OSTIS-2016
(Open Semantic Technologies for Intelligent Systems)

УДК 621.391

МИКРОСЕРВИСНЫЙ ПОДХОД К ПРОЕКТИРОВАНИЮ РЕЧЕВЫХ ИНТЕРФЕЙСОВ

Житко В.А.[*], Лобанов Б.М.[**]

[*]*Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь*

zhitko.vladimir@gmail.com

[**]*Объединённый институт проблем информатики НАН Беларуси, Минск*

lobanov@newman.bas-net.by

В статье описывается модель построения прикладных программных систем с речевым интерфейсом, основанная на применении сервис-ориентированной архитектуры и микросервисов. Такой подход обеспечивает минимизацию расходов на интеграцию различных по реализации и подходам компонентов в единую прикладную программную систему.

Ключевые слова: речевой интерфейс; сервис-ориентированная архитектура, микросервисы.

Введение

В связи с бурным ростом рынка мобильных и встроенных приложений и систем, разработкам в сфере упрощения ввода текстового запроса информации уделяется большое внимание. Наиболее перспективными в этой ситуации являются разработки, связанные с распознаванием речи, – современная альтернатива ручному вводу с клавиатуры. Кроме того, голосовой ввод команд и вопросов даёт возможность использовать систему в качестве источника справочной информации одновременно с основной деятельностью (получение справки при выполнении технических работ, управлении автомобилем и т.д.).

Использование одновременно с распознаванием речевого синтеза позволяет снизить нагрузку на зрительную систему пользователя путём восприятия результатов работы системы не через графический интерфейс, а посредством голоса.

Однако в этом направлении существуем масса различных проблем:

- большая вычислительная сложность и огромный объем баз данных, необходимых для корректного решения задачи распознавания речи;
- большое разнообразие используемых мобильных устройств и интернет-платформ.

Одним из возможных решений преодоления указанных трудностей является перенос на удалённый сервер систем распознавания и синтеза речи. В связи с этим наиболее перспективным выглядит применение активно развивающихся в настоящее время облачных интернет-технологий, позволяющих легко наращивать по мере необходимости производительность и объем используемых баз данных.

Целью данной работы является разработка моделей и средств построения естественно-языковых и речевых интерфейсов к прикладным системам, путём интеграции различных существующих компонентов в единую систему на основе применения микросервисов. В качестве основной модели была принята сервис-ориентированная архитектура программной системы [Barry, D. 2003] с применением подхода использования микросервисов [Chris, R. 2014] [Sam, N. 2015]. Использование микросервисов при проектировании позволяет без дополнительных затрат использовать компоненты построенные на базе облачных технологий. На сегодняшний день существует ряд справочных систем обладающих речевым интерфейсом (Siri, Voice Actions и др.), однако все они являются закрытыми коммерческими проектами и использовать данные наработки в других проектах не представляется возможным.

1. Сервис-ориентированная архитектура

В качестве основного подхода к проектированию приложений с естественно-языковым интерфейсом была выбрана сервис-ориентированная архитектура. Такой выбор обусловлен необходимостью

интеграции большого количества компонентов, участвующих в работе естественно-языкового интерфейса. На сегодняшний день существует множество разработок в сфере естественно-языкового общения, будь то синтез речи, распознавание, обработки и пр. Это позволяет построить из представленных компонентов системы, обладающие естественно-языковым интерфейсом, а выбранный подход обеспечивает лёгкую интеграцию компонент, построенных на отличающихся принципах за счёт слабой связанности самих компонент. Это позволяет также беспрепятственно менять компоненты на аналогичные, не перестраивая всего приложения.

Сервис-ориентированная архитектура (SOA, англ. service-oriented architecture) — модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных (англ. loose coupling) заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам.

2. Использование микросервисов

В качестве подхода при декомпозиции логики приложения на сервисы был выбран подход микросервисов.

Микросервисы — это архитектурный стиль, при котором сложное приложение разбивается на мелкие, независимые процессы, взаимодействующие через методы API, независимых от языка.

Архитектура приложения построенного на микросервисах обладает рядом свойств. Вот некоторые из них:

- разбиение на компоненты через сервисы;
- сервисы определяются бизнес-задачами;
- логика реализована в методах сервиса, а для передачи сообщений используются простейшие каналы передачи.

