Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Шумилов Пётр Евгеньевич

Автоматическая категоризация инцидентов

Курсовая работа

Научный руководитель:
ст.преп. Я. А. Кириленко

Санкт-Петербург
2018

Оглавление

Введение

На сегодняшний день, подавляющее большинство разного рода компаний и организаций имеют систему технической поддержки. Будь это крупный оператор связи или служба доставки цветов, немалую долю в качестве предоставляемого ими продукта играет то, насколько хорошо этот товар или услуга поддерживаются. Ключевым фактором для конечного потребителя является скорость, с которой он может решить те или иные вопросы касаемо предоставляемого ему продукта. С другой стороны, перед поставщиком товара или услуги стоит задача как можно быстрее понять, что именно волнует пользователя. Критерием скорости, в большей мере, является точность, с которой проблема может быть категоризована, так как правильно классифицированная заявка быстрее попадёт на обработку к компетентному специалисту.

Существуют практики, при которых пользователь самостоятельно пытается категоризировать своё обращение. В некоторых случаях такие методы показывают удовлетворительную эффективность, например, в небольших компаниях, которые имеют узкий спектр продуктов или услуг. Однако существуют случаи, когда вышеописанные методы неприемлемы из-за высокого разнообразия категорий, в которых рядовому пользователю зачастую трудно разобраться. Неправильно категоризованная проблема, приводит к потере времени на перенаправление инцидента от одного специалиста к другому. По мнению автора этой работы, выходом из подобных ситуаций может стать налаживание процесса автоматической категоризации обращений с помощью некоего программного обеспечения, способного регулярно обучаться на ранее успешно разрешенных инцидентах и выдавать категорию на основе текста обращения. Наличие такого программного обеспечения позволить существенно уменьшить участие пользователя в определении категории своей проблемы, тем самым повысит эффективность работы технической поддержки.

Также немаловажной проблемой, в рассматриваемой предметной области, являются устаревшие интерфейсы продуктов, обеспечивающие

работу технической поддержки. Чаще всего это связано с тем, что интеграция и сопровождение такого программного обеспечения являются довольно длительным и дорогостоящим процессом, в течение которого ранее актуальные подходы к построению интерфейсов становятся устаревшими. Примером является то, что даже на сегодняшний день не все крупные игроки рынка вышеописанного ПО имеют адаптированные под мобильные устройства интерфейсы. Данная проблема вдохновила автора этой работы на поиск вариантов решений, что далее вылилось в совершенно новый канал взаимодействия с технической поддержкой — мессенджеры.

Вышеописанные проблемы были озвучены сотрудниками петербургского отделения компании SAP, что далее повлекло за собой рождение данного проекта. Представленные выше идеи были воплощены в жизнь автором данной работы в рамках стажировки. Хочется отметить, что заложенный в работу идейный и архитектурный потенциал практически не исчерпаем, поэтому сегодня проект продолжает жить и развиваться.

1. Постановка задачи

Целью данной работы является разработка продукта для автоматической категоризации инцидентов и организации взаимодействия пользователя с технической поддержкой посредством общения с чат-ботом в нескольких популярных мессенджерах. Для достижения данной цели были поставлены следующие задачи:

- Провести анализ требований со стороны потенциального заказчика;
- Разработать архитектуру продукта;
- Провести базовые интеграционные испытания для оценки трудозатрат на те или иные модули;
- Найти и обучить классификатор;
- Найти платформу для построения чат-ботов с поддержкой ведения дилогов на естественном языке;
- Разработать сценарий взаимодействия пользователя с чат-ботом.

2. Требования к системе

В этом разделе описываются требования к системе с обсуждением причин, по которым они были выдвинуты. Требования были разделены на две категории — функциональные и нефункциональные.

2.1. Функциональные требования

2.1.1. Поддержка мессенджеров

Одной из основных задач разрабатываемой системы является налаживание дополнительного канала взаимодействия между пользователем и технической поддержкой. Подход, в основании которого лежит идея фокусирования на одном мессенджере может оказаться недальновидным, так как приведёт к жёсткой зависимости от стороннего ПО. Изменения в API, в публичной политике компании-разработчике или законодательстве страны, на территории, которой работает мессенджер (например, Telegram [10]), может критически отразится на судьбе разрабатываемого продукта. Поэтому одним из основных требований стала возможность поддержки широкого спектра различных мессенджеров.

