Оглавление

[Введение 3](#_Toc275255687)

[1. Исследовательская часть 4](#_Toc275255688)

[1.1. Анализ существующих систем 5](#_Toc275255689)

[1.1.1. ICQ 5](#_Toc275255690)

[1.1.2. Windows Live Messenger 5](#_Toc275255691)

[1.1.3. Skype 6](#_Toc275255692)

[1.1.4. Google Talk 6](#_Toc275255693)

[1.1.5. IRC 6](#_Toc275255694)

[1.1.6. Веб-чаты 7](#_Toc275255695)

[1.2. Подведение итогов обзорного анализа 8](#_Toc275255696)

[1.3. Вывод 9](#_Toc275255697)

[2. Конструкторская часть 12](#_Toc275255698)

[2.1. Терминология 12](#_Toc275255699)

[2.2. Общая структура системы 13](#_Toc275255700)

[2.3. Обзор протокола ENCP 14](#_Toc275255701)

[2.4. Структура клиентской программы 16](#_Toc275255702)

[2.5. Структура серверной программы 17](#_Toc275255703)

[2.6. Каналы управления (бизнес-процессы) 19](#_Toc275255704)

[2.6.1. Подключение нового участника конференции 19](#_Toc275255705)

[2.6.2. Обмен сообщениями 21](#_Toc275255706)

[2.6.3. Синхронизация клиентского списка 22](#_Toc275255707)

[2.7. Спецификация датаграмм ENCP 22](#_Toc275255708)

[3. Технологическая часть 23](#_Toc275255709)

[4. Заключение 24](#_Toc275255710)

[5. Приложение А 25](#_Toc275255711)

[6. Список литературы 26](#_Toc275255712)

# Введение

Хотим сделать чат (почему?)

# Исследовательская часть

Многопользовательские сетевые конференции появились достаточно давно (так, протокол IRC был создан в 1988 году), и к настоящему времени существует множество реализаций, с различными протоколами и функциональностью. Наиболее популярные системы общения включают:

* **Windows Live Messenger** (бывший MSN Messenger)
* **Skype**
* **Google Talk**
* **IRC** (Internet Relay Chat)
* **Веб-чаты**
* **Веб-чаты**
* и т.д

Общая функциональность, присутствующая в любой реализации онлайн-чата, включает:

* Список пользователей, отображающий всех людей, в данный момент подключенных к серверу;
* Возможность отправлять текстовые сообщения, видимые как всем пользователям, так и одному выбранному собеседнику (личные сообщения);
* Ведение истории сообщений, полученных в текущей сессии, и ее отображение в окне клиентской программы

Используя информацию, представленную в **[1]**, проведем обзорный анализ перечисленных выше реализаций.

## Анализ существующих систем

### ICQ

Являясь одной из самых популярной программ для моментального обмена сообщениями через Интернет (службой активно пользуется около 15 миллионов человек, а зарегистрированных учётных записей около 400 миллионов)[[1]](#footnote-1), программа **ICQ** реализует закрытый фирменный протокол **OSCAR**. Протокол обеспечивает такую функциональность, как многопользовательские беседы, передачу файлов и онлайн-игры. Хотя фирма-владелец **ICQ** старается пресечь попытки создать альтернативные клиенты для своего протокола, в частности, постоянно внося в него изменения, существует ряд альтернативных клиентов (программы **QIP**, **Miranda IM**), которые чуть ли не более популярны, чем официальный клиент (особенно на территории России и стран СНГ).

### Windows Live Messenger

Данный продукт является клиентским приложением для протокола **Microsoft Notificaion Protocol (MSNP)** от компании **Microsoft**. Приложение поддерживает как стандартный дуплексный режим общения двух пользователей, так и возможность объединения пользователей в группы и проведения многопользовательской текстовой конференции.

Одним из плюсов используемого протокола является возможность передачи нарисованных в области ввода текста изображений, что существенно облегчает общение, если необходимо показать собеседнику какую-нибудь формулу или схему. Кроме того, **MSNP**, в отличие от **OSCAR**, является открытым протоколом, что позволяет создавать для него альтернативные клиенты.

### Skype

**Skype** – приложение и протокол, предназначенные прежде всего для IP-телефонии. **Skype** очень популярен в России, так как  поддерживает возможность бесплатного видео- и голосового общения через Интернет. **Skype** поддерживает возможность создания многопользовательской конференции (аудио, видео, текст), но при этом кол-во пользователей конференции ограниченно числом 25.

Протокол **Skype** является закрытым, и исходный код официального клиента не публикуется.

### Google Talk

Программа мгновенного обмена сообщениями, разработанная компанией **Google**.

**Google Talk** позволяет общаться с помощью голосового чата и текстовых сообщений. Особенностью является тесная интеграция с почтовой службой **Gmail** (например, по **Google Talk** приходят уведомления о новых сообщениях). Для использования **Google** **Talk** обязательно наличие учётной записи **Gmail**.

**Google Talk** основан на открытом протоколе **XMPP**, но исходный код официального клиента не публикуется.

### IRC

**IRC** *(Internet Relay Chat — ретранслируемый интернет-чат)* — одна из старейших систем многопользовательских текстовых конференций (появилась в 1988 г.)

**IRC** предоставляет возможность как группового, так и личного общения. Для группового чата в **IRC** предназначены каналы, на которых пользователи могут собираться и вести общение.

Система **IRC** очень популярна, но ориентирована исключительно на текстовое общение.

Протокол **IRC** является открытым и описан в **RFC 1459**. Сам протокол содержит множество ограничений: длина сообщений ограничена 510 символами; текст передается в 8-битной кодировке (стандарт не определяет способ передачи текста в многобайтной кодировке); отсутствует поддержка передачи файлов и т.п. Подобная отсутствующая функциональность реализуется клиентскими приложениями поверх протокола.

