1. Добрый день, господа. Представляю вашему вниманию систему дистанционного голосового управления пользовательской электроникой «Red Sia». Под пользовательской электроникой я подразумеваю всё, что мы в той или иной степени можем назвать компьютерами: ПК, планшеты, телефоны, плееры, and so on.
2. В мире уже существует большое количество самых разных систем голосового управления. На слуху у всех такие системы, как Siri, Cortana, Google Speech. В большей степени эти систем ориентированы на мобильные устройства. Компания LG выпускает стиральную машинку с голосовым управлением. Почти в каждый автомобиль сейчас встроена подобная система.
3. Однако у подобных продуктов существует ряд одинаковых недостатков: такие системы по большей части рассчитаны на работу по принципу «запрос-ответ», о диалоге речь не идёт. Эти системы не умеют строить ответы, ориентируясь на своё эмоциональное состояние, исходя из отношения к пользователю, а также брать во внимание предыдущие запросы. В них отсутствует понятие «разорванного мышления».
4. Red Sia – это совершенно новая точка восприятия подобных систем. Основной упор при реализации архитектуры был сделан именно на построение «живых диалогов» с пользователем. Также данная архитектура позволяет обрабатывать как моментальные, так и асинхронные команды, а также выбирать лексику и модель поведения исходя из своего личного отношения к пользователю.
5. Давайте пройдёмся по архитектуре, её модулям и способам взаимодействия между ними. Как только поступает новая команда, она сразу же попадает в модуль RSCore. Этот модуль является управляющим классом всей системы. Он содержит список модулей комплексной обработки голосовых команд. Список нужен для параллельной обработки нескольких голосовых команд. Для новой команды сразу же создаётся новый SE, выполняется, и, в зависимости от результата его работы запускаются остальные SE модули, но об этой логике позже. По окончанию работы SE, RSCore удаляет этот модуль из памяти.
6. Теперь давайте разберём исполняющий модуль, он же SE. Данный модуль хранит четыре списка: IPT, IPAT, IST, ISV. Теперь уместно рассказать о принципе выполнения команды. Обработка и выполнение проходят в два этапа: распознавание и выполнение. Во время распознавания генерируется скрипт на основе введённых пользователем голосовых данных. Во время выполнения этот скрипт асинхронно