2.1.2. Поддержка сервис-десков

Большинство представленных сегодня на рынке сервис-десков [1] обладают схожей функциональностью и практически одинаковым жизненным циклом инцидента. Исходя из этого, для обеспечения максимальной конкурентоспособности продукта была поставлена цель потенциально поддерживать неограниченное количество сервис-десков

2.1.3. Размещение

Некоторые компании (потенциальные заказчики), в зависимости от устройства своего IT-ландшафта, предпочитают либо полностью, ли-

[1] ориг(англ.) Service Desk — тех поддержка. Под этим словосочетанием обычно понимают конкретное ПО от конкретного производителя, которое обеспечивает работу технической поддержки — создание, редактирование, разрешение и закрытие инцидентов

бо частично размещать свои бизнес приложения в облаке (SaaS, IaaS, PaaS), другие же предпочитают разворачивать ПО на своих внутренних мощностях (on-premise). Исходя из этой ситуации, было выдвинуто требование о возможности размещения отдельных элементов приложения как в облаке, так и на мощностях заказчика.

2.1.4. Аутентификация

Так как продукт преимущественно ориентирован на корпоративных клиентов, была поставлена задача в поддержке аутентификации пользователя с помощью корпоративного почтового адреса (email) или номера телефона сотрудника, в зависимости от возможностей IT-инфраструктуры заказчика

2.1.5. Локализация

Важным критерием успешного развития продукта является ориентированность на глобальные рынки, однако локальные потребители также играют немалую роль. Поэтому было выдвинуто требование поддержки не только английского языка, в качестве основного языка интерфейса, но и других языков. При этом внедрение нового языка в меню выбора локали, должно быть реализовано с помощью добавления файла в директорию с другими локалями.

2.1.6. Команды и описание проблемы на естественном языке

Для добавления привлекательности продукту, было выдвинуто требование поддерживать не только командный интерфейс взаимодействия с чат-ботом (например, ”/new” - команда, инициирующая создание нового инцидента), но также и поддерживать возможность взаимодействия с ботом на естественном языке (например, «список моих инцидентов» инициирует вывод списка открытых инцидентов пользователя)

2.2. Нефункциональные требования

2.2.1. Классификатор

Одной из основных целей продукта является обеспечения высокой точности категоризации инцидентов, поэтому к классификатору были выдвинуты следующие требования:

- обучение на 200 тыс размеченных обращений не должно занимать более 1 суток;
- точность на подготовленных данных должна быть не менее 80%;
- является продуктом SAP (опционально).

2.2.2. Чат-бот платформа

Так как разрабатываемая система прежде всего ориентирована на корпоративных клиентов, немаловажным фактором является стоимость её обслуживания. Для минимизации затрат клиента было выдвинуто требование в поиске платформы для построения чат-ботов с максимальной прозрачной системой ценообразования для корпоративных заказчиков.

3. Обзор существующих решений

В ходе анализа требований был сформирован список необходимых компонентов и параметров, которыми должно обладать существующие решение. Далее в этом разделе приводится обзор выбранных компонентов и их аналогов. Также хочется отметить, что в процессе анализа требований перед автором данной работы встал вопрос существования рынка систем подобных разрабатываемой. Этому вопросу посвящён первый подраздел.

3.1. Полные аналоги системы

Анализ рынка программного обеспечения, решающего задачи, схожие с поставленными перед автором работы, показал, что прямого аналога разрабатываемая система на данный момент не имеет. Объяснить подобную ситуацию можно тем, что взаимодействие с тех поддержкой посредствам чат-ботов хоть и является крайне перспективным подходом, однако крупные игроки рынка (HP [8], SAP [9] и др.) по производству сервис-десков не ставят перед собой первоочередной задачей интеграцию с мессенджерами «из коробки». Связано это с тем, что эти продукты чаще всего имеют крайне развитое API, что позволяет переложить все вопросы на конкретного эксплуатанта, т.е. на нашего потенциального клиента.

На сегодняшний день существуют сервис-дески, которые предоставляют возможность взаимодействия через чат-ботов [5], однако они имеют множество недостатков, такие как:

- поддержка только 1 мессенджера (например, Telegram);
- отсутствие поддержки общения на естественном языке;
- нет возможности взаимодействовать с сателитными системами;
- низкая популярность таких сервис-десков.