IRC, в отличие от перечисленных выше систем общения, не требует регистрации пользователя в системе; при подключении к серверу достаточно ввести псевдоним (ник).

### Веб-чаты

Описанные выше протоколы многопользовательских конференций реализуются в клиентских приложениях, устанавливаемых на компьютер пользователя. Существуют также так называемые веб-чаты, для участия в которых необходимо лишь наличие веб-браузера. Взаимодействие с сервером в таких чатах основано на протоколе HTTP.

Первоначально веб-чаты представляли собой страницу с разговором, содержащую форму, посредством которой введённый текст отсылался на сервер. Сервер добавлял новые сообщения в текстовую область, удалял старые и обновлял файл; сама страница регулярно перезагружалась. Такой чат осуществлялся с заметной задержкой в десятки секунд, из-за того, что сервер не мог сам сообщить клиенту об изменениях — клиент мог только запрашивать данные сам с некоторой периодичностью, обновляя страницу целиком или - позднее - отправляя асинхронные HTTP-запросы.  Другие системы чатов не имели такого недостатка, поскольку клиентские приложения могли устанавливать дуплексное соединение с сервером.

Веб-чаты завоевали популярность, отчасти благодаря отсутствию необходимости установки специального клиентского приложения на компьютер пользователя. Стоит отметить, что различные веб-чаты не совместимы между собой, так как не существует общего стандарта на реализацию протокола веб-чата поверх HTTP.

Большинство веб-чатов не требуют регистрации пользователя в системе.

К веб-чатам можно отнести продукт **Google Wave**, работающий в веб-браузере. **Wave** - экспериментальное средство общения, призваное соединить в себе функции электронной почты, мгновенной передачи сообщений, чата, веб-форума, вики, совместного создания и редактирования документов с системой управления версиями и социальной организацией сети. Компания **Google** позиционировала свой продукт как "убийцу электронной почты", чем несколько дезориентировала публику; в результате **Wave** так и не получил широкого применения.

## Подведение итогов обзорного анализа

Проведенный обзорный анализ позволяет составить следующий перечень функций, которые могут присутствовать в реализациях многопользовательских конференций:

* передача текстовых сообщений;
* передача личных сообщений в дополнение к групповому общению;
* передача файлов;
* передача изображений, нарисованных пользователем в программе-клиенте;
* голосовая и видео-связь в реальном времени;
* многопользовательские игры.

Клиентские приложения могут быть реализованы как:

* полновесные приложения, устанавливаемые на компьютер пользователя;
* веб-страницы, не требующие установки.

В различных реализациях присутствуют следующие требования или ограничения, накладываемые на участников общения:

* необходимость регистрации в системе для создания аккаунта, хранящего личную информацию о пользователе;
* необходимость использования только официального клиента (в случае, если протокол является закрытым);
* необходимость использования исключительно специального клиентского приложения или же исключительно браузера для доступа к системе;
* ограничение на число участников конференции (например, в Skype - 25).

## Вывод

Для реализации в рамках данного курсового проекта решено создать продукт со следующими параметрами:

* передача текстовых сообщений;
* передача личных сообщений в дополнение к групповому общению;
* передача изображений, нарисованных пользователем в программе-клиенте
  + Из всех перечисленных выше вариантов приложений только **Windows Live Messenger** обеспечивает эту возможность; в то же время она незаменима при передаче формул или схем;
* использование в качестве клиентского приложения как веб-страницы, так и устанавливаемой на компьютер пользователя программы;
* отсутствие необходимости регистрации пользователя в системе
  + Одним из минусов многих приложений, представленных выше, является необходимость наличия аккаунта, никак не связанного с услугами чата. Требование регистрации может служить препятствием в ситуациях, когда необходимо срочно организовать конференцию, участники которой ранее не работали с данной системой и не имеют аккаунтов. Система **IRC** не требует регистрации, что является одной из причин ее популярности (хотя отдельные каналы могут вводить обязательную регистрацию);
* отсутствие жестких ограничений на число участников конференции
  + **Skype** устанавливает ограничение в 25 участников, не всегда оправданное ограничениями технологий связи. Столкновение с такими ограничениями, как правило, приводит к разочарованию пользователей в системе, поэтому предполагается, что количество пользователей будет ограниченно только ресурсами сервера.

Отличительной чертой разрабатываемого продукта будет возможность одновременного редактирования несколькими пользователями одного изображения, отсутствующая в перечисленных выше популярных продуктах.

В целях обеспечения высокой скорости отклика веб-клиента разрабатываемый протокол будет основан на технологии **WebSocket [2]**, позволяющей веб-странице установить полнодуплексное соединение с сервером поверх **TCP**. Использование этой технологии позволит ликвидировать такой недостаток веб-чатов, как относительно длительные задержки обновления. **WebSocket** также легко реализуется в обычных приложениях; это, в конечном счете, позволит пользователю выбирать, какой тип клиентской программы ему удобнее – веб-клиент или обычное приложение.

Таким образом, в данном курсовом проекте будут разработаны:

* протокол многопользовательской конференции с поддержкой передачи изображений;
* серверное приложение, работающее по данному протоколу;
* клиентские приложения, работающие по данному протоколу – веб-клиент и обычное настольное приложение.

