Онтологический подход и обработка языковой информации в задаче роботизированной поддержки граждан при обращении в кол-центры и на «горячие линии»

А. С. Гончаров, email: artemgoncharov6003@gmail.com [1]
В. В. Гаршина, email: garshina.veronika@gmail.com [2]

[1] ФГБОУ ВО «МИРЭА – Российский технологический университет»
[2] Воронежский государственный университет

Аннотация. *В работе исследуется возможность роботизации обработки обращений граждан в кол-центры и на «горячие линии» организаций различных профилей по вопросам медицинского, социального и бытового обслуживания. Исследование заключается в определении возможности преобразования речевой информации (аудиозаписи телефонного обращения гражданина) в текст с последующим автоматическим извлечением из него ключевых фактов и размещением их в фактологической базе данных, созданной на основе онтологического подхода. Изучается возможность обеспечения жизненного цикла заявок, создаваемых по обращениям граждан, посредством автоматического формирования, отправки и получения коротких текстовых сообщений (смс).*

Ключевые слова: *онтология задачи; обращения граждан; заявка; жизненный цикл заявки; заявитель; исполнитель заявки; аудиозапись; преобразование аудиозаписи в текст; распознавание речи; лексический парсер; выделение фактов из текста; графовая база данных; граф знаний; правила вывода; смс; Protégé; GraphDB; Python; Yargy-парсер; Arduino; GPRS-модуль SIM900.*

Введение

В современных условиях интеллектуальная обработка входящих телефонных звонков и последующее за ней снижение нагрузки на диспетчеров кол-центров является важной и социально значимой задачей. Особенно острый характер эта проблема приобрела в условиях вынужденной самоизоляции, вызванных коронавирусом COVID-19: многочисленные случаи недозвонов на «горячие линии» массово

отмечались в социальных сетях и ряде местных и региональных изданий.

Целью работы является разработка программного (программно-аппаратного) решения для упрощения и ускорения обработки обращений граждан в службы поддержки и оказания помощи.

Для достижения поставленной цели определены следующие задачи:

- Исследовать возможность преобразования аудиозаписи в текст, для этого рассмотреть программные библиотеки, которые могут быть задействованы для распознавания речи;
- Изучить методы выделения ключевой информации из текстов на русском языке, для этого исследовать доступность и функциональные возможности популярных лексических анализаторов (парсеров);
- Создать онтологическую модель задачи (в объёме, достаточном для выполнения проекта), используя онтологический подход к моделированию и представлению данных в виде графов;
- Изучить возможность наполнения онтологии выделенными фактами, сохранения их в графовой базе данных, возможность вывода на их основе новых знаний посредством аксиом логического вывода;
- Изучить технологии автоматического формирования, отправки и получения смс, выбрать необходимые программные и аппаратные инструменты;
- Определить целевую архитектуру системы и реализовать программно-аппаратное решение.

Первые результаты, полученные в ходе решения сформулированных выше задач, опубликованы в работе [1].

1. Анализ существующих решений

Анализ существующих решений показал: несмотря на то, что с середины прошлого века до настоящего времени технологии распознавания речи прошли большой путь развития [2], основанные на них программные решения предназначены для взаимодействия типа «вопрос-ответ», и притом только в самом общем, широко используемом контексте. Применительно к конкретной предметной области без специальной подготовки и настройки они не будут полезны, т. е., эти решения не являются самодостаточными. По той же причине не являются самодостаточными и лексические анализаторы: до того, как использовать их для извлечения из текстовой информации фактов требуемого качества, их необходимо изучить и настроить.

Например, ни один существующий лексический парсер, находящийся в открытом доступе, не сможет без специальной программной настройки выделить из простого предложения «*Прошу вызвать доктора по адресу улица Ржевская, дом 9, квартира 101*» название специальности и адрес. Для того чтобы это произошло, нужно создать и наполнить словарь специальностей, настроить контекстно-сводные грамматики – правила, по которым из предложения будут выделены факты, реализовать связующий программный код.

Решения с использованием голосовых ботов и чатботов в последнее время активно развиваются [3]. Оба типа решений предполагают активный обмен информацией с клиентом. Вопросы бота могут быть неоднозначны или сложны для восприятия человеком, а ответы человека, соответственно, могут быть сложны и запутанны для восприятия их ботом. Кроме того, такие системы не способны к мультизадачности: множество вопросов пользователя сбивает их с толку, а если клиент резко меняет тему или точку зрения, то бот становится вовсе беспомощным. Несовершенство подобных решений ограничивает их повсеместное использование.

Однако известны и эффективные решения с использованием голосовых ассистентов. Разработчики робота, принимающего заявки на поверку счётчиков расхода воды и передающего данные в контролирующий орган, отмечают [4], что все сценарии диалогов сведены к тому, чтобы человек отвечал только «да» или «нет», либо называл конкретное число, но даже при таких ограничениях клиента возможны неоднозначности.

