**Санкт-Петербургский Государственный Университет**

**Математико-механический факультет**

Кафедра системного программирования

Малышев Виталий Владиславович

Дипломная работа

**Реализация SIP-телефонии для мобильных устройств с операционной системой Android**

Допущено к защите

Зав. кафедрой

д.ф.-м.н, проф. А.Н. Терехов

Научный руководитель

 д.т.н., проф. кафедры информатики В.О. Сафонов

Рецензент

аспирант кафедры информатики В.С. Соловьев

г. Санкт-Петербург

2010 год

**St. Petersburg State University**

**Faculty of Mathematics and Mechanics**

Chair of Software Engineering

Malyshev Vitaly Vladislavovich

Graduate paper

**SIP-telephony implementation for mobile devices powered by Android operating system**

Admitted to proof

Head of the chair

Dr. of Phys. and Math. Sci., Professor A.N. Terekhov

Scientific advisor

 Doctor of Sciences, Professor of Computer Science V. O. Safonov

Reviewer

Post-graduate student of Computer Science V.S. Soloviev

St. Petersburg

2010

Оглавление

[**1.** **Введение** 5](#_Toc262661010)

[**1.1.** **Постановка задачи** 5](#_Toc262661011)

[**1.2.** **Этапы работы** 8](#_Toc262661012)

[**1.3.** **Введение в предметную область** 10](#_Toc262661013)

[1.3.1. Общие сведения о SIP телефонии 10](#_Toc262661014)

[1.3.2. Принципы SIP протокола 11](#_Toc262661015)

[1.3.3. Архитектура сети 12](#_Toc262661016)

[1.3.4. Сообщения протокола SIP 13](#_Toc262661017)

[1.3.5. Сравнение SIP с H.323 15](#_Toc262661018)

[1.3.6. Сравнение SIP с Skype 17](#_Toc262661019)

[**2.** **Описание предлагаемого решения** 19](#_Toc262661020)

[**2.1.** **Выбор платформы решения** 19](#_Toc262661021)

[**2.2.** **Описание использования приложения** 20](#_Toc262661022)

[**3.** **Реализация решения** 24](#_Toc262661023)

[**3.1.** **Реализация низкоуровнего модуля, работающего со звуком напрямую** 24](#_Toc262661024)

[3.1.1. AudioRecordWrapper 26](#_Toc262661025)

[3.1.2. AudioTrackWrapper 27](#_Toc262661026)

[3.1.3. QueuedBuffer 28](#_Toc262661027)

[**3.2.** **Принцип работы “Handover”** 29](#_Toc262661028)

[**3.3.** **Архитектура проекта “MC Client”** 31](#_Toc262661029)

[**3.4.** **Ключевые классы элементов архитектуры.** 33](#_Toc262661030)

[3.4.1. Activity. 33](#_Toc262661031)

[3.4.2. SipServiceWrapper 35](#_Toc262661032)

[3.4.3. Service 36](#_Toc262661033)

[3.4.4. SIPEngine 39](#_Toc262661034)

[**4.** **Реализация и применение решения** 43](#_Toc262661035)

[**5.** **Сравнительный анализ** 45](#_Toc262661036)

[**6.** **Заключение** 47](#_Toc262661037)

[**7.** **Список литературы** 48](#_Toc262661038)

1. **Введение**
	1. **Постановка задач­­­и**

 Для пользователей персональных компьютеров уже давно стало привычным голосовое общение через Интернет, но для мобильных устройств качественных решений до недавнего времени просто не существовало. Однако развитие современных мобильных технологий, а именно появление мобильных устройств с быстрым доступом в Интернет и работающих под операционной системой Android[1], открывает новые возможности.

 В июле 2005 Google приобрела небольшую компанию Android, Inc., занимающуюся развитием собственных проектов. В то время было немного известно про компанию Android, Inc. – только то, что компания занималась разработкой программного обеспечения для мобильных телефонов. Как раз тогда и пошли слухи о том, что Google собирается выйти на рынок мобильных устройств.

 Под руководством Google команда разработала операционную систему, основанную на ядре Linux, гибкую и предлагающую широкие возможности по конфигурации. В ноябре 2007 Open Handset Alliance (OHA)[2] - консорциум множества компаний, куда входит и Google, поставил перед собой задачу разработки открытых стандартов для мобильных устройств. Тогда же OHA и выпустил свой первый продукт – операционную систему Android, основанную на ядре Linux версии 2.6. Основными особенностями данной системы являлась открытость и многозадачность. Официальный выход первого смартфона HTC Dream (G1), под управлением ОС Android, на рынок США состоялся 22 октября 2008 года.

 После появления первого устройства под управлением Android, стало возможным предоставить пользователю возможность использовать Интернет-телефонию так же, как и GSM телефонию. Широкополосный доступ в Интернет, многозадачность операционной системы дали возможность пользователю забыть о проблемах, связанных с Интернет связью. До Android самые удобные приложения Интернет-телефонии предлагала ОС iPhone, но с ее ограниченной многозадачностью невозможно использовать другие приложения вместе с Интернет-телефонией, что означает, например, невозможность читать документы и одновременно принимать звонки.

 На данный момент существует ряд решений по обеспечению Интернет-телефонии для Android, однако все они обладают существенными ограничениями по использованию.

 Данная работа ставит целью проведение анализа существующих решений Интернет-телефонии для ОС Android и создание приложения SIP телефонии под Android, удовлетворяющего следующим требованиям:

* Приложение должно работать на всех существующих версиях ОС Android[3]. Следует учитывать старые версии Android (1.1 и 1.5), так как общий процент устройств под управлением данных версий ОС достаточно велик, что видно из рисунка 1.