# Конструкторская часть

## Терминология

Перечислим термины, использующиеся при описании структуры и функционирования разрабатываемой системы (синонимы и взаимозаменяемые слова перечисляются через запятую):

* Онлайн-конференция, конференция, чат (chat) - способ сетевого общения, позволяющий нескольким участникам обмениваться сообщениями в реальном времени
* Участник конференции, участник чата - пользователь, подключившийся к конференции при помощи клиентской программы и участвующий в общении.
* Серверная программа, сервер - центральный компонент разрабатываемой системы, реализованный в виде отдельного приложения и обеспечивающий взаимодействие клиентских программ, подключающихся к нему.
* Клиентская программа, клиент - приложение, выполняющееся на компьютере пользователя, подключающееся к серверной программе и предоставляющее пользователю графический интерфейс для общения в чате.
* Веб-приложение - программа, как правило, написанная с использованием HTML и JavaScript, и выполняющаяся в среде веб-браузера, обмениваясь в процессе работы данными со своей серверной частью.
* Текстовый чат - элемент конференции, позволяющий участникам обмениваться текстовыми сообщениями.
* Лог чата - элемент графического интерфейса текстового чата, отображающий тектстовые сообщения и системные уведомления в порядке их поступления. Не путать с логом работы сервера.
* Лог работы сервера - файл, в который сервер записывает все происходящие события и поступающие от клиентов датаграммы. Не путать с логом чата.
* Доска рисования - элемент конференции, представляющий собой поверхность для рисования и набор инструментов рисования. Изменения, сделанные на доске одним участником, моментально становятся видны другим участникам конференции.

## Общая структура системы

Разрабатываемая система реализует классическую централизованную клиент-серверную архитектуру. Как показано на рисунке 2.1 , в системе имеется один сервер и произвольное число клиентов.

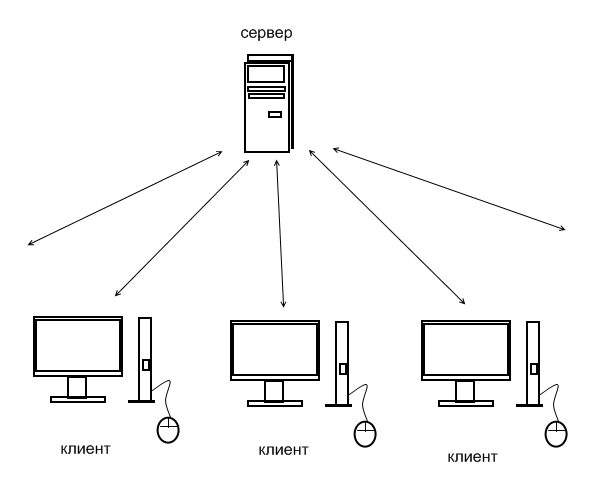


Рисунок 2.1 Классическая клиент-серверная архитектура

При этом сервер и клиенты могут выполняться как на разных компьютерах, так и на одном.

## Обзор протокола ENCP

Experimental Network Chat Protocol ( ENCP ) - протокол, используемый в системе для обеспечения функционирования онлайн-конференции.

Протокол ENCP описывает сообщения (датаграммы ENCP), которыми обмениваются серверная и клиентские программы в процессе работы, и порядок обработки этих сообщений, задавая таким образом функционирование системы во времени.

Для передачи датаграмм между приложениями используется стандартный протокол WebSocket[[2]](#footnote-2), поддерживаемый современными веб-браузерами. WebSocket является относительно новой технологией, позволяющей установить дуплексное TCP-соединение между веб-приложением и сервером; наличие такого соединения устраняет необходимость регулярного опроса сервера веб-приложением, что снижает нагрузку на сеть и одновременно увеличивает скорость реакции клиентской программы на серверные события. Датаграммы ENCP передаются по каналу WebSocket в сериализованном виде в нотации JSON[[3]](#footnote-3). Схема передачи датаграмм представлена на рисунке 2.2.

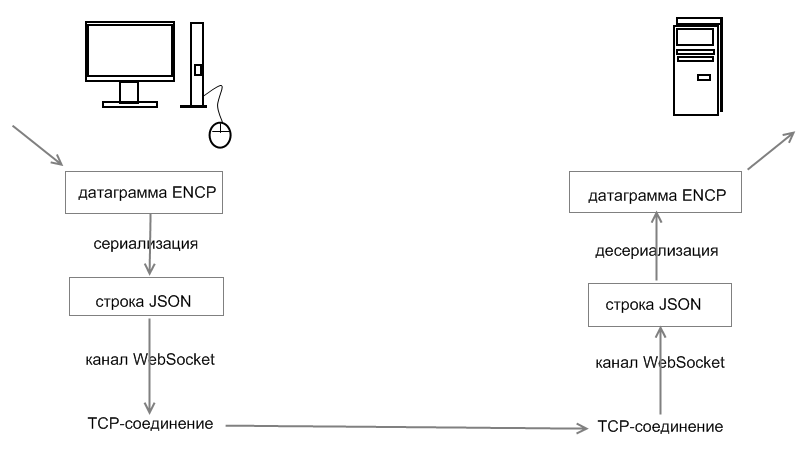


Рисунок 2.2 Схема передачи датаграмм

С точки зрения ENCP, клиент может находиться в четырех различных состояниях: "отключен", "подключен к серверу", "аутентификация" и "участник конференции" (см. рисунок 3).