Ко всему вышесказанному, хочется добавить, что далеко не редкостью является ситуация, когда компания имеет несколько различных сервис-десков разного назначения. Использование вышеупомянутых продуктов ведёт к потери идеи централизованного канала, организованного через чат-бота.

3.2. Классификатор инцидентов

Одним из основных требований, которое было выдвинуто в сторону классификатора — высокие показатели точности категоризации. Существует два пути развития ситуации:

- разработка собственно классификатора с использование алгоритмов машинного обучения;
- использование готового классификатора как сервиса.

Первый вариант требует высоких трудозатрат и знания предметной области, при этом нет никаких гарантий, что даже при помощи привлеченного специалиста, разбирающегося в предметной области, получится в короткие сроки добиться нужной точности. Поэтому выбор пал на второй вариант.

3.2.1. Amazon Comprehend

Amazon Comprehend [1] — это сервис обработки текстов, в котором применяются технологии машинного обучения. Данный сервис определяет язык текста, извлекает места, бренды или события, определяет степень позитивности или негативности текста и многое другое, однако не имеет возможности обучаться на основе выборки состоящей из пар текст-категория и соответственно выдавать категорию по тексту инцидента.

3.2.2. Microsoft Azure Text Analytics

Microsoft Azure Text Analytics [2] обладает крайней схожей функциональностью, что и сервис от Amazon, но имеет такие же недостатки.

3.2.3. SAP Service Ticket Intelligence

SAP Service Ticket Intelligence [7] — продукт, специальной разработанный для построения классификаторов по принципу «текст-категория». Основным приемуществом перед другими продуктами является ориентированность на работу с обращениями в техническую поддержку.

3.3. Чат-бот платформы

Чат-бот платформа — это сервис, позволяющий создавать чат-ботов для разных платформ и языков на разных устройствах. Обычно такие системы используется для упрощения построения диалогов с пользователем, так как имеют возможность извлечения сущностей из переписки.

3.3.1. Dialogflow

Dialogflow [3] — продукт компании Google, позволяющий создавать чат-ботов с поддержкой ведения диалога на естественном языке через различные мессенджеры. Данный продукт является одним из наиболее популярных представителей рассматриваемого рынка, однако не удовлетворяет требованиям по прозрачности ценообразования для корпоративных клиентов.

3.3.2. Recast.ai

Recast.ai [6] — система аналогичная продукту от компании Google и практически ей не уступающая по функциональности, при этом удовлетворяет всем выдвинутым к разрабатываемому продукту требованиям.

4. Архитектура

С учетом требований и анализа существующих решений, была разработана базовая архитектура системы (на Рис. 1 синим цветом обозначены компоненты, которые были разработаны автором данной работы, фиолетовым цветом — компоненты, разработанные сторонними разработчиками):