Рисунок 1: Соотношение числа устройств, работающих под управлением конкретной версии ОС Android. Данные приведены по состоянию на 01.04.2010

* Приложение должно уметь осуществлять переключение типа звонка с SIP на GSM и обратно – так называемый “Handover”. Т.е. при наличии широкого Интернет канала звонок идет по SIP, а если ширины канала начинает не хватать, то идет переключение звонка с SIP на GSM незаметно для пользователя. Как только ширина канала становится снова достаточной, идет переключение обратно на SIP.
* Приложение должно работать на заднем плане, не ограничивая пользователя в работе с коммуникатором.
* Приложение не должно уступать другим приложениям по качеству звука и скорости работы.
* Приложение должно использовать по максимуму возможности Dialer - приложения для совершения GSM звонков, встроенного в ОС Android (пример возможностей - телефонная книга, учет совершенных звонков).
	1. **Этапы работы**

Разработку SIP телефонии возможно разбить на несколько этапов:

1. **Получение доступа к звуку на ранних версиях ОС**

Так как одним из основных требований является работа на всех версиях ОС, то необходимо написать низкоуровневый модуль, работающий со звуком напрямую - записывать голос с устройства и передавать звук в динамики – на ранних версиях ОС (Android 1.1 и Android 1.5) такой возможности не было.

1. **Сборка низкоуровнего модуля в виде библиотеки, которую возможно подключить и использовать в приложениях под Android**

В Android есть механизм подключения низкоуровневыхбиблиотек в формате “.so” – формате понятном Linux, на ядре которого базируется Android. Сборка осуществляется с помощью компилятора arm gcc.

1. **Адаптация java библиотеки SIP телефонии для Android**

В качестве базы для обмена SIP сообщений используется Open Source библиотека “MjSip”[4]. К данному этапу уже необходимо создать простейшее приложение, которое позволяет звонить и принимать звонки через Интернет.

1. **Обеспечение дополнительных возможностей SIP клиента**

Для удобства использования необходимо обеспечить возможность конфигурации клиента и возможность управления звонками. На данном этапе должна быть реализована возможность совершения автоматического и ручного “Handover” – переключение звонка с GSM на SIP и обратно.

 Мною были спроектированы и реализованы этапы 1, 3, 4, в ходе чего были получены результаты, которые легли в основу данной работы. Этап 2 реализовали Смирнов Олег и Семаков Семен.

 Стоит отметить, что проект по разработки SIP телефонии для Android является коммерческим, поэтому исходный код проекта является коммерческой тайной компании ООО “e-Legion Ltd.”.

 В связи с этим в данной работе будут приведены описания только тех частей проекта, что отражают основные результаты.

 Проект, являющийся результатом разработка SIP телефонии для Android, описан в разделе “Реализация и применение решения”.

* 1. **Введение в предметную область**
		1. Общие сведения о SIP телефонии

 SIP (*Session Initiation Protocol* — протокол установления сессии) - стандарт установления и завершения пользовательского Интернет-сеанса, включающего обмен мультимедийным содержимым (видео - и аудиоконференция, мгновенные сообщения).

 В модели взаимодействия открытых систем SIP является сетевым протоколом прикладного уровня.

 Протокол описывает, каким образом клиентское приложение (например, MC Client) запрашивает начало соединения у другого, возможно физически удалённого, клиента, находящегося в той же сети, используя его уникальное имя. Протокол определяет способ согласования между клиентами об открытии каналов обмена на основе других протоколов, которые могут использоваться для непосредственной передачи информации (например, RTP). Допускается добавление или удаление таких каналов в течение установленного сеанса, а также подключение и отключение дополнительных клиентов (то есть допускается участие в обмене более двух сторон — конференцсвязь). Протокол также определяет порядок завершения сеанса. Схему организации работы по протоколу SIP можно видеть на рисунке 2.



Рисунок 2: Пример сети на базе протокола SIP

 Наряду с устаревшим H.323, SIP — один из протоколов, лежащих в основе Voice over IP (VoIP).

* + 1. Принципы SIP протокола

В основе протокола лежат следующие принципы:

* *Простота*: включает в себя только шесть методов (функций)
* *Независимость* от транспортного уровня, допустимо использовать UDP, TCP, ATM и т. д.
* *Персональная мобильность пользователей*. Пользователи разрешается перемещаться в пределах сети без ограничений. Данное свойство достигается путем присвоения пользователю уникального идентификатора. При этом набор предоставляемых услуг остается неизменным. О своих перемещениях пользователь сообщает с помощью сообщения **REGISTER**.
* *Масштабируемость сети*. Структура сети на базе протокола SIP позволяет легко ее расширять и увеличивать число элементов.
* *Расширяемость протокола*. Протокол характеризуется возможностью дополнять его новыми функциями при появлении новых услуг.
* *Интеграция в стек существующих протоколов Интернет*. Протокол SIP является частью глобальной архитектуры мультимедиа, разработанной комитетом IETF. Кроме SIP, данная архитектура включает в себя протоколы RSVP, RTP, RTSP,SDP.
* *Взаимодействие с другими протоколами сигнализации*. Протокол SIP может быть использован совместно с другими протоколами IP-телефонии, протоколами ТфОП (Телефонная сеть общего пользования), и для связи с интеллектуальными сетями.
	+ 1. Архитектура сети

 Протокол SIP имеет клиент-серверную архитектуру. Клиент выдает запросы, с указанием того, что он хочет получить от сервера. Сервер принимает и обрабатывает запросы, выдает ответы, содержащие уведомление об успешности выполнения запроса, уведомление об ошибке или информацию, запрошенную клиентом.

 Обслуживание вызова распределено между различными элементами сети SIP. Основным функциональным элементом, реализующим функции управления соединением, является абонентский терминал. Остальные элементы сети могут отвечать за маршрутизацию вызовов, а иногда служат для предоставления дополнительных сервисов.

