“The Arius” project

prepared by
Serhii Dubovyk

Problem statement

The goal is to build a question-answering system for gathering additional information about SoftServe and its projects in company`s showrooms.

The system named “Arius” should provide an invisible interface to its users. It should be able to get voice input in natural English language and give answers as a short voice message or as a web page, video or presentation displayed on the screen. In the case of showing a response on the screen, the system should give a short voice message to the user, which is related to the question topic.

According to the location of the system, it should work when there are multiple conversations in the microphones range.

Also, it is important to be obvious for users, that the system is online and ready to answer their questions. So, Arius should indicate its state and attracts visitors attention.

Scope

The Arius system user interface will include a microphone for user voice input, speakers for the brief or ‘placeholder’ answers and two wireless screens for showing requested data as a web page, video or presentation.

When a group of visitors come into the showroom and have some more questions, they should easily notice the Arius, so when in waiting mode there should be some animation, which will show, that the system is online. It should not be anthropomorphic as it can be scary and simply too strange for the users.

We assume that there will be about 6-7 people in the showroom simultaneously, so the system should be able to detect when there is a regular conversation between visitors and when it is a request for the Arius. To do it, we will use a keyphrase at the beginning of each request: “OK, Arius”. The probability of using this phrase in typical conversation is quite low so that it can be used as a trigger of a request.

The requests, which the Arius gets should be related to the SoftServe and its projects or some basic real-world questions.

After getting a request, the system should give some voice information, which will confirm receiving of the request and give some interaction to the user. While that, the system provides search for the data about the requested topic in the local storages or SoftServe`s web-site and when the best match is found, it is displayed on the screen. While searching for the data, the “search screen” will be shown to the user. It`ll contain some randomly-chosen tips how to use Arius. It will allow to do some kind of interaction with the user while he is waiting for a response. If the data is not found, the system should show a message about this to the user or tell it with a voice, and then wait to one of two possible commands: search for the data on the internet or cancel the request and get to the waiting mode.

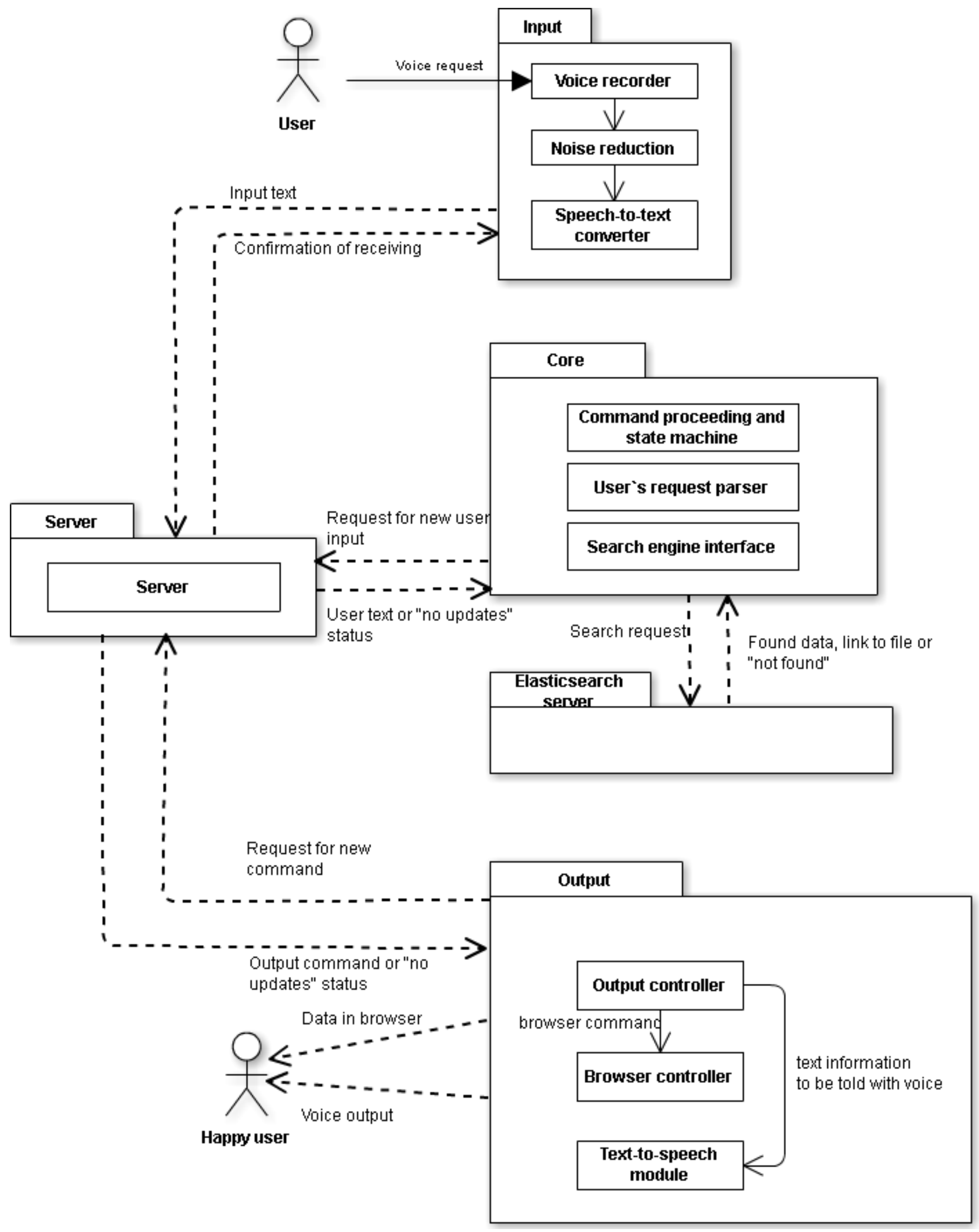
Files with the data we search in can be stored on local machine or a remote server. As a search server, we will use the Elasticsearch, which is an open-source project.