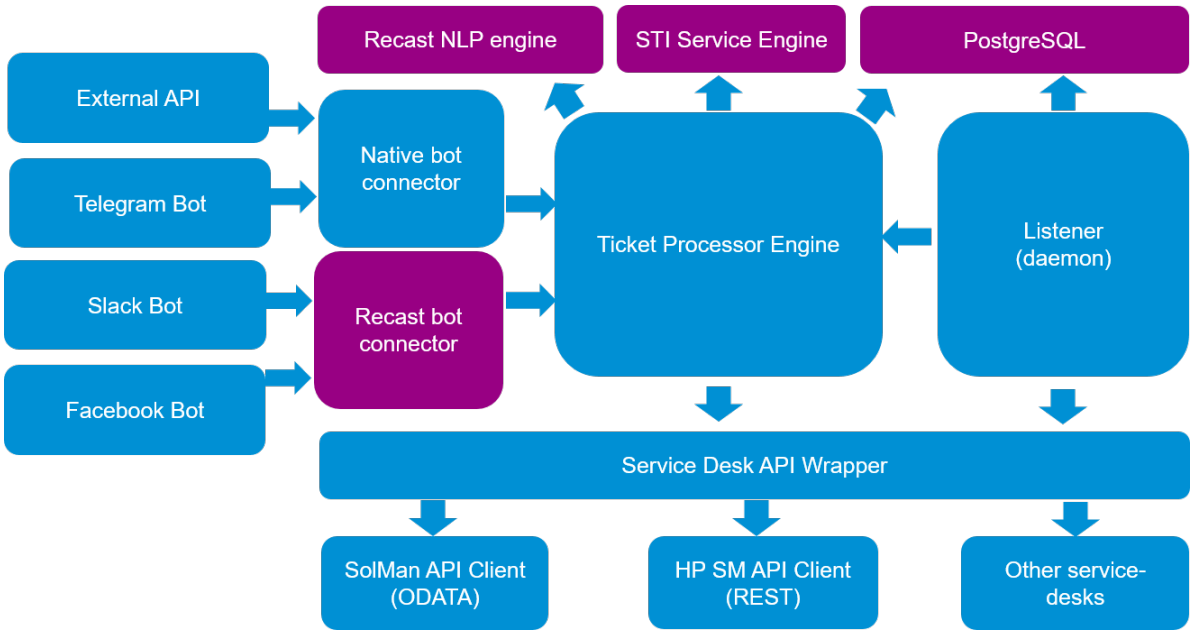

Рис. 1: Архитектура

4.1. Основные компоненты

4.1.1. Bot connector

Одним из основных требований, предъявленных к системе было обеспечение поддержки как можно большего количества различных мессенджеров. Однако в последнее время конкуренция между сервисами быстрого обмена сообщениями сильно обострилась, и каждый участник гонки пытается завлечь потенциального пользователя с помощью введения в систему новых сущностей, таких как видео-сообщения или анимированные стикеры. Возникшая ситуация приводит к тому, что для качественной поддержки большого количества мессенджеров необходима такая прослойка как **бот-коннектор**, которая будет занимать-

ся отображением сущностей конкретного мессенджера в обобщенные сущности, которыми далее будет удобно оперировать в компоненте, реализующим бизнес логику. Сервис для построения чат-ботов recast предоставляет свой бот-коннектор, который упрощает вышеописанную процедуру отображения сущностей.

Однако возможности бот-коннектора recast ограничены, в связи с тем, что он был построен с оглядкой на сущности мессенджера от Facebook, API которого хоть и является довольно развитым, но не дотягивает до возможностей, например, бот-API Telegram. В связи с этим возникла потребность в разделении бот-коннектора на 2 части: **Native bot-connector** и **Recast bot-connector**, первая из которых предназначена для «ручного» добавления какого-либо мессенджера.

Такая возможность реализована с помощью введения абстрактного класса Messenger, методами которого можно покрыть довольно широкий спектр бот-API. Например, некоторые бот-API поддерживают различные подходы к оповещению бота о новых сообщениях со стороны пользователя, такие как polling (подход, при котором бот периодический опрашивает мессенджер с целью получения информации о наличии новых сообщений) или webhook (подход, при котором мессенджер сам отправляет информацию о новых сообщениях, на ранее обговоренную точку входа). В общем случае, имплементация методов класса Messenger позволяет инкапсулировать все возможные тонкости во взаимодействии с конкретным мессенджером.

4.1.2. Service Desk API Wrapper

Для выполнения вышеописанных требований касаемо поддержки широкого спектра различных сервис-десков было принято решение о внедрении некой прослойки, схожей по принципу с бот-коннектором, т.е. решающей задачу отображения сущностей конкретного сервис-деска в сущности бизнес логики. Аналогично бот-коннектору был введён абстрактный класс ServiceDesk, имплементация методов которого добавляет поддержку нового сервис-деска.

Однако, помимо всего, к системе было выдвинуто требование в под-

держке гибкого размещения. Возникло понимание, что большинство потенциальных заказчиков размещают свои сервис-дески on-premise, что ведёт к определённым трудностям с доступом, если, например, разрабатываемое приложение будет исполнятся в облаке. Обычно для решения подобных проблем применяются средства, создающие туннель между внутренней сетью компании и облаком, такие как SAP Cloud Connector или AWS Direct Connect, однако из-за высоких требований, предъявляемых к безопасности туннелей, для разработчиков процесс построения взаимодействия сильно усложняется. В связи с этим был внедрён абстрактный класс RequestServiceDesk, который позволяет инкапсулировать запросы из облака в on-premise систему.