* + 1. Сообщения протокола SIP

 Сообщения протокола SIP (запросы и ответы), представляют собой последовательности текстовых строк, закодированных в соответствии с документом RFC 2279. Структура и синтаксис сообщений SIP идентичны используемым в протоколе HTTP. Структура сообщений протокола SIP:

|  |
| --- |
| **Стартовая строка** |
| **Заголовки** |
| Пустая строка |
| **Тело сообщения** |

* Стартовая строка — начальная строка любого SIP-сообщения. Если сообщение является запросом, в ней указывается тип запроса, адресат и номер версии протокола. Если сообщение является ответом на запрос, в ней указывается номер версии протокола, тип ответа и его короткая расшифровка.
* Заголовки сообщений содержат информацию, необходимую для обработки сообщения (информация об отправителе, адресате, пути следования и пр.)
* Тело сообщения содержит описание сеансов связи. Не все запросы содержат тело сообщения (например, запрос **BYE**). Все ответы могут содержать тело сообщения, но содержимое тела в них бывает разным.

Пример запроса **INVITE**:

INVITE sip:nikoli@zenit.chempion.spb.ru SIP/2.0

Record-Route: <sip:nikoli@10.0.0.10;lr>

Via: SIP/2.0/UDP 10.0.0.10;branch=z9hG4bK3af7.0a6e92f4.0

Via: SIP/2.0/UDP 192.168.0.2:5060;branch=z9hG4bK12ee92cb;rport=5060

From: "78128210000" <sip:78128210000@vladimir.spb.ru>;tag=as149b2d97

To: <sip:nikoli@zenit.chempion.spb.ru>

Contact: <sip:78128210000@vladimir.spb.ru>

Call-ID: 3cbf958e6f43d91905c3fa964a373dcb\_zenit.chempion.spb.ru

CSeq: 103 INVITE

Max-Forwards: 16

Date: Wed, 10 Jan 2001 13:16:23 GMT

Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY

Supported: replaces

Content-Type: application/sdp

Content-Length: 394

v=0

o=root 3303 3304 IN IP4 10.0.0.10

s=session

c=IN IP4 10.0.0.10

t=0 0

m=audio 40358 RTP/AVP 0 8 101

a=rtpmap:0 PCMU/8000

a=rtpmap:8 PCMA/8000

a=rtpmap:101 telephone-event/8000

a=fmtp:101 0-16

a=silenceSupp:off - - - -

a=sendrecv

* + 1. Сравнение SIP с H.323[5]

 Рекомендации ITU-T, входящие в стандарт H.323, определяют порядок функционирования абонентских терминалов в сетях с разделяемым ресурсом, не гарантирующих качества обслуживания (QoS). Стандарт H.323 не связан с протоколом IP, однако, большинство реализаций основано на данном протоколе. Набор рекомендаций определяет сетевые компоненты, протоколы и процедуры, позволяющие организовать мультимедиасвязь в пакетных сетях. Стандарт H.323 определяет четыре основных компонента, которые вместе с сетевой структурой позволяют проводить двусторонние (точка - точка) и многосторонние (точка - много точек) мультимедиаконференции.

1. Терминал может представлять собой ПК или автономное устройство, способное выполнять мультимедиаприложение. Требуется, чтобы он обеспечивал звуковую связь и дополнительно поддерживал передачу видео или данных. Вследствие того, что основной функцией терминала является передача звука, он играет ключевую роль в предоставлении сервиса IP- телефонии.
2. Шлюз (gateway) не входит в число обязательных компонентов сети H.323. Он необходим только в случае, когда требуется установить соединение с терминалом другого стандарта. Связь обеспечивается трансляцией протоколов установки и разрыва соединений, а также форматов передачи данных. Шлюзы H.323 сетей широко применяются в IP телефонии для сопряжения IP сетей и цифровых или аналоговых коммутируемых телефонных сетей.
3. Привратник (gatekeeper) выступает в качестве центра обработки вызовов внутри своей зоны и выполняет важнейшие функции управления вызовами. Зона определяется как совокупность всех терминалов, шлюзов и MCU под управлением данного привратника. Привратник необязательный компонент сети H.323, однако, если он присутствует в сети, то терминалы и шлюзы должны использовать его услуги.
4. Сервер многосторонней конференции (Multipoint Control Unit) обеспечивает связь трёх или более H.323 терминалов. Все терминалы, участвующие в конференции, устанавливают соединение с MCU. Сервер управляет ресурсами конференции, согласовывает возможности терминалов по обработке звука и видео, определяет аудио и видеопотоки, которые необходимо направлять по многим адресам.

 В результате появления стандарта H.323, описывающего механизмы взаимодействия устройств обеспечивающих передачу голоса и видео по IP сетям, появилась возможность объединять в сети устройства различных производителей, что эффективно для сетей специальной связи.

 Если сравнивать SIP с H.323, то SIP легче читается человеком и структурирован в отношении запросов и откликов.

 Сторонники SIP также заявляют о нём как о более простом по сравнению с H.323. Однако некоторые считают, что, в то время как первоначально целью SIP была простота, в своём сегодняшнем виде он стал так же сложен, как и H.323. Другие считают, что SIP — протокол без состояний, который тем самым даёт легко реализовать восстановление при отказе и другие возможности, которые затруднены в протоколах с состояниями, таких как H.323. SIP и H.323 не ограничены голосовой связью, они могут обслуживать любой сеанс связи, от голосового до видеосеанса или приложений будущего.

|  |  |  |
| --- | --- | --- |
| **Параметр сравнения** | **SIP** | **H.323** |
| **Дополнительные услуги** | Набор услуг, поддерживаемых обоими протоколами примерно одинаков |
| **Персональная мобильность пользователей** | Имеется хороший набор средств поддержки мобильности | Персональная мобильность поддерживается, но менее гибко |
| **Расширяемость протокола** | Удобная расширяемость, простая совместимость с предыдущими версиями | Расширяемость поддерживается, но существует ряд сложностей |
| **Масштабируемость сети** | Оба протокола обеспечивают хорошую масштабируемость сети |
| **Время установления соединения** | Достаточно одной транзакции | Требуется несколько транзакций. |
| **Сложность протокола** | Простой, мало запросов, текстовый формат сообщений | Сложный, много запросов и протоколов, двоичное представление сообщений |