Каждый микросервис решает задачи, связанные с единственным или несколькими неразделимыми объектами домена (The Single Responsibility Principle [DeMarco, 1979], [PageJones, 1988]). При этом любой сервис, по своей природе, открыт для расширения, но его трудно модифицировать, особенно не имея доступа к исходному коду (The Open Closed Principle [Bertrand Meyer, 1988]). В силу того, что сервисы взаимодействуют между собой по контракту, то имеется возможность заменять одну реализацию сервиса, на другую (The Liskov Substitution Principle [Barbara, L. 1988]), если только он соответствует контракту. При проектировании микросервисов, также как и в объектно-ориентированном программировании, имеет смысл по возможности разделять сервисы на более мелкие, тем самым реализуя принцип разделения интерфейсов (Interface segregation principle [Martin, Robert 2002]). Ну и безусловно, используемые микросервисы не должны зависеть от других используемых компонент (The Dependency Inversion Principle [Robert C. 1996]).

Использование микросервисов обладает следующими преимуществами:

- масштабируемость - микросервисную архитектуру очень легко масштабировать;
- отказоустойчивость - надёжность достигается за счёт запуска дополнительных процессов;
- обновляемость - малая связанность сервисов и их относительная простота делает процесс тестирования и обновления технологий довольно простой задачей.

Основная трудность при использовании архитектуры микросервисов — разбиение бизнес-логики на независимые сервисы и организация взаимодействия между ними.

2.1. Распознавание речи

Для распознавания речевого сигнала в системе используется компонент, в основе которого лежит сервис распознавания речи, разработанный компанией Google.

На сегодняшний день компания Google является лидером по предоставлению облачных технологий распознавания речи [Manjoo F., 2011]. В течение последних пяти лет активно развивалась облачная технология распознавания речи Google Voice, и к настоящему времени существуют технологии распознавания речи для большинства европейских языков, включая русский, а также японский и китайский языки. Одним из немаловажных компонентов системы распознавания речи Google Voice является обучающая выборка звукозаписей человеческого голоса. Для системы Google Voice источником таких записей являются различные сервисы, предоставляемые Google, использующие речевые технологии, к ним относятся система распознавания речи и команд в системе Android, сервис диктовки писем Google Mail, телефонная справочная система Goog411 и др. [Singhal, A., 2011]. Таким образом, обучающая выборка постоянно пополняется новыми образцами голосов с их особенностями, как произношения, так и эффектами, вносимыми техническими особенностями записи и передачи голоса на различных устройствах. К примеру, в 2011 г. обучающая выборка для английского языка составляла примерно 230 млрд записей слов извлеченных из реальных запросов [Enge, E. 2011]. Для обработки таких объемов информации требуется около 70 лет процессорного времени, однако с использованием облачной технологии Google время сокращается до одного дня [Singhal, A., 2011]..

Используемая модель распознавания включает в себя три части: акустическую, лексическую и языковую модели [Enge, E. 2011]. Акустическая

модель ответственна за распознавание фонем, она учитывает все возможные варианты произношения, а также другие особенности, такие как тип используемого микрофона (качество записи), фоновые шумы, возраст и пол говорящего и многое другое. Наиболее важным в данном случае является объем обучающей выборки, чем он больше, тем лучше будет результат распознавания. В лексической модели фонемы объединяются в слова на основе словарей, в которых указаны различные варианты произношения слов. Языковая модель объединяет слова, используя статистический подход. На основе анализа поисковых запросов, текстов Интернета выделяются вероятности взаимного расположения слов в предложении.

В составе разработанной прикладной программы в процессе распознавания речи задействованы три микросервиса:

- сервис записи и детектор речи;
- сервис сжатия и обработки речевого сигнала;
- сервис по работе с распознаванием речи Google;

Детектор речи является необходимым дополнительным компонентом, обусловленным используемой архитектурой удаленного распознавания речи. В этом случае канал передачи данных (в нашем случае Интернет-соединение) является «бутылочным горлышком», ограничивающим максимально возможный объем передачи данных в заданный промежуток времени. При передаче по каналу слишком длинного отрезка речевого сигнала происходит недопустимо длительная задержка ответной реакции системы распознавания. Кроме того, при этом возникает значительный риск получения ошибочных результатов распознавания. Ввод коротких отрезков речи возможен при использовании ручного стартстопного режима, предоставляемого системой Google Voice. Однако для решения поставленной здесь задачи стенографирования устной речи этот режим оказывается крайне неудобным. Обычно диктовка осуществляется короткими фразами, заканчивающимися паузами. После произнесения каждой из них пользователь, как правило, желает убедиться в правильности распознавания и при необходимости повторить её более чётко.