4.1.3. Listener

Подавляющее большинство сервис-десков, представленных сегодня на рынке, не предусматривают сценарий, при котором система генерирует внешний вызов (например, http-запрос) при каких-либо изменениях в тикете. Это приводит к тому, что единственным выходом для получения обновлений по какому-либо инциденту является периодический опрос сервис-деска и сравнения только что полученных данных с теми, которые были получены ранее.

Стоит отметить, что слишком частые обращения к сервис-деску периодически могут приводить к отказу в обслуживании, так как вышеописанный сценарий является генератором паразитного трафика и может мешать стабильной работе системе обработки инцидентов. Поэтому было принято решение разрабатывать listener с использованием готового решения[4] для планирования с возможностью вытеснение задач на основе различных метрик (например, на основе частоты изменений в том или ином инциденте).

4.1.4. Ticket Processor Engine

Наиболее монументальной частью вышепредставленной архитектуры является модуль Ticket Processor Engine, который реализует бизнес

логику и объединяет все остальные модули, в том числе и не попавшие на диаграмму.

Как и большинство программных продуктов, создающихся не только в учебных или лабораторных целях, разрабатываемая система не ограничилась использованием строго одного паттерна проектирования, однако можно сказать, что основополагающим стал паттерн State. Это связано с тем, что наиболее удобно рассматривать взаимодействие с ботом как некий конечный автомат, переходами в котором являются определённые действия пользователя. Такой подход позволяет строго задать все возможные сценарии использования, а также обеспечить высокую степень гибкости (так как в одно из состояний автомата можно встроить другой автомат).