* + 1. Сравнение SIP с Skype[6]

 SIP и Skype предлагают два различных подхода к VoIP связи. SIP обладает гибкостью, Skype прост в установке, использовании и обладает прозрачным сетевым доступом; оба предлагают отличное качество звука.

|  |  |  |
| --- | --- | --- |
| **Параметр сравнения** | **SIP** | **Skype** |
| **Лицензия** | Публичная | Проприетарная |
| **Аутентификация** | Каждый VoIP SIP провайдер предлагает свой собственный сервер регистрации для своих подписчиков | Единый централизованный сервер регистрации, поддержка пользователей по всему миру |
| **Архитектура** | Клиент-серверная – есть сервер регистрации и агенты пользователей | Распределенная – есть вершины (клиент) и супервершины (вершины с большими аппаратными ресурсами). |
| **Масштабируемость сети** | Хорошая | Отличная – число супервершин возрастает, как только возрастает число вершин |
| **Кодеки** | Чаще всего G.711, для корпоративных сетей G.729 | Для узких каналов связи iLBC, для широких iSAC |
| **Проход через сетевые защиты(Firewalls)** | STUN server анализирует доверенность агента – не всегда хорошо работает | Проблемы отсутствуют |
| **Защищенность** | Все сообщения передаются по Интернету в виде обычного текста | Каждый пакет шифруется |

1. **Описание предлагаемого решения**
	1. **Выбор платформы решения**

 В качестве платформы для разработки приложения SIP телефонии был выбран Android.

 Поскольку разработка приложения проводилась в ходе рабочего проекта, то при выборе платформы рассматривались как технические, так и экономические факторы, например многозадачность операционной системы, отсутствие приложений SIP телефонии на момент начала проекта и удобство разработки мобильных приложений.

 Для удобства большая часть разработки велась на платформе Windows для настольных компьютеров, сборка низкоуровнего модуля в виде библиотеки велась на платформе Linux, так как компилятор arm gcc существовал только для Linux. При этом стоит отметить, что при переносе приложения на платформу Android возник ряд технических проблем:

* Невозможность работать со звуком через стандартный API в ранних версиях Android SDK. Отсутствие такой возможности позволило написать первое приложение VoIP телефонии для Android, в дальнейшем данный факт превратился в конкурентное преимущество (работа на всех версиях ОС). Для решения данной проблемы пришлось написать низкоуровневый модуль, работающий со звуком напрямую.
* Отсутствие некоторых функций. Так из соображений безопасности Android запрещает писать обработку нажатия красной кнопки сброса вызова. Однако для пользователя в плане удобства использование кнопки критично. Для решения данной проблемы был написан модуль, который слушает все события системы и перехватывает требуемое.
	1. **Описание использования приложения**

 Приложение SIP телефонии, написанное в ходе дипломной работы, будет далее обозначено как “MC Client”. MC Client ориентирован на использование точек доступа Wi-Fi. Когда пользователь заходит в приложение впервые (рисунок 3), он вводит свои данные для SIP регистрации (такие, как адрес сервера, порт, имя пользователя – рисунок 5) в разделе настроек (рисунок 4).

 ** **

Рисунок 3: Главное окно приложения Рисунок 4: Настройки (settings)

 ** **

Рисунок 5: Настройки соединения с сервером (SIP Settings)

 Далее, когда пользователь находится вблизи точки доступа Wi-Fi, идет автоматическая попытка соединения через найденную точку доступа. В случае успеха (рисунок 6), пользователь регистрируется на сервере, который он указал в настройках.



Рисунок 6: Сообщение об успешной регистрации

 MC Client очень тесно интегрируется со стандартным dialer (приложение для GSM звонков), находящемся в операционной системе Android. Происходит это следующим образом:

1. Пользователь запускает приложение, и идет успешная регистрация на сервере
2. Пользователь запускает стандартный dialer системы и набирает некий номер
3. Он нажимает кнопку звонить
4. Данное событие перехватывается и предлагается пользователю выбор (рисунок 7)
5. Позвонить по GSM(GSM call)

В данном случае идет стандартный звонок через Dialer по GSM, средствами операционной системы Android. Т.е. все происходит так, как если бы приложение ни на что не влияло

1. Позвонить по SIP(Enterprise Call)

В указанном случае, звонок, инициируемый dialer, - обрывается. Берется номер, набранный пользователем, и считается, что он набирал SIP адрес.



Рисунок 7: Окно выбора типа звонка

 К номеру подставляются данные из настроек приложения, чтобы получился правильный SIP адрес (адрес сервера и порт). По составленному SIP адресу идет дозвон на сервер средствами реализации протокола SIP.

1. Визуально звонок выглядит одинаково для звонков GSM и SIP (рисунок 8 и 9)

  

 Рисунок 8: Дозвон до абонента Рисунок 9: Разговор с абонентом

 Результаты SIP звонков отображаются в логе dialer, как если бы пользователь говорил по GSM(кто звонил, сколько времени говорил и другая информация о разговоре).

1. **Реализация решения**
	1. **Реализация низкоуровнего модуля, работающего со звуком напрямую**

 Основное преимущество проекта SIP телефонии “MC Client” перед конкурентами состоит в работе на всех версиях Android. В частности, для версий 1.1 и 1.5 было необходимо реализовывать низкоуровневый модуль, работающий со звуком напрямую. Реализации для версий 1.1 и 1.5 очень схожи и отличаются лишь в нескольких константах и параметрах, передаваемых методам, поэтому далее будет описана лишь реализация для Android 1.5.