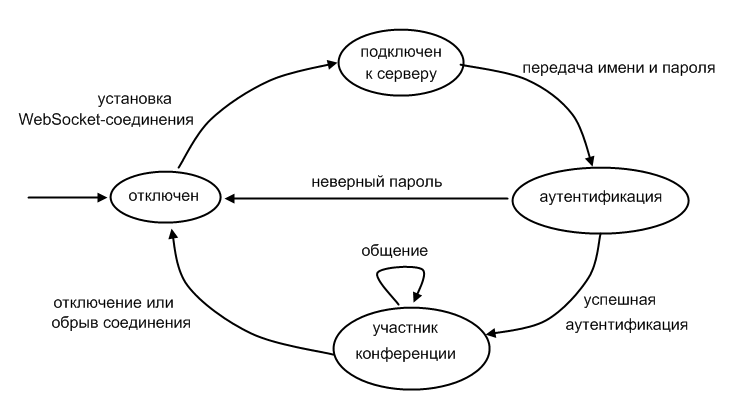


Рисунок 2.3 Схема состояний клиента с точки зрения протокола ENCP

Состояние "отключен" является начальным. При входе в конференцию клиент проходит через состояния "подключен к серверу" и "аутентификация", и, при успешном прохождении аутентификации попадает в состояние "участник чата". В этом состоянии клиент находится во время общения. При выходе из конференции, инициированном пользователем или же произошедшем при обрыве связи, клиент возвращается в начальное состояние.

Каждая датаграмма ENCP, описанная в протоколе, имеет свой код, состоящий из буквы "М" и числа. Более подробное описание датаграмм будет приведено позже.

## Структура клиентской программы

Разбиение клиентской программы на модули и их взаимодействие представлено на рисунке 2.4.

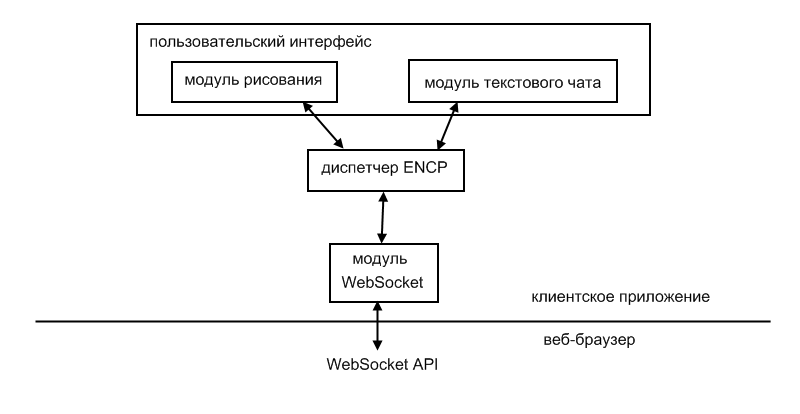


Рисунок 2.4 Структура клиентской программы

Основные модули клиентской программы:

* Модуль WebSocket отвечает за управление сокетом, используемым для подключения к серверу. Модуль использует WebSocket API, предоставляемый веб-браузером. Взаимодействие модуля с модулями более высокого уровня заключается в передаче и приеме датаграмм ENCP; при этом данный модуль берет на себя работу по сериализации и десериализации датаграмм в нотацию JSON.
* Диспетчер ENCP выполняет первичную обработку датаграмм, поступающих от сервера. В задачу диспетчера входит определение типа датаграммы и вызов соответствующей процедуры-обработчика из модулей верхнего уровня.
* Модуль текстового чата отвечает за отображение текстовых сообщений, написанных участниками чата, и уведомлений сервера в логе чата, а также содержит интерфейс для ввода и отправки пользователем текстовых сообщений.
* Модуль рисования отвечает за графическую часть конференции, а именно поддержку общей доски рисования, реализацию различных инструментов рисования, составление и обработку датаграмм ENCP для синхронизации доски рисования.

## Структура серверной программы

Разбиение серверной программы на модули и их взаимодействие представлено на рисунке 2.5.

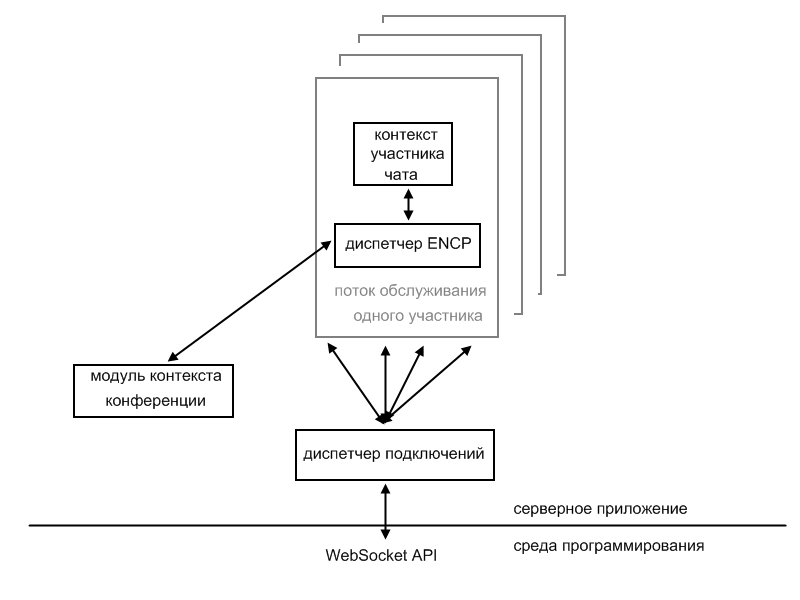


Рисунок 2.5 Структура серверной программы

Серверное приложение является многопоточной программой со следующими потоками выполнения:

* Главный поток, в котором выполняется диспетчер подключений;
* Набор обслуживающих потоков (по одному для каждого подключенного участника чата), выполняющих код диспетчера ENCP;
* Отладочный поток, выполняющий код интерактивной отладочной консоли.