When there is data on the screen, the system changes its behavior. In this state it can process only commands related to the data representation on the screen, for example: “Scroll down,” “Zoom In/Out,” “Zoom” and an escape command, after which Arius gets back to the waiting mode.

Also, it is important that data for user`s request will be shown only on about ½ of screen`s height(about 50-60 cm) – it will make consuming information more comfortable as the user won`t be required to bent down to see bottom lines on the page. As there is a lot of free space on the screen, it is possible to use that space for showing information about SoftServe`s top projects which are recognized as most successful but wasn`t requested by the visitor.

Architecture

On the following diagram you can see the general structure of the Arius system.



The architecture is based on a server module which is used for storing input data or commands between modules and three “worker” modules which communicate with the server via REST API: input, core (controller) and an output module. Data transfer between modules and server is asynchronous, so each command can be interrupted during its execution in case of receiving a new command or data.

INPUT MODULE

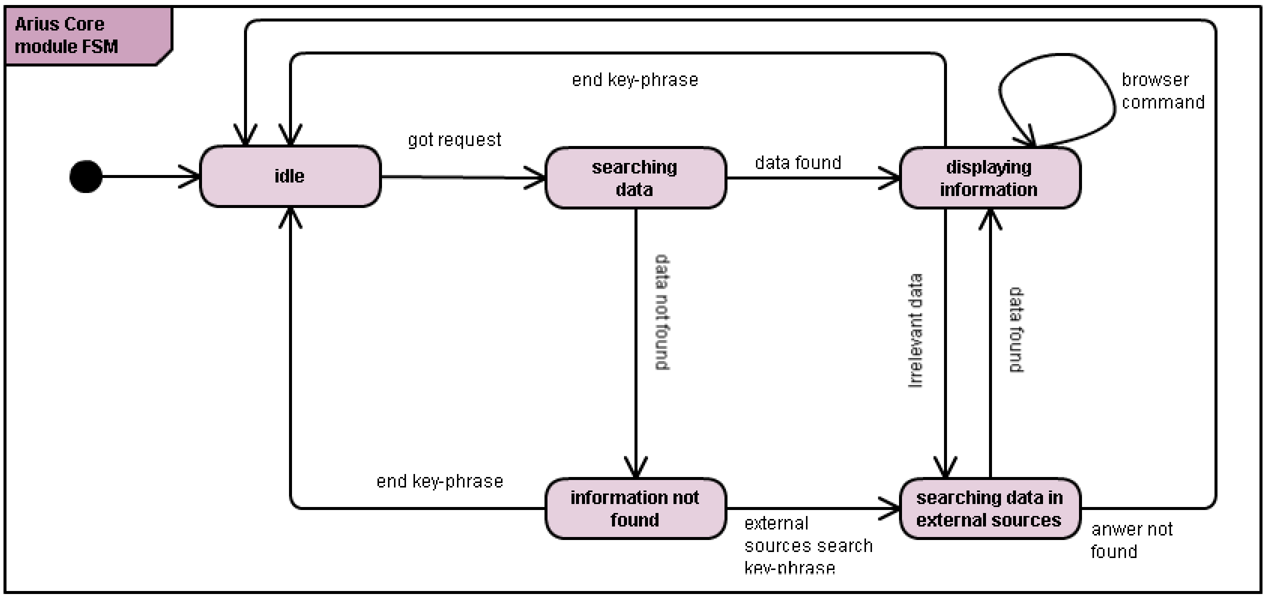
This module is responsible for handling all voice data which can be gathered with an input device and converting it to text. It can be implemented as an application which gets input from system sound input device and then uses Nervana Neon to reduce noise and get text input.