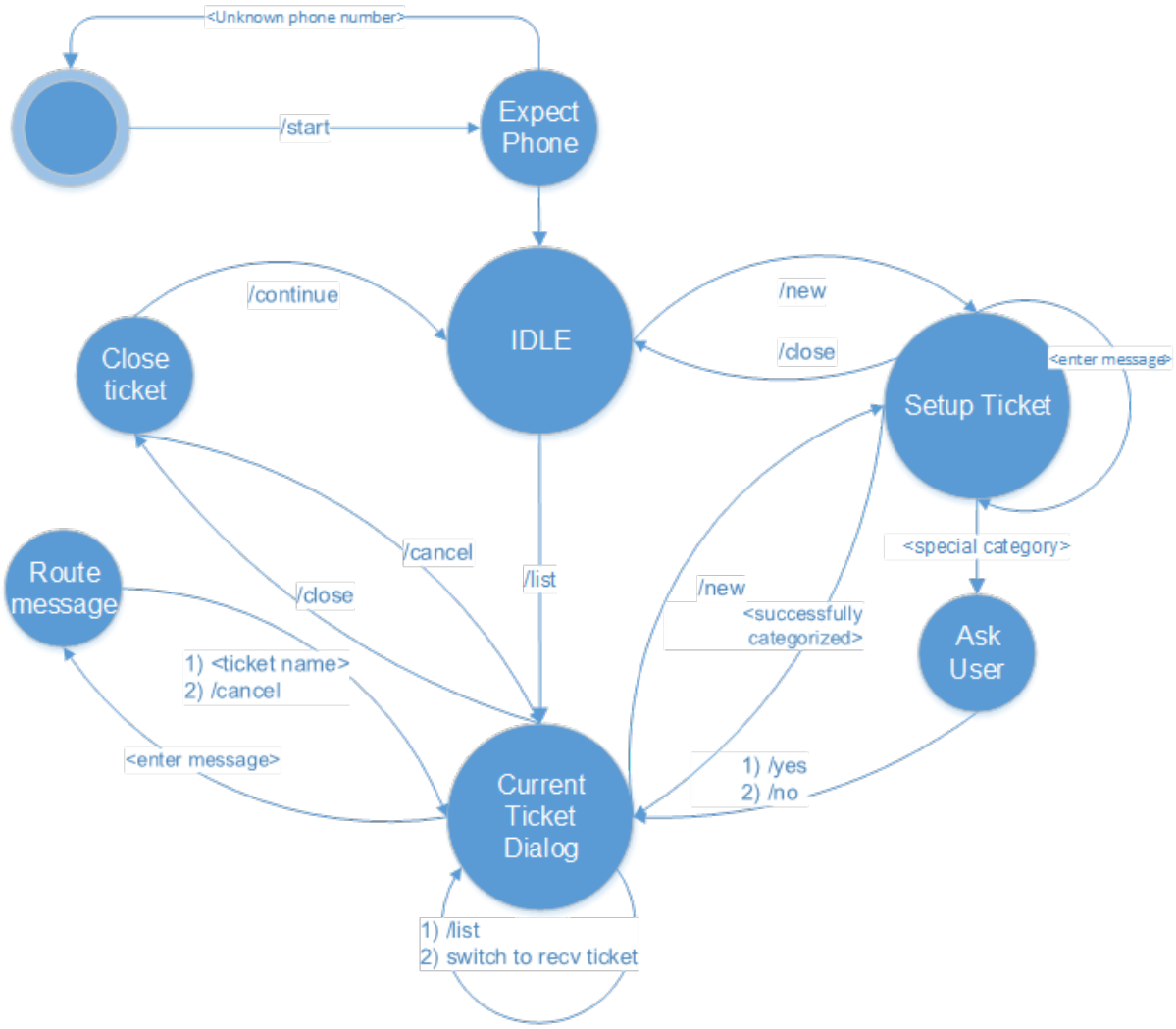

Рис. 2: Сценарий взаимодействия

Также хочется добавить, что для достижения высокой степени изменяемости был выбран подход, при котором бизнес логика оперирует максимально высокоуровневыми сущностями, такими как User, Ticket, Category и пр. Подобная практика успела хорошо себя зарекомендовать в течение жизни данного проекта.

4.2. Используемые технологии

Из-за определенных ограничений, связанных с особенностями размещения разрабатываемой системы, выбор между технологиями оказался невелик. Базовой платформой для хостинга проекта стала SAP Cloud Platform, которая пока что имеет ограниченный набор так называемых билдпаков [2]. Помимо этого стояли задачи как можно более быстрого развертывания процесса разработки. Поэтому основным претендентом стал Node.js по ряду следующих причин:

- неблокирующий ввод/вывод;
- гибкая типизация (JavaScript — динамически типизированный язык, однако использования «ручной» проверки типов там где это критично позволяет добиться баланса);
- веб-ориентированность со всеми вытекающими.

[2]ориг(англ.) buildpack — пакет для развертывания контейнера с окружением под определенный стек технологий.

Заключение

В ходе выполнения данной работы были достигнуты следующие результаты:

- проведён анализ требований со стороны потенциального заказчика;
- разработана архитектуру продукта;
- проведены базовые интеграционные испытания для оценки трудозатрат на те или иные модули;
- Выбран и обучен классификатор;
- Выбрана платформа для построения чат-ботов с поддержкой ведения диалогов на естественном языке;
- разработан сценарий взаимодействия пользователя с чат-ботом.

Список литературы

[1] amazon.com. Выявление закономерностей и взаимосвязей в тексте. — 2018. — URL: https://aws.amazon.com/ru/comprehend/.

[2] azure.microsoft.com. Определение тональности, ключевых фраз и языка текста. — 2018. — URL: https://azure.microsoft.com/ru-ru/services/cognitive-services/text-analytics/.

[3] dialogflow.com. Build natural and rich conversational experiences. — 2018. — URL: https://dialogflow.com/.

[4] lykmapipo@github. A job scheduler utility for kue. — 2018. — URL: https://github.com/lykmapipo/kue-scheduler.

[5] okdesk.ru. Telegram бот службы поддержки. — 2017. — URL: https://okdesk.ru/blog/telegram-bot.

[6] recast.ai. The collaborative platform to build, train, deploy and monitor intelligent bots for developers. — 2018. — URL: https://recast.ai/.

[7] sap.com. SAP Service Ticket Intelligence. — 2018. — URL: https://www.sap.com/products/service-ticket-intelligence.html.

[8] wikipedia. HP Service Manager software. — 2018. — URL: https://en.wikipedia.org/wiki/HP_Service_Manager_software.

[9] wikipedia. SAP Solution Manager. — 2018. — URL: https://en.wikipedia.org/wiki/SAP_Solution_Manager.

[10] РБК. Битва за Telegram. — 2018. — URL: https://www.rbc.ru/story/5950cd3c9a79477ac23ef59d.