Министерство образования Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра теоретических основ информатики

УДК 681.03

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, доцент, д.т.н.
_____Ю.Л. Костюк
«___»_____2003 г.

Фофонов Алексей Владиславович

# РАЗРАБОТКА АРХИТЕКТУРЫ И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМЫ КОНТРОЛЯ ДОСТУПА

Дипломная работа

Научный руководитель,
заведующий 15 отделом
НИИАЭМ, д.т.н. _____ А. Я. Клименко

Исполнитель,
студент гр.1481 _____ А.В. Фофонов

Электронная версия дипломной работы помещена
В электронную библиотеку. Файл
Администратор

Томск - 2003

# Реферат

Дипломная работа 57 с., 21 рис., 14 источников, 3 прил.

СИСТЕМЫ КОНТРОЛЯ И УПРАВЛЕНИЯ ДОСТУПОМ, КАРТОЧКИ-ИДЕНТИФИКАТОРЫ, БИОМЕТРИЧЕСКАЯ ИДЕНТИФИКАЦИЯ, КЛАССИФИКАЦИЯ СИСТЕМ КОНТРОЛЯ И УПРАВЛЕНИЯ ДОСТУПОМ

(1) Объект исследования – системы контроля и управления доступом.

(2) Цель работы – разработать эффективную архитектуру системы контроля и управления доступом.

(3) Метод исследования – экспериментальный (на ЭВМ).

(4) Результат – разработана общая архитектура системы контроля и управления доступом, реализованы и испробованы все основные архитектурные механизмы.

# Содержание

## Перечень условных обозначений

- СКУД – Системы Контроля и Управления Доступом;
- КУД – Контроль и управление доступом;
- СКД – Системы Контроля Доступа;
- ПИН – Персональный Идентификационный Номер;
- СРВ – Системы Реального Времени;
- АРМ – Автоматизированное Рабочее Место;
- UML – Unified Modeling Language – Унифицированный язык моделирования;
- CASE – Computer Aided System Engineering – автоматизированное проектирование систем;
- MFC – Microsoft Foundation Class library – базовая библиотека классов Микрософт.

# Введение

В настоящее время на предприятиях, где необходимо контролировать и ограничивать доступ людей в различные помещения, широкое применение нашли автоматизированные системы контроля и управления доступом (СКУД). Эти системы предназначены для обеспечения санкционированного прохода в помещения и охраняемые зоны. Помимо своих основных функций по организации доступа к ресурсу, СКУД помогают решить многие другие задачи. Сюда можно отнести учет рабочего времени, быстрое определение местонахождения сотрудников, управление лифтами, вентиляцией, пожарной сигнализацией и многое другое.

Сегодня главным направлением развития систем контроля и управления доступом является их интеллектуализация, передача максимально возможного количества функций по сбору, обработке информации и принятию решений, аппаратным средствам СКУД и компьютерам. Освобождение человека от рутинного труда особенно важно в процессе обеспечения безопасности объектов, где цена ошибки, а иногда и элементарной невнимательности, очень велика. С другой стороны важно обеспечить оператора полной и точной информацией о происходящих на объекте событиях и удобными средствами для безошибочного и своевременного принятия оперативных решений.

Но системы контроля доступа могут быть использованы не только для снижения издержек предприятия при организации системы безопасности. Их применяют и как средства автоматизации маркетинга в гостиницах, на курортах и т. д. При этом возможность доступа в различные помещения рассматривается как некоторый оплачиваемый ресурс, что позволяет говорить о такой системе как об автоматизированной системе управления маркетингом. Особенностью этих СКУД является реализация механизма расчетов пользователей системы с ее владельцем.

***Целью данной работы*** является разработка гибкой архитектуры системы контроля и управления доступом к оплачиваемому ресурсу, поддерживающей схему расчетов пользователей-клиентов с владельцем СКУД. Под архитектурой здесь понимается организационная структура системы, включающая в себя разделение системы на части, связи между этими частями, механизмы взаимодействия и основные принципы проектирования системы (см.[1]).

***Актуальность темы***. Существующие в настоящее время СКУД слишком специализированы. Например, в одних системах недостаточно поддерживаются функции оплаты используемых ресурсов, в других – обслуживания владельцев дружественных

систем и т.д. Поэтому встала проблема разработки системы, которая бы комплексно решала все задачи СКД. Комплексное проектирование системы позволит решить и все возникающие при ее внедрении проблемы безопасности.

***Для достижения поставленной цели необходимо решить следующие задачи***:

- Провести аналитическое исследование систем контроля доступа и на его основе осуществить построение модели предметной области;
- Разработать, реализовать и протестировать набор механизмов, обеспечивающих выполнение требований, предъявляемых к системам доступа;
- Разработать архитектуру модулей системы на основе созданных механизмов;
- Провести комплексное тестирование архитектуры, подтверждающее возможность ее использования.

В первой главе производится обзор предметной области, раскрываются основные понятия, состав и принципы работы, приводится классификация СКУД. Вторая глава посвящена объяснению выбора подходов и инструментов, а также описанию структуры системы и модели предметной области. В третьей главе изложены базовые механизмы, положенные в основу архитектуры программного обеспечения системы контроля доступа. Четвертая и пятая главы соответственно описывают архитектуру приложения-сервера и приложений-клиентов. В шестой главе рассматриваются принципы взаимодействия этих модулей.

# 1. Исследование предметной области

## 1.1. Состав СКУД

Система контроля и управления доступом обычно состоит (см. [2]) из серверов СКУД – обычных компьютеров, которые управляют подключенными к ним контроллерами СКУД. ***Контроллер*** (контрольная панель) – это специализированный высоконадежный компьютер. В нем хранится информация о конфигурации, режимах работы системы, список людей, которые имеют право доступа к ресурсу, а также их привилегии доступа к этому ресурсу. В простых случаях минимальный вариант контроллера может быть встроен в считыватель, турникет, замок или другое исполнительное устройство.

Следующим важным звеном в СКУД являются такие устройства, как ***считыватели***, которые можно подключить к контроллерам. Считыватель представляет собой устройство, которое позволяет считывать информацию, записанную на карточке. Эту информацию он передает контроллеру, который и принимает решение о допуске человека к ресурсу. Можно настроить контроллер так, что он будет запрашивать подтверждение принятого решения у компьютера.

Любой считыватель предполагает ответную часть – ***ключ-идентификатор***, который содержит информацию, с помощью которой происходит идентификация человека. Каждой карточке приписан некоторый уровень доступа, в соответствии с которым пользователь имеет право получить доступ к тому или иному ресурсу в определенные промежутки времени. Классификация ключей представлена в §1.2.

Для повышения надежности идентификации кроме считывателей к контроллеру может подключаться ***клавиатура*** для набора персонального идентификационного номера (***ПИН-кода***).

Другой тип устройств, которые можно подключить к контроллеру – это ***охранные панели***. Это также специализированный контроллер, который отслеживает состояние охранных датчиков (датчики на дверях, окнах, объемные датчики и другие). Если состояние какого-либо датчика изменяется, то информация об этом тут же поступает в основной контроллер.

У охранной панели может быть набор реле, с помощью которых она осуществляет управление исполнительными устройствами: электромеханическими замками, турникетами, лифтами, автоматическими воротами и т.д.

## 1.2. Способы идентификации

Существует два различных направления в способах идентификации. Это идентификация с использованием электронных карт, и идентификация, использующая биометрические параметры человека. Сейчас применяются следующие типы карт, каждому из которых соответствует определенный тип считывателя (см. [3]):

- ***магнитные карты*** – считываются, при проведении в определенном направлении и с определенной скоростью по щели считывателя. Магнитная полоса с записанной на ней информацией нанесена на одну из сторон пластиковой карточки. Современные магнитные полосы изготовлены из материалов, требующих сильных магнитных полей для записи информации и, соответственно, для ее уничтожения, поэтому можно не бояться случайного размагничивания. Однако магнитные карты достаточно чувствительны к внешним воздействиям другого рода – загрязнению, влаге, царапинам. Еще один недостаток связан с необходимостью точного позиционирования в считывателе. Средний срок службы магнитных карт составляет около года, затем магнитный слой стирается. Поэтому магнитные карточки применяют, как правило, в системах, где предусмотрена частая замена карт, например, в отелях или на автостоянках.
- ***бесконтактные радиочастотные (PROXIMITY) карты*** – наиболее перспективный на сегодняшний день тип карт. Бесконтактные карточки действуют на расстоянии и не требуют четкого позиционирования, что обеспечивает их устойчивую работу и удобство использования, высокую пропускную способность. Для считывания информации с бесконтактной карточки ее достаточно просто поднести к считывателю. Считыватель генерирует электромагнитное излучение определенной частоты и, при внесении карты в зону действия считывателя, это излучение через встроенную в карте антенну питает чип карты. Получив необходимую энергию для работы, карта пересылает на считыватель свой идентификационный номер с помощью электромагнитного импульса определенной формы и частоты. При этом карточка может находиться в кармане или в бумажнике.
- ***карты Виганда*** – названные по имени ученого, открывшего сплав, обладающий прямоугольной петлей гистерезиса. Внутри карты размещены отрезки проволоки из этого сплава, которые при перемещении мимо считывающей головки позволяют считать информацию. Эти карты более долговечны, чем магнитные, но и более

дорогие. Один из недостатков – то, что код в карту занесен при изготовлении раз и навсегда.

- ***штрих-кодовые карты*** – на карту наносится штриховой код. Существует более сложный вариант – штрих-код закрывается материалом, прозрачным только в инфракрасном свете, считывание происходит в инфракрасной области.
- ***Touch-memory*** – металлическая таблетка, внутри которой расположен чип ПЗУ. При касании таблетки считывателя, из памяти таблетки в контроллер пересылается уникальный код идентификатора. Достаточно дешевы и удобны.

К биометрическим способам идентификации относят (подробнее см. в [4]):

- ***Сканирование отпечатков пальцев*** – сканирование отпечатков пальцев является самым удобным методом, а применяемые при этом устройства – самыми дешевыми. Преимуществом является и надежность сканирования отпечатков пальцев: несанкционированный доступ возможен примерно в одном случае из миллиона, а отказ в доступе уполномоченному пользователю возникает примерно в 3% случаев и связан в основном с неправильным уходом за сканером.
- ***Геометрия ладони и кисти рук –*** сканируются не линии, как у отпечатков пальцев, а геометрия руки: форма ладони или кисти, длина пальцев и т. д. В принципе, по надежности этот метод практически не уступает предыдущему, но подобные системы занимают гораздо больше места, что затрудняет их использование на обычном компьютере, да и стоят они дороже.
- ***Сканирование глаза –*** различают два типа: сканирование радужной оболочки и сканирование сетчатки. Первый метод более простой и удобный, но и менее надежный. Второй является самым надежным, но и самым дорогим.
- ***Идентификация по голосу –*** Преимуществом является удобство использования. Но этот метод имеет низкую надежность, так как для того, чтобы голос человека значительно изменился, достаточно простудиться.
- ***Подпись –*** человек расписывается на специальном устройстве типа графического планшета. Компьютер сравнивает полученную написанную информацию с той, которая хранится в его базе, и в зависимости от результатов сравнения предоставляет доступ или отказывает в нем. Саму подпись легко подделать, но современные считыватели измеряют еще и характеристики движения руки при письме, что повышает надежность метода.

- ***Геометрия лица, Клавиатурный почерк*** **–** эти методы слабо разработаны, реально действующих систем не существует.

## 1.3. Принципы работы систем контроля доступа

В основе работы систем контроля и управления доступом заложен принцип сравнения тех или иных идентификационных признаков, принадлежащих конкретному физическому лицу или объекту, с данными, заложенными в систему [5].

Каждый из сотрудников (посетителей) получает карту доступа или брелок, содержащий индивидуальный код, присваиваемый при выдаче карты доступа в бюро пропусков. В качестве кода могут использоваться также биометрические данные человека.

При проходе на охраняемую территорию или в охраняемое помещение производится считывание данных с носителя кода через считыватели. Информация о посетителе передается в систему, где производится анализ и дается сигнал, адекватно реагирующий на сложившуюся ситуацию: <Проход разрешен>, <Проход запрещен>, <Повторный проход по одной карте>, вывод сигнала <Тревога> на пульт охранника при нарушении охраняемой территории без соответствующих прав и т.д.

При необходимости вмешательства охраны в сложившуюся ситуацию на экран компьютера поста охраны выводится тревожный сигнал и инструкция, определяющая действия персонала в данной ситуации. Причем, система тут же может отреагировать на тревожную ситуацию, заблокировав замки в охраняемое помещение и пути прохода по точкам доступа.

Для анализа произошедших событий имеется возможность просмотра и распечатки протокола событий за определенный период времени. Для исключения злоупотреблений в использовании карт и ужесточения проходного режима в особо важные зоны имеется ряд функций, позволяющих: (см. [5])

- исключить двойной проход в зону по одной карте (различают возможности блокирования повторного прохода на определенное время – для систем, не оборудованных считывателями на выходе и запрет на вход в несмежную зону для полных систем контроля доступа);
- разрешить доступ только по 2-м картам (войти могут только два человека, встретившись вместе и обладающих соответствующими полномочиями);

- ограничить количество лиц в помещении и зоне (при превышении установленного порогового значения контроллер не пропустит в зону очередного человека);
- установить режим <вход под принуждением> (незаметно для окружающих охране подается сигнал тревоги);
- охраннику дается право на самостоятельное принятие решения о разрешении на проход посетителя (при считывании карты на монитор охранника выводится фотография владельца, которая сличается с изображением, выдаваемым видеокамерой);
- установить режим счетчика на использование карты (количество чтений карты на конкретном считывателе ограничивается);
- установить скрытый контроль в помещении (подать сигнал тревоги на пульт охраны при проникновении в защищаемое помещение и отсутствии соответствующих прав, причем для злоумышленника факт обнаружения остается неведомым).

## 1.4. Классификация СКУД

Рассмотрим некоторые важные классификации систем контроля и управления доступом, представленные в [6].