When there is one of the key-phrases recognized we record the input voice and convert it to text. Then, after receiving the end key-phrase or long(>5 sec) silence input module sends a POST-request to the server module with the user input. If the message was delivered it gets HTTP 200 code in response, if the message wasn`t delivered it`s sent again.

CORE MODULE

This module is a main part of the Arius system. It sends requests to server module with given intervals and as soon as new data from input module is available it gets that data.

Below there is a state machine which describes the behavior of the core module in dependence of user`s actions.



When in **"idle"** mode core module gets a new user command it provides sense extraction from the input and uses received key-phrases to get relevant documents via Elasticsearch. While searching information, core module gets to "searching data" state. The search process is run in a separate thread, so it can be aborted in case of receiving such command (e.g. "Stop", "Cancel"). In case of receiving abort command, core module gets back to "idle" state.

If some relevant data was found, "core" module sends a link to the found file to the server in a format of a command to open a file (e.g. {"type": "OPEN_PDF", "body": "data.pdf"}).

In case when there is **no relevant data**, core module sends to server a command to open a "not found" message, gets to "information not found" state and then waits for user input:

1) "Thanks, Arius" (or other end key-phrases) to end the operation and get to "idle" mode
2) "Ok, Arius" (or other start key-phrases) to get a new request

3) or “Search more, Arius” (or other external search commands) to provide search in external sources, like the internet search engines(e.g. Google).

If 3rd option was chosen, the system sends a request to an external search engine and the **first link is used as a response data source**. And the main module behaves as the data was found in the first iteration — it sends a request with the data to the server module and changes FSM state to “displaying information”.

In the **“displaying information” mode**, Arius can handle two groups of commands: browser control commands and requests for other sources of data. When core module gets an input with commands like “scroll down” or “zoom in” (or any other forms of such commands) it sends to the server a command to operate with the browser. Also, system can get a request that the data is irrelevant. In that case, Arius repeats a request to the data module and uses the [n+1] result, where n — amount of irrelevant responses, if n exceeds some limit, set by the administrator of the system, Arius gets into the “idle” mode. So, in the “displaying information” module, it can handle only commands about the data representation, choosing next response or about finishing the current session.

OUTPUT MODULE

This module is responsible for the giving responses to the user. It should be run on the host, which is connected to the screens and the audio device.

This module is used for displaying data in a browser or giving a voice output as well. Output module has a BrowserController object wich uses Selenium WebDriver to operate with a browser and a VoiceOutput for giving a voice output. For voice output, Festival Text-to-Speech will be used.

Output module will have two threads:
1) Updater – for getting commands from a server with some intervals of time
2) Worker – checks if new command available and if so, handles it.

Such architecture allows to receive new commands in each moment of time, so the system can not be “frozen” with some long command (e.g. open large webpage).

While the Arius is in idle mode the output module should turn on some kind of screensaver to attract and involve people. Probably, it can be an image of multiple streams moving and changing in dependence of visitor`s behavior. This idle-mode part can be completely autonomous from another system as it should just capture noise and maybe user`s movement and react to it. It will be realized as a JavaScript program which will use OpenCV or just mics to detect people and react to them. One of the possible behaviors in the “idle” mode is showing random muted videos while the user isn`t detected and as Arius detects user it will print out an invitation to talk and start using it. Another opportunity is to use OpenCV and track users with some kind of “eyes”.