Принципиальное отличие созданного программного робота, являющегося объектом настоящего исследования, от указанных видов телефонных «ботов» заключается в том, что как таковой диалог между клиентом и роботом не предусматривается: гражданин просто оставляет голосовое обращение со своей просьбой в свободном формате. Специальное коммуникационное оборудование сохраняет аудиозапись обращения, которой далее занимается программный робот. Особенность решения в том, что человек не скован жёсткими рамками возможных вариантов ответов: главное, чтобы в его речи были слова о том, какая помощь ему требуется и по какому адресу её нужно оказать.

2. Описание предлагаемого решения

Проектирование и реализация системы выполняется на основе **онтологического подхода** [5]. С одной стороны, это позволяет формализовать и представить понятия, составляющие словарь задачи, в формате, понимаемом программными системами и принятом для обмена знаниями между ними (т. е., согласно определению Томаса Грубера [6],

«специфицировать концептуализацию»), а с другой – в соответствии с парадигмой «открытого мира» [5] – оставить возможность дальнейшего расширения и уточнения онтологии, поскольку на момент проектирования и реализации не вся информация может быть в ней отражена. Эти возможности обусловливают выбор данного подхода.

Факты, наполняющие онтологию, на самом простом уровне имеют форму так называемых «триплетов»: subject – predicate – object. Данные такого вида удобно хранить и обрабатывать в графовых базах данных: в них факты представляются в виде графов, где вершинам соответствуют объекты, а рёбрам – отношения между ними. В качестве хранилища выбрано решение компании OntoText – **GraphDB** (graphdb.ontotext.com). Для работы с данными в ней используется язык SPARQL.

Онтология, построенная для конкретной задачи, называется **онтологией задачи**.

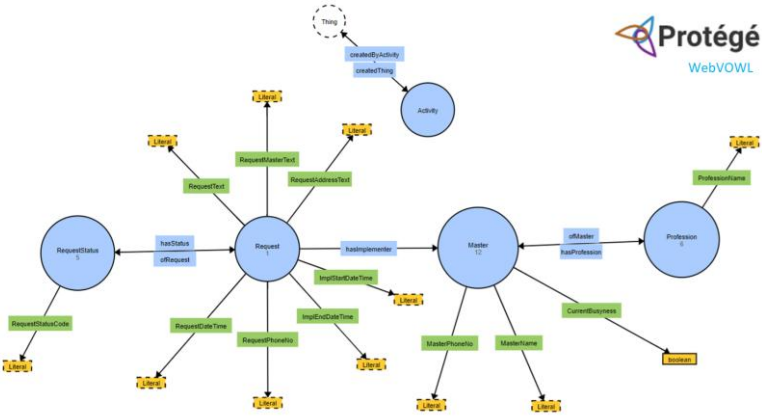


Рис. 1. Онтологическая модель задачи

В онтологию поставленной задачи введены следующие классы (рис. 1):

- **Request** (Заявка) – сущность, соответствующая звонку и, вместе с тем, заявке;
- **Master** (Мастер) – сущность, соответствующая специалисту;
- **Profession** (Специальность) – сущность, представляющая специальность, которой может владеть мастер;
- **RequestStatus** (Статус заявок) – сущность, представляющая один из возможных статусов заявки: Сформирована, В обработке, Назначен мастер, Выполнена, Отклонена;

– **Activity** (Активность) – сущность, соответствующая техническому процессу, в рамках которого осуществляется генерация новых заявок.

При обращении человека, нуждающегося в помощи или услуге, осуществляется запись и сохранение его речи в виде аудиофайла, который затем принимается в обработку программным роботом. Сначала выполняется распознавание речи, т. е. представление обращения в текстовом виде. Далее выполняется анализ предложений текста и извлечение из них ключевых фактов, наполнение онтологии задачи и порождение новых значимых сущностей – в частности, заявок, поддержка жизненного цикла которых представляет достаточно интересную и относительно самостоятельную подзадачу.

На рис. 2 представлена в общем виде функциональная схема предлагаемого решения.

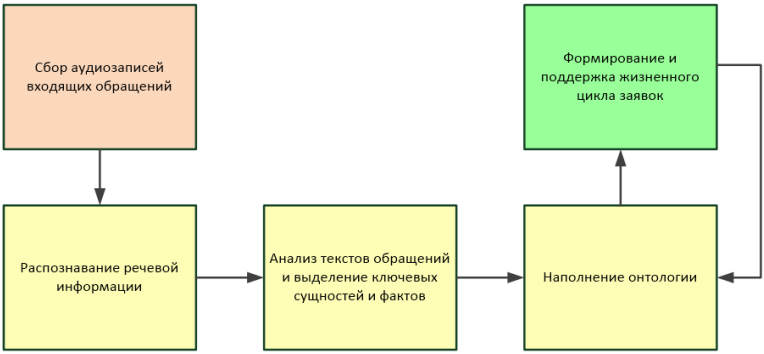


Рис. 2. Функциональная схема решения

Рассмотрение специального коммуникационного оборудования и программного обеспечения, используемого в кол-центрах и на «горячих линиях» для записи и сохранения голосовых обращений, выходит за рамки данной работы.