Классификация СКУД по способу управления:

- ***автономные*** – для управления одним или несколькими преграждающими устройствами без передачи информации на центральный пульт и без контроля со стороны оператора. Обычно это простейшие СКУД, точнее электронные замки, которые ограничивают доступ в помещения. К преимуществам таких систем можно также отнести возможность легкого удаления номера ключа из энергозависимой памяти системы при его утере, таким образом, нашедший ключ никогда не сможет им воспользоваться. Автономные системы нашли применение, как правило, на небольших объектах (входы в жилые дома, коттеджи и т.п.). Существуют и автономные системы контроля доступа с функциями охраны.
- ***централизованные*** (сетевые) – для управления преграждающими устройствами за счет обмена информацией с центральным пультом, для контроля (управления) со стороны оператора. Сетевые системы контроля применяются там, где требуется постоянный контроль состояния объекта, возможность оперативного

вмешательства в работу системы и получение различных статистических данных о движении персонала. Управление доступом в сетевых системах в основном осуществляется автоматически, на основе различных объектных и временных ограничений доступа, задаваемых как для отдельных владельцев ключей, так и для групп владельцев, выделенных по какому-либо признаку с помощью специальной программы. Оператор имеет возможность работать с базами данных пользователей, регистрировать и редактировать права доступа. При запущенной программе все события, происходящие в системе, выводятся на монитор в режиме реального времени и протоколируются для последующего получения отчетов по каждому пользователю. Система позволяет получить полный набор стандартных отчетов о перемещениях сотрудников, а также вести учет рабочего времени. Сети связи в системе защищены от злоумышленников аппаратно и программно. Сетевые системы оптимальны для применения в небольших и средних офисах или предприятиях (до 256 контролируемых точек прохода).

- ***универсальные*** – включающие функции как автономных, так и сетевых систем, работающие в сетевом режиме под управлением центрального устройства управления и переходящие в автономный режим при возникновении отказов в сетевом оборудовании, в центральном устройстве или обрыве связи.

По количеству контролируемых точек доступа различают:

- системы малой емкости (менее 16 точек);
- системы средней емкости (не менее 16 и не более 64 точек);
- системы большой емкости (64 точки и более).

Классификация по виду объектов контроля:

- для контроля доступа к физическим объектам;
- для контроля доступа к информации.

## 1.5. Системы контроля доступа в автоматизации маркетинга

В настоящее время наиболее широко распространено понимание «системы контроля доступа» как средства организации контрольно-пропускного режима на предприятии. В этих системах в качестве пользователей понимаются сотрудники предприятия – владельца СКУД, и наибольшее внимание уделяется непосредственному

обеспечению безопасного доступа в зоны и помещения. Такое использование СКУД, способствует уменьшению расходов предприятия на организацию безопасности.

Использование систем контроля доступа как средств автоматизации маркетинга преследует уже несколько иные цели: получение прибыли за счет продажи возможности доступа к ресурсу. В этом смысле система контроля доступа становится специализированной системой автоматизации маркетинга. Подробнее о карточных системах в автоматизации маркетинга изложено в [7].

В таких системах пользователи понимаются более широко – как предприятия и частные лица. Здесь уже огромное значение приобретает механизм организации расчетов этих пользователей с владельцем СКУД, который вовсе отсутствует в первичном понимании систем контроля доступа, где в качестве пользователей выступали сотрудники.

Если в первом случае группировка пользователей востребуется лишь как средство более удобного управления доступом, то в системах автоматизации маркетинга, она приобретает большую значимость благодаря внедрению понятия счета. Один счет может использоваться как одним человеком, так и группой людей (предприятие, семья и пр.) и даже группой групп (ассоциация групп), что задает возможности построений сложных иерархий.

В целом понимание СКУД как системы автоматизации маркетинга можно считать обобщением первичного понятия, так как все взаимодействие сотрудников с системой можно интерпретировать через механизм счетов. Например, для сотрудников стоимость доступа может быть нулевой. В противном случае, счета естественным образом позволяют оценивать интенсивность использования системы данным сотрудником.

Системы автоматизации маркетинга нагружают дополнительным смыслом и различные отчеты, формируемые СКУД, позволяя на их основе производить статистические исследования востребованности ресурса в конкретной точке доступа. Эти исследования могут использоваться для гибкой настройки системы на нужды пользователей. Статистика доступа конкретного пользователя или группы позволяет эффективно стимулировать постоянных клиентов с помощью различных схем скидок и поощрений.

Использование СКУД открывает широкие возможности по автоматизации работы различных гостиниц, курортов, дискотек и других учреждений, торгующих доступом в различные помещения. Например, система контроля доступа, автоматизирующая работу гостиницы, позволяет не только снизить издержки на обеспечение безопасности, но и

организовать платное использование сауны, бара, ресторана или других сервисов гостиницы. СКУД горнолыжного курорта может организовать удобный для клиентов платный доступ к подъемникам и т.д.

## 1.6. Особенности систем контроля доступа, как систем реального времени

В силу своей специфики, системы контроля доступа являются системами реального времени (СРВ). СРВ, как аппаратно-программный комплекс, включает в себя датчики, регистрирующие события на объекте, модули ввода-вывода, преобразующие показания датчиков в цифровой вид, пригодный для обработки этих показаний на компьютере, и, наконец, компьютер с программой, реагирующей на события, происходящие на объекте. Всякая СРВ ориентирована на обработку внешних событий. Ее основная задача – реагировать в предсказуемые времена, на непредсказуемый поток внешних событий. Это означает, что система должна отреагировать на событие, произошедшее на объекте, своевременно, то есть в течение времени, критического для этого события. Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается ошибкой для систем реального времени [8].

Кроме этого, система должна успевать реагировать на одновременно происходящие события. Даже если два или большее число внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение временных интервалов, критических для этих событий [8]. Различают системы реального времени двух типов – системы жесткого реального времени и системы мягкого реального времени. Системы жесткого реального времени не допускают никаких задержек реакции системы, так как:

- результаты могут оказаться бесполезными в случае опоздания;
- может произойти катастрофа в случае задержки реакции;
- стоимость опоздания может оказаться бесконечно велика.

Системы мягкого реального времени характеризуются тем, что задержка реакции допустима, хотя и может привести к увеличению стоимости результатов и снижению производительности системы в целом. СКУД относят именно к этому типу систем. Вообще, основное отличие между системами жесткого и мягкого реального времени

можно выразить так: система жесткого реального времени никогда не опаздывает с реакцией на событие, система мягкого реального времени – не должна опаздывать с реакцией на событие.

Понимание системы контроля доступа как системы реального времени требует от разработчика использования ряда специфических механизмов, оказывающих существенное влияние на архитектуру всей системы.

# 2. Моделирование СКУД

## 2.1. Подходы и инструментарий

Основной целью любого разработчика программных систем является написание эффективного приложения. Эффективного во всех смыслах: надежного, удобного, расширяемого. Поэтому, чем удобнее и понятнее используемый подход, тем меньше придется разработчику совершить ошибок, меньше затратить времени на написание программы, проще ее будет понять и расширить.

На сегодняшний день для написания любых программ часто используется объектно-ориентированный подход, позволяющий разработчику абстрагироваться от алгоритма, как последовательности указаний. Объектно-ориентированный подход предлагает программисту представлять абстрактные сущности своей программы в виде взаимодействующих между собой объектов. Такой подход более близок и понятен человеку, поскольку он наиболее удобно отражает окружающий реальный мир. Объектное мышление позволяет представлять и понимать все более сложные системы.

В погоне за дальнейшей эффективностью и снижением времени разработки программных систем, одного существования такого подхода и соответствующих объектно-ориентированных языков программирования оказывается мало. Требуются специальные средства, помогающие продумать, промоделировать систему, избежать фатальных ошибок, проявляющихся, когда множество строк программного кода уже написано. Нужны средства, позволяющие визуально изобразить объекты и их взаимодействия, нужны методики, позволяющие последовательно найти и изучить взаимодействующие объекты, продумать процесс разработки и адаптировать его к изменяющимся потребностям.

Модели позволяют наглядно продемонстрировать структуру и поведение системы. Они необходимы для визуализации и управления архитектурой системы, минимизации рисков. Модели позволяют добиться лучшего понимания систем, что приводит к их упрощению и возможностям повторного использования. Система может быть описана с разных точек зрения, для чего используются различные модели, каждая из которых является семантически значимой абстракцией системы. Различают структурные модели, представляющие организацию системы, и поведенческие, отражающие ее динамику.

Необходимость универсализации подходов к построению моделей привела к созданию в 1997 году специального языка для описания. Универсальный язык

моделирования (UML = Unified Modeling Language) получил широкое распространение и был своевременно стандартизован.

С появлением стандарта в моделировании, возникли CASE-средства, позволяющие визуализировать этот процесс, объединить модели с документацией и даже генерировать части программного кода. Одним из наиболее известных средств этого типа является Rational Rose компании Rational. Созданное авторами UML это CASE-средство наиболее полно поддерживает нотации языка и его диаграммы. Оно позволяет организовать генерацию заголовков классов и некоторых простейших функций на основе созданных моделей, что позволяет вносить даже существенные изменения в существующую структуру системы. Универсальность языка моделирования позволяет сгенерировать участки кода и описаний на любом объектно-ориентированном языке Delphi, Java, C++, Visual Basic и других.

Для системы контроля доступа были созданы необходимые структурные и поведенческие модели на языке UML с помощью CASE-средства Rational Rose 2001. Подробно о нотации UML можно прочитать в [9],[10]. Наиболее тесно Rational Rose интегрируется со средой разработки Microsoft Visual Studio 6.0, поэтому в качестве языка программирования для генерации кода был выбран Visual C++.

## 2.2. Структура СКУД.

Как уже отмечалось выше, основным направлением развития СКУД является их интеллектуализация, агрегирование максимально возможного количества функций по сбору, обработке информации и принятию решений. Системы КУД способны автоматизировать множество процессов, связанных с организацией доступа к ресурсу. Сюда входят регистрация субъектов (пользователей и персонала) и объектов (ресурсов) СКУД, непосредственное предоставление доступа к ресурсу, организация контроля работы персонала, сбор и предоставление статистики о функционировании системы и многое другое. В автоматизации маркетинга системы контроля доступа расширяют свои возможности за счет организации системы счетов и платежей за использование ресурса.

СКУД, ориентированные на обслуживание большого числа клиентов, обычно имеют модульную структуру, позволяющую организовать рабочие места для различных служб, обеспечивающих эффективное функционирование системы. Модульная схема обеспечивается за счет использования архитектуры клиент-сервер. Количество и

функциональность модулей зависят от назначения системы и производителя. Например, набор модулей может быть таким:

- ***«бюро пропусков»*** - рабочее место менеджера по работе с клиентами. Эта служба занимается регистрацией новых клиентов, выдачей электронных карт, созданием счетов, назначением прав доступа группам и отдельным пользователям.
- ***«АРМ кассира»*** - специализированное рабочее место, предназначенное для приема платежей за использование услуг, предоставляемых системой.
- ***«АРМ оператора»*** - рабочее место оператора (охранника). Оператор отвечает за корректное функционирование системы в течение своей смены. Он контролирует работу системы, реагирует на внештатные ситуации.
- ***«АРМ администратора»***. Администратор осуществляет настройку системы, прием карт-идентификаторов, регистрацию персонала.
- ***«Генератор отчетов»***. Используется для построения различных отчетов о работе системы.

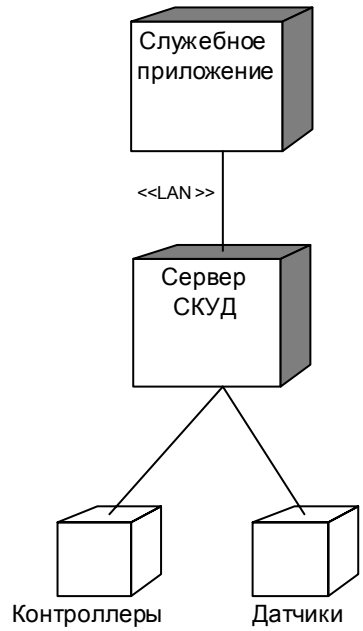


Рис.1. Диаграмма развертывания СКУД.

Для небольших систем, где роль обслуживающего персонала играет лишь один человек, вся необходимая функциональность может быть сведена в единый модуль.

Модули (служебные приложения) взаимодействуют с центральным сервером СКУД (см. рис.1), который исполняет роль некоторого диспетчера, обрабатывающего запросы приложений и события контроллеров, датчиков и других исполнительных устройств.

В качестве среды взаимодействия служебных приложений и сервера СКУД могут выступать локальная вычислительная сеть или адресное пространство одного компьютера.

## 2.3. Описание предметной области

Модель предметной области представлена на рисунке 2.

Группа представляет собой любое объединение людей (фирма, семья и т.д.). Каждый человек в системе обязательно приписан какой-либо группе. Доступ

пользователей к своим учетным записям осуществляется по паролю. Когда новый клиент желает зарегистрироваться, для него создается группа, и он считается ее администратором. Администратор имеет следующие полномочия:

- Добавлять в группу новых членов,
- Настраивать права и расписания доступа членов группы,
- Создавать и настраивать счета,
- Назначать счета для использования членам группы,
- Устанавливать ассоциации для своей группы.

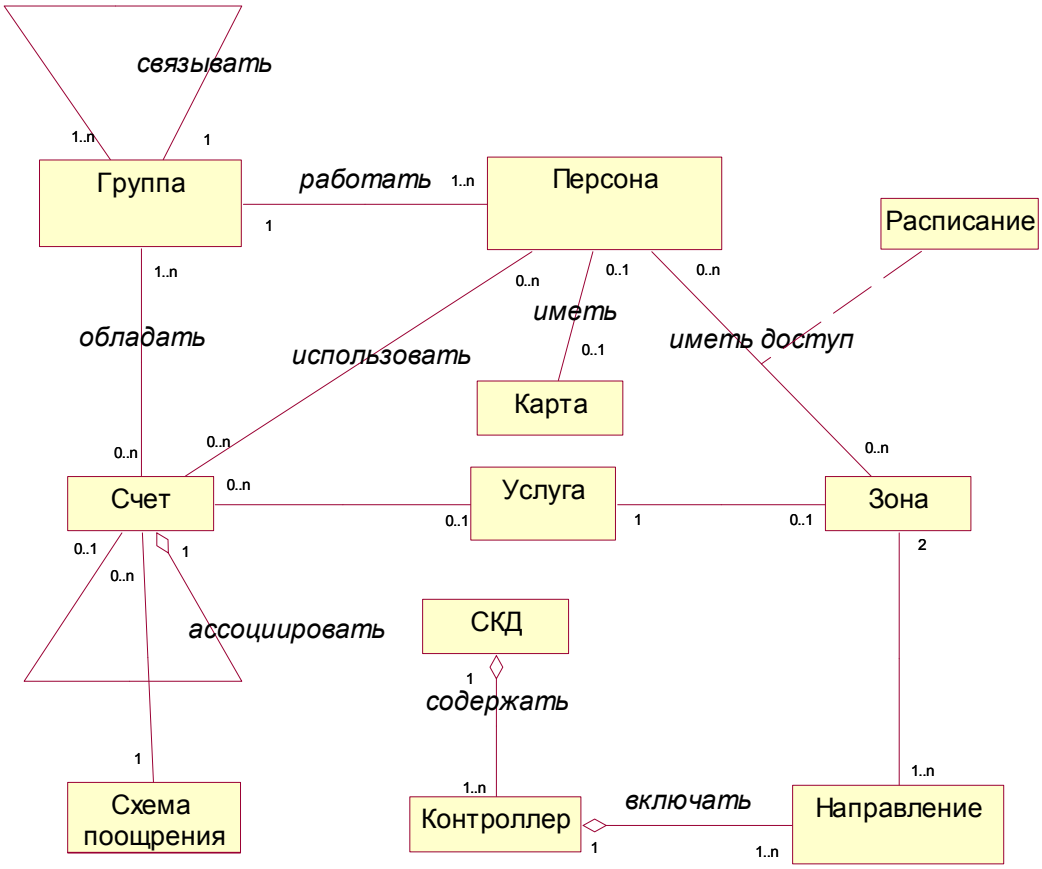


Рис.2. Модель предметной области.