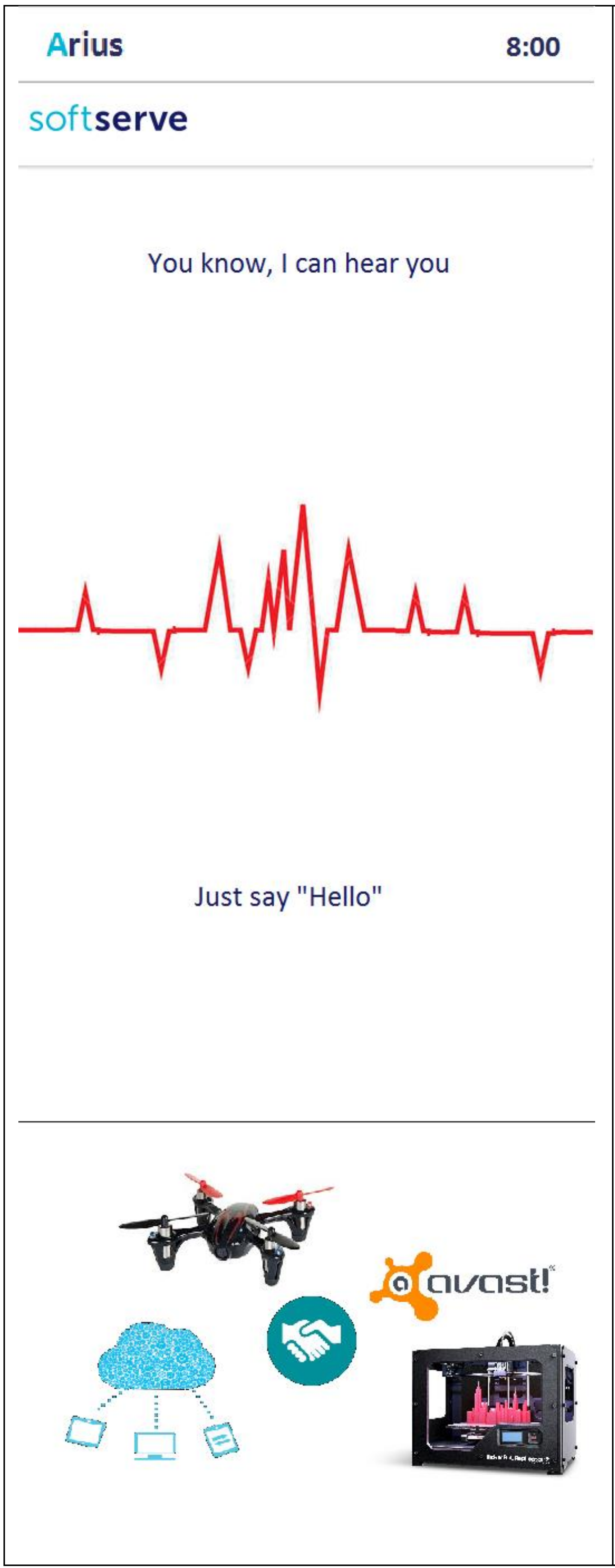
DATA MODULE

This module will be represented as an Elasticsearch server and a client to it in the form of Python class, which will be used in the core module. As core module gets a command to find some data the request will be sent to Elasticsearch server and as a response file names and percentage of correlation with the request will be given.

Search can be provided in HTML files of the local copy of SoftServe`s site as well as in the local pdf`s, presentations or text documents. All these types of files can be indexed directly by Elasticsearch. In case of using of videos, we can use solutions based on Toronto Deep Learning to get tags for videos or mark them manually, depending on the amount of data.