3. Распознавание речевой информации

Типовые системы распознавания речи включают следующие этапы [7]:

– Шумоочистка и отделение полезного сигнала.

Следует учитывать, что в случае, если исходный речевой сигнал не сильно зашумлен, то некоторые эпизоды речи (затихающий голос) могут быть приняты за шум, из-за чего могут возникать пропуски слов при распознавании.

– Акустическая адаптация.

Выполняется на основании оцененного уровня помех и искажений и использует акустическую модель. Акустическая модель позволяет оценить фрагмент речи с точки зрения схожести на звуковом уровне.

– Определение участков, содержащих речь, и их распознавание.

С использованием языковой модели происходит выделение фонетических и просодических (тембр, высота и сила голоса, мелодика, темп, пауза, модуляции голоса, ритм, логическое ударение, дикция) вероятностных характеристик для синтаксического, семантического и прагматического анализа. Определяются части речи, формы слов и статистические связи между словами.

– Декодирование.

В декодере – основном блоке системы распознавания – сопоставляется входной речевой поток с результатами, полученными на основе акустической и языковой моделей, и определяется наиболее вероятная последовательность слов, которая и является конечным результатом распознавания.

Для распознавания аудиозаписей была выбрана открытая библиотека **speech_recognition**. Библиотека может обрабатывать аудиозаписи только в формате wav, поэтому для преобразования аудиофайлов других форматов (например, mp3) к формату wav используется программа-конвертер с открытым кодом FFmpeg.

Библиотека speech_recognition содержит несколько разных методов распознавания. Наилучшим образом проявил себя метод **recognize_google**, использующий Google Speech API (рис. 3).

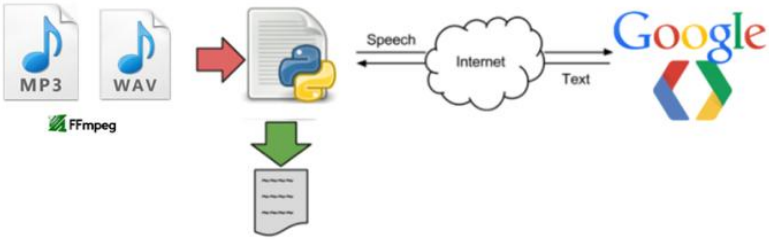


Рис. 3. Схема обращения к Google Speech API

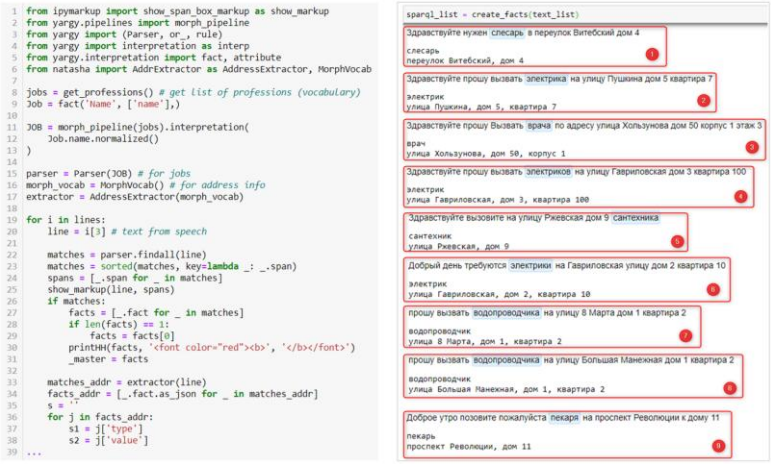
Следует отметить, что в качестве других облачных сервисов для распознавания речи могут использоваться, например, решения Yandex Speechkit (https://cloud.yandex.ru/services/speechkit) и SaluteSpeech (https://sbercloud.ru/ru/aicloud/salutespeech, разработчик SberDevices) – они дают высокое качество распознавания, но являются платными. В качестве офлайн решения может использоваться библиотека Vosk (https://alphacephei.com/vosk) с открытым исходным кодом.

Результатом обработки языковой информации, представленной в аудиозаписях обращений граждан, являются предложения в текстовом виде.

4. Анализ текстов обращений и выделение ключевых фактов

Для анализа текстовых предложений на русском языке могут использоваться разнообразные лексические парсеры (программные библиотеки, выделяющие данные с определёнными свойствами из текстовых массивов). Анализ наиболее популярных: **Томита-парсер** (разработка Yandex), **DeepPavlov** (разработка МФТИ) и **Yargy-парсер** [8, 9] – показал преимущество использования последнего: он реализован на Python и использует библиотеку **natasha**, в которой уже настроены правила для извлечения имён, адресов, дат, сумм денег, правила для извлечения названий организаций и географических объектов.