Для того, чтобы клиент мог воспользоваться системой, ему выдается карта. На каждую учетную запись может быть выдано не более одной карты. Несколько учетных записей не могут использовать одну карту.

Для того чтобы построить иерархию групп, между ними можно устанавливать связи. Группа, установившая связь, разрешает использовать свои счета для установления

ассоциации между ними. Группа обязательно содержит хотя бы одного члена (ее администратора).

Каждая зона (ресурс) связана с некоторой услугой. С другой стороны, группой могут быть созданы различные счета для различных услуг. Услуга задает цену единицы ресурса. Для того чтобы производить каскадное обновление, можно устанавливать ассоциацию счетов. Счету может быть приписана определенная схема поощрения, влияющая на стоимость использования единицы ресурса.

В зависимости от типа контроллера-турникета у него может быть несколько направлений доступа. Каждое направление связано с двумя зонами (зона, откуда осуществляется доступ, и зона, куда осуществляется доступ).

# 3. Архитектурные механизмы

Архитектура программного обеспечения СКУД строится на основе ряда механизмов, определяемых требованиями, предъявляемыми к системе.

Трактовка СКУД как системы реального времени требует реализации механизмов диспетчеризации, межобъектного взаимодействия и средств работы с таймерами. Параллелизм в обработке одновременно происходящих внешних событий должен обеспечивается за счет использования многопоточности. Клиент-серверный подход вносит необходимость реализации механизма и способов взаимодействия между сервером и приложениями, а общие требования безопасности и надежности заставляют выбирать особые способы хранения данных и работы с ними.

Некоторые архитектурные механизмы используют нестандартный подход к созданию объектов, требующий передачи строкового имени класса создаваемого объекта. Этот подход также может быть рассмотрен как вспомогательный механизм (или механизм более низкого уровня).

Все нижеописанные архитектурные механизмы были реализованы на Visual C++ 6.0. Список прилагаемых файлов-модулей приведен в Приложении Б.

## 3.1. Организация взаимодействия объектов

***Диспетчер сообщений.*** Для реализации механизма межобъектного взаимодействия был создан специальный объект – диспетчер. Диспетчер «знает» все объекты, которые желают обмениваться сообщениями, а эти объекты в свою очередь «знают» о существовании диспетчера. Помимо таблицы зарегистрированных объектов, важной частью диспетчера является очередь сообщений. Принцип отправки сообщения можно описать следующим образом (см. рис.3):

- Объект-отправитель создает сообщение и инициализирует его данными. Затем он вызывает функцию отправки сообщения (SendEvent) диспетчера.
- Диспетчер осуществляет постановку сообщения в очередь, проверяет, запущен ли поток обработки очереди сообщений. Если поток не запущен, то диспетчер его запускает. После этого управление возвращается объекту отправителю.

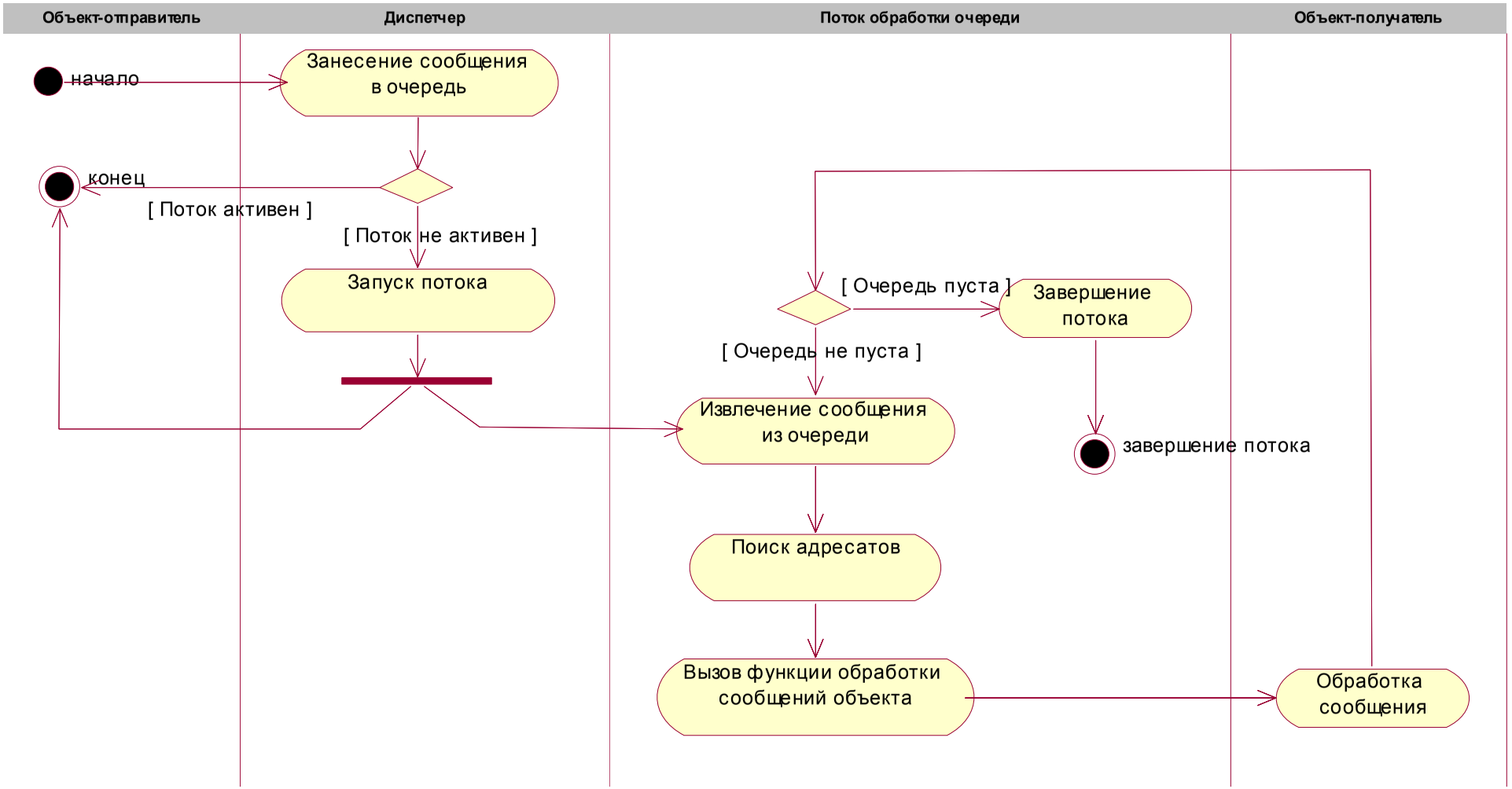


Рис.3. Диаграмма деятельности, отражающая принцип обработки сообщения.

- Поток обработки сообщений извлекает следующее сообщение из очереди, ищет объект-получатель по таблице зарегистрированных объектов и вызывает функцию обработки сообщения получателя (ProcEvent), передавая ей в качестве параметра объект сообщение. Когда управление возвращается диспетчеру, поток осуществляет проверку наличия сообщений в очереди. Если сообщения отсутствуют, то поток завершается, в противном случае поток осуществляет обработку сообщения.

Перед вызовом функции ProcEvent получателя осуществляется запуск таймера. Если к моменту таймаута управление не возвращено объекту диспетчера, то поток прерывается и запускается снова со следующего элемента очереди сообщений. Если же получатель знает, что время обработки сообщения превышает время таймаута, то он запускает свой поток и возвращает управление диспетчеру.

Очередь сообщений диспетчера представляет собой массив элементов сообщений. Сообщения в очередь укладываются последовательно. При достижении конца очереди следующее сообщение записывается на первое место, таким образом, очередь зациклена. Обработка очереди потоком прекращается, когда номер следующего обрабатываемого элемента равен номеру следующего добавляемого элемента, что означает, что в очереди отсутствуют сообщения.

Выделение объекта диспетчера с его очередью сообщений разрешает все проблемы синхронизации многопоточного приложения, при условии, что взаимодействие объектов осуществляется только через очередь обработки сообщений диспетчера.

Каждый взаимодействующий объект в системе характеризуется тремя значениями: номером класса, номером объекта и дополнительным кодом. Для уникальной идентификации объекта используются первые два значения. Номера объектов выдаются диспетчером последовательно, с заполнением пустот (т.е. объект занимает первый свободный номер). Дополнительный код призван выражать пользовательскую нумерацию объектов. Кроме того, он может быть использован для поиска объектов в системе (Seek, Select).

Трудоемкость поиска элемента по таблице взаимодействующих объектов является логарифмической от числа объектов, принадлежащих классу искомого. Это объясняется тем, что диспетчер работает с классами, как элементами массива фиксированной длины, а с объектами класса как с расширяемым массивом. Поиск по списку объектов осуществляется дихотомически.

Проведенное тестирование показало, что диспетчер способен поддерживать неограниченное количество объектов. Единственным «узким» местом может выступить очередь сообщений: поскольку она зациклена, то ее переполнение может закончиться потерей ряда начальных сообщений.

***Диспетчер событий.*** Для реализации возможности подписки одних объектов на события других диспетчер сообщений был расширен функциями диспетчера событий. Это выразилось в добавлении таблицы подписчиков и функций работы с событиями. Объект, желающий заявить о событии, создает объект-сообщение и заполняет часть его полей, касающихся события. После этого осуществляется вызов функции уведомления о событии (ThrowEvent). Эта функция находит всех подписчиков, и отправляет им сообщения о событии, удаляя их подписные записи из таблицы. Если объект желает получить событие снова, он должен опять на него подписаться.

Объекты имеют возможность подписываться на события монопольно (для этого в объекте сообщения предусмотрен соответствующий атрибут). При возникновении события диспетчер определяет наличие монопольных подписок, и рассылает сообщения, уведомляя подписчиков, была ли заявлена монопольная подписка и является ли он монопольным подписчиком. В зависимости от этого объект принимает решение о соответствующей реакции на событие.

***Диспетчер таймеров.*** Для снижения загруженности системы таймерами было решено организовать службу таймеров на основе диспетчера сообщений. Для этого были добавлены таблицы таймеров объектов. При запуске диспетчера автоматически запускается периодический таймер. После истечения каждого периода счетчик каждого объекта, заказавшего таймер (SetTimer), уменьшается на единицу. При достижении нулевого значения объекту посылается сообщение об истечении таймаута, при этом запись таймера не удаляется и может быть инициализирована снова повторным вызовом SetTimer. Для удаления записи таймера используется функция DeleteTimer.

Если вызывается деструктор объекта, зарегистрированного у диспетчера, то происходит удаление регистрации данного объекта, удаление всех его подписок (как входящих, так и исходящих), а также удаление всех его таймеров. Кроме этого организуется просмотр очереди сообщений диспетчера и удаление всех сообщений, направленных удаляемому объекту.

Трудоемкость поиска по таблице событий и таблице таймеров логарифмическая от числа всех элементов соответствующей таблицы. На рис.4 представлена диаграмма классов системы межобъектного взаимодействия.

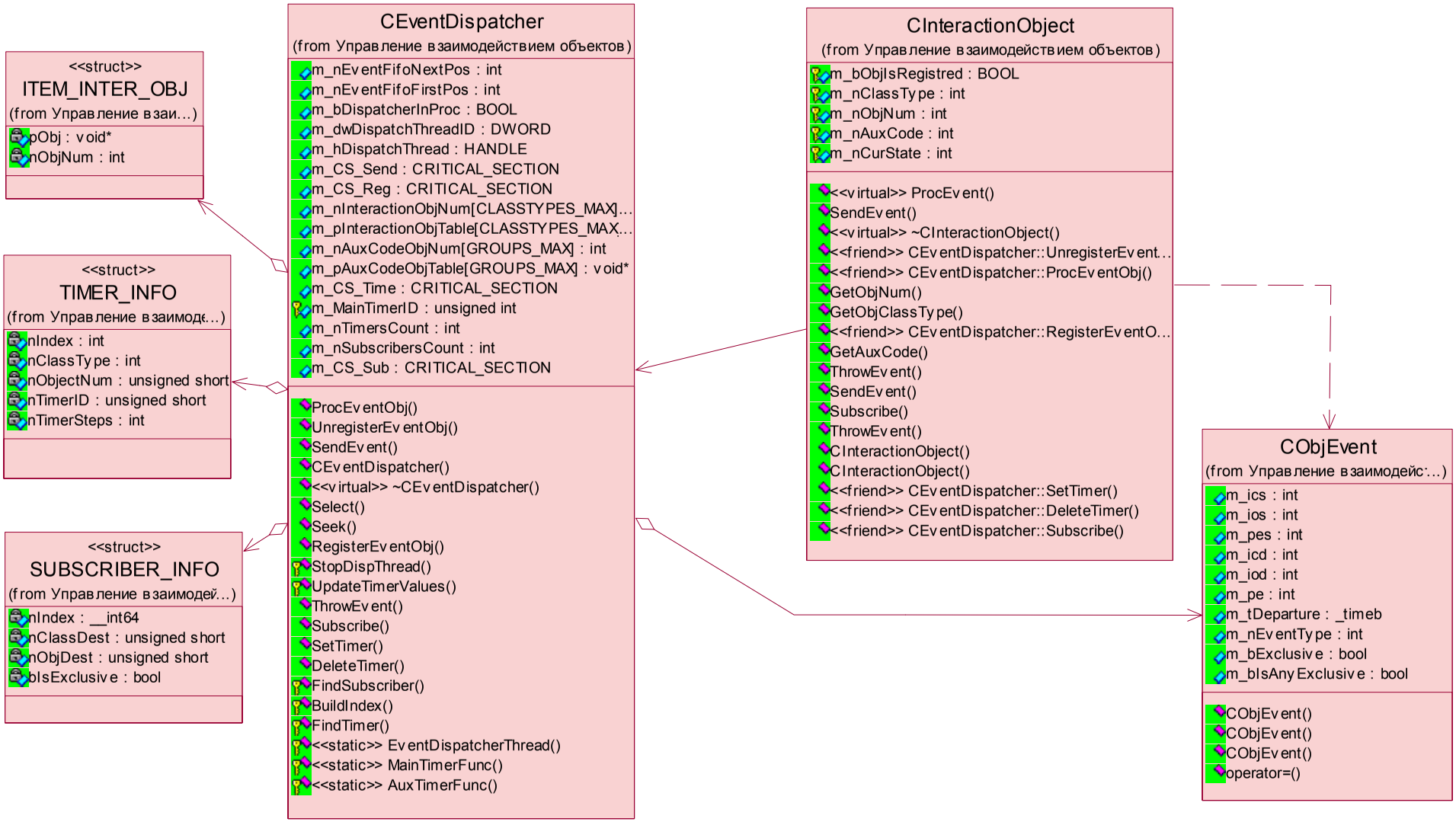


Рис.4. Диаграмма классов механизма межобъектного взаимодействия.