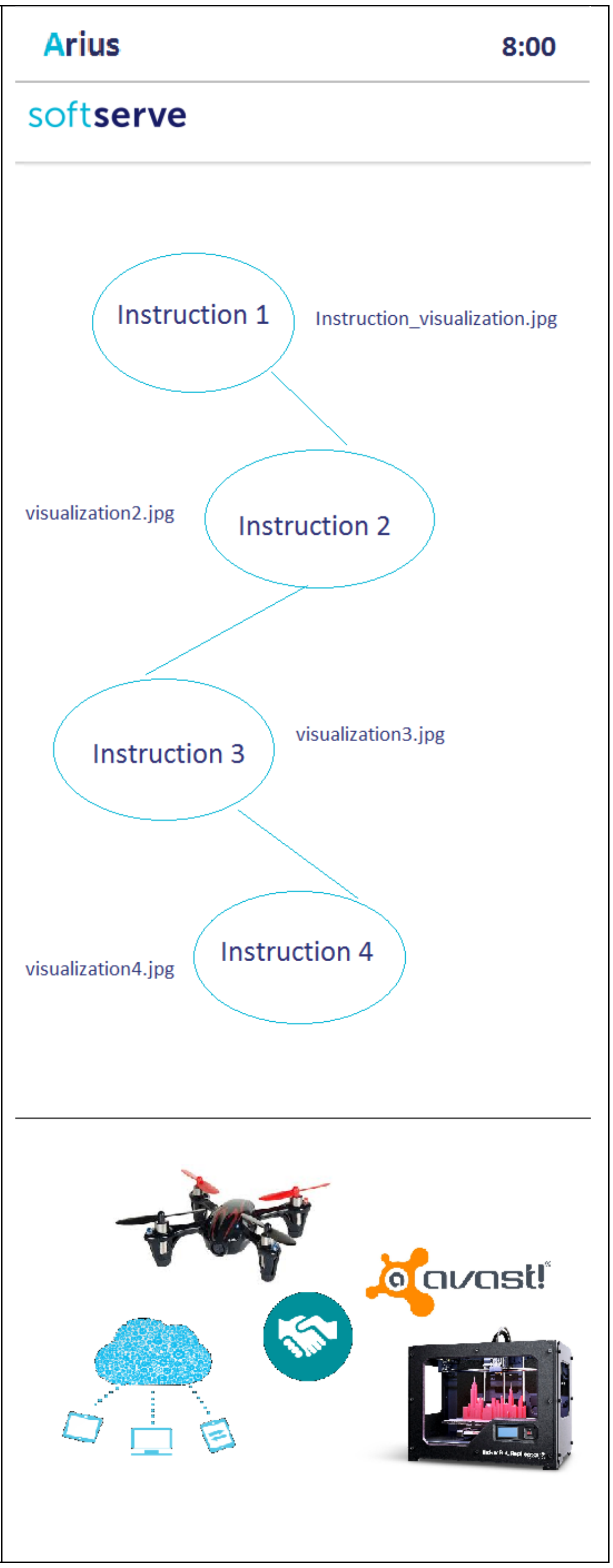
UI

In order to provide comfortable reading, we reserve 2/3 part of a display for the main content. The bottom part of a display will contain a grid of pictures of SoftServe's projects.

