return event >= E_30_DC_OPENED && event <= E_34_ALLTRANSFERED ||
                event == E_13_CMD_ABOR ;
      }

      void A4_DataTransfer::CalcState( OptState& state )
      {
         using namespace datatransf_automaton ;
```

```
        switch( GetState())
        {
        case S_1_PREPAREDATA:

           if( e( 32 ))
           {
              object_.z32() ;
              state.reset( S_2_CREATING_DC ) ;
              Trace( L"Transition 1" ) ;
           }
           else if ( e( 33 ))
           {
              object_.z33() ;
              state.reset( S_4_COMPLETED ) ;
              Trace( L"Transition 2" ) ;
           }

           break;
        case S_2_CREATING_DC:

           if( e( 30 ))
           {
              state.reset( S_3_SENDING ) ;
              Trace( L"Transition 1" ) ;
           }
           else if ( e( 31 ))
           {
              object_.z35() ;
              state.reset( S_4_COMPLETED ) ;
              Trace( L"Transition 2" ) ;
           }
           else if ( e( 13 ))
           {
              object_.z38() ;
              object_.z36() ;
              state.reset( S_4_COMPLETED ) ;
              Trace( L"Transition 3" ) ;
           }

           break;
        case S_3_SENDING:

           if( e( 34 ))
           {
              object_.z36() ;
              state.reset( S_4_COMPLETED ) ;
              Trace( L"Transition 1" ) ;
           }
           else if ( e( 13 ))
           {
              object_.z38() ;
```

```
            object_.z36() ;
            state.reset( S_4_COMPLETED ) ;
            Trace( L"Transition 2" ) ;
         }
         else if ( e ( 31 ))
         {
            object_.z35() ;
            state.reset( S_4_COMPLETED ) ;
            Trace( L"Transition 3" ) ;
         }

         break;
      case S_4_COMPLETED:

         break;

      }
   }

   void A4_DataTransfer::OnStateChanged()
   {
      using namespace datatransf_automaton ;

      switch( GetState())
      {
      case S_1_PREPAREDATA:

         object_.z30() ;

         break;
      case S_2_CREATING_DC:

         object_.z31() ;

         break;
      case S_3_SENDING:

         object_.z34() ;

         break;
      case S_4_COMPLETED:

         object_.z37() ;

         break;

      }
   }

   ////////////////////////////////////////////////////////////////////////
   //
```

```
// Main Automaton
//

A2_MainAutomaton::A2_MainAutomaton( UserSite& us )
   : Base      ( main_automaton::S_1_INITIALIZATION )
   , object_   ( us )
   , initAuto_ ( us )
   , tranfAuto_( us )
{
}

void A2_MainAutomaton::Init( ITracer * tracer )
{
   Base::Init( tracer ) ;
}

// Перенаправление сообщений о событиях для вложенных автоматов
BOOL A2_MainAutomaton::OnEvent( Event event )
{
   BOOL eventProcessed = FALSE ;

   if( event >= E_3_TRANSFCOMPLETED && event <= E_19_RETRCOMPLETED )
   {
      eventProcessed = TRUE ;
      event_.reset( event ) ;
      CalcState() ;
   }
   else
   {
      if ( initAuto_ .ProcessEvent( event ))
         eventProcessed = TRUE ;

      if ( tranfAuto_.ProcessEvent( event ))
         eventProcessed = TRUE ;
   }

   return eventProcessed ;
}

void A2_MainAutomaton::CalcState( OptState& state )
{
   using namespace main_automaton ;

   switch( GetState())
   {
   case S_1_INITIALIZATION:

      if( e( 16 ))
      {
         state.reset( S_2_READY ) ;
         Trace( L"Transition 1" ) ;
```

```
            }
            else if( e( 17 ))
            {
               state.reset( S_5_CLOSING ) ;
               Trace( L"Transition 2" ) ;
            }

            break ;
         case S_2_READY:

            if( e( 12 ))
            {
               object_.z9() ;
               Trace( L"Transition 1" ) ;
            }
            else if( e( 4 ))
            {
               object_.z11() ;
               state.reset( S_5_CLOSING ) ;
               Trace( L"Transition 2" ) ;
            }
            else if( e( 5 ))
            {
               state.reset( S_1_INITIALIZATION ) ;
               object_.z2() ;
               Trace( L"Transition 3" ) ;
            }
            else if( e( 6 ))
            {
               object_.z3() ;
               Trace( L"Transition 4" ) ;
            }
            else if( e( 7 ))
            {
               object_.z4() ;
               Trace( L"Transition 5" ) ;
            }
            else if( e( 8 ))
            {
               object_.z5() ;
               Trace( L"Transition 6" ) ;
            }
            else if( e( 9 ))
            {
               object_.z6() ;
               Trace( L"Transition 7" ) ;
            }
            else if( e( 10 ))
            {
               object_.z7() ;
               Trace( L"Transition 8" ) ;
```