Класс CEventDispatcher реализует диспетчер событий, сообщений и службу таймеров. Класс CObjEvent представляет собой объект событие, содержащий информацию об отправителе, получателе, типе сообщения, структуре события и времени отправления. Класс CInteractionObject задает общий интерфейс для всех взаимодействующих объектов в системе. Структуры ITEM_INTER_OBJ, TIMER_INFO, SUBSCRIBER_INFO представляют, соответственно, записи в таблице объектов, таблице таймеров и таблице подписчиков. Данный механизм реализован в модулях InteracitonObject.cpp(h), EventDispatcher.cpp(h), ObjEvent.cpp(h).

## 3.2. Сохранение объектов

***Сохранение объекта в файл.*** Некоторые объекты системы требуют сохранения на диск и последующего восстановления. Для этого применяется механизм сериализации (преобразования объекта в последовательную форму и обратно). Принцип работы этого механизма следующий:

- При вызове функции сохранения в файл (Save) объект прежде всего записывает строку с именем своего класса. После этого происходит вызов функции сериализации (Serialize) с параметром «сохранить».
- В этой функции объект осуществляет запись на диск всех своих атрибутов. После этого объект считается сохраненным.
- При вызове функции загрузки объекта (Load) из файла считывается имя класса загружаемого объекта. Эта строка передается механизму параметризованного создания объектов, который осуществляет создание соответствующего объекта. После чего вызывается функция Serialize с параметром «загрузить».
- В функции Serialize происходит считывание из файла атрибутов сохраняемого объекта.

Таким образом, мы можем хранить в одном файле произвольное число объектов. Однако, чтобы получить доступ к какому-либо объекту, нам придется последовательно загрузить все предыдущие. Поэтому сериализацию удобно использовать лишь для сохранения состояния одиночных объектов, либо групп объектов, используемых одновременно.

На рис.5 представлена диаграмма классов, отображающая структуру данного механизма.

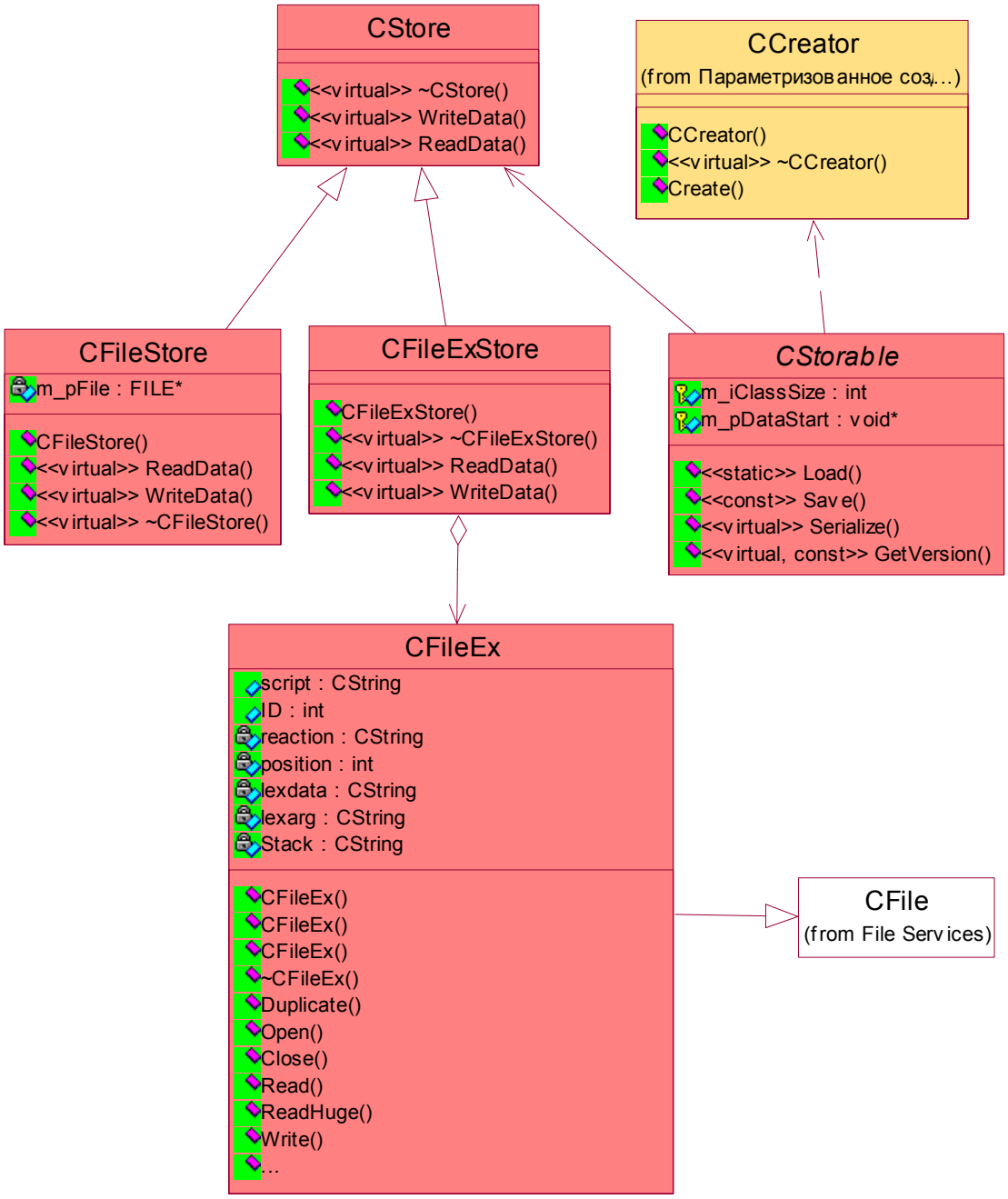


Рис.5. Диаграмма классов механизма сериализации.

Для использования этого механизма все классы, желающие, чтобы их объекты сохранялись, должны наследовать от класса, задающего интерфейс для сериализации (CStorable, модуль Storable.cpp(h)). Класс CStore задает интерфейс для использования различных типов файловых хранилищ. Например, класс CFileStore использует для работы с файлами функции API, а класс CFileExStore – функции класса CFileEx (реализация в модуле CStore.h). Кроме них могут быть использованы любые классы-обертки, обеспечивающие работу с файлами.

Этот механизм является полностью объектно-ориентированным, так как только сам объект знает порядок загрузки/сохранения своих атрибутов. Сериализация реализована и используется в стандартной библиотеке классов MFC. Однако MFC-реализация этого механизма может быть неприемлема в случаях, когда применяется множественное наследование или предполагается обеспечить кроссплатформенность приложений. В [11] описана альтернативная реализация этого подхода, без привлечения MFC. Она требует использования механизма параметризованного создания объектов, который будет описан отдельно. Кроме этого, требуется, чтобы компилятор умел определять информацию о классе объекта во время исполнения.

Чтобы ускорить процесс записи объекта в файл, было принято следующее решение. Вместо того чтобы записывать на диск по-очереди все свои атрибуты, объект записывает себя целиком путем копирования области памяти. Этот способ позволяет сэкономить на времени записи, но при этом копируются не только атрибуты, но и таблицы виртуальных функций объекта и всех его базовых классов. Однако таблицы виртуальных функций объекта меняются лишь от компиляции к компиляции. Данный подход может применяться лишь в случаях, когда необходимо проводить постоянные сохранения объектов.

***Склад.*** Сохранение в файл не позволяет нам организовать быстрый поиск и извлечение объектов. Для решения этой задачи был разработан специализированный класс CStock (см. рис.6), являющийся контейнером одно- или двуключевых записей и реализующий возможности сохранения и поиска объектов (модуль CStock.cpp(h)). Склад может хранить только объекты одного типа. Для этого он инициализируется именем класса хранимых объектов, положением и длиной ключей в записи, характеризующей объект.

Для помещения объектов на склад используется механизм, похожий на сериализацию. Каждый объект, желающий сохраняться на складе, должен иметь функции, задаваемые интерфейсом CStockItem (модуль CStockItem.cpp(h)). При добавлении объекта на склад, вызывается функция преобразования (Transform), которая по аналогии с функцией Serialize осуществляет запись атрибутов объекта и их извлечение. Однако в качестве хранилища атрибутов здесь выступает не файл, а строка (область памяти). Полученная строка добавляется в таблицу элементов и включается в индекс в соответствии со своим ключом (ключами).

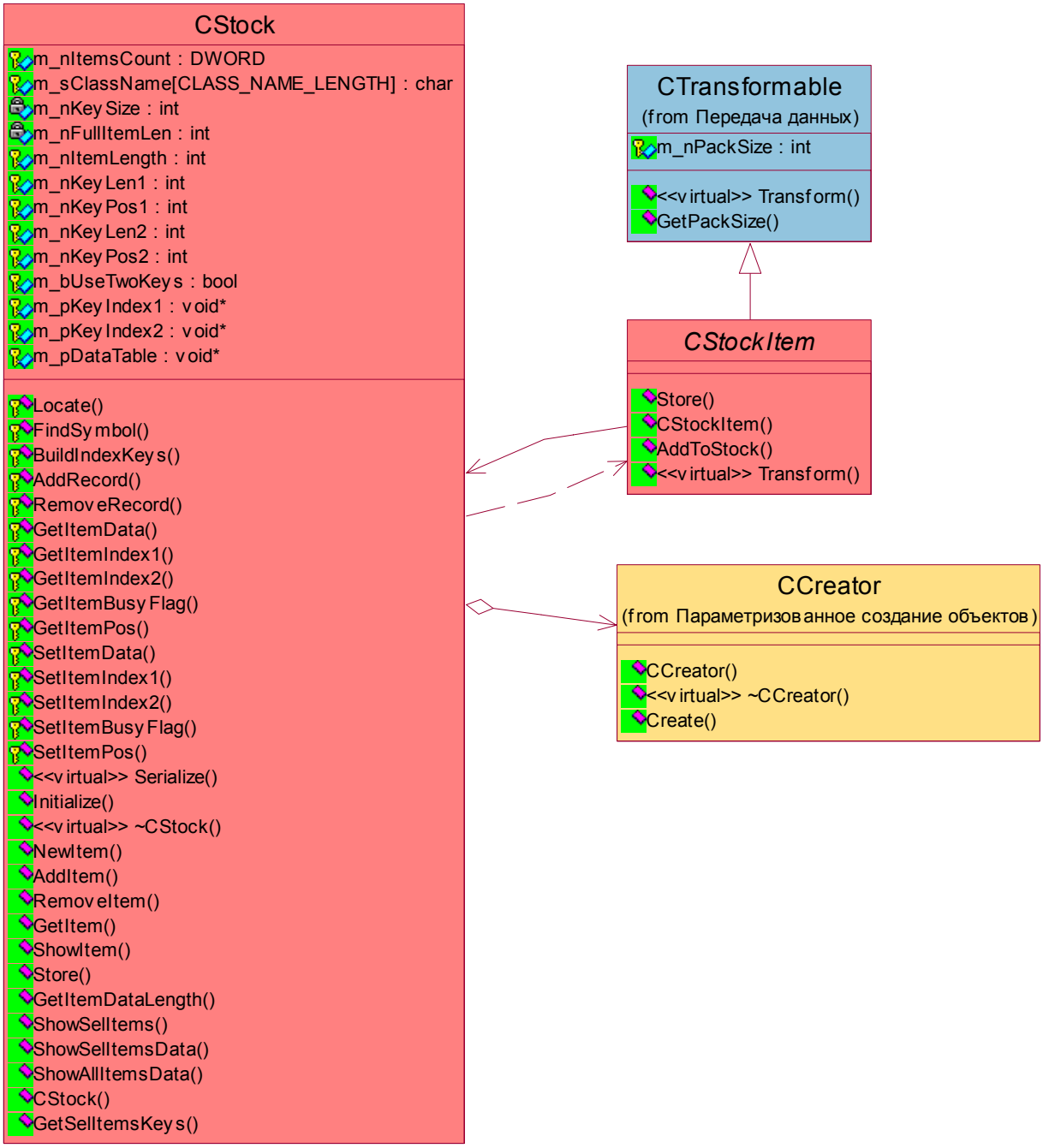


Рис.6. Диаграмма классов механизма складирования объектов.

Класс CStock имеет два индекса (по одному на каждый ключ), которые используются для быстрого поиска элементов. Поиск элементов по индексам осуществляется с помощью дихотомии, с использованием лексикографического сравнения, что дает возможность использования в качестве ключей любых типов, в том числе и строк.

Со склада можно извлекать либо один элемент, характеризуемый двумя ключами, либо все элементы, имеющие одинаковые значения какого-либо одного ключа.

Реализованы возможности извлечения элемента со склада с блокировкой записи и без блокировки. Для создания объектов извлекаемых элементов используется механизм параметризованного конструирования объектов.

## 3.3.Параметризованное создание объектов

Для параметризации создаваемого объекта именем класса использовался механизм, описанный в шаблоне проектирования «Абстрактная фабрика» (см. [12]). Применяемый подход также описан в [11].