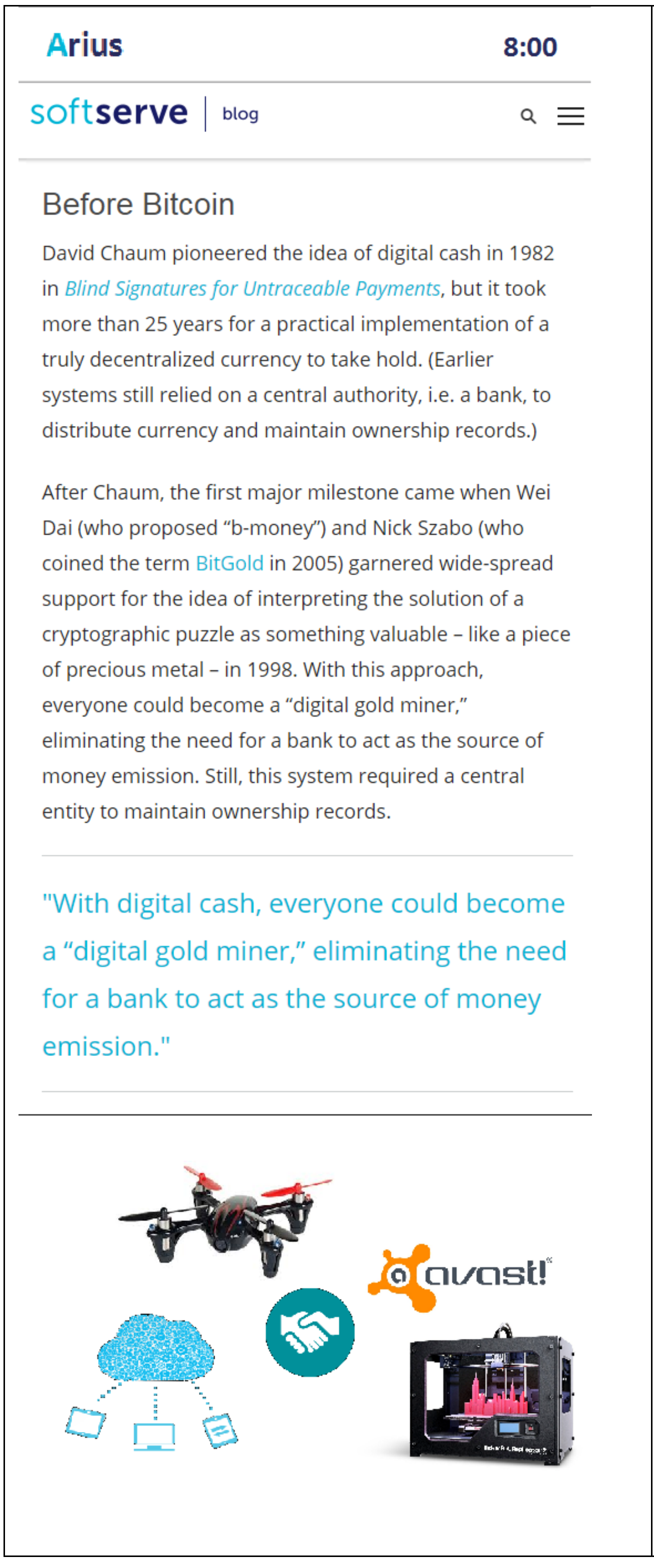


You know, I can hear you

Just say "Hello"



It will make the interface more interesting. Each picture will be smoothly replaced by other in a short period of time. To avoid distraction we will make our footer darker when we give some answer to the user. In our interface, we follow the SoftServe corporate style. It is a good solution. First of all, because Arius should represent SoftServe to customers. Another reason to follow corporate style is integrality of our interface.