В предлагаемом решении Yargy-парсер и библиотека natasha используются для извлечения названия профессии (специальности) и адресной информации, которые могут содержаться в тексте, полученном на основе аудиозаписи обращения гражданина (рис. 4). Вдохновение для реализации проекта в этой части придала работа Кукушкина А. [10], в ней на примерах показывается, как из текстовых данных выделить ключевые факты.



```
from ipymarkup import show_span_box_markup as show_markup
from yargy.pipelines import morph_pipeline
from yargy import (Parser, or_, rule)
from yargy import interpretation as interp
from yargy.interpretation import fact, attribute
from natasha import AddrExtractor as AddressExtractor, MorphVocab

jobs = get_professions() # get list of professions (vocabulary)
JOB = fact('Name', ['name'],)

JOB = morph_pipeline(jobs).interpretation(
    JOB.name.normalized()
)

parser = Parser(JOB) # for jobs
morph_vocab = MorphVocab() # for address info
extractor = AddressExtractor(morph_vocab)

for i in lines:
    line = i[3] # text from speech

    matches = parser.findall(line)
    matches = sorted(matches, key=lambda _: _.span)
    spans = [_.span for _ in matches]
    show_markup(line, spans)
    if matches:
        facts = [_.fact for _ in matches]
        if len(facts) == 1:
            facts = facts[0]
        printHH(facts, '<font color="red"><b>', '</b></font>')
        _master = facts

    matches_addr = extractor(line)
    facts_addr = [_.fact.as_json for _ in matches_addr]
    s = ''
    for j in facts_addr:
        s1 = j['type']
        s2 = j['value']
...
```

```
sparql_list = create_facts(text_list)

Здравствуйте нужен слесарь в переулок Витебский дом 4
слесарь
переулок Витебский, дом 4                                    1

Здравствуйте прошу вызвать электрика на улицу Пушкина дом 5 квартира 7
электрик
улица Пушкина, дом 5, квартира 7                             2

Здравствуйте прошу Вызвать врача по адресу улица Хользунова дом 50 корпус 1 этаж 3
врач
улица Хользунова, дом 50, корпус 1                           3

Здравствуйте прошу вызвать электриков на улицу Гавриловская дом 3 квартира 100
электрик
улица Гавриловская, дом 3, квартира 100                      4

Здравствуйте вызовите на улицу Ржевская дом 9 сантехника
сантехник
улица Ржевская, дом 9                                        5

Добрый день требуются электрики на Гавриловская улицу дом 2 квартира 10
электрик
улица Гавриловская, дом 2, квартира 10                       6

прошу вызвать водопроводчика на улицу 8 Марта дом 1 квартира 2
водопроводчик
улица 8 Марта, дом 1, квартира 2                             7

прошу вызвать водопроводчика на улицу Большая Манежная дом 1 квартира 2
водопроводчик
улица Большая Манежная, дом 1, квартира 2                    8

Доброе утро позовите пожалуйста пекаря на проспект Революции к дому 11
пекарь
проспект Революции, дом 11                                   9
```

Рис. 4. Фрагмент кода и примеры извлечения фактов: адреса и специальности

Библиотека natasha – одна из набора Python-библиотек (https://natasha.github.io), созданных в рамках проекта Natasha и служащих для обработки текстов на естественном русском языке.

Преимущества использования библиотеки natasha также отмечаются в работе [11], посвящённой извлечению фактографической информации о процессе протекания пандемии COVID-19 из открытых новостных статей интернета. Авторы упомянутой работы на ряде примеров показывают, как в результате NER-анализа с помощью библиотеки Natasha успешно получают все топонимы: названия организаций, городов и регионов.

В нашей задаче, в которой требуется распознавание более детализированных локаций, сложность представляют комплексные названия улиц, например: улица 45-ой Стрелковой Дивизии, улица 26 Бакинских Комиссаров, 4-я улица 8 Марта, 5-й проезд Марьиной рощи, проезд Верхний Михайловский Поперечный и т. п.

5. Наполнение онтологии фактами и аксиомы семантического вывода

Одновременно с разбором текстов формируется список предложений на языке SPARQL (рис. 5).

```
# Создание активити процесса генерации сущностей (Activity20210330100438)
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.cs-vsu-gas.org/ex-onto/test#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
insert {:Activity20210330100438 rdf:type :Activity, owl:NamedIndividual}
where {filter not exists {:Activity20210330100438 rdf:type :Activity}}
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.cs-vsu-gas.org/ex-onto/test#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
insert {
    :Request20210330100438_8925 rdf:type :Request , owl:NamedIndividual ;
    :hasStatus :01_Сформирован ;
    :RequestPhoneNo '9034494910' ;
    :RequestDateTime '20210325080629' ;
    :RequestText 'Здравствуйте прошу вызвать электриков на улицу Гавриловская дом 3 квартира 100' ;
    :RequestAddressText 'улица Гавриловская, дом 3, квартира 100' ;
    :RequestMasterText 'электрик' ;
    :createdByActivity :Activity20210330100438 .
}
where {filter not exists {:Request20210330100438_8925 rdf:type :Request}}
```

Рис. 5. Пример текста сгенерированной команды на языке SPARQL

Далее предложения на SPARQL передаются веб-сервису REST API сервера базы данных GraphDB, который выполняет их, создавая новые факты в указанном репозитории графовой базы данных и тем самым наполняя онтологию.