Основные модули серверной программы:

* Диспетчер подключений управляет серверным сокетом, принимает входящие подключения и создает новые потоки для обслуживания клиентов.
* Модуль контекста конференции содержит информацию, необходимую для работы конференции в целом, в том числе:
  + Список зарегистрированных участников, содержащий регистрационные имена, пароли, права и дополнительную информацию по каждому участнику чата;
  + Список участников, подключенных к серверу в настоящий момент;
  + Историю последних сообщений - фиксированное число последних текстовых сообщений чата;
  + Полную историю изменения доски рисования с момента последней ее очистки.
* Диспетчер ENCP, выполняющийся в каждом потоке обслуживания. Каждый диспетчер работает со своим клиентским сокетом, и принимает и обрабатывает датаграммы ENCP.

С каждым потоком обслуживания, и, соответственно, с каждым диспетчером ENCP связан свой контекст участника чата.

## Спецификация датаграмм ENCP

## Протокол ENCP представляем собой стек из двух протоколов: ENCP/C (Chat) и ENCP/T (Transport). Первый определяет пакеты сообщений, на которых строится реализация чата, а второй - способ передачи этих сообщений.

## ENCP/T

## 1. Данные, передаваемые по протоколу.

## На самом высоком уровне, приложения (реализующие ENCP/C) считают, что используют протокол ENCP/T для передачи объектов.

## [ прога1 ] -- <объект> --> ...канал ENCP/T... -- <объект> --> [ прога2 ]

## ENCP/T не специфицирует, какие именно объекты передавать. Ему пофигу. Этим занимается ENCP/C. Задача ENCP/T - молча передать, что ему вручили, и не вякать.

## Под объектами тут понимаются экземпляры типов, нативных для используемых языков программирования. Прежде всего - словарей, у которых ключи и значения являются строками. Но значения могут быть также и числами, и массивами строк, и массивами словарей, у которых ключи и значения... Все, что может быть представлено в нотации JSON.

## Перейдем на уровень ниже, и увидим, что ENCP/T как раз и занимается сериализацией и десериализацией вверенных ему объектов в JSON.

## [ прога1 ] -- <объект> --+ +-- <объект> --> [ прога2 ]

## | |

## сериализация десериализация

## | ^

## V |

## JSON --> ...канал...--> JSON

## 

## Что такое канал на этой диаграмме?

## Канал - это веб-сокет (WebSocket). Веб-сокеты занимаются передачей текстовых строк в UTF-8. Сериализация в JSON порождает текстовую строку ASCII, а мы знаем, что любая строка ASCII является строкой UTF-8.

## Еще к слову: строки, передающиеся по WebSocket, окружаются ограничителями - байтами 0x00 в начале и 0xFF в конце. Джаваскриптовое API делает это само, а Питон, не знающий, что такое веб-сокеты, не делает. Поэтому эти байты добавляем и убираем мы сами. Конечно, можно написать уровень абстракции и сказать, что так оно и было))

## Ну и процедуру рукопожатия (handshake) протокола веб-сокетов мы на Питоне делаем ручками. Ту самую процедуру, где нужно делить на число пробелов.

## ENCP/C

## Протокол ENCP/C использует ENCP/T для передачи объектов между клиентом и сервером для достижения высшей цели - обмена сообщениями в виде чата.

## Передающиеся объекты, в общем случае, являются словарями. Все объекты JavaScript и так уже являются словарями, а Питон имеет соответствующий тип данных.

## Как минимум, передающиеся объекты-словари должны иметь поле 'type', указывающее на тип сообщения. Предполагается, что функция-диспетчер (привет, К.Л.! привет, А.В.!) по значению этого поля определит, какой функции второго уровня передавать объект, а та уже сможет разобраться с остальными полями.

## Некоторые сообщения содержат еще и поле subtype для дальнейшего мультиплексирования.

## Тип сообщения определяется парой значений полей type и subtype (если subtype есть); назовем эти два поля типообразующими. Типообразующие поля присутствуют всегда; если в тексте встретится предложение "клиент формирует сообщение с полями X и Y", это значит, что кроме полей X и Y в сообщении есть поле type, и, возможно, subtype.

## Если мы когда-нибудь доберемся до реализации нескольких чат-комнат на одном сервере, то у всех сообщений будет обязательное поле "room\_id". Текст данного документа написан без учета возможности наличия нескольких чат-комнат; в частности, нет описания сообщений подключения к конкретной чат-комнате, выхода из комнаты, получения списка комнат...

## Некоторые сообщения идут только от клиента к серверу или только от сервера к клиенту.

## Некоторые ходят в обоих направлениях, но имеют различный набор полей в зависимости от направления.

## Общая схема для большинства сообщений такова: клиент посылает сообщение серверу, сервер дополняет его некоторыми полями (например именем отправителя) и рассылает всем клиентам, подключенным к серверу (в т.ч. тому, от которого оно только что пришло!)

## Каждому типу сообщений соответствует код - буква M и порядковый номер по приведенному ниже списку.

## Для краткости фраза "формирует сообщение М1" сокращена в данном тексте до "формирует М1".

## Сообщения, реализованные в настоящий момент:

## M1. Текстовое сообщение

## { 'type': 'text', 'sender': sender, 'value': value }

## sender: имя (ник) отправителя. Изначально отсутствует, добавляется сервером.

## value: строка-текст сообщения.

## Жизненный цикл сообщения:

## 1. Пользователь набирает сообщение и жмет Enter

## 2. Программа-клиент формирует M1 с полем value, содержащем введенный текст, и отправляет серверу.

## 3. Сервер добавляет поле sender, содержащее имя (ник) отправителя, опционально проводит валидацию value (чтобы не включали злобный JavaScript и т.п.) и отправляет всем клиентам.