```
            }
            else if( e( 11 ))
            {
               object_.z8() ;
               Trace( L"Transition 9" ) ;
            }
            else if( e( 14 ))
            {
               state.reset( S_4_DATATRANSFERING ) ;
               Trace( L"Transition 10" ) ;
            }
            else if( e( 18 ))
            {
               state.reset( S_5_CLOSING ) ;
               Trace( L"Transition 11" ) ;
            }
            else if( e( 15 ))
            {
               object_.z12() ;
               state.reset( S_3_FILETRANSFERING ) ;
               Trace( L"Transition 1" ) ;
            }
            break ;
         case S_3_FILETRANSFERING:

            if( e( 13 ))
            {
               object_.z13() ;
               state.reset( S_3_FILETRANSFERING ) ;
               Trace( L"Transition 1" ) ;
            }
            else if( e( 4 ))
            {
               object_.z10() ;
               object_.z11() ;
               state.reset( S_3_FILETRANSFERING ) ;
               Trace( L"Transition 2" ) ;
            }
            else if( e( 18 ))
            {
               object_.z10() ;
               state.reset( S_3_FILETRANSFERING ) ;
               Trace( L"Transition 3" ) ;
            }
            else if ( e( 19 ) && !object_.x1())
            {
               state.reset( S_2_READY ) ;
               Trace( L"Transition 4" ) ;
            }
            else if ( e( 19 ) && object_.x1())
            {
```

```
            state.reset( S_5_CLOSING ) ;
            Trace( L"Transition 5" ) ;
         }

         break ;
      case S_4_DATATRANSFERING:

         if ( e( 3 ) && !object_.x1())
         {
            state.reset( S_2_READY ) ;
            Trace( L"Transition 1" ) ;
         }
         else if( e( 3 ) && object_.x1())
         {
            state.reset( S_5_CLOSING ) ;
            Trace( L"Transition 2" ) ;
         }
         else if( e( 4 ))
         {
            object_.z10() ;
            object_.z11() ;
            state.reset( S_4_DATATRANSFERING ) ;
            Trace( L"Transition 3" ) ;
         }
         else if( e( 18 ))
         {
            object_.z10() ;
            state.reset( S_4_DATATRANSFERING ) ;
            Trace( L"Transition 4" ) ;
         }

         break ;
      case S_5_CLOSING:
         break ;
      }
   }

   void A2_MainAutomaton::OnStateChanged()
   {
      using namespace main_automaton ;

      switch( GetState())
      {
      case S_1_INITIALIZATION :

         initAuto_.Init( GetTracer()) ;

         break ;
      case S_2_READY :
         break ;
      case S_3_FILETRANSFERING :
```

```
            break ;
         case S_4_DATATRANSFERING :

            tranfAuto_.Init( GetTracer()) ;

            break ;
         case S_5_CLOSING :

            object_.z1() ;

            break ;
         }
      }

      } // namespace user_site




      ////////////////////////////////////////////////////////////////////////
      //
      // RSAutomaton.h
      // Объявление класса A5_RemoteServer
      //
      ////////////////////////////////////////////////////////////////////////

      #pragma once
      #include "BaseAutomaton.h"

      ////////////////////////////////////////////////////////////////////////
      //
      // Remote server automaton
      //

      class RemoteServer ;

      namespace remote_server
      {

         namespace main_automaton
         {
            enum State
            {
               S_1_CONNECT       = 1,
               S_2_WAIT4GREETINGS   ,
               S_3_PROC_USER        ,
               S_4_PROC_PASS        ,
               S_5_READY4RETR       ,
               S_6_SETPARAMS        ,
               S_7_TRANSFERING      ,
               S_8_CLOSE
```

```
      } ;
   } // main_automaton

   enum Event
   {
      E_1_REPLY            = 1,
      E_2_REPLY               ,
      E_3_REPLY               ,
      E_4_REPLY               ,
      E_5_REPLY               ,
      E_6_CONNECTIONERROR     ,
      E_7_CONNECTED           ,
      E_8_RETRIEVE            ,
      E_9_ABORT
   } ;

   struct A5_RemoteServer
      : public BaseAutomaton< main_automaton::State, Event >
   {
      typedef BaseAutomaton< main_automaton::State, Event > Base ;