Результаты выполнения можно видеть в пользовательском интерфейсе GraphDB примерно в таком виде (рис. 6).

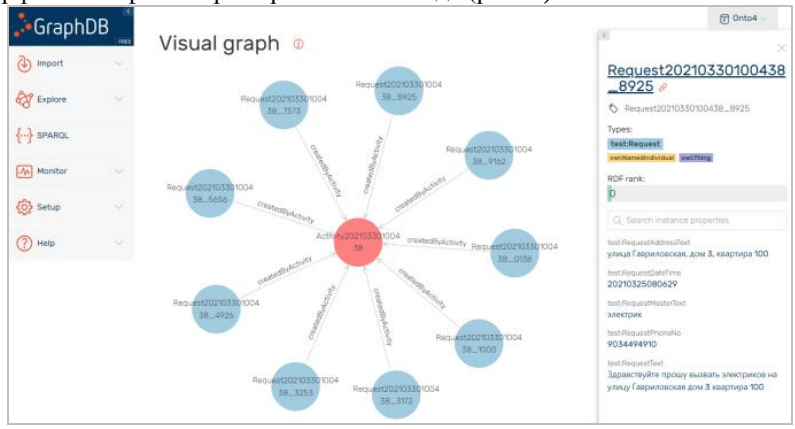


Рис. 6. Пример представления фактов в графовой базе данных

GraphDB – одно из немногих хранилищ триплетов, позволяющее определить аксиомы логического вывода на графах и выполнить семантический вывод новых фактов (знаний) в любом масштабе в режиме реального времени. Благодаря онтологическому подходу унифицируются операции по логическому выводу и упрощается добавление новых аксиом, при этом аксиомы сохраняются в том же формате, что и сами данные.

При создании репозитория в GraphDB указывается, какой набор аксиом логического вывода будет использоваться. Для онтологии можно указать стандартный набор аксиом (на основе RDFS), а можно определить собственную аксиоматическую систему. Специальные алгоритмы логического вывода (они называются «ризонерами», от англ. semantic reasoner) на основании аксиом для существующего графа автоматически формируют (выводят) новые факты (знания).

В GraphDB используется собственный семантический механизм рассуждений.

Ниже приведён пример аксиомы, используемой для определения доступных мастеров (рис. 7). Для упрощения считается, что профильный специалист занят, если у него в работе находится хотя бы одна заявка. На основании правила вывода определяется занятость специалиста.

```
Prefices
{
    rdf : http://www.w3.org/1999/02/22-rdf-syntax-ns#
    rdfs : http://www.w3.org/2000/01/rdf-schema#
    owl : http://www.w3.org/2002/07/owl#
    onto : http://www.ontotext.com/
    xsd : http://www.w3.org/2001/XMLSchema#
    test  :  http://www.cs-vsu-gas.org/ex-onto/test#
}

Axioms
{
…
     <test:hasImplementer>      <rdf:type>  <owl:ObjectProperty>
     <test:hasStatus>           <rdf:type>  <owl:ObjectProperty>
     <test:RequestStatusCode>       <rdf:type>  <owl:DatatypeProperty>

     <test:CurrentBusyness>         <rdf:type>  <owl:DatatypeProperty>
}

Rules
{
…

Id: infr1
     req <test:hasStatus> st
     st <test:RequestStatusCode> "03"^^xsd:integer
     req <test:hasImplementer> master
     ----------------------
     master <test:CurrentBusyness> "true"^^xsd:boolean

…
}
```

Рис. 7. Пример описания аксиомы логического вывода

Следующий пример показывает, как логические правила вывода содействуют облегчению понимания и уменьшению количества условий в SPARQL-запросах, на примере определения доступности профильного специалиста. Если в фильтре запроса, представленного на рис. 8, используется 4 предиката и фильтр, то в фильтре запроса, основанного на использовании введённой аксиомы, остался всего 1 предикат (рис. 9). Т. е., количество логических условий в фильтре сократилось в 5 раз, при этом запрос в более простой форме выглядит более понятно.

```
# Определение свободных мастеров (без правил вывода)
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.cs-vsu-gas.org/ex-onto/test#>
select ?profname ?name
where {
    ?master rdf:type :Master; :MasterName ?name; :hasProfession ?prof .
    ?prof :ProfessionName ?profname .
    filter not exists {
        ?req rdf:type :Request; :hasStatus ?st; :hasImplementer ?master .
        ?st :RequestStatusCode ?stcode .
        filter (?stcode = 03)
    }
}
order by ?profname ?name
```