 Всего в модуле есть 3 ключевых сущности: AudioRecordWrapper (рисунок 10) – native обертка для записи звука с микрофона, AudioTrackWrapper (рисунок 10) – native обертка для проигрывания звука на динамиках и QueuedBuffer (рисунок 11) – буфер, через который проходит звук, прежде чем попасть на целевую компоненту.



Рисунок 10: Поля и методы классов AudioRecordWrapper и AudioTrackWrapper



Рисунок 11: Буфер хранения информации

 Далее будет описаны основные методы ключевых сущностей. В данном разделе не будет описана инициализация и использование ключевых сущностей низкоуровнего модуля в дипломном проекте “MC Client”, так как это стандартная процедура работы с native классами через JNI[7].

* + 1. AudioRecordWrapper

~AudioRecordWrapper() – деструктор, при уничтожении объекта остановить запись звука и уничтожить все связанные переменные.

audioRecordCallback(int event, void\* user, void \*info) – callback метод о событии, присылаемом ОС Android. В частности, получение новой порции информации с микрофона, которую требуется положить в буфер.

handleEvent(int event, void\* voidInfo) – метод обработки события, которое пришло от ОС Android.

onMoreData(const android::AudioRecord::Buffer\* info) – при получении новой порции информации положить ее в буфер.

read(char\* data, int maxSize) – прочитать данные из буфера.

start() – запустить запись звука.

stop() – остановить запись звука.

* + 1. AudioTrackWrapper

~AudioTrackWrapper() – деструктор, при уничтожении объекта остановить проигрывание звука и уничтожить все связанные переменные.

audioTrackCallback (int event, void\* user, void \*info) – callback метод о событии, присылаемом ОС Android. В частности, получение новой порции информации от динамиков о том, что проигралось некое количество информации и требуется высылать еще.

handleEvent(int event, void\* voidInfo) – метод обработки события, которое пришло от ОС Android.

onMoreData(android::AudioTrack::Buffer\* info) – если вызван данный метод, значит проиграна часть информации на динамиках и ее можно выбросить из буфера.

write(const char\* data, int size) – отправить данные из буфера на динамик.

start() – запустить проигрывание звука на динамиках.

stop() – остановить проигрывание звука на динамиках.

* + 1. QueuedBuffer

~QueuedBuffer() – деструктор, при уничтожении объекта уничтожить все связанные переменные.

clear() – в режиме блокировки сбросить все индексы – такие как индекс чтения, индекс записи и размер активных данных в буфере.

get(char\* data, int size) – записать в data данные из буфера размером в size.

put(const char\* data, int size) – записать в буфер данные из data размером size.

lock(), unlock() – обеспечение режима блокировки. Чтобы обращаться с переменными одновременно мог только один поток (очевидно, что запись звука и проигрывание звука совершаются в разных потоках).

putNoOverrun(char data) – очистить буфер и заполнить его данными из data.

* 1. **Принцип работы “Handover”**

 Еще одной отличительной особенностью проекта SIP телефонии “MC Client” является так называемый “Handover”. Во всех конкурентных решениях такая особенность отсутствует. Суть заключается в следующем: когда доступен широкий Интернет канал, - для связи с абонентом использовать SIP телефонию, когда канала достаточной ширины нет в наличии, - использовать GSM связь. Есть 2 типа “Handover”:

1. Автоматический
2. Плохое качество SIP звука

 Во время SIP звонка сервер периодически пересылает клиенту SIP сообщения, содержащие 2 величины, определяющие качество звука – “Jitter” (нежелательные фазовые и/или частотные случайные отклонения передаваемого сигнала) и “Packet Loss” (процент пакетов, которые не дошли до сервера). Полученные величины сравниваются с величинами, указанными в настройках (изображено на рисунке 12). При превышении их серверу посылается сообщение о совершении Handover, завершается SIP звонок и автоматически принимается GSM звонок от сервера. В итоге для пользователя будет совсем небольшая задержка, и качество связи все время будет на достойном уровне.



Рисунок 12: Настройки пороговых величин Jitter и Packet Loss. Значения в процентах.

1. Слабый сигнал Wi-Fi

 Во время разговора с абонентом все время отслеживается сила сигнала Wi-Fi. Если разговор идет по SIP, и сигнал Wi-Fi стал слабым, то серверу будет послано сообщение о “Handover” с SIP на GSM – заканчивается текущий SIP звонок и автоматически принимается входящий GSM звонок от сервера. Если разговор идет по GSM, и сигнал стал сильным, то серверу будет послано сообщение о “Handover” с GSM на SIP – заканчивается текущий GSM звонок и автоматически принимается входящий SIP звонок от сервера.

1. Ручной

 Во время разговора пользователь всегда может самостоятельно выбрать опцию “Handover” и переключить тип сигнала.

* 1. **Архитектура проекта “MC Client”**

Архитектура проекта изображена на рисунке 13.

****

Рисунок 13: Ключевые сущности архитектуры

 **Activity** – это визуальный пользовательский интерфейс. Например, это список элементов меню, которые пользователь имеет возможность выбирать. Всегда есть activity переднего плана и activities заднего плана. Когда создается новая activity, она выходит на передний план, смещая на задний план ту, которая раньше была передним планом. При уничтожении activity переднего плана, на передний план выходит первая в очереди заднего плана. Если системе будет не хватать ресурсов, то самые неприоритетные из activity заднего плана (система сама устанавливает приоритеты) будут уничтожены. Каждая activity живет в собственном процессе.

 **Service** – не имеет визуального пользовательского интерфейса и работает на заднем плане в течение неопределенного времени. Например, сервис может играть музыку, в то время как пользователь может использовать браузер. Пользователь не видит списка песен и состояния проигрывания, однако музыку он слышит. Каждый сервис живет в собственном процессе.