## 4. Клиент, получив M1, использует его содержимое для добавления строки в лог чата.

## Комментарии:

## - возможно, данное сообщение будет дополнено полем 'private\_to', чтобы реализовать передачу приватных сообщений. В таком случае сервер на шаге 3 передаст сообщение только клиенту, указанному в этом поле, вместо массовой рассылки.

## М2. Уведомление о новом участнике чата

## { 'type': 'notify', 'subtype': 'user\_joined', 'user': user }

## user: имя (ник) нового участника

## Жизненный цикл сообщения:

## 1. Программа-клиент нового участника производит первое (за сеанс) обращение к серверу. Выполняется процедура рукопожатия WebSocket.

## 2. По завершении рукопожатия сервер присваивает участнику имя (ник), формирует M2 с этим ником и отправляет всем клиентам.

## 3. Клиент, получив M2, добавляет в лог чата фразу "к нам пришел %user%" и добавляет нового участника в отображаемый список сидящих в чат-комнате людей.

## Комментарии:

## - скорее всего, процедура задания имени будет переработана. Вероятный сценарий: Клиент после рукопожатия отправляет серверу сообщение M4 "меня зовут %user%", после чего сервер выполняет рассылку М2. До этого момента клиент не считается участником чата.

## M3. Уведомление об уходе участника

## { 'type': 'notify', 'subtype': 'user\_left', 'user': user }

## user: имя (ник) покинувшего чат-комнату участника

## Жизненный цикл сообщения:

## 1. Участник закрывает страницу в браузере или программу-клиент, таким образом обрывая подключение по веб-сокету.

## 2. Сервер фиксирует закрытие веб-сокета, формирует M3 с именем участника, сокет которого закрылся, и рассылает всем клиентам.

## 3. Клиент, получив M3, добавляет в лог чата фразу "нас покинул %user%" и удаляет user'а из отображаемого списка участников.

## M4. Запрос о задании имени

## { 'type': 'set-name', 'new\_name': new\_name }

## new\_name: новое имя участника

## Жизненный цикл сообщения:

## 1. Пользователь решает сменить свой ник и вводит новый ник в программу-клиент.

## 2. Программа-клиент формирует М4 с полем new\_name и отправляет серверу.

## 3. Сервер изменяет имя клиента в своем внутреннем списке и смотрит, было ли отправлено сообщение M2 для данного клиента (является ли клиент участником чата) - см. комментарий к М2

## 4а Если клиент не является участником чата, то сервер формирует сообщение М2 с новым именем клиента; далее goto жизненный цикл М2.

## # замечание: такого не может быть после введения регистрации/авторизации. В коде эта альтернатива игнорируется.

## 4б Если клиент является участником чата, то сервер формирует сообщение М8.

## M5. Запрос списка участников.

## { 'type': 'roommates', 'list': roommates }

## roommates - массив строк - ников участников чата. Добавляется сервером.

## TODO : отправлять не массив строк, а массив объектов {'nick': nick, 'color': color}, чтобы клиент мог знать, кто каким цветом рисует.

## Жизненный цикл сообщения:

## 1. Программа-клиент раз в 30 секунд, а так же при приходе сообщения типа notify, решает, что ей неплохо бы синхронизировать свой список участников чата с серверным.

## 2. Программа-клиент формирует М5 без полей (кроме 'type', разумеется) и отправляет серверу.

## 3. Сервер дополняет M5 полем roommates и отправляет сообщение клиенту, запросившему список (а не всем клиентам!)

## 4. Клиент, получив М5 со списком, использует его для обновления отображаемого списка участников чата. Кроме того, при подключении нового клиента сервер формирует и отправляет ему сообщение М5 со списком.

## М6. Графическое сообщение должно содержать изображение, нарисованное клиентом, целиком.

## Как мы его разместим в юникодной строке - это тот еще вопрос.

## Скорее всего, реализовано не будет.

## М7. Изменение общей доски для рисования

## { 'type': 'public\_drawing', 'sender': sender, 'commands': [drawing\_command, ...] }

## Где sender добавляется сервером, drawing\_command = { 'color': color, 'tool': tool, 'param': param },или drawing\_command = 'clearall' где tool - имя инструмента ('pencil', 'line', 'rect', ...), а param = {'p1': {'x': x1, 'y': y1}, 'p2': {'x': x2, 'y': y2}}.

## Интерпретация param и color в зависимости от tool:

## tool | интерпретация x1, y1, x2, y2, color

## pencil | рисуется отрезок из точки (x1,y1) в точку (x2,y2) цветом color

## line | --//--

## rect | рисуется, но не закрашивается, прямоугольник с противоположными углами в точках (x1,y1) и (x2,y2)

## fillrect | то же, но прямоугольник закрашивается

## Жизненный цикл A:

## 1. Пользователь рисует закорючку на доске рисования. В процессе рисования клиентский буфер команд (draw\_history) пополняется командами рисования.

## 2. По завершении операции рисования (пользователь отпустил кнопку мыши) программа-клиент формирует сообщение М7 с содержимым буфера команд (одно сообщение содержит массив, представляющий буфер целиком), отправляет М7 на сервер и очищает буфер.

## 3. Сервер, получив М7, дополняет его именем пользователя (sender) и рассылает всем клиентам, кроме клиента-отправителя. Затем сервер очищает поле commands (commands = []) и отправляет М7 клиенту-отправителю.

## Сервер сохраняет команды в истории команд рисования, чтобы выдавать новым участникам уже готовую картинку.

## Серверная история команд очищается, если commands содержит 'clearall'.

## 4. Клиент, получив М7 от сервера, выполняет содержащиеся в нем команды рисования одна за другой, применяя их к общей доске для рисования. Дополнительно, клиент может отобразить имя пользователя, указанное в поступившем сообщении, в надписи "Автор последней закорючки".