      A5_RemoteServer( RemoteServer& rs ) ;
      LPCWSTR GetPrefix() const { return L"Remote Server Automaton: "; }

   private:
      void CalcState     ( OptState& state ) ;
      void OnStateChanged() ;

   private:
      RemoteServer&  object_ ;
   };

} // namespace remote_server




////////////////////////////////////////////////////////////////////////
//
// RSAutomaton.cpp
// Определение класса A5_RemoteServer
//
////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "RemoteServer.h"
#include "RSAutomaton.h"

namespace remote_server
{
```

```
A5_RemoteServer::A5_RemoteServer( RemoteServer& rs )
   : Base( main_automaton::S_1_CONNECT )
   , object_( rs )
{
}

void A5_RemoteServer::CalcState( OptState& state )
{
   using namespace main_automaton ;

   switch( GetState())
   {
   case S_1_CONNECT :

      if( e( 6 ))
      {
         object_.z2() ;
         state.reset( S_8_CLOSE ) ;
         Trace( L"Transition 1" ) ;
      }
      else if( e( 7 ))
      {
         state.reset( S_2_WAIT4GREETINGS ) ;
         Trace( L"Transition 2" ) ;
      }

      break ;
   case S_2_WAIT4GREETINGS:

      if( e( 2 ))
      {
         state.reset( S_3_PROC_USER ) ;
         Trace( L"Transition 1" ) ;
      }
      else if( e( 6 ))
      {
         object_.z2 () ;
         state.reset( S_8_CLOSE ) ;
         Trace( L"Transition 2" ) ;
      }

      break;
   case S_3_PROC_USER :

      if( e( 2 ) && !object_.x1())
      {
         object_.z11() ;
         state.reset( S_5_READY4RETR ) ;
         Trace( L"Transition 1" ) ;
      }
```

```
            else if( e( 3 )) // требуется пароль для подтверждения
            {
               state.reset( S_4_PROC_PASS ) ;
               Trace( L"Transition 2" ) ;
            }
            else if( e( 1 ) || e ( 4 ) || e( 5 ) || e ( 6 ))
            {
               object_.z2() ;
               state.reset( S_8_CLOSE ) ;
               Trace( L"Transition 3" ) ;
            }
            else if( e( 2 ) && object_.x1())
            {
               object_.z15() ;
               object_.z17() ;
               state.reset( S_5_READY4RETR ) ;
               Trace( L"Transition 4" ) ;
            }

            break ;
         case S_4_PROC_PASS :

            if( e( 2 ) && !object_.x1())
            {
               object_.z11() ;
               state.reset( S_5_READY4RETR ) ;
               Trace( L"Transition 1" ) ;
            }
            else if ( e( 1 ) || e( 3 ) || e( 4 ) || e( 5 ) || e( 6 ))
            {
               object_.z2 () ;
               state.reset( S_8_CLOSE ) ;
               Trace( L"Transition 2" ) ;
            }
            else if ( e( 2 ) && object_.x1())
            {
               object_.z15() ;
               object_.z17() ;
               state.reset( S_5_READY4RETR ) ;
               Trace( L"Transition 3" ) ;
            }

            break ;
         case S_5_READY4RETR :

            if( e( 6 ))
            {
               state.reset( S_8_CLOSE ) ;
               Trace( L"Transition 1" ) ;
            }
            else if ( e( 8 ) && !object_.x1())
```

```
            {
               state.reset( S_6_SETPARAMS ) ;
               Trace( L"Transition 2" ) ;
            }

            break ;
         case S_6_SETPARAMS :

            if( e( 2 ))
            {
               object_.z9() ;
               state.reset( S_6_SETPARAMS ) ;
               Trace( L"Transition 1" ) ;
            }
            else if( e( 1 ) || e( 3 ) || e( 4 ) || e( 5 ) || e( 6 ))
            {
               object_.z16() ;
               state.reset( S_8_CLOSE ) ;
               Trace( L"Transition 2" ) ;
            }
            else if ( e( 9 ))
            {
               object_.z15() ;
               object_.z17() ;
               state.reset( S_5_READY4RETR ) ;
               Trace( L"Transition 3" ) ;
            }
            else if( object_.x4() && object_.x6())
            {
               object_.z7() ;
               state.reset( S_6_SETPARAMS ) ;
               Trace( L"Transition 4" ) ;
            }
            else if ( object_.x5() && object_.x6())
            {
               object_.z8() ;
               state.reset( S_6_SETPARAMS ) ;
               Trace( L"Transition 5" ) ;
            }
            else if ( object_.x2() && object_.x6())
            {
               object_.z5 () ;
               state.reset( S_6_SETPARAMS ) ;
               Trace( L"Transition 6" ) ;
            }
            else if( object_.x3() && object_.x6())
            {
               object_.z6 () ;
               state.reset( S_6_SETPARAMS ) ;
               Trace( L"Transition 7" ) ;
            }
```