 Так как activity и сервис живут в разных процессах, то нельзя напрямую обращаться к сервису из activity (иначе activity будет ждать окончания действий сервиса и не будет реагировать на действия пользователя). Для разрешения указанной ситуации существует специальное понятие ServiceWrapper. Он является посредником между сервисом и activity и связывает их. Происходит это следующим образом: при создании activity требуется создать в ней ServiceWrapper и установить связывание между ServiceWrapper и сервисом (которое является системным, т.е. система берет на себя такое связывание). Потом, все обращения к сервису идут через ServiceWrapper, вызывая его методы.

 Сервис может вызвать создание новой Activity путем отправления соответствующего Intent сообщения, которое обрабатывается системой. Сам по себе Intent – это объект, который содержит в себе описание абстрактных операций, которые необходимо выполнить. Таким образом, возможно посылать сообщения, которые потом обрабатываются системой Android, Android смотрит в манифест – файл (в котором описаны настройки приложения), если находит соответствие между activity и intent, то создает данный activity.

 **SIPEngine** – это отдельный модуль, ответственный за передачу звука по протоколу SIP. Часть, основанная на общении с сервером посредством протокола SIP, основана на Open-Source библиотеке MjSip. В данном модуле есть методы, позволяющие совершать звонки, а также реагировать на сообщения от сервера (такие как, успешный звонок, отвергнутый звонок, таймаут по времени соединения с сервером и т.д.).

 Сервис взаимодействует с SIPEngine тем, что вызывает его методы, связанные со звонком. Т.е. пользователь нажал кнопку позвонить, данное событие поймано в Activity, Activity передает через ServiceWrapper команду к сервису позвонить, а сервис обращается к SIPEngine с просьбой выполнить звонок.

* 1. **Ключевые классы элементов архитектуры.**
		1. Activity.

 Рассмотрим класс Sipdroid – главное окно(Activity) нашего приложения и его основные методы (все методы можно видеть на рисунке 14).



Рисунок 14:Константы и методы класса Sipdroid

*onCreate(Bundle savedInstanceState)* – первый метод, который вызывается при создании сущности. В нем происходит обработка функционала и решение о том, что из него показываться в главном окне (есть функционал, доступный в режиме разговора, а есть доступный вне разговора). Также, происходит решение о том, требуется ли показывать пользователю диалог об активации приложения.

*onDestroy()* – последний метод, который вызывается при разрушении сущности. Здесь происходит освобождение всех связанных с Sipdroid сущностей (например, ServiceWrapper).

*onCreateOptionsMenu(Menu menu)* – метод, инициализирующий меню, появляющегося на нажатии на кнопку “Menu”. Из ресурсов приложения (которые хранятся в формате xml) берутся элементы меню “Настройки”, “Спрятать”, “Выйти”.

*onOptionsItemSelected(MenuItem item)* – метод, который вызывается при нажатия на кнопку меню. Определяется нажатый элемент меню и обрабатывается событие, соответствующее данному элементу.

*closeFromSipServiceListener()* – метод, который закрывает сервис. При выходе из приложения необходимо сначала закрыть сервис, получить ответ об успешном закрытии и только после этого закрыть окно приложения, которое показывалось пользователю.

*reinitializeService()* – перезапускает сервис. Отправка через ServiceWrapper (так как сервис и Activity живут в разных процессах, то необходимо пользоваться посредником) сообщения сервису о необходимости перезапуска.

*isRegistered()* – спросить у сервиса через ServiceWrapper, зарегистрирован ли пользователь в данный момент или нет. В зависимости от регистрации пользователя на главном окне показывается различный функционал (например, незарегистрированному пользователю недоступен “Handover”).

* + 1. SipServiceWrapper

****

Рисунок 15:Константы и методы класса SipServiceWrapper

 Все методы и классы приведены на рисунке 15. Далее будет приведено описание основных методов.

SipServiceWrapper(Context context, ISipServiceListener listener) – связать через контекст Activity и сервис и зарегистрировать listener в качестве слушателя сообщений от сервиса. В качестве listener чаще всего выступают различные Activity, которым требуется получить от сервиса ответ.

SipServiceWrapper(Context context) – тоже самое, что и конструктор, описанный выше, только без регистрации слушателя сообщений от сервиса.

*unbind()* – если есть сервис, и есть слушатель сообщений от него, тогда отсоединить слушателя от сервиса. Вызывается в основном при уничтожении слушателя, чтобы сервис не посылал сообщения уничтоженным Activity.

 Остальные методы класса просто пересылают сообщения сервису о необходимом функционале.

* + 1. Service



Рисунок 16: Константы и методы класса SipService

 Полный перечень методов класса можно видеть на рисунке 16. Далее следует описание лишь основных методов.

*loadConfig()* – метод преобразует данные из xml файла в объект класса UAConfig, класса представляющего конфигурацию клиента. Загрузка конфигурации происходит при инициализации сервиса. Большинство функционала клиента опирается на введенные пользователем данные, поэтому важно, чтобы во весь период жизни сервиса возможно было обратиться к конфигурации.

*onStart(Intent intent, int startId)* – метод, который вызывается при запуске сервиса. При запуске сервиса, например, необходимо проверить, не ожидает ли пользователя голосовая почта.

*onCreate()* - вызывается при создании экземпляра сервиса. В указанном методе проверяется состояние активации, не изменился ли номер SIM карты пользователя с последнего раза, начинается отслеживание уровня Wi-Fi сигнала и происходит инициализация ключевых переменных.

onDestroy() - вызывается при уничтожении экземпляра сервиса. Необходимо остановить все службы, запущенные в onCreate(), а также оповестить всех слушателей о закрытии сервиса, что они могли выполнить необходимые действия.

*httpsRequestDone(int result)* – связано с функцией CallBack. Смысл состоит в следующем – формируется специальный https запрос, который содержит в себе имя пользователя, зашифрованный пароль, имя адресата, и указанный запрос посылается на сервер. Сервер перезванивает адресату, который автоматически берет трубку, и соединяет с пользователем по GSM связи.