Arius 8:00

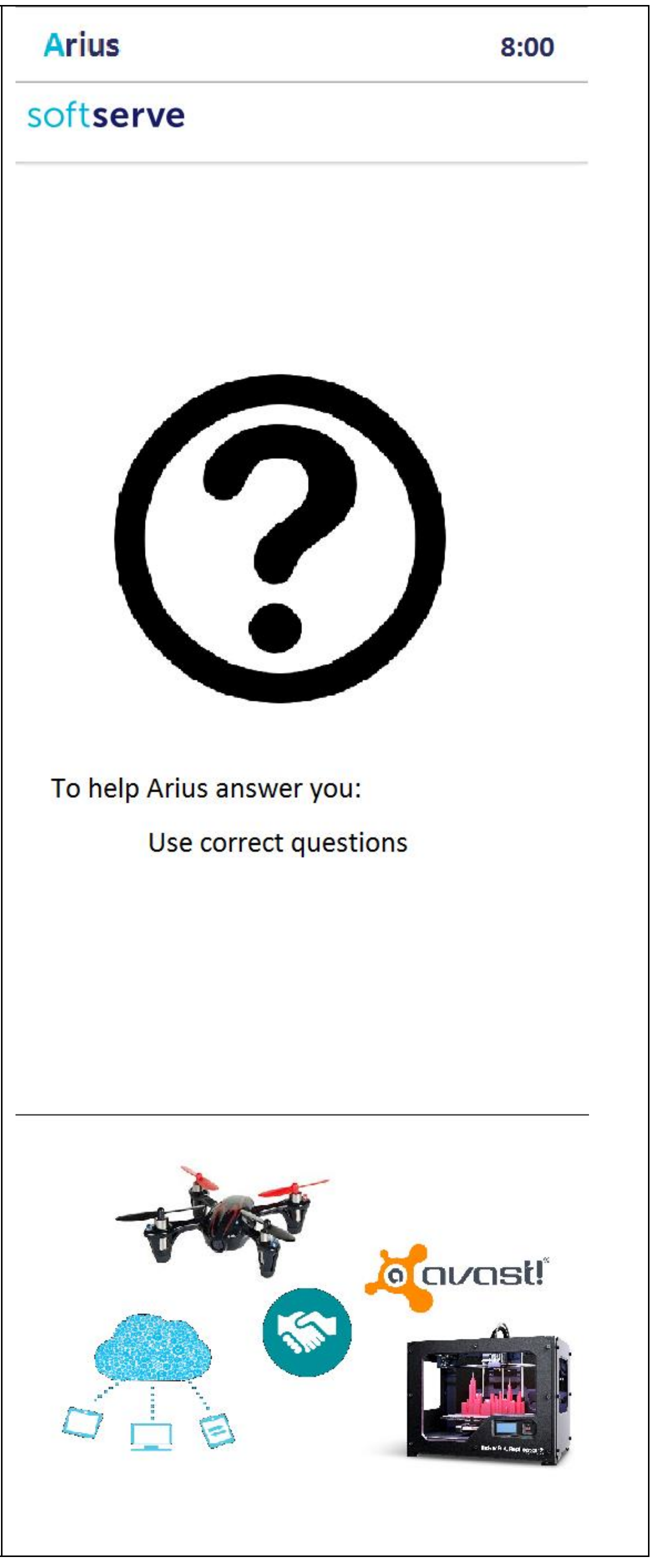
softserve | blog

Before Bitcoin

David Chaum pioneered the idea of digital cash in 1982 in *Blind Signatures for Untraceable Payments*, but it took more than 25 years for a practical implementation of a truly decentralized currency to take hold. (Earlier systems still relied on a central authority, i.e. a bank, to distribute currency and maintain ownership records.)

After Chaum, the first major milestone came when Wei Dai (who proposed “b-money”) and Nick Szabo (who coined the term BitGold in 2005) garnered wide-spread support for the idea of interpreting the solution of a cryptographic puzzle as something valuable – like a piece of precious metal – in 1998. With this approach, everyone could become a “digital gold miner,” eliminating the need for a bank to act as the source of money emission. Still, this system required a central entity to maintain ownership records.

"With digital cash, everyone could become a “digital gold miner,” eliminating the need for a bank to act as the source of money emission."



Arius 8:00

softserve

To help Arius answer you:

Use correct questions

Most of the resources which we use are in corporate style. The last reason is lightness and simplicity of design, which will help to attract user but not to overload him.

We also reserve a small part of a display in the top to leave there static elements, such as clock and name of this project-system.

It is a way to make some borders for the main content, what will help to catch an eye on it. It also provides some functionality, which will make the interface more interactive.

Main changes during using of Arius user notice in the main content area.

If Arius "notice" somebody in the room it reacts. Arius don't use voice to avoid interrupting people it there is some conversation, but show on the screen that it can hear somebody and it is ready for use.

After short "Hello", Arius welcome user and show him information how to use him. Those instructions are short and easy to understand. They are presented in few steps with using visualization.

User can ask Arius about anything. When system gets new message about searching information, it shows "loading" screen with short advice how to use it properly.

Roadmap

To deliver Arius in two months, we have the following plan. Here is just a vision of the process of development, the more detailed timetable you can find in “Effort Estimation” page.

The first thing to do is to define internal interaction protocols between the modules of the Arius.
The next steps are to:

1) build basic voice input system(without noise reduction, etc)
2) build basic output controller which will be capable of interacting with browsers via Selenium WebDriver
3) implement a skeleton of the core module, which will contain state machine and will be able to get user`s requests and send some data to the output module. These tasks can be done simultaneously, as there are three developers and the tasks do not cross. When completed this step, we can have some kind of Arius MVP.

Then, a method to fill in the Elasticsearch database should be developed, as well as Elasticsearch server should be installed and set up. For that time, parsing only text files will be enough as a first step.

After that, it is important to develop the core of data module. That means it should be able to get a query from the core module and send back a link to a most relevant file or a short text information. Also, it will be important to be able to analyze the query to form the database request.