Задача детектора речи заключается в автоматической локализации полезного сигнала, т. е. в определении начала и конца произнесённой фразы. Это обеспечивает автоматический пофразовый ввод речи. В экспериментальной программной системе использована библиотека SPTK для реализации механизма определения начала и конца фразы. Для этого производиться запись небольшого отрезка сигнала в кольцевой буфер, далее этот отрезок проверяется на наличие частоты основного тона, и если таковая находится, то начинается запись сигнала в файл. Аналогичным образом определяется конец фразы.

Полученный полезный сигнал в модуле сжатия и обработки сигнала подготавливается для отправки на удаленный сервер распознавания речи Google Voice. Для этого сигнал кодируется в открытом формате FLAC (Free Lossless Audio Codec). При этом происходит сжатие сигнала кодеком, что уменьшает объем передаваемых данных и, как следствие, сокращает время ожидания результата распознавания. Использование open-source-кодека имеет и другие преимущества: простота использования сторонними разработчиками, единообразие получаемого сервером формата данных. Каждый компонент программы работает асинхронно и имеет свои пулы данных для нивелирования эффектов, связанных с задержками в работе каждого компонента. Такие задержки появляются как на этапе записи и кодирования речевого сигнала в файл, так и отправки его на удаленный сервер.

3. Экспериментальная прикладная программа

Для оценки предлагаемой модели был разработан программный комплекс, объединивший в себе ряд режимов (приложений) демонстрирующих различные аспекты использования речевых интерфейсов.

В качестве входных требований к проекту были взяты следующие положения:

- программное средство реализуется в виде приложения под ОС Windows, Linux;
- работа с программным средством должна вестись не только в ручном стартстопном режиме (как это предусмотрено системой Google Voice), но и в автоматическом режиме путём дополнительного включения в его состав детектора речи, определяющего начало и конец голосового сигнала;
- программное средство должно обладать простым и понятным графическим интерфейсом;
- программное средство должно характеризоваться минимальными требованиями по установке и настройке.

В связи с этим были выбраны следующие средства реализации:

- язык программирования C++ обеспечивает максимальную производительность при захвате и обработке речевого сигнала;
- библиотека построения пользовательских интерфейсов Qt Quick позволяет в кратчайшие сроки разрабатывать кроссплатформенные пользовательские интерфейсы;
- библиотека OpenAL для реализации однотипной работы с аудио устройствами под различными платформами.

В экспериментальной прикладной системе выделены следующие микросервисы:

- сервис записи и детектор речи;

- сервис работы с Google Voice;
- сервис работы с Google Translate;
- сервис работы с синтезаторами речи (Windows SpeechAPI, CMU Sphinx);
- сервис загрузки и обработки графического пользовательского интерфейса;
- сервис динамической загрузки логики работы режимов;
- и др.

В приложении присутствуют следующие режимы:

- телеграф
- сказка
- переводчик
- калькулятор

Согласно микросервисному подходу каждый логический элемент системы должен представлять собою независимую сущность, общающуюся с системой посредством асинхронных сообщений или событий, в реализуемом приложении это набор сервисов, перечисленный ранее. Режимы в приложении представляют собою компоненты на стороне клиента, использующие функциональность сервисов. Режимы реализованы в виде отдельных модулей (написанных на QML) и подгружаемых динамически после старта и инициализации основного приложения. Таким образом, перечень и логика работы режимов может быть изменена без необходимости повторной компиляции или сборки приложения. Это возможно благодаря динамической загрузке модулей режимов и интерпретации языка QML и JavaScript используемых для описания модулей.

При таком подходе приложение можно логически строго разделить на три основных слоя:

- слой поддержки микросервисной архитектуры (инфраструктура микросервисов);
- слой микросервисов;
- слой клиентских приложений (в рамках реализуемого приложения это режимы).

Ниже приведена общая схема приложения с реализацией микросервисной архитектуры.