*fireStateChanged(final int state, final String result)* – оповещает всех слушателей об изменении состоянии сервиса. Используется, например, когда инициализируется звонок абоненту, чтобы Activity могли перерисовать свое графическое представление.

*wifiStateChanged(boolean on)* – получения события об изменении состояния беспроводной сети. Здесь происходит обработка функционала, связанного с беспроводной сетью, например, “Handover”.

*showCallTypeSelector(String phone)* – показать диалог выбора(Enterprise Call, GSM Call). Когда пользователь нажимает кнопку звонить, данное событие перехватывается вместе с введенным номером, обрабатывается в зависимости от настроек (в настойках, например, можно указать, чтобы звонить всегда по SIP) и предлагается пользователю визуальный выбор типа звонка.

 У класса есть методы, которые выглядят как onUa\*, - это callback методы из SipdroidEngine. Например, когда требуется завершить разговор, сервис обращается к SipdroidEngine о прекращение звонка, происходит обработка, и сервису возвращается событие о завершении обработки.

 Остальные методы обращаются к SipdroidEngine за выполнением указанного действия.

* + 1. SIPEngine



Рисунок 17: Константы и методы класса SipdroidEngine

 Полный список методов приведен на рисунке 17, далее описаны только ключевые.

SipdroidEngine(SipdroidEngineListener a\_EngineListener) – конструктор, который инициализирует слушателя (например, сервис), чтобы в дальнейшем оповещать слушателя о всех произошедших событиях.

*GetState()* – получить текущее состояние объекта класса. Например, с его помощью сервис может узнать, произошла ли инициализации объекта класса и, в случае успешного ответа, продолжить собственную инициализацию.

*StartEngine(UAConfig config)* – получить конфигурацию клиента и подготовить объект класса SipProvider (класс, входящий в библиотеку MjSip) для передачи SIP сообщений серверу. Данный метод вызывается при запуске сервиса.

*reloadConfig(UAConfig config)* – перечитать конфигурацию клиента и выполнить реинициализацию. Необходимо вызывать данный метод при перезагрузке сервиса, так как сервис прочитает новую конфигурацию из файла, и конфигурации сервиса и объекта класса не будут совпадать.

*getMyNumber()* – позволяет получить полное имя клиента из введенных данных в настройках клиента. Согласно протоколу SIP формируется в виде {user\_name}@{ip\_address}:{port}. Используется, например, при отправке сообщений на сервер о регистрации клиента.

*register (int expire\_time)* – отослать запрос на сервер о регистрации. Содержит параметр expire\_time, который определяет, через сколько времени при отсутствии сообщений со стороны клиента считать, что клиент неактивен. Регистрация используется при старте системы или при возникновении соединения по Wi-Fi.

*loopRegister(int expire\_time, int renew\_time, long keepalive\_time)* – посылать периодические сообщения о регистрации. Чтобы сервер знал, что клиент активен, необходимо посылать периодически ему сообщения.

*unregister ()* – послать сообщение на сервер об окончании активного периода. Используется, например, при закрытии клиента.

*isRegistered()* – вспомогательный метод, который определяет состояние регистрации на текущий момент. Большая часть функционала опирается на данный метод.

*listen ()* – перейти в режим прослушивания входящих звонков. Только в данном режиме возможно получать сообщения о звонке от сервера. Используется при инициализации объекта класса и при изменениях состояния звонка – например, когда разговор с текущим абонентом завершен, необходимо перейти в режим ожидании звонков.

*call (String target\_url)* – совершить SIP звонок на номер. Здесь происходит обращение библиотеке, ответственной за передачу звука по протоколу RTP. Используется в том случае, когда клиент хочет инициализировать разговор с другим абонентом.

*answercall ()* – ответить на входящий звонок. В данном методе происходит обращение к библиотеке, ответственной за передачу звука по протоколу RTP.

*rejectcall ()* – отклонить входящий звонок. В данном методе происходит обращение к библиотеке, ответственной за передачу звука по протоколу RTP.

*getTransportString (short transport)* – получить текстовое представление транспортного протокола. Используется при инициализации SipProvider, который отвечает за коммуникацию SIP сообщений с сервером.

 У класса есть методы, которые выглядят как onUa\*, - это callback методы из UserAgent. В указанных методах соответствующие события передаются всем слушателям, например, сервису.

1. **Реализация и применение решения**

 Приложение написано на языке Java для платформы Android с помощью Android SDK 1.6.

 При изучении рынка было обнаружено несколько библиотек с открытым кодом, реализующих функциональность SIP телефонии. Приложения для Android пишутся на языке Java, поэтому интересовали библиотеки, написанные на Java.

* MjSip
* JSIP[8]
* Jain SIP[9]

 Библиотеки основываются на правилах установления сессии (SIP, Session Initial Protocol), описанных в стандарте RFC 3261[10], поэтому алгоритм один и тот же во всех библиотеках. Была выбрана библиотека MjSip, так как она наиболее полно реализует требующиеся возможности.

 Звук идет по протоколу RTP[11]. В качестве кодека звука используется G.711[12], так как данный кодек наиболее просто реализуется и обеспечивает хорошее качество звука.

 Отличительной особенностью приложения является работа приложения на всех версиях ОС Android. Для указанных целей реализована низкоуровневая схема получения непрерывных данных с микрофона и отправления данных на динамики мобильного устройства. Низкоуровневая реализация доступа к звуку собрана в виде библиотеки с помощью компилятора “arm gcc” в формате “.so”, понятном для Android, отдельно для Android версий 1.1 и 1.5. Библиотеки успешно подключены к проекту “MC Client” и используются в проекте.

 Также особенностью работы приложения является функциональность “Handover”, которая позволяет практически незаметно для пользователя переключать тип звонка с SIP на GSM или обратно, в зависимости от ширины доступного интернет канала или качества звука.