These two tasks are dependent one on another, so while creating a parsing tool, it is possible to improve the quality of voice input and to add to the output module opportunity to choose, how to present the data: display it in the browser or ‘say’ it with Text-to-Speech system.

When previous stages are accomplished, the next goal is to complete the core module. It should be capable of behavior shown on the 6th page.

After this, Arius will be a working prototype. The next step will be to develop a tool to analyze and tag video files or to do it manually and to add this information into the database. Simultaneously, the tool for managing entries in the database will be developed.

Effort estimation

Schedule of the development		
Development step	**Sub-step**	**Required time(man-days)**
Pre-development	Internal protocols	3
	Internal architecture, classes structure	3
Pre-development (review)		3
Pre-development (documentation)		1
Pre-development(total)		9
Early development, super-MVP	Input (network sub-module)	1
	Input (basic voice input)	3
	Input (documentation)	1
	Output (network sub-module)	1
	Output (browser control)	2
	Output (documentation)	1
	Core (state machine)	2
	Core (network sub-module)	1
	Core (simple interaction)	2
	Core (documentation)	1
Early development, super-MVP (testing)		6
Early development, super-MVP (total)		21
Elasticsearch deployment		2

Elasticsearch filling tool development		3
Elasticsearch filling tool (documentation)		1
Data module development	Network sub-module	1
	Query parsing	3
	Elasticsearch interface	3
	Documentation	1
Data module and Elasticsearch development (testing)		6
Data module and Elasticsearch development (total)		20
Voice input improve (noise reduction)		6
Voice input improve (noise reduction) (testing)		3

Schedule of the development

Development step	**Sub-step**	**Required time(man-days)**
Output (tts system, presentation type selection)		2
Output (tts system, presentation type selection) (documentation)		1
Output (tts system, presentation type selection) (testing)		1
Input & output improvement (total)		8
Completing the core module		6
Completing the core module (testing)		3
Completing the core module (documentation)		1
Completing the core module (total)		9
Animation for idle mode design		6

Animation for idle mode design (testing)		1
Animation for idle mode design (total)		7
Documentation (total)		3
Compiling into package, basic refactoring		9
Global testing		9
Man-days left to 2 months		30

Executive Summary

Nowadays invisible interfaces are one of the fast developing directions of the UX and it seems that they will become more and more popular and wide-spread. So it is logical to use them in SoftServe`s marketing to declare that the company is always on the edge of technologies.

The goal of the Arius project is to create a system capable of giving answers to questions asked with natural English by voice. Answers should be in the form of web-pages, presentations or documents. In case of short answer, it can be given in a voice form. Also, when Arius receives a question it should react to it to show that he got it: show some animation and give short voice response (probably related to the question topic).

Arius will look like a set of screens, speakers and a microphone attached to one of the room`s walls. While in waiting mode it will display some interactive animations and muted videos in order to attract user`s attention.

When a group of visitors come into the showroom and have some additional question, they can easily activate Arius with “OK, Arius” (or any other form of showing their interest to the system) phrase, ask their question with natural language and get a short voice answer or detailed response in a form of the web page, video or presentation on the screens. The browser can be controlled by voice commands to change the data representation. Then, also by user`s voice command, Arius can be sent into the waiting mode.

The system performs a search on local resources. However, in the case of lack of required information on the local storage, it is possible to use external search engines (e.g. Google) to find the necessary data on the internet. Also, if the response given by Arius did not satisfy the user, it is quite easy to try to improve the answer by loading another web page or video relevant to the request.

The project will be developed with Python as a back-end language as it can be run on multiple systems and its high speed of development. Priority OS is Debian-based Linux distributives. To recognize user`s speech it`s possible to use Nervana`s Neon deep-learning framework or Kaldi toolkit. Modules of the system (input, output, core, data, server) will be interconnected REST API and Flask

framework as it is a simple and unified method which allows to run the system as on a single PC as on multiple. In order to present requested data to user, webpages with target content (e.g. slides, pdf`s) will be generated with output module and showed to user.

The system can be run on multiple hosts due to its modularity. Because of this, there is no need in a powerful machine for the input/output purposes, while main modules can be run on the server.

To accomplish this project, we need a team of 3 developers and two months. In that case, it will be possible to deliver a working Arius system. Also, in the case of using technologies listed above, there will be no license payments for the system.