Класс CAbstractFactory задает общий интерфейс для создания продукта (объекта) различными фабриками объектов. Класс CFactory инициализируется создаваемым продуктом и реализует интерфейс абстрактной фабрики. Класс CFactoryListItem представляет собой элемент списка фабрик. Класс CCreator осуществляет перебор списка фабрик для создания конкретного продукта, определенного переданным ему параметром. Диаграмма классов представлена на рис.7

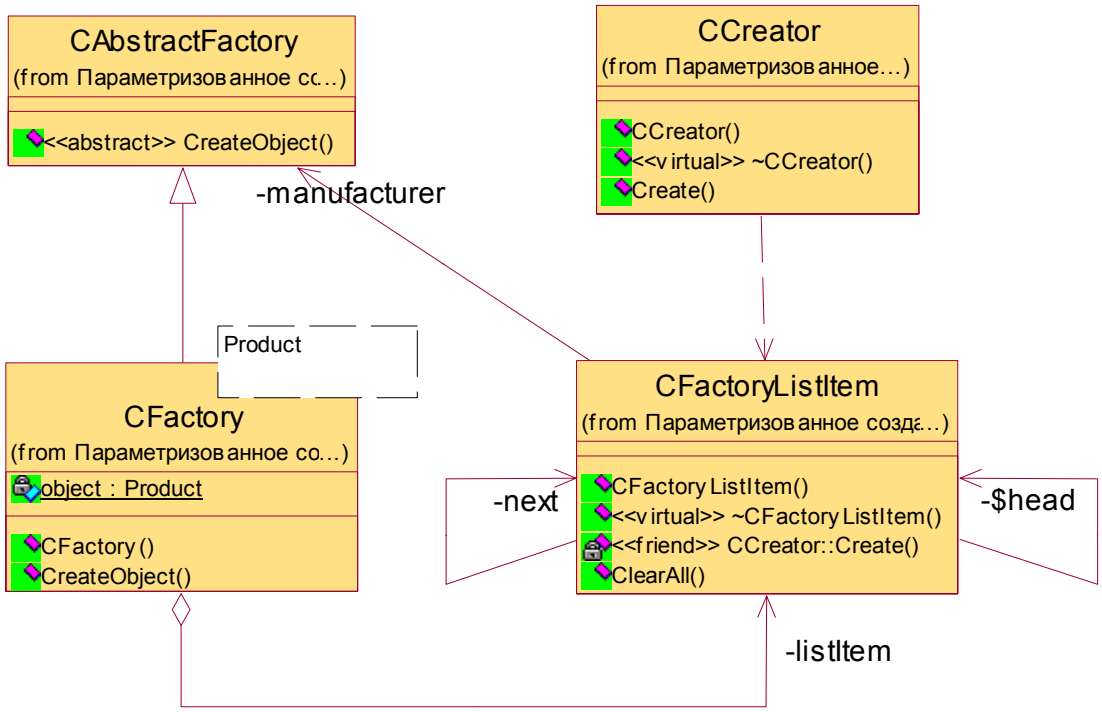


Рис.7. Диаграмма классов механизма параметризованного создания объектов.

Все описанные классы реализованы в модуле dynamic.cpp(h)

Общая последовательность действий выглядит следующим образом:

- Создается новый объект класса CFactory, параметризованный классом объекта-продукта. При этом автоматически осуществляется создание нового элемента CFactoryListItem и его добавление к существующему списку фабрик.
- Вызывается функция Create класса CCreator, которой, в качестве параметра, передается строка, содержащая имя класса создаваемого объекта. Класс CCreator осуществляет просмотр списка фабрик и вызывает функцию создания продукта для каждой фабрики.
- Фабрика проверяет, совпадает ли переданный параметр с именем класса ее продукции, и в зависимости от этого создает либо нет объект. Для создания объекта используется конструктор по-умолчанию.
- Как только очередная фабрика возвращает непустое значение, класс CCreator прерывает просмотр списка фабрик и, в свою очередь, возвращает созданный объект. Если ни одна фабрика не смогла создать запрашиваемый объект, то возвращается пустой указатель.

Для корректной работы с продуктами, использующими множественное наследование, необходимо осуществить динамическое преобразование созданного объекта к его типу. Объекты-фабрики можно создавать единожды при старте программы и уничтожать при ее завершении. С другой стороны, если параметризованное создание объектов производится редко, то фабрики можно создавать и удалять по мере необходимости.

## 3.4. Работа с файлами

Необходимость обеспечения гибкой и надежной работы с файлами в механизмах сохранения объектов требует разработки специальной системы для обработки особых ситуаций – исключений. Они представляют собой некоторые события, которые прерывают нормальный процесс выполнения программы. Многие стандартные библиотеки учитывают возможность возникновения исключений, но обработкой их должен заниматься непосредственно разработчик.

Стандартная библиотека классов Microsoft (MFC) предоставляет особый класс для работы с файлами CFile, который умеет генерировать исключения при работе с файлами. Но, при его прямом использовании в коде программы, придется «защищать» операторами обработки исключения каждую файловую операцию, что приведет к непомерному росту

объема программы. Это послужило стимулом для написания собственного класса работы с файлами, который является наследуемым от CFile и дополняет его специальными возможностями.

Помимо непосредственной защиты самих операций программисту - пользователю предоставлена возможность самому назначать функцию обработки для каждого файлового объекта или воспользоваться процедурой обработки по умолчанию. Для обеспечения более эффективного и гибкого управления исключениями был создан специальный язык. При возникновении исключения в режиме интерпретации будет выполняться специальная программа на этом языке - управляющий скрипт.

Разработанный язык позволяет однозначно сопоставить набору стимулов (причин исключения) последовательность действий, которые необходимо выполнить. Причем необходимые действия или последовательности действий могут повторяться любое число раз. Такая конструкция языка является вполне достаточной: количество стимулов конечно, и, сопоставив каждому стимулу последовательность действий, мы определим реакцию системы на все возможные файловые исключения для данного файлового объекта. Использование последовательности, а не единого действия, повышает гибкость языка за счет конструирования сложных действий на основе простых. Операция повторения позволяет строить сложные реакции на основе неоднократного выполнения блоков действий: например, ожидание освобождения файла при совместном его использовании и пр. Для стимулов, требующих одинаковой обработки, предусмотрена возможность сопоставления реакции их набору.

Поскольку специфика исключений дает возможность однозначно определять их причину, поиск имени процедуры, определяющей реакцию на стимул, был вынесен в функции транслитератора. Это, помимо ускорения обработки скрипта, упростило и сам язык. Он перешел, без потери функциональности, к одному из стандартных типов – «списку с одной ассоциативной операцией» со следующей q –грамматикой (более подробно описано в [13]):

$$< S > \rightarrow a < S >$$

$$< S > \rightarrow * < A > < S >$$

$$< S > \rightarrow \varepsilon$$

$$< A > \rightarrow a$$

$$< A > \rightarrow (< S >)$$

В качестве операции (*) в нем выступает повторение действий – элементов списка (***a***), а ***ε*** – пустая цепочка. Рассмотрим типичный пример управляющей программы обработки исключений:

shareViolation = 5*(ask ,wait ,tryAgain), notify, terminate;

При возникновении исключения ***shareViolation*** (ошибка совместного использования) пять раз выполнится блок функций: ***ask*** – спросить пользователя о необходимости ожидания, ***wait*** – подождать определенное время, ***tryAgain*** – попробовать снова открыть файл. После неудачи на пятой попытке вызывается функция ***notify***, которая уведомит пользователя о невозможности открыть файл и, наконец, функция ***terminate***, которая завершит работу программы.

Использование разработанного языка позволяет нам, помимо создания набора стандартных действий, предоставлять пользователю возможность регистрации своих собственных функций обработки исключений, что обуславливает дальнейшую расширяемость класса и приспособление его к любым потребностям.

Представление управляющего скрипта в виде строки разрешает нам подменять реакцию на файловые исключения прямо во время исполнения программы. Кроме этого, при наличии хорошо разработанной библиотеки функций, можно организовать систему, загружающую управляющий скрипт из файла и таким образом изменять схему обработки файловых исключений без рекомпиляции программы.

Данный механизм реализован в модуле FileEx.h

## 3.5. Взаимодействие приложений

***Команды.*** Архитектура «клиент-сервер» подразумевает, что приложения-клиенты посылают запросы серверу, а тот их выполняет. Часто к серверу выдвигаются требования организации протоколирования этих запросов. Для реализации подобной схемы был использован шаблон проектирования «Команда». Он инкапсулирует запрос как объект, позволяя тем самым задавать параметры клиентов для обработки соответствующих запросов, ставить запросы в очередь или протоколировать их, а также поддерживать отмену операций.

Все команды имеют общий интерфейс, задаваемый классом CCommand (см. рис.8). В этом классе определена виртуальная функция Execute, которая инициирует выполнение

команды. Такой подход позволяет выполнить команду, не зная, какая это команда и что она должна сделать.

Если требуется выполнить не одну команду, а несколько, то можно определить класс CMacroCommand как наследник класса CCommand, дополнив его операциями добавления и удаления команды и переписав операцию Execute таким образом, чтобы она осуществляла последовательное выполнения списка команд.

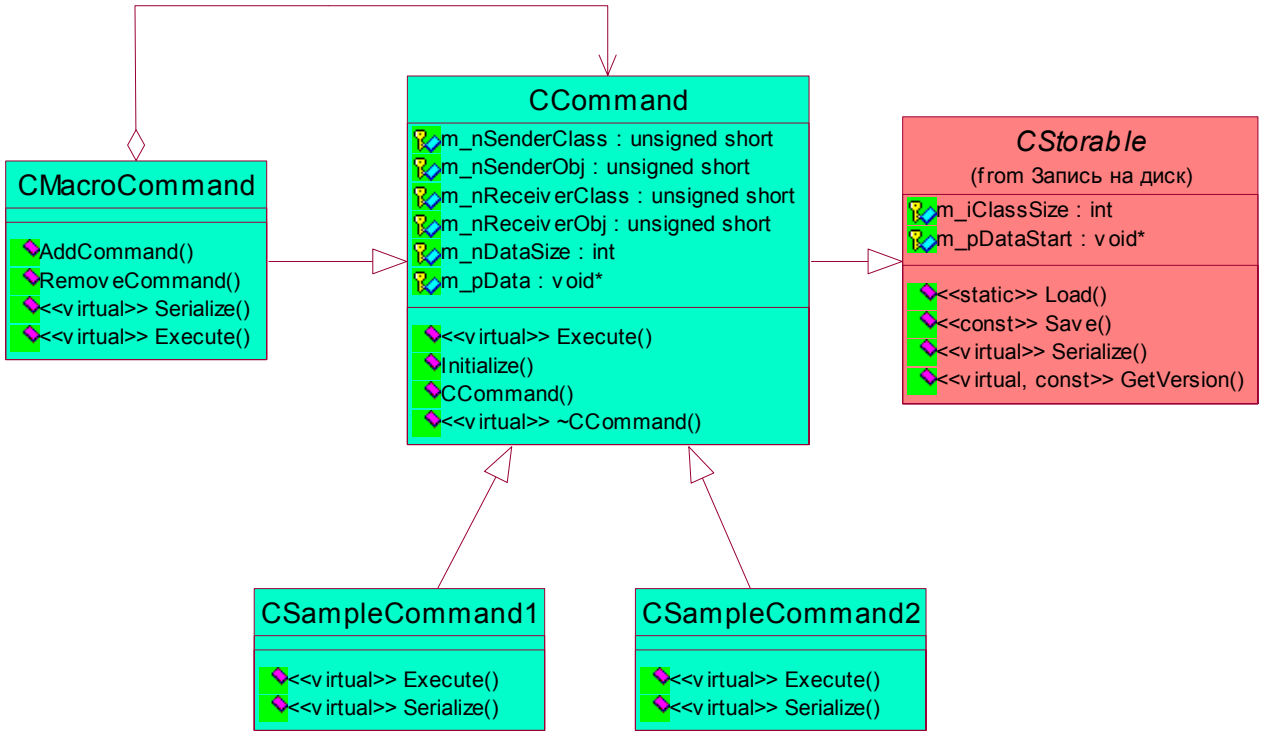


Рис. 8. Диаграмма классов механизма команд.

Для того чтобы организовать протоколирование команд воспользуемся механизмом сохранения объектов в файл, описанным выше. Для этого унаследуем класс CCommand (модуль Command.h) от класса CStorable. Это позволит осуществить последовательную запись команд в файл. Протокол может быть использован для восстановления состояния системы в случае сбоя.

***Передача объектов.*** Сервер и клиенты производят обмен данными. Часто эти данные представляют собой различные объекты. Для того чтобы организовать передачу объекта от одного приложения другому, можно воспользоваться механизмом, аналогичным сериализации, и осуществлять преобразование объекта в строку. Тогда весь процесс передачи объекта будет выглядеть следующим образом:

- Приложение-отправитель создает объект, инициализирует его данными и вызывает функцию преобразования в строку (Transform).
- Полученная строка передается приложению-получателю вместе с именем класса объекта ( или другим идентификатором, определяющим объект).
- Получатель создает объект и вызывает функцию преобразования, передавая ей полученную строку.

Использование этого механизма предполагает, что и отправитель и получатель знают о классе передаваемых объектов. Существенным преимуществом же является тот факт, что только сами объекты знают, как себя упаковать. Это позволяет не разрабатывать дополнительный формат пакета для передачи каждого объекта и облегчает возможности замены объектов и их повторного использования, поскольку изменить потребуется лишь логику функций класса объекта.

Для того чтобы иметь возможность унифицированного преобразования в строку был разработан класс CTransformable (модуль Transformable.h). Этот класс также был использован в механизме сохранения объектов на склад (см. рис.6).

# 4. Архитектура приложения-сервера СКУД

Используя вышеописанные принципы, была разработана схема для представления архитектуры приложения-сервера СКУД. На рисунке 9 отражены пакеты, определяющие функциональную группировку классов, и показаны зависимости между ними.

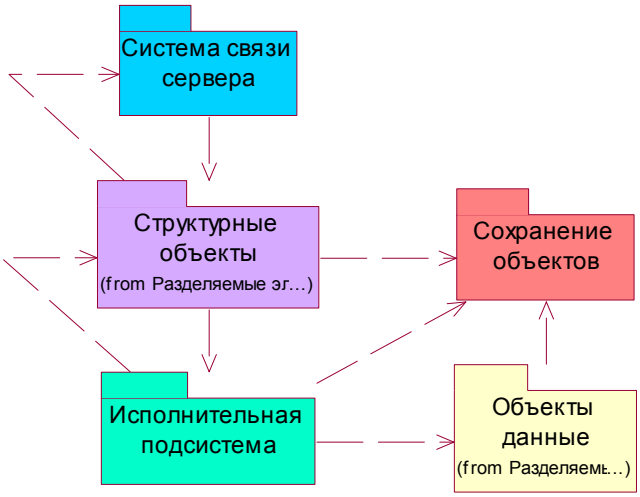


Рис.9. Пакеты и связи приложения-сервера СКУД.