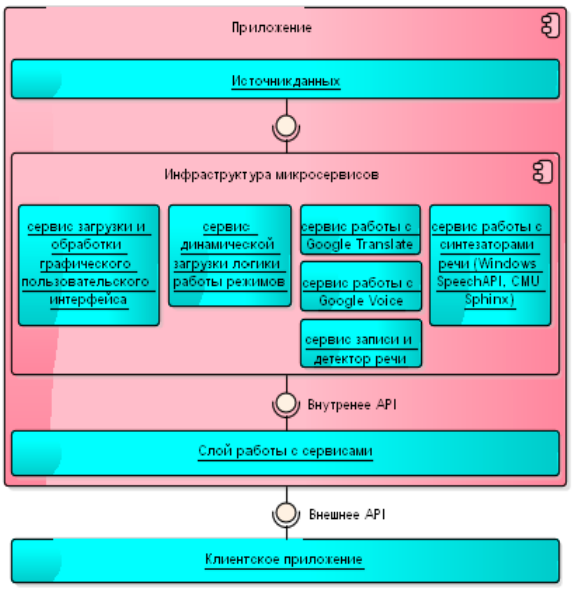

Рисунок 1 – Общая схема структуры приложения

Ниже представлен пользовательский интерфейс приложения:

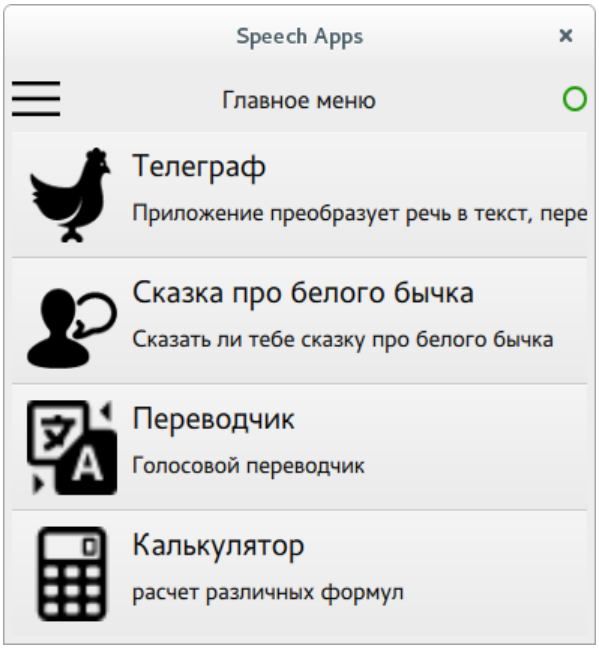

Рисунок 2 – Основное меню приложения

В приложении реализованы базовые настройки аудио входа и выхода, языка интерфейса и языка пользователя:

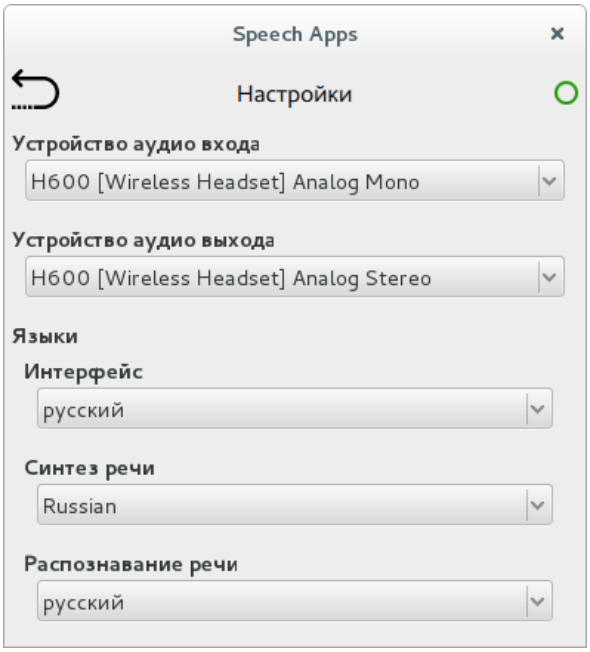

Рисунок 3 – Базовые настройки приложения

В режиме «Телеграф» приложение будет повторять любые сказанные пользователем фразы. При этом происходит их автоматическая запись, распознавание и отправка распознанного текста на вход синтезатора речи:

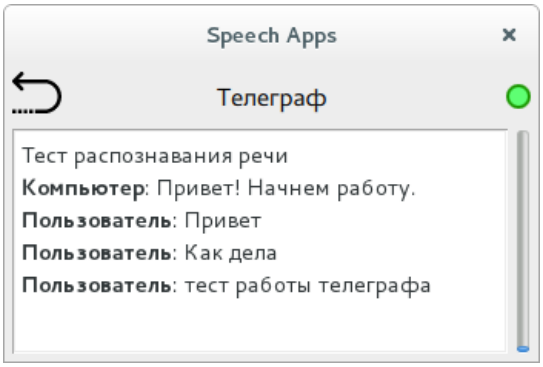

Рисунок 4 – Режим «Телеграф»

В режиме «Сказка» приложение демонстрирует элементарный диалог в стиле докучной сказки:

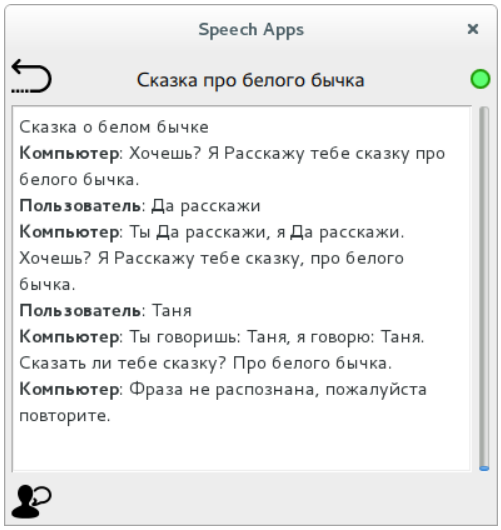

Рисунок 5 – Режим «Сказка»

В режиме «Переводчик» приложение отправляет распознанный текст на вход переводчика, а результат выводит в форме синтезированного текста:

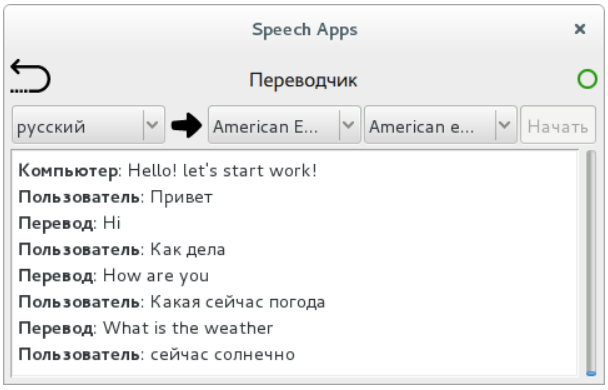

Рисунок 6 – Режим «Переводчик»

В режиме «Калькулятор» реализован ввод математических формул и дальнейший их расчёт с выводом результата в виде синтезированного текста. В приложении реализованы базовые математические операции:

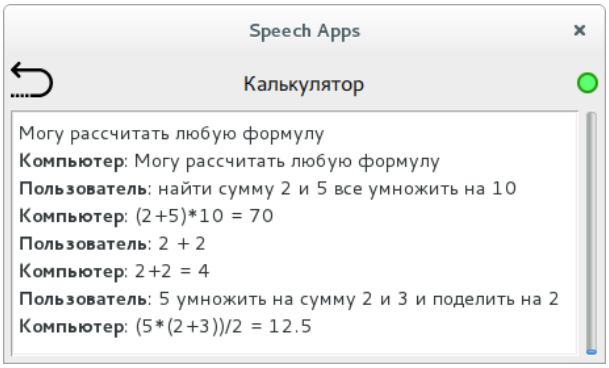

Рисунок 7 – Режим «Калькулятор»

Заключение

Представленная модель речевого интерфейса с разбиением на отдельные компоненты со строго заданным функционалом, позволяет упростить разработку, а в дальнейшем – сопровождение, различных систем с речевым интерфейсом.

Компоненты системы распознавания и синтеза речи по тексту предоставляют конечному пользователю возможность устно задавать вопрос и слышать ответ на него от системы, а не просто вводить вопрос через клавиатуру и читать ответ с экрана компьютера. Это делает естественно-языковой интерфейс еще более привлекательным для пользователя.