Рис. 8. Пример SPARQL запроса без использования аксиом

```
# Определение свободных мастеров (с правилами вывода)
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.cs-vsu-gas.org/ex-onto/test#>
select ?profname ?name
where {
    ?master rdf:type :Master; :MasterName ?name; :hasProfession ?prof .
    ?prof :ProfessionName ?profname .
    filter not exists {
        ?master :CurrentBusyness true
    }
}
order by ?profname ?name
```

Рис. 9. Пример SPARQL запроса с использованием аксиомы

6. Поддержка жизненного цикла заявок

Выполнение сгенерированных SPARQL-команд приводит к наполнению онтологии, хранимой в репозитории GraphDB, новыми фактами. Новым заявкам присваивается первоначальный статус: «**01_Сформирована**». Программный робот выявляет заявки с таким статусом и переводит их в статус «**02_В_обработке**», определяя в соответствии с требованиями заявки занятость специалистов соответствующих специальностей, далее формирует и направляет им смс о новых заявках с кодами подтверждений. Специалиста, первым отправившего код подтверждения, робот назначает исполнителем заявки и сообщает ему об этом посредством смс, заявка переходит в статус «**03_Назначен_мастер**», а заявителю приходит смс о том, что его обращение принято в работу и следует ожидать звонка специалиста для уточнения деталей.

Также программный робот контролирует сроки исполнения заявок, на которые назначены специалисты: если с момента начала работы прошло некоторое время (например, сутки), то робот отправляет смс исполнителю, запрашивая код подтверждения, соответствующий завершению или отмене заявки. По получении этого кода робот переводит заявку в один из финальных статусов: «**04_Выполнена**» или «**05_Отклонена**».

7. Управление информацией на основе смс

Смс (транслитерация от англ. сокр. SMS – Short Message Service – «служба коротких сообщений») – технология приёма и передачи коротких текстовых сообщений с помощью сотового телефона, один из стандартов сотовой связи [12].

Преимуществами использования смс являются быстрое время доставки (обычно не более 10 секунд), возможность отправки сообщения на выключенный или находящий вне зоны действия сети телефон, а также на телефон, занятый в данный момент разговором абонента. При этом смс почти не нагружают сотовую сеть.

Технология SMS поддерживается основными сотовыми сетями (GSM, NMT, D-AMPS, CDMA, UMTS). SMS была создана как составная часть стандарта GSM Phase 1.

Текст смс может состоять из алфавитно-цифровых символов, знаков пунктуации и служебных знаков. Максимальный размер сообщения в стандарте GSM – 140 байт, поэтому при использовании 7-битной кодировки (латинский алфавит и цифры) максимальная длина сообщения может быть 160 знаков, 8-битной кодировки (немецкий и французский языки) – 140 знаков, 2-байтной кодировки UCS-2 (русский, китайский, арабский и др.) – не более 70 знаков.

При передаче сообщения программным путём в исходной среде, как правило, используется кодировка UTF-8, поэтому требуется конвертация текста из UTF-8 в соответствующую кодировку: например, для текста на русском языке – в UCS-2 (Юникод). Однако основная сложность заключается не в конвертации текста, а в том, что для корректной передачи сообщения по сети его необходимо дополнительно представить в PDU-формате, в соответствии с протоколом SMPP (Short Message Peer-to-Peer) – протоколом одноранговой передачи коротких сообщений [13]. В SMPP сообщение представляется в виде заголовка PDU и тела PDU, в двоичном формате.

Кроме того, при передаче сообщения из программного кода на устройство его требуется представить в бинарной форме. Таким образом, получается очень непростая цепочка преобразований текста на русском языке: UTF-8 → UCS-2 → PDU → binary. Для латинского

алфавита цепочка выглядит существенно проще: UTF-8 → binary. Поскольку предполагается, что получатели смс сообщений – прежде всего, русскоязычные граждане, требуется реализация более универсальной длинной цепочки. Минусом здесь является не только сложность реализации, но и то, что длина сообщений после всех преобразований будет более короткой (не более 50 символов), что в ряде случаев может требовать разбивки смс на части.

Многие компании в целях информирования используют смс-рассылки. Например, МЧС таким образом может массово проинформировать население о грядущих погодных условиях, банк – сообщить владельцам о действиях со счетами их банковских карт, магазины – привлечь покупателей сведениями о скидках и акциях, службы доставки – уведомить о статусе заказа, и т. д.

В ряде случаев смс может содержать ключевое слово или цифру, для обратной связи с клиентом. Технология двустороннего обмена смс позволяет наладить коммуникацию с клиентом, при этом взаимодействие происходит по инициативе пользователя. Использование номера для приема смс позволяет компаниям проводить опросы, розыгрыши, а также автоматизировать смену паролей или проверку баланса. От пользователя требуется лишь отправить смс с ключевым словом/цифрой на мобильный номер. В России массовая рассылка смс операторами связи правомерна, что подтверждено Высшим арбитражным судом РФ (https://pravo.ru/news/view/9726/).