 Приложение предоставляет пользователю гибкие возможности клиента, однако конкурентные решения предлагают более широкие возможности настройки клиента. В частности, в конкурентных решениях есть возможность задавать способ кодирования звука, в то время как “MC Client” использует только G.711.

В данный момент приложение используется фирмой Comdasys, Inc.

1. **Сравнительный анализ**

 На данный момент существует ряд приложений, реализующих VoIP телефонию на операционной системе Android. Большая часть является закрытыми продуктами, созданными для решения конкретных задач:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Параметр сравнения | MC Client | Skype Lite[13] | Fring[14] | Sip Agent[15] | Sipdroid[16] |
| Возможность совершения звонков через интернет | ✓ | 🗶 | ✓ | ✓ | ✓ |
| Работа на всех версиях ОС | ✓ | ✓ | 🗶 | 🗶 | 🗶 |
| Тесная интеграция с GSM связью | ✓Обеспечивается функциональностью “Handover” | 🗶 | 🗶 | 🗶 | 🗶 |
| Возможность высокого сжатия звука | 🗶G.711 | 🗶Звука нет | 🗶 | ✓Speex[17] | ✓Speex |
| Высокое качество звука | ✓ | 🗶Звука нет | ✓ | 🗶 | ✓ |
| Использование стандартной встроенной телефонной книги | ✓ | 🗶 | ✓ | ✓ | 🗶 |
| Возможность чата | 🗶 | ✓ | ✓ | 🗶 | 🗶 |
| Поддержка протоколов, отличных от SIP | 🗶 | ✓Skype | ✓ICQ, SIP, Google Talk, AIM, MSN Messenger, Yahoo,Twitter | 🗶 | 🗶 |
| Использование произвольного сервера | ✓ | 🗶skype.com | 🗶fring.com | ✓ | ✓ |
| Тонкие возможности настройки клиента | ✓ | 🗶 | 🗶 | ✓ | ✓ |

 Таким образом, ни одно из существующих готовых решений не предоставляет функциональности, отвечающей задачам, поставленным в данной дипломной работе. Также дипломный проект MC Client долгое время являлся единственным клиентом VoIP телефонии на ОС Android (до выхода Android версии 1.6[18], где была предоставлена через API возможность получить доступ к звуку).

1. **Заключение**

 В рамках данного дипломного проекта был предложен способ получить доступ к звуку на всех версиях ОС Android.

 Также была предложена концепция “Handover” - функционала клиента, который позволяет незаметно для пользователя переключать тип звонка с SIP на GSM или обратно, в зависимости от ширины доступного интернет канала или качества звука.

 Описанный способ получения доступа к звуку и функционал “Handover” были реализованы в проекте SIP телефонии “MC Client”, который является конечным результатом дипломной работы.

 Реализованное решение имеет практическую ценность и уже сейчас используется в коммерческой системе Comdasys, Inc.

 Был произведен анализ приложений, решающих схожие задачи, и проведено сравнение представленных в них возможностей с возможностями предложенного решения.

 Предложенное решение решает поставленные задачи лучше, чем имеющиеся в открытом доступе приложения, однако конкурентные решения предлагают более широкие возможности конфигурации клиента. Предложенное решение разрабатывалось для использования на мобильных устройствах на платформе Android с учётом её особенностей.

 В перспективе, планируется внедрить возможность текстового чата в клиент, а также возможность совершения видео звонков. Для уменьшения трафика между клиентом и сервером планируется использовать более прогрессивный кодек звука iLBC[19]. Также планируется сделать весь трафик более защищенным путем внедрения TLS[20] для шифрования SIP сообщений и внедрить протокол SRTP[21] вместо незащищенного RTP для передачи звука.

1. **Список литературы**

[1] Android operating system

<http://source.android.com/>

[2] Open Handset Alliance – OHA - Members

<http://www.openhandsetalliance.com/oha_members.html>

[3] Android platform versions – statistics

<http://developer.android.com/resources/dashboard/platform-versions.html>

[4] MjSip, A complete java-based implementation of a SIP stack

<http://www.mjsip.org/>

[5] H.323, recommendation from the ITU Telecommunication Standardization Sector (ITU-T)

<http://www.itu.int/rec/T-REC-H.323/e>

[6] Skype and SIP comparison

<http://www.rtx.dk/Default.aspx?ID=949>

[7] JNI, Java Native Interface

<http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/jniTOC.html>

[8] Java SIP Library, JSIP

<http://sourceforge.net/projects/jsip/>

[9] Jain SIP, Java specification for SIP Signaling

<http://wiki.java.net/bin/view/Communications/JainSIP>

[10] RFC 3261

<http://www.ietf.org/rfc/rfc3261.txt>

[11] Standard 64, RTP: A Transport Protocol for Real-Time Applications

<http://tools.ietf.org/html/rfc3550>

[12] G.711, Audio Codec

<http://www.itu.int/rec/T-REC-G.711/e>

[13] Android, Skype Lite Source <http://share.skype.com/sites/skypegear/2009/01/video_skype_lite_on_android.html>

[14] Android, Fring for Android

<http://www.androlib.com/android.application.com-fring-xmnt.aspx>

[15] Android, SIP Agent application

<http://www.androlib.com/android.application.com-bw-sip-ui-xqCB.aspx>

[16] Android, Sipdroid client

<http://www.androlib.com/android.application.org-sipdroid-sipua-BCw.aspx>

[17] Speex, Audio Codec

<http://www.speex.org/>

[18] Android 1.6 released

<http://android-developers.blogspot.com/2009/09/android-16-sdk-is-here.html>

[19] iLBC

<http://www.ilbcfreeware.org/>

[20] TLS, Transport Layer Security

<http://www.ietf.org/rfc/rfc2246.txt>

[21] SRTP, Secured RTP

<http://www.ietf.org/rfc/rfc3711.txt>