Разбиение системы на отдельные независимые компоненты дает возможность интеграции сторонних разработок и проектов, производить интеграцию различных подходов и методов в рамках одного проекта, что позволяет эффективно использовать их лучшие стороны.

Библиографический список

[Barry, D. 2003] D. K. Barry, Web Services and Service-Oriented Architectures: The Savvy Manager's Guide: Morgan Kaufmann, 2003.

[Chris, R. 2014] Chris Richardson. Pattern: Microservices architecture. http://microservices.io/patterns/microservices.html, 2014

[Sam, N. 2015] Sam Newman. Building Microservices. O'Reilly Media, 2015

[Public Cloud Service Definition, 2010] Public Cloud Service Definition. Public Version 1.5 // WMWare, Inc. [Electronic resource]. – 2010. Mode of access : http://www.vmware.com/files/pdf/VMware-Public-Cloud-Service-Definition.pdf.

[Mell, P. 2011] Mell, P. The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology / P. Mell, T. Grance // U.S. Department of Commerce [Electronic resource]. – NIST Special Publication, 2011.

[DeMarco, 1979] Structured Analysis and System Specification, Tom DeMarco, Yourdon Press Computing Series, 1979

[PageJones, 1988] The Practical Guide to Structured Systems Design, 2d. ed., Meilir PageJones, Yourdon Press Computing Series, 1988

[Bertrand Meyer, 1988] Object Oriented Software Construction, Bertrand Meyer, Prentice Hall, 1988, p 23

[Barbara, L. 1988] Barbara Liskov, "Data Abstraction and Hierarchy," SIGPLAN Notices, 23,5 (May, 1988).

[Martin, Robert 2002] Martin, Robert (2002). Agile Software Development: Principles, Patterns and Practices. Pearson Education.

[Robert C. 1996] The Dependency Inversion Principle, Robert C. Martin, C++ Report, May 1996

[Manjoo F., 2011] Manjoo, F. Now You're Talking / Farhad Manjoo // The Slate Group. – Washington Post Company, 2012. – [Electronic resource] – Mode of access : http://www.slate.com/articles/technology/technology/2011/04/ now_youre_talking.single.html. – Date of access : 01.08.2012.

[Singhal, A., 2011] Singhal, A. Knocking down barriers to knowledge / Amit Singhal // Google Official Blog [Elec-tronic resource]. – 2011. – Mode of access : http://googleblog.blogspot.com/2011/06/knocking-down-barriers-to-knowledge.html. – Date of access : 01.08.2012.

[Enge, E. 2011] Enge, E. Search Algorithms with Google Director of Research Peter Norvig / E. Enge // Stone Temple Consulting [Electronic resource]. – 2011. – Mode of access : http://www.stonetemple.com/search-algorithms-with-google-director-of-research-peter-norvig. – Date of access : 01.08.2012.

MICROSERVICE DESIGN APPROACH FOR DEVELOPING SPEECH USER INTERFACES

B.M. Lobanov[*], V.A. Zhitko[**]

[*]*United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, Republic of Belarus*

lobanov@newman.bas-net.by

[**]*Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus*

zhitko.vladimir@gmail.com

In work describes design approaches to make system with speech user interface. Main flow is to use service oriented architecture and micro services business logic decompositions. Also describes components for text-to-speech and voice recognition. Primary goal of this work is making easy building applications with speech user interfaces.

Main Part

Microservices is modern approach to build scalable flexible applications and computer services. This allow to build application as ecosystem with independent small services, called microservices. Microservice is application component with weak relations and performs only one business function.

There are three main layout in such applications. They are: microservice infrastructure, list of microservices and client applications. All this layout communicate with each other via unified protocol. It could be REST api, json api, xml api or something else.

Describe experimental system with a number of examples of user speech interfaces. It include text-to-speech, speech-to-text transformations.

Additional third-party component is natural Russian voice input. For this component used Google voice recognition service with made some changes. So component can recognize non-stop user speech. Also used as third-party component Google translate service.

Another important third-party component is voice output. This component is top required for making natural language interface friendlier for users. Also, as previous component, it built in a traditional technology. But some tasks are hard to solve in traditional ways, for example correct detect the stress in words, sometimes for this need understand meaning of text content, and this could be solve by semantic approaches.

Conclusion

In the given paper short description of model and methods for designing and prototyping natural language interfaces with voice input and output.