## Жизненный цикл Б:

## 1. При подключении к чату нового участника пользователь отправляет ему всю серверную историю рисования в виде сообщения М7, но без поля sender.

## 2. См. п.4 цикла А.

## М8. Уведомление о смене ника

## { 'type': 'notify', 'subtype': 'user\_renamed', 'new\_nick': new\_nick, 'old\_nick': old\_nick }

## Жизненный цикл:

## 1. Сервер, получив сообщение М4, в пункте ж.цикла 4б формирует М8 и рассылает всем клиентам.

## 2. Клиент, получив сообщение М4, добавляет в лог чата фразу "%old\_name% теперь известен под именем %new\_name%" и изменяет отображаемый список участников соответствующим образом.

## М9. Отправление только что вошедшему в чат пользователю последних N сообщений

## {'type': 'notify', 'subtype': 'last\_messages', 'messages':self.websocket.lastNMessages}

## где lastNMessages массив словарей:

## {'sender': datagram['sender'], 'value': datagram['value'], 'time': datetime.datetime.now().strftime('%H:%M:%S')}

## Жизненный цикл:

## 1. Сервер, узнав, что к конференции присоединился пользователь отсылает ему массив последних 10 сообщений.

## 2. Клиент, получив сообщение М9, уведомляет пользователся о новых сообщениях и выводит их в окно чата.

## М10. Вход на сервер

## {'type': 'login', 'nick': chat\_nick, 'password': password}

## Жизненный цикл:

## 1. Пользователь решает подключиться к чату. Он вводит адрес сервера, логин, пароль, и жмет "Войти".

## 2. Клиент устанавливает WebSocket-соединение с сервером. После успешного установления и проведения handshake (т.е. в обработчике onopen) клиент формирует и отправляет на сервер сообщение М10.

## 3. Сервер, получив М10, медитирует над его содержимым; результатом медитации является сообщение М11.

## М11. Результат входа на сервер

## {'type': 'login\_result', 'logged\_in': logged\_in, 'is\_super': is\_super, 'color': color, 'message': message}

## 1. Сервер получает М10.

## 2. Выполняются следующие проверки:

## (1) есть ли указанный ник в списке зарегистрированных пользователей;

## (2) если (1), то правильно ли указан пароль;

## (3) является ли уже этот пользователь участником чат-комнаты.

## 3. Если (1) неверно, сервер выполняет регистрацию пользователя в своей базе, сохраняя ник и пароль. После этого считается, что условия (1) и (2) выполнены.

## 4. Вход на сервер считается успешным при одновременном выполнении первых двух условий и невыполнении третьего.

## 5а. Если вход успешен, то сервер формирует М11 с полем logged\_in = True. Поле color содержит значение цвета, ассоциируемого с пользователем.

## Сервер отправляет М11 клиенту; затем сервер выполняет прочие действия, связанные с входом клиента в чат, такие как отправка последних сообщений.

## 5б. Если вход не успешен, то сервер формирует М11 с logged\_in = False. В поле message заносится причина отказа во входе (эту строку увидит пользователь). Сервер отправляет клиенту сформированное сообщение и разрывает связь

## Каналы управления (бизнес-процессы)

### Подключение нового участника конференции

На рисунке 2.6 представлен канал управления для подключения нового участника конференции. Все события сопровождаются передачей ENCP датаграмм.

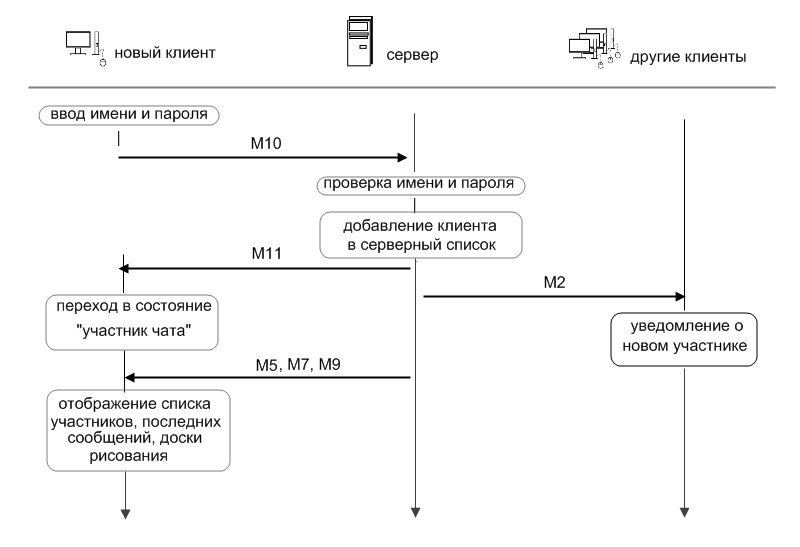


Рисунок 2.6 Канал управления подключения нового клиента

На рисунке 2.7 изображён алгоритм аутентификации нового участника конференции.