Смс-рассылка может также использоваться для информирования сотрудников компании о проводимых мероприятиях, планах, поставленных задачах, получения обратного отклика. Таким образом, смс-рассылка позволяет наладить эффективные коммуникации внутри компании и способствовать росту эффективности управления персоналом.

Для реализации процессов управления информацией на основе смс в части аппаратной поддержки рассматриваемого решения первоначально была выбрана программно-аппаратная платформа Arduino и модуль GPRS. В качестве GSM/GPRS-модуля был рассмотрен SIM900 в силу его невысокой цены, доступности в РФ и совместимости с платформой Arduino. Связующее ПО между Arduino и модулем SIM900 реализуется на C++ с использованием AT-команд, наиболее актуальные из которых представлены в таблице ниже.

Таблица

Примеры основных AT-команд модуля SIM900

Команда	Описание	Пример
AT+CMGF=1 AT+CMGF=0	Установка режима приёма/передачи смс: 1 – в текстовом формате (в режиме передачи не поддерживается кириллица), 0 – в формате PDU (поддерживает национальные кодировки, но более сложный в реализации) PDU = Protocol Data Unit	AT+CMGF=1 OK
AT+CSCS="GSM" AT+CSCS="UCS2"	Кодировка символов Для чтения смс используется кодировка GSM, для отправки – UCS2	AT+CSCS="GSM" OK
AT+CMGL="REC READ" AT+CMGL="REC UNREAD" AT+CMGL="ALL"	Чтение смс. REC READ – прочитать ранее прочитанные REC UNREAD – прочитать непрочитанные ALL – прочитать все	AT+CMGL="ALL" +CMGL: 1,"REC UNREAD","+71234567890","21/04/16,11:28:48+12" Test
AT+CMGR=xx	Чтение смс по заданному номеру	AT+CMGR=2 +CMGR: "REC READ","+71234567890", "21/04/16,11:30:04+12" 0422043504410442 OK
AT+CMGD=xx	Удаление смс по заданному номеру	AT+CMGD=1 OK
AT+CMGS=nn >text in PDU format, length(text) = nn	Отправка смс в формате PDU nn – количество передаваемых символов сообщения + заголовка (=14) Ограничение: не более 70 (по факту – 50)	AT+CMGS=22 >0011000B911732547698F0001AFF08042204350441044 2 +CMGS: 183 OK

Например, команда для отправки слова «Тест» на номер +71234567890 выглядит следующим образом (рис. 10).

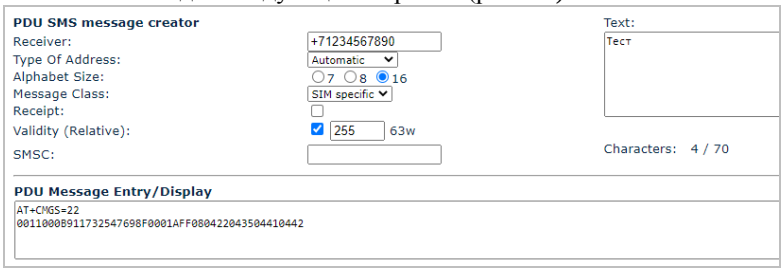


Рис. 10. Пример представления слова «Тест» в виде смс в PDU-формате

Подробное описание AT-команд для SIM900 приведено на сайте компании-производителя SIMCom (www.simcom.ee).

В качестве альтернативы вместо использования программно-аппаратной связки Arduino и GPRS-модуля может быть рассмотрен вариант с использованием мобильного устройства (смартфона) и мобильного приложения, реализующего функции отправки и получения смс и взаимодействующего с сервером. Например, мобильное приложение **Airmore** (https://airmore.com/) позволяет получать и отправлять сообщения, файлы, управлять контактами, дублировать изображение с устройства на компьютер. Взаимодействие между Airmore и серверной стороной при этом может обеспечиваться библиотекой **pyairmore**, реализованной на Python. Обмен данными между мобильным устройством с ОС Android и сервером с программным роботом осуществляется по беспроводной сети.

В этом варианте упрощается реализация функций отправки и получения заявок, поскольку они возлагаются на мобильное приложение, снимаются ограничения на длину смс, работа с хранилищем смс также становится проще. С другой стороны, возникает необходимость установки мобильного приложения и настройки синхронизации между мобильным устройством и сервером.

8. Варианты реализации решения

Ввиду доступности библиотек, предназначенных для обработки языковой информации, реализованных либо предоставляющих интерфейсы на языке Python, он был выбран в качестве основного языка для разработки программной логики решения.

Модуль на Python реализует следующие функции программного робота:

- обнаружение новых аудиозаписей, их преобразование вплоть до создания новых фактов в графовой базе;
- обработка заявок в статусе 01_Сформирована;
- обработка заявок в статусе 03_Назначен_мастер;
- обработка входящих смс;
- формирование и отправка исходящих смс.

Модуль на Arduino посредством платы Arduino UNO осуществляет трансляцию АТ-команд GPRS-модулю.

GPRS-модуль (SIM900) с активированной SIM-картой выполняет отправку и получение смс, взаимодействуя с платой Arduino UNO посредством АТ-команд.