```
            else if ( !object_.x2() && !object_.x3() && !object_.x4()
                  && !object_.x5() && object_.x6())
            {
               object_.z10() ;
               state.reset( S_7_TRANSFERING ) ;
               Trace( L"Transition 8" ) ;
            }

            break ;
         case S_7_TRANSFERING :

            if( e( 6 ))
            {
               object_.z16() ;
               state.reset( S_8_CLOSE ) ;
               Trace( L"Transition 1" ) ;
            }
            else if ( e( 2 ) || e( 4 ) || e( 5 ))
            {
               object_.z13() ;
               state.reset( S_5_READY4RETR ) ;
               Trace( L"Transition 2" ) ;
            }
            else if ( e( 1 ))
            {
               object_.z12() ;
               state.reset( S_7_TRANSFERING ) ;
               Trace( L"Transition 3" ) ;
            }
            else if ( e( 9 ))
            {
               object_.z18() ;
               object_.z15() ;
               object_.z17() ;
               state.reset( S_5_READY4RETR ) ;
               Trace( L"Transition 4" ) ;
            }

            break ;
         case S_8_CLOSE :

            // no transitions

            break ;
         }
      }

      void A5_RemoteServer::OnStateChanged()
      {
         using namespace main_automaton ;
```

```
        switch( GetState())
        {
        case S_1_CONNECT :

           object_.z1() ;

           break ;
        case S_3_PROC_USER :

           object_.z3() ;

           break ;
        case S_4_PROC_PASS :

           object_.z4() ;

           break ;
        case S_8_CLOSE :

           object_.z14() ;

           break ;
        }
     }

     } // namespace remote_server
```

Приложение 3. Пример протокола работы сервера

```
[19:41:43.237] UserSite: Computer connected: 127.0.0.1:21; connection id: 0
[19:41:43.237] User Main Automaton: Automaton started at state: 1
[19:41:43.237]   User Init Automaton: Automaton started at state: 1
[19:41:43.237] UserSite: -->220 Service ready for new user.
[19:41:43.257] UserSite: <-- USER valery
[19:41:43.257]   User Init Automaton: Transition 1
[19:41:43.257]   User Init Automaton: Entered state: 2
[19:41:43.257]   User Init Automaton: Transition 1
[19:41:43.257]   User Init Automaton: Entered state: 3
[19:41:43.257] UserSite: -->331 User name okay, need password.
[19:41:43.267] UserSite: <-- PASS password1
[19:41:43.267]   User Init Automaton: Transition 1
[19:41:43.267]   User Init Automaton: Entered state: 4
[19:41:43.267]   User Init Automaton: Transition 1
[19:41:43.267]   User Init Automaton: Entered state: 5
[19:41:43.267] UserSite: -->230 User logged in, proceed.
[19:41:43.267] User Main Automaton: Transition 1
[19:41:43.267] User Main Automaton: Entered state: 2
[19:41:43.277] UserSite: <-- REST 0
[19:41:43.277] UserSite: -->202 Command not implemented, superfluous at this site.
[19:41:43.297] UserSite: <-- PWD
[19:41:43.297] UserSite: -->257 "/" Created.
[22:21:32.996] User Main Automaton: Transition 5
[22:21:33.006] UserSite: <-- SYST
[22:21:33.006] UserSite: -->202 Command not implemented, superfluous at this site.
[22:21:33.016] UserSite: <-- PORT 127,0,0,1,19,8
[22:21:33.026] UserSite: -->200 Command okay.
[22:21:33.026] User Main Automaton: Transition 6
[22:21:33.026] UserSite: <-- LIST
[22:21:33.026] User Main Automaton: Transition 10
[22:21:33.026] User Main Automaton: Entered state: 4
[22:21:33.026]   User Data Transfer Automaton: Automaton started at state: 1
[22:21:33.026] UserSite: -->150 File status okay; about to open data connection.
[22:21:33.026]   User Data Transfer Automaton: Transition 1
[22:21:33.026]   User Data Transfer Automaton: Entered state: 2
[22:21:33.036]   User Data Transfer Automaton: Transition 1
[22:21:33.036]   User Data Transfer Automaton: Entered state: 3
[22:21:33.036] UserSite: -->226 Closing data connection.
[22:21:33.036]   User Data Transfer Automaton: Transition 1
[22:21:33.036]   User Data Transfer Automaton: Entered state: 4
[22:21:33.036] User Main Automaton: Transition 1
[22:21:33.036] User Main Automaton: Entered state: 2
```