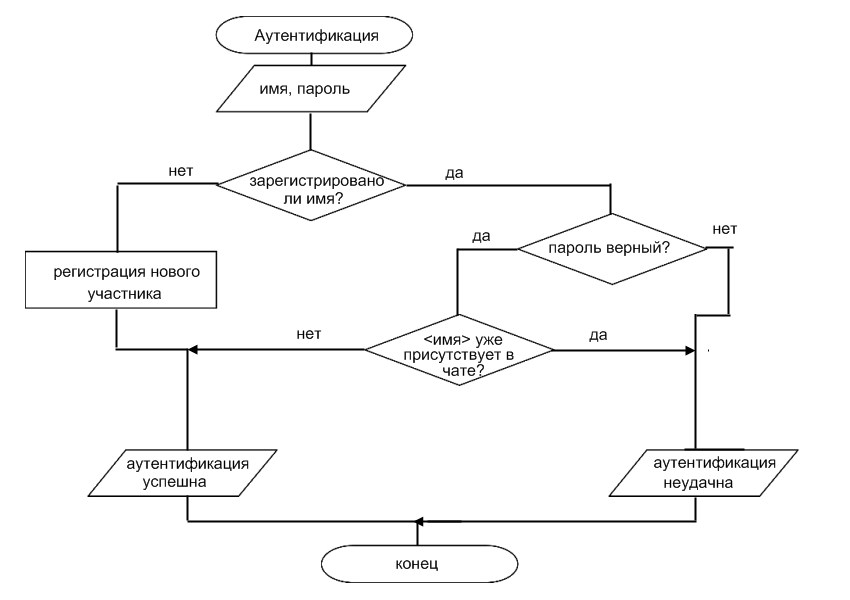


Рисунок 2.7 Алгоритма аутентификации клиента

### Обмен сообщениями

На рисунке 2.8 изображён канал управления обмена сообщениями. Сообщения могут быть как текстовые, так и связанные с рисованием общего изображения. Все сообщения проходя через сервер сохраняются в лог работы сервера.

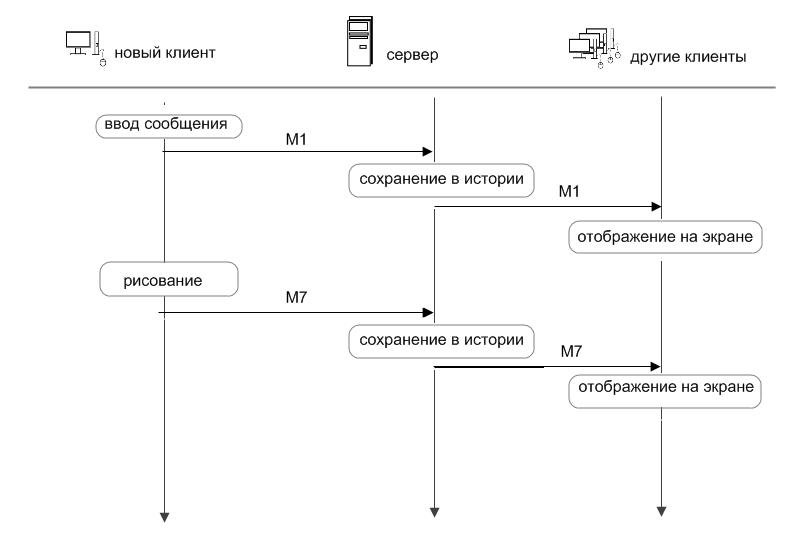


Рисунок 2.8 Канал управления обмена сообщениями

### Синхронизация клиентского списка

На рисунке 2.9 изображен канал управления синхронизации клиентского списка.

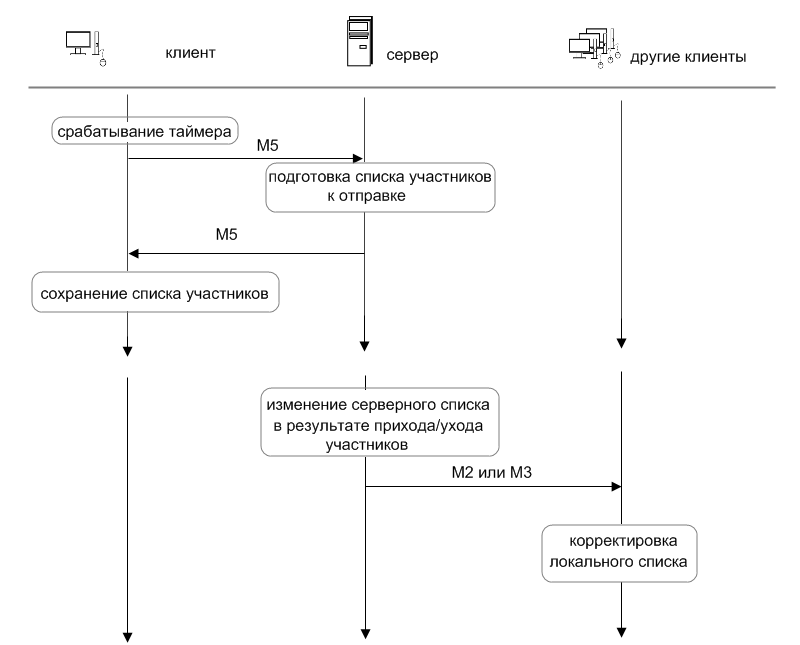


Рисунок 2.9 Канал управления клиентского списка

# Технологическая часть

# Заключение

# Приложение А

# Список литературы

1. Свободная энциклопедия Wikipedia, <http://en.wikipedia.org/>
2. Протокол WebSocket (черновик стандарта), <http://www.whatwg.org/specs/web-socket-protocol/>

1. http://www.digitaltrends.com/how-to/guide-to-instant-messenger-clients/ [↑](#footnote-ref-1)
2. Подробнее о стандарте WebSocket можно прочитать по данному адресу: <http://dev.w3.org/html5/websockets/> [↑](#footnote-ref-2)
3. Подробнее о нотации JSON можно прочитать по данному адресу: <http://www.json.org/> [↑](#footnote-ref-3)