Мобильное приложение Airmore и взаимодействующий с ним модуль могут рассматриваться в качестве замены связки Arduino+GPRS-модуль.

Заключение

В ходе исследования были рассмотрены и изучены варианты роботизации процесса обработки обращений граждан в кол-центры и на «горячие линии» по вопросам оказания услуг и помощи. Ускорение обработки обращений граждан и снижение нагрузки на диспетчеров могут достигаться путём обработки языковой информации.

Среди потенциальных пользователей системы: медицинские организации и учреждения; службы психологической помощи; центры социальной защиты населения; ветеринарные организации; волонтёрские организации; управления жилищно-коммунального хозяйства; ремонтные подразделения; службы дорожной помощи; страховые организации; службы юридической помощи; службы доставки продуктов, лекарств и т. д.; «горячие» линии и кол-центры других профилей.

Поэтому в общем случае задача более ёмкая. В частности, из речевой информации нужно выбирать не только профиль специалиста, но и виды работ или помощи, в которых нуждается заявитель, при этом обращение может не сводиться к какому-то одному, а предполагать несколько разнопрофильных специалистов. Расширение модели задачи и продуцируемой ею фактологической базы может обеспечиваться без изменения реализованных решений благодаря онтологическому подходу, заложенному в основу системы.

Список литературы

1. Гончаров А. С. Роботизация обработки обращений граждан по вопросам социального и бытового обслуживания / А. С. Гончаров, М. А. Гончарова // Наука без границ. – 2021. – № 6(58). – С. 40-52 – Электронный журнал – Режим доступа: https://elibrary.ru/item.asp?id=46277024
2. Speech Recognition Through the Decades: How We Ended Up With Siri [Электронный ресурс]: PCWorld: сайт [Melanie Pinola, дата публикации 02.11.2011]. – Режим доступа: https://www.pcworld.com/article/243060/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html
3. Как голосовые боты меняют клиентский сервис [Электронный ресурс]: Cossa: сайт [Валерия Минкевич, Voximplant, дата публикации 01.10.2021]. – Режим доступа: https://www.cossa.ru/trends/294366
4. Кейс «Метрологии». Как заменить диспетчеров на бота и обрабатывать в 5 раз больше заявок. [Электронный ресурс]: aimylogic.com: сайт. – Режим доступа: https://aimylogic.com/ru/cases/smart-ivr-metrologiya?ga=2.146378416.540714361.1630680941-1987544937.1630680941
5. Горшков С. Введение в онтологическое моделирование. – Тринидата, 2016. – 166 с. [Электронный ресурс]: trinidata.ru: сайт. – Режим доступа: https://trinidata.ru/files/SemanticIntro.pdf
6. Gruber T. R. Ontolingua: A mechanism to support portable ontologies. / Technical Report KSL 91-66. – Stanford University, Knowledge Systems Laboratory. 1992. – 61 p. [Электронный ресурс]: Электронная библиотека и поисковая машина по научным публикациям и препринтам CiteSeerX. – Режим доступа: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.9819&rep=rep1&type=pdf
7. Мазуренко И. Л. Компьютерные системы распознавания речи // Интеллектуальные системы. – Москва, 1998. – Т.3. Вып. 1-2. – С. 117-134.
8. Yargy-парсер и библиотека Natasha. Извлечения структурированной информации из текстов на русском языке [Электронный ресурс]: Хабр: сайт [блог Кукушкина А., дата публикации 14.03.2018]. – Режим доступа: https://habr.com/ru/post/349864
9. Проект Natasha. Набор качественных открытых инструментов для обработки естественного русского языка (NLP) [Электронный ресурс]: Хабр: сайт [блог Кукушкина А., дата публикации 24.08.2020]. – Режим доступа: https://habr.com/ru/post/516098

10. Yargy парсер. Извлечение структурированной информации из текстов на русском языке [Электронный ресурс]: Youtube: видеохостинг [канал Кукушкина А., дата публикации 31.10.2018]. – Режим доступа: https://youtu.be/NQxzx0qYgK8

11. Извлечение фактографической информации о пандемии из открытых источников сети Интернет / Е. Ю. Акулинина [и др.] // Математическая биология и биоинформатика. Информационные и вычислительные технологии в биологии и медицине. – 2022. – Т. 17. № 2. – С. 423–440. – doi: 10.17537/2022.17.423 – Электронный журнал – Режим доступа: https://www.matbio.org/article_ref.php?id=514

12. SMS [Электронный ресурс]: Wikipedia: свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/SMS

13. Отправка SMS-сообщений в формате PDU, теория с примерами на C#, часть 1 [Электронный ресурс]: hardisoft.ru: сайт [блог StarXXX, дата публикации 15.10.2010]. – Режим доступа: http://hardisoft.ru/soft/samodelkin-soft/otpravka-sms-soobshhenij-v-formate-pdu-teoriya-s-primerami-na-c-chast-1