- «Система связи сервера» - пакет, отвечающий за связь клиентов и сервера, а также сервера и оборудования СКУД, такого как турникеты, датчики и т.д.. Он организует прием входящих сообщений и отправку исходящих. Входящие сообщения интерпретируются как события, их публикация осуществляется от лица структурных объектов.
- Пакет «Структурные объекты» содержит объекты, являющиеся агентами внешних элементов системы: контроллеров, приложений-клиентов и других. Агенты используют систему связи для общения с внешним миром.
- Пакет «Объекты-данные» осуществляет группировку ряда пакетов, представляющих такие элементы предметной области, как карты, счета, группы, ресурсы и персоны.
- Пакет «Исполнительная подсистема» отвечает за выполнение запросов от приложений и обработку событий контроллеров. Он представляет собой адаптированную к системе реализацию механизма команд. Команды работают как с объектами-данными, так и со структурными объектами.

- Пакет «Сохранение объектов» показывает, каким образом сервер использует механизмы сохранения объектов. Этот пакет используется командами, структурными объектами и объектами-данными.

## 4.1.Пакет «Система связи сервера»

Рассмотрим состав (см. рис.10) и принципы работы системы связи приложения-сервера СКУД.

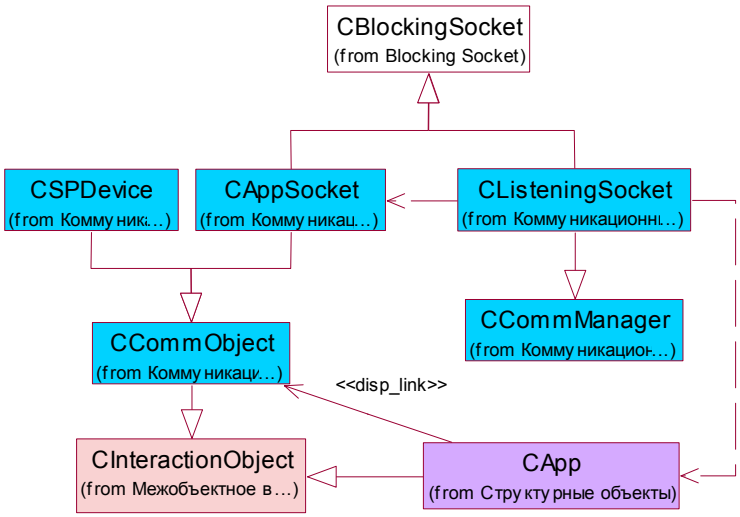


Рис.10. Пакет «Система связи сервера».

Класс CCommManager задает общий интерфейс для приема входящих соединений. При приеме соединения создается объект класса CCommObject, через который и осуществляется дальнейшее взаимодействие с подсоединившимся приложением. В данном случае в качестве коммуникационных объектов были выбраны сокеты (для их объектно-ориентированного представления используется класс CBlockingSocket, являющийся оберткой функций API Winsock). Однако в случае необходимости сокеты могут быть заменены любым другим средством межпроцессного взаимодействия (каналами, файловыми очередями и др.).

Кроме коммуникационного объекта создается структурный объект, представляющий собой образ приложения в системе (CApp). Образ приложения знает реквизиты своего коммуникационного объекта и может общаться с ним через диспетчер, что отражено на диаграмме стрелкой с пометкой «disp_link».

Связь с контроллерами обеспечивается при помощи связной платы, подключенной к компьютеру. Класс CSPDevice представляет в системе драйвер связной платы. Он тоже

наследует интерфейс коммуникационного объекта. Такой подход позволяет унифицировать обработку внешних сигналов в системе.

Каждый пакет данных, полученный коммуникационным объектом, считается системным событием. Например, событие «вставлена карта» от контроллера или событие «новый клиент» от приложения. Данные об объекте-источнике и типе события заложены в формат пакета. Используя эти сведения, коммуникационный объект осуществляет публикацию события от лица источника.

Коммуникационный объект осуществляет отправку только готовых пакетов, упаковкой же данных в пакет со стороны сервера занимается образ приложения CApp.

## 4.2.Структурные объекты

Все структурные элементы системы контроля доступа имеют специальных агентов, представляющих интересы данных элементов внутри программы-сервера. Взаимодействие между агентами и другими объектами сервера происходит с помощью диспетчера (интерфейс CInteractionObject). Для обеспечения надежности системы структурные объекты сохраняются в файл (интерфейс CStorable). В случае необходимости можно передать объект со всеми его данными другому приложению в виде строки (интерфейс CTransformable).

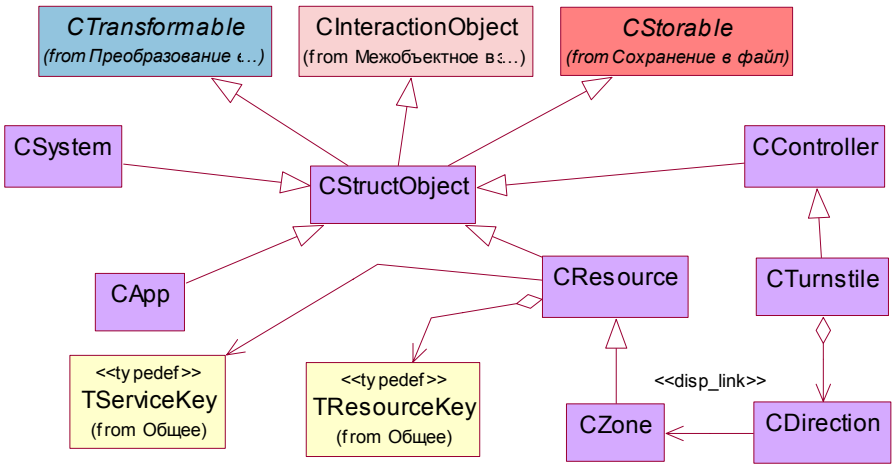


Рис.11. Пакет «Структурные объекты».

На рис.11 представлены классы пакета «Структурные объекты». Класс CSystem представляет собой систему и является источником системных событий. Класс CApp является источником для объектов-агентов приложений, подключенных к серверу. Класс CController задает общий интерфейс для различных типов контроллеров, а CResource для

различных ресурсов. Таким образом, наследуя от этих классов, мы получаем возможность создания не только СКУД, но и некоторых других систем автоматизации маркетинга.

Для ресурсов-помещений в качестве контроллера должен использоваться класс турникета (CTurnstile), а класс CZone в качестве ресурса. Для конкретного класса CTurnstile важно направление доступа: с направлением доступа связана соответствующая зона-ресурс. Все ресурсы в системе последовательно перенумерованы для организации возможности ограничения доступа конкретного пользователя к выбранному ресурсу (TResourceKey). Ключи используются для организации поиска связанных элементов на складах.

## 4.3. Объекты-данные

В отличие от структурных элементов эти объекты не регистрируются у диспетчера, не общаются между собой, они пассивны. Это просто хранилища данных о некоторых элементах предметной области, таких как группы, персоны, карты и счета. Все объекты-данные хранятся на складах, поэтому для каждого такого элемента выделяется ключ – уникальный идентификатор, с помощью которого объект можно найти на складе. Помимо списков объектов предметной области, хранящихся на складах в виде одноключевых записей, используются списки отношений. Они хранят двуключевые элементы, представляющие связи объектов предметной области.

Объекты-данные можно разделить на два подпакета: «Клиенты» и «Счета».

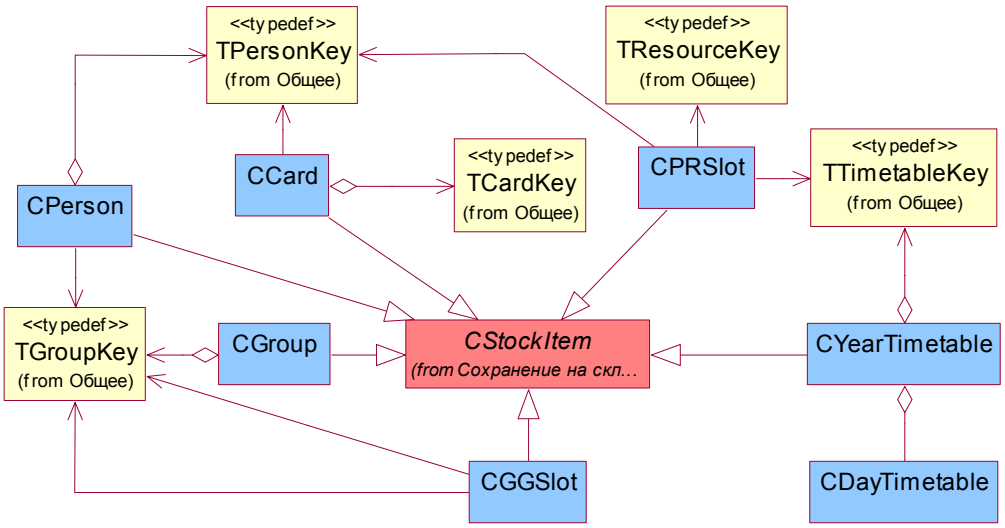


Рис.12. Пакет «Клиенты» из пакета «Объекты-данные».

Пакет «Клиенты» (см. рис.12) задает логическую структуру данных о клиентах-пользователях системы. Так, класс CPerson представляет собой сведения о персонах, CGroup – о группах, CCard – о картах. Класс CGGSlot задает возможность ассоциаций групп путем установления связи между ними. Для определения прав доступа конкретной персоны к определенному ресурсу используется отношение CPRSlot. Наличие элемента этого отношения говорит о возможности доступа конкретного клиента к определенному ресурсу с использованием расписания. Класс CDayTimetable представляет собой расписание доступа на один день, а CYearTimetalbe – на год.

Другой пакет – «Счета» (см. рис.13) задает реализацию объектов-счетов (CAccount), схем поощрения (CBonus) и услуг (CService). Класс CGASlot задает связь между счетами и группами, а класс CPASlot – связь между группами и персонами. Описание основных элементов предметной области представлено в §2.3.

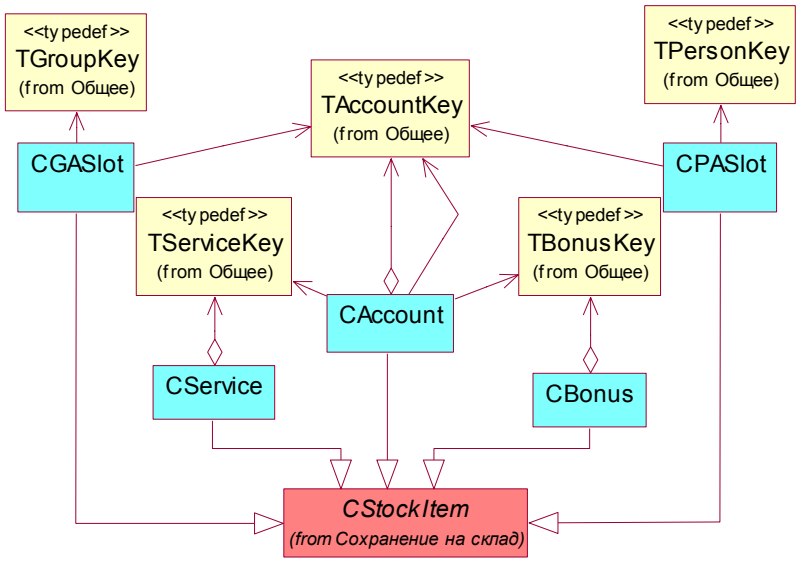


Рис.13. Пакет «Счета» из пакета «Объекты-данные».

Все классы, представляющие одноключевые записи, обладают возможностью генерировать уникальные в пределах системы ключи своего типа.

## 4.4.Исполнительная подсистема

Исполнительная подсистема сервера построена на основе механизма команд, с учетом использования диспетчера и требований протоколирования запросов (см. рис 14).

Класс CCommandHandler отвечает за создание и запуск команд в системе. Он подписан на ряд событий и имеет таблицу, сопоставляющую имя класса конкретной

команды определенному событию. В случае возникновения события обработчик команд проверяет, зарегистрирована ли за этим событием команда. Если соответствие найдено, то осуществляется создание экземпляра соответствующей команды с помощью механизма параметрического создания объектов. После создания команды происходит ее инициализация входными данными, на основе которых команда определяет своего получателя. Запуск команды осуществляется с помощью отправки ей сообщения.

Каждая команда имеет своего получателя (объект, который должен быть уведомлен о результате выполнения команды). Когда необходимая последовательность действий выполнена, команда осуществляет запись необходимых данных в файл протокола, отправляет результат получателю и завершается. Команды, требующие длительных вычислений, могут запускаться в отдельном потоке.

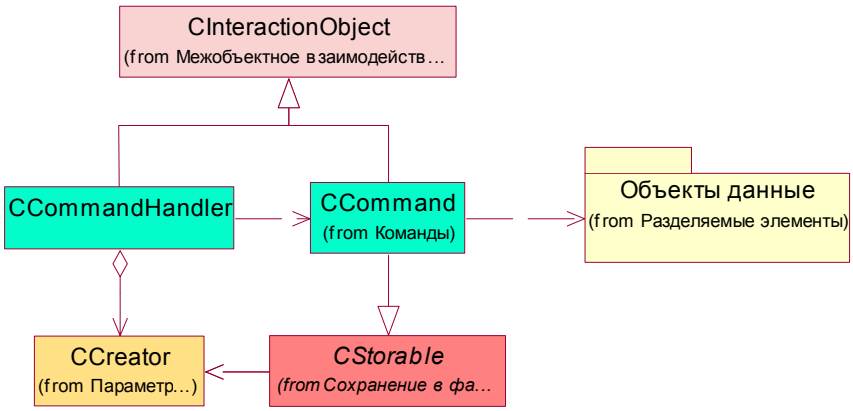


Рис.14. Пакет «Исполнительная подсистема».

Команды являются активными объектами системы, содержащими всю логику запросов приложений. Каждый объект-команда регистрируется у диспетчера и поэтому может осуществлять обмен сообщениями с другими объектами, подписываться на события и т.д.

Использование механизма параметрического создания объектов позволяет строить обобщенные команды. Например, вместо специфичных команд для создания каждого пассивного объекта (объекты-данные) можно написать единую команду. Для этого необходимо лишь передать код склада и строку, содержащую упакованный объект. Склад автоматически создаст объект нужного класса, а общий интерфейс преобразования позволит наполнить его данными, после чего объект можно благополучно добавить на склад. Применяя аналогичный подход, мы получим команды, реализующие запросы «Получить», «Демонстрировать», «Модифицировать», «Удалить» и др. Можно обобщить команды не только для объектов данных, но и для структурных объектов.

Наравне с обобщенными командами присутствуют и специфические команды. Например, команда «Новая группа» требует не только создания объекта «группа», но и объекта «персона», представляющего администратора данной группы. Вообще специфичные команды применимы в тех случаях, где семантика предметной области предполагает выполнение нескольких действий, либо единственное действие требует особого внимания.

Подход с использованием команд удобен тем, что позволяет простым образом добавить новую команду или модифицировать одну из существующих.

## 4.5.Сохранение объектов

Этот пакет показывает, каким образом приложение-сервер СКУД использует механизмы сохранения объектов (см. рис.15).

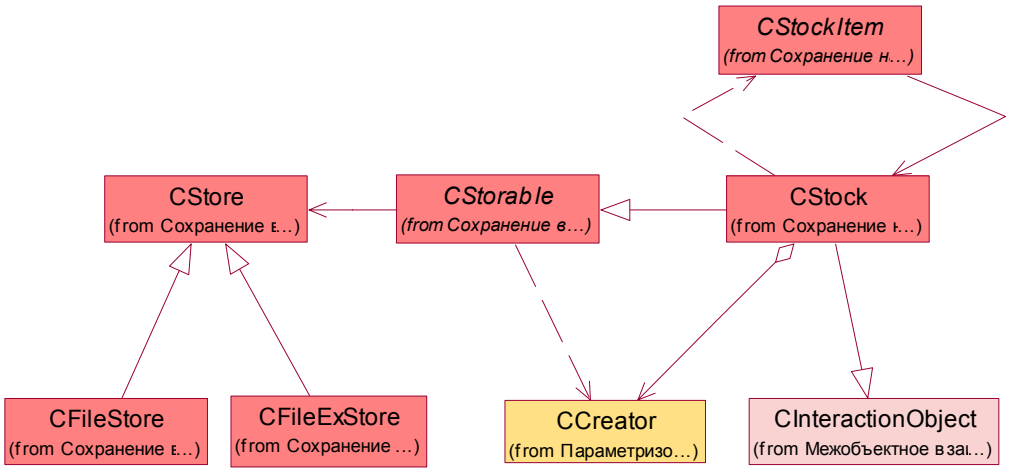


Рис.15. Пакет «Сохранение объектов».

Различные типы объектов в системе используют различные механизмы сохранения.

- Структурные объекты «живут» в системе постоянно. Они зарегистрированы у диспетчера, общаются с другими объектами, подписываются на события. Их «образ жизни» не позволяет им храниться на складе, поэтому для сохранения они используют сериализацию в файлы.
- Объекты-данные, в отличие от структурных объектов, многочисленны и пассивны. Они лишь хранят информацию о соответствующих элементах предметной области и не требуют регистрации у диспетчера. При работе с ними большую важность приобретает механизм поиска, поэтому для их сохранения и используется склад.

Для того чтобы обеспечить доступ к своим записям, склад должен быть зарегистрирован у диспетчера. Наследование же интерфейса сериализации позволяет организовать сохранение множества объектов на диск более удобным и быстрым способом.

- Объекты-команды, являющиеся носителями логики запросов приложений. Эти объекты, так же как и структурные, являются «живыми», но «живут» они временно. Для их сохранения применяется сериализация в отдельные файлы. А после того, как команда выполнилась, организуется ее сохранение в общий файл протокола.

# 5. Архитектура приложения-клиента СКУД

Несмотря на то, что различных типов приложений может быть несколько, можно выделить и описать их общую часть, каркас приложения. Разработанная архитектура предполагает, что приложение-клиент, так же как и сервер, является многопоточным. Поэтому для обеспечения межобъектного взаимодействия используются те же механизмы, которые применялись в сервере. В каркас приложения были включены следующие пакеты (см. рис.16):

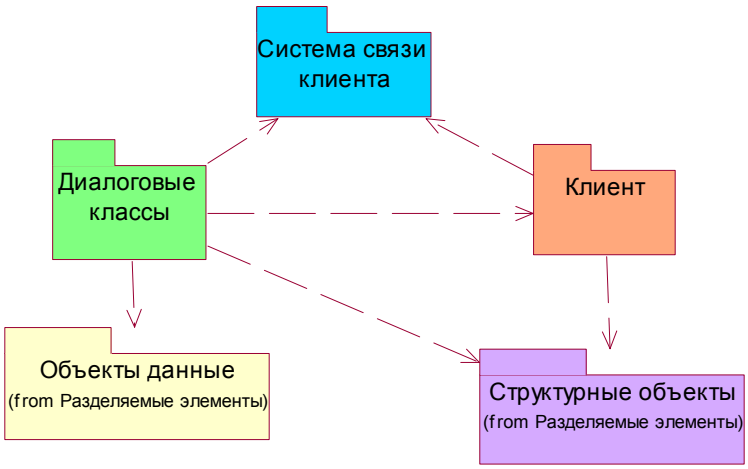


Рис.16. Пакеты и связи приложения-клиента СКУД.

- «Система связи клиента» - обеспечивает прием и публикацию информации от приложения-сервера.
- «Клиент». Этот пакет состоит из единственного класса, представляющего собой само приложение клиент. Он первым инициализируется при создании приложения и отвечает за установление связи между сервером и клиентом.
- «Диалоговые классы» - объединение классов, занимающееся визуализацией и представлением информации, полученной от сервера. Сюда входят как основной диалог, так и классы «помощников» (wizard), обеспечивающие возможность построения необходимых запросов приложению-серверу.
- С пакетами «Объекты-данные» и «Структурные объекты» мы уже познакомились при рассмотрении архитектуры сервера. Присутствие их в приложении объясняется используемым способом передачи объектов от приложения приложению, который описан в архитектурных механизмах.

## 5.1. Система связи приложения-клиента

Система связи клиента использует те же коммуникационные объекты, что и сервер (см рис.17). Это оказалось возможным благодаря унифицированной работе коммуникационных объектов. Как уже говорилось при рассмотрении сервера, коммуникационный объект расценивает все всходящие сообщения как события и осуществляет их публикацию. Тип сообщения и источник события заданы в формате пакета. Если источник события не указан, то коммуникационный объект производит публикацию от своего имени. Для отправки же сообщений используются лишь готовые пакеты.

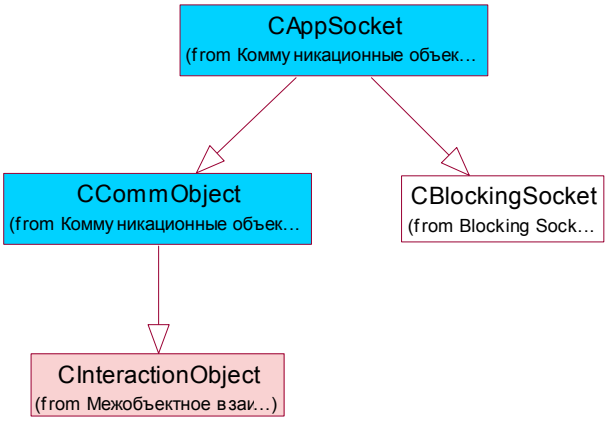


Рис.17. Пакет «Система связи клиента».

В качестве реализации коммуникационного объекта мы рассматриваем сокеты.

## 5.2. Пакеты «Клиент» и «Диалоговые классы»

Так как пакет «Клиент» состоит лишь из одного класса, мы рассмотрим его вместе с пакетом «Диалоговые классы» (см. рис.18). Для того чтобы интегрировать класс приложения (CClientApp) в структуру СКУД, мы наследуем его от структурного класса CApp. Это автоматически заставляет его регистрироваться у диспетчера. В связи с тем, что класс CClientApp является первым классом, создаваемым приложением, вся начальная инициализация помещена в его функцию InitInstance. Здесь создается диспетчер, коммуникационный объект, инициализируется функция подключения к серверу, устанавливаются начальные подписки и т.д.

Класс CDialogClass является основным окном в системе. Он использует интерфейс CInteractoionObject для получения необходимых событий, после чего отображает

произошедшие изменения. Параллельно с этим классом могут существовать другие диалоговые классы, отображающие любую информацию.

Приложение-клиент может содержать также диалоговые классы «помощников» (CWizard), которые шаг за шагом позволят пользователю внести необходимые изменения (создать группу, добавить нового члена группы и т.д).

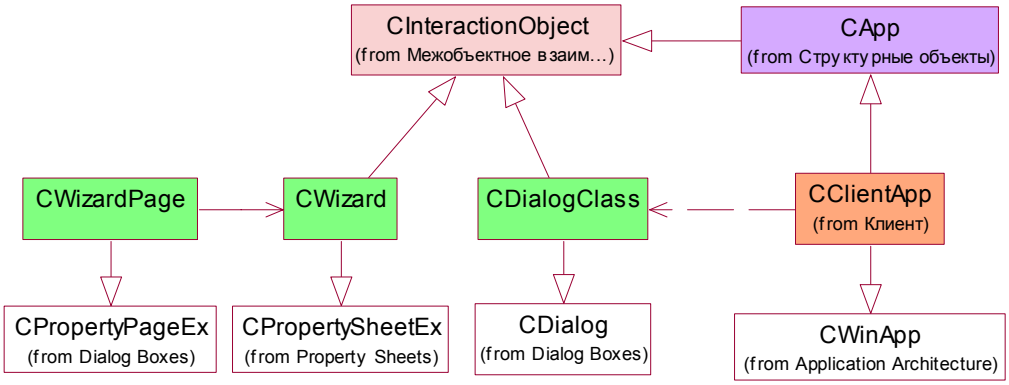


Рис.18. Пакеты «Клиент» и «Диалоговые классы».

Для того чтобы отображать необходимую информацию, диалоговые классы должны «знать» о структурных объектах и объектах-данных.

# 6. Принципы взаимодействия клиента и сервера

## 6.1. Установление связи

Схематичная последовательность действий объектов при установлении связи изображена на рис.19 Данная диаграмма не отображает наличие диспетчера как организатора межобъектного взаимодействия.

После запуска сервера, во время инициализации объектов создается экземпляр класса CCommManager, который ожидает входящего соединения. При запуске приложения-клиента создается коммуникационный объект CCommObject, который пытается соединиться с сервером. Экземпляр класса CCommManager обнаруживает попытку соединения, создает коммуникационный объект класса CCommObject, и принимает на него входящее соединение. После этого он создает структурный объект-приложение (CApp), составляет и отправляет пакет, уведомляющий приложение об успешном соединении. Этот пакет получает объект-приложение (CClientApp), который отправляет серверу регистрационное требование в виде особого запроса (команды).

На это событие подписан обработчик команд сервера. Он инициализирует и запускает соответствующую команду. Команда, выполняясь, загружает в соответствующий структурный объект CApp регистрационную информацию приложения, определяет его тип и устанавливает необходимые подписки. После этого она отправляет уведомление о регистрации получателю, в качестве которого выступает этот же структурный объект CApp. Структурный объект создает пакет и передает его коммуникационному объекту. Коммуникационный объект обеспечивает передачу данных приложению.

Приложение, получив уведомление о регистрации, активирует свои основные функции, запрашивает у сервера с помощью команд необходимые данные и т.д.

## 6.2. Выполнение запроса

Схематичная последовательность действий представлена на рис.20.

Один из диалоговых классов приложения отправляет через коммуникационный объект запрос и подписывается на ответ. Например, это может быть запрос на получение объекта из какого-нибудь склада. Коммуникационный объект сервера публикует полученный запрос как некоторое событие. На это событие подписан объект обработчик команд, который создает, инициализирует и запускает соответствующую команду.

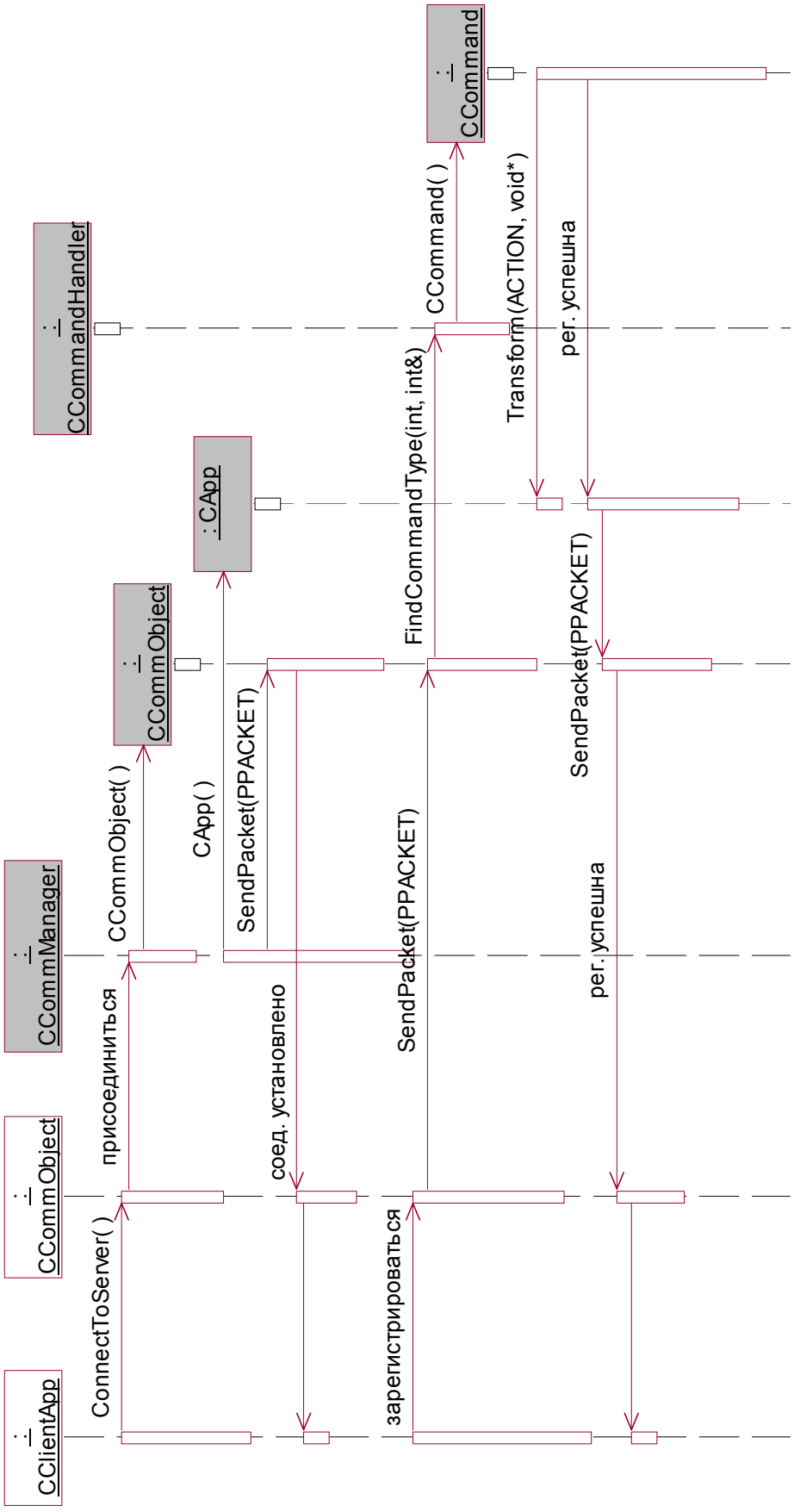


Рис.19. Диаграмма последовательности для регистрации агента приложения на сервере.

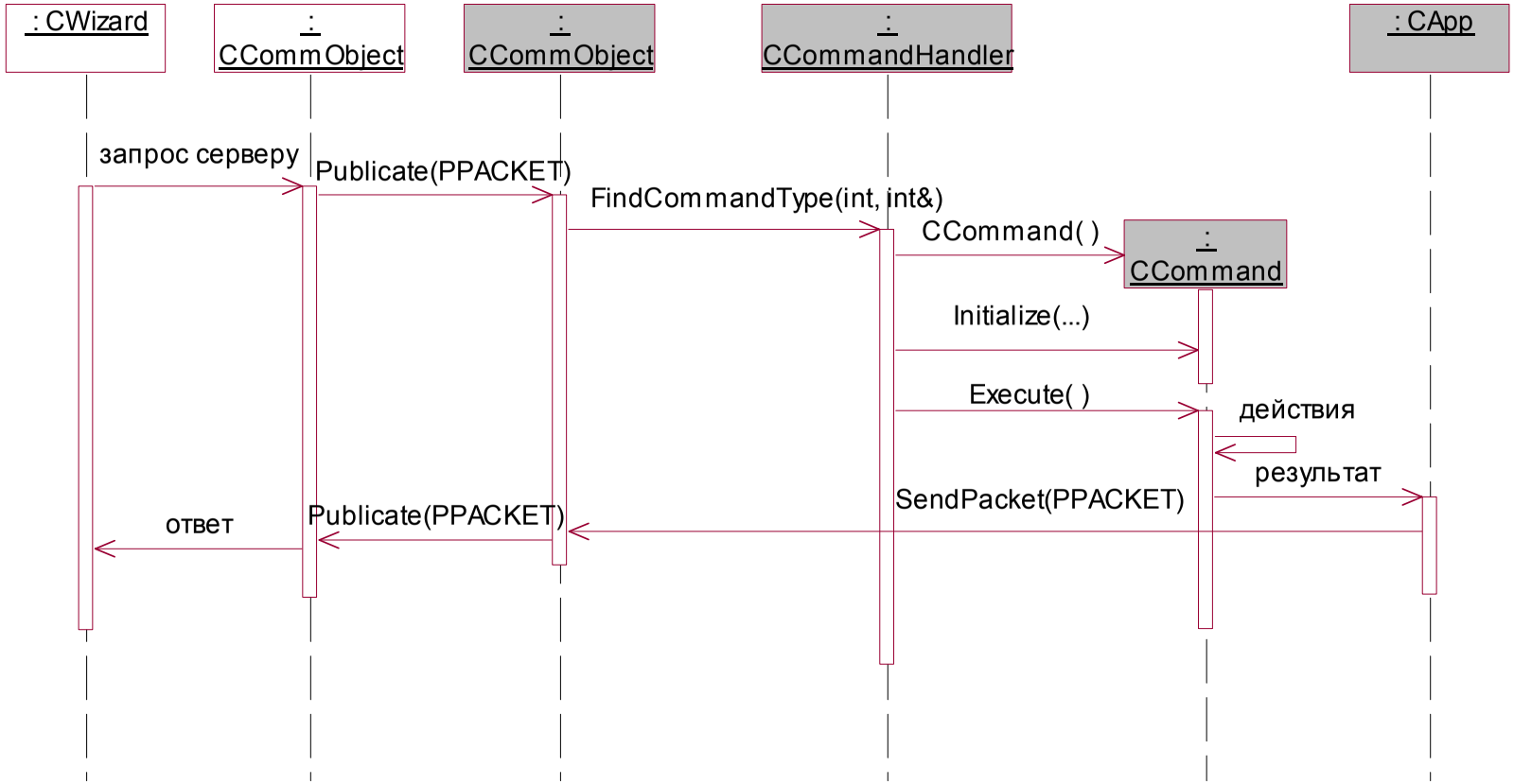


Рис.20. Диаграмма последовательности отправки запроса на сервер.

Команда, выполняясь, находит необходимый склад, извлекает из него объект, преобразует его в строку и отправляет ее получателю, в качестве которого выступает агент приложения. Агент осуществляет упаковку данных в пакет и, через коммуникационный объект, отправляет пакет приложению. Коммуникационный объект приложения публикует событие ответ на команду. Диалоговый класс получает сообщение о событии и осуществляет необходимую обработку.

## 6.3. Подписка приложения на события

Во время регистрации агент приложения подписывается на специфические для данного приложения события. Например, для менеджера клиентов это могут быть события: создание новой группы, добавление члена в группу и др. Как только такое событие происходит, агент о нем уведомляется. При этом он осуществляет упаковку полученной информации в пакет и отправляет его через коммуникационный объект приложению . Коммуникационный объект приложения публикует данное событие. Все подписчики уведомляются и производят необходимую его обработку. Например, отображение новой группы в списке групп.

На рисунке 21 изображена соответствующая диаграмма последовательности

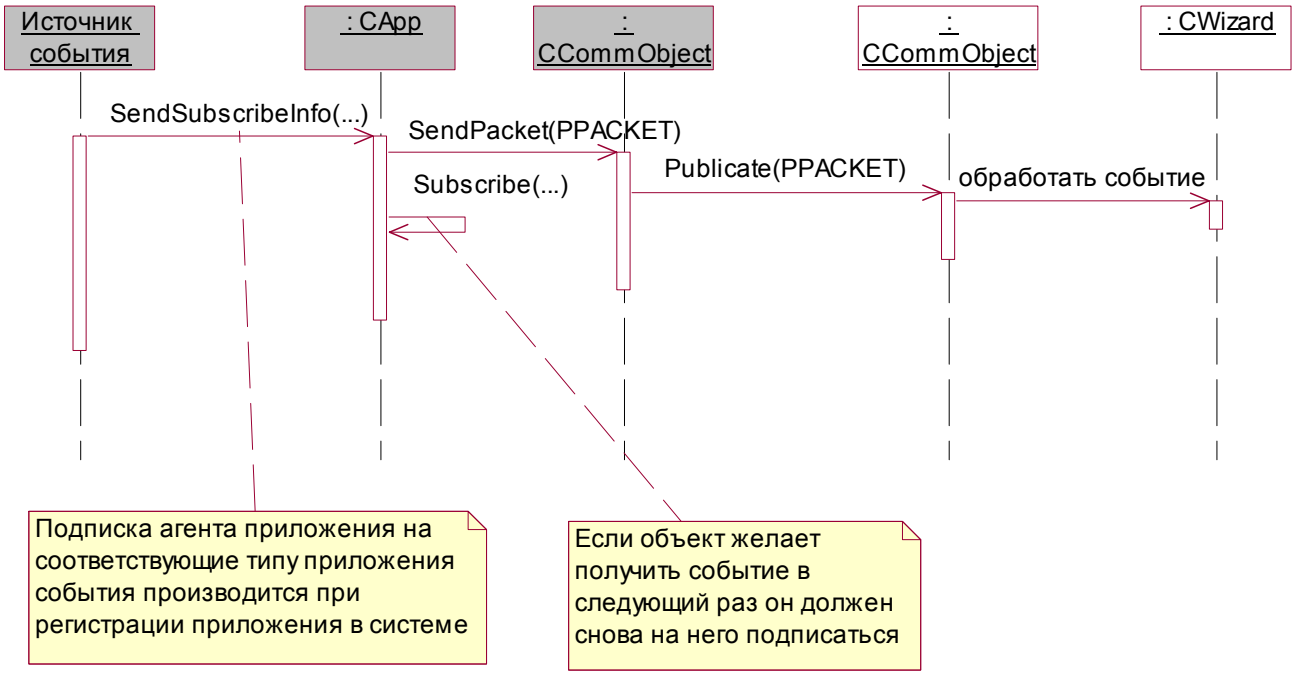


Рис.21. Диаграмма последовательности, отражающая получение подписки.

# Заключение

В результате выполненных работ была создана гибкая архитектура системы контроля доступа, поддерживающая схему расчетов пользователей за использование ресурсов. Реализованы на языке C++ и протестированы все используемые архитектурные механизмы. Получены следующие результаты:

- Проведено системное исследование предметной области. На его основе была построена соответствующая модель, учитывающая схему расчетов клиентов за предоставленные услуги;
- Разработаны, реализованы и протестированы основные архитектурные механизмы (межобъектное взаимодействие, сериализация в файл, сохранение на склад, параметризованное создание объектов, команды, механизмы взаимодействия приложений)
- Разработана архитектура модулей программного обеспечения и созданы соответствующие структурные модели этих модулей.
- Реализованы тестовые приложения, подтверждающие жизнеспособность как разработанной архитектуры в целом, так и правомерность применения предложенных механизмов.

Практическая ценность работы состоит в следующем:

- Разработанная архитектура модулей программного обеспечения системы контроля доступа может быть использована в качестве базы для построения целого ряда систем автоматизации маркетинга.
- Созданные механизмы могут быть повторно применены в любом проекте, выдвигающем соответственные требования.

## Список использованных источников


1. Энциклопедия UML. - Режим доступа: http://ooad.asf.ru/standarts/uml/spr/Architecture.asp, свободный.
2. О Системах управления доступом. - Режим доступа: http://www.gamma.kz/gt/sud.html, свободный.
3. Карточки – идентификаторы, для систем контроля доступа. - Режим доступа: http://www.avtolik.ru/access/systems/identifikotor.htm, свободный.
4. Давлетханов М. Внимание, чужой, 2003. - Режим доступа: http://daily.sec.ru/dailypblshow.cfm?rid=5&pid=6637&pos=1&stp=50, свободный.
5. Принципы работы систем контроля доступа. - Режим доступа: http://www.secret-c.ru/uphp/default.php?id=41, свободный.
6. ГОСТ Р 51241-98 Средства и системы контроля и управления доступом. Классификация. Общие технические требования. Методы испытаний. Введен 01.01.2000. - Режим доступа: http://www.nist.ru/hr/doc/gost/51241-98.htm, свободный.
7. Гильманов А.А., Клименко А.Я., Странгуль О.Н., Тарасенко В.П. Карточные технологии в автоматизации маркетинга. – Томск: Издательство НТЛ, 2000. –380 с.
8. Жданов А.А. Современный взгляд на ОС реального времени. - Режим доступа: http://asutp.interface.ru/articles/display_topic_threads.asp?ForumID=13&TopicID=299, свободный.
9. Унифицированный язык моделирования. - Режим доступа: http://penguin.photon.ru/doc/uml.shtml, свободный.
10. Якобсон А., Буч Г., Рамбо Дж. UML: специальный справочник – СПб, Питер, 2001
11. Holub A. Roll your own persistence implementations to go beyond MFC frontier// Microsoft Systems Journal, 1996, June issue.(MSDN Library 2001)
12. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования – СПб: Питер, 2001. – 386 с.
13. Льюис Ф., Роэенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. -М.: Мир, 1979. - 654 с.

14. Трофимов С.А. CASE-технологии: Практическая работа в Rational Rose.-М.: Бином-Пресс, 2002 г. - 288 с.

# ПРИЛОЖЕНИЕ А. Генерация заголовков классов по модели

Моделирование системы контроля и управления доступом осуществлялось на универсальном языке моделирования (UML), с помощью CASE-средства Rational Rose 2001.Об использовании UML в Rational Rose можно прочитать в [14].

Файл модели acs.mdl содержит необходимые диаграммы классов. Для осуществления генерации кода заголовков классов необходимо:

1. Создать компоненту в модели реализации.
2. Выбрать необходимый язык программирования.
3. Ассоциировать классы с необходимой компонентой.
4. В контекстном меню компоненты выбрать позицию «Update code».
5. В мастере обновления кода выбрать классы, требующие обновления.
6. Нажать кнопку «Finish».

Rational Rose произведет генерацию классов по информации из модели. Для языка Visual C++ будут сгенерированы и добавлены в проект заголовочные файлы (*.h) и файлы реализации (*.cpp)

При необходимости Rational Rose позволяет осуществить обновление модели по коду. Для этого:

1. В контекстном меню компоненты выбрать позицию «Update Model».
2. Нажать кнопку «Finish».

Rational Rose произведет загрузку существующих классов проекта в модель.

# ПРИЛОЖЕНИЕ Б. Список прилагаемых файлов

.\model\acs.mdl – Файл модели Rational Rose.

.\arcmech\dispatcher\: Механизм сохранения объектов:

- EventDispatcher.cpp(h) – реализация класса CEventDispatcher;
- InteractionObject.cpp(h) – реализация класса CInteractionObject;
- ObjEvent.cpp(h) – реализация класса CObjEvent.

.\arcmech\saveobj\: Механизм сохранения объектов:

- CStorable.cpp(h) – реализация класса CStorable;
- CStore.cpp(h) – реализация классов CStore, CFileStore, CFileExStore;
- FileEx.h – реализация класса CFileEx;
- CStock.cpp(h) – реализация класса CStock;
- CStockItem.cpp(h) – реализация CStockItem.

.\arcmech\param\: Механизм параметризованного создания объектов:

- Dynamic.cpp(h) – реализует классы CCreator, CAbstractFactory, CFactory, CFactoryListItem.

.\arcmech\app\: Механизм взаимодействия приложений:

- Transformable.cpp(h) – реализация CTransformable;
- Command.cpp(h) – реализация CCommand.

.\middleware\

- Blocksock.cpp(h) – реализация классов CBlockingSocket, CBlockingSocketException, CSockAddr.

.\archtest\: тестовые проекты.

- ClientManager – тестовое приложение-клиент;
- Server – тестовое приложение-сервер.

# ПРИЛОЖЕНИЕ В. Дискета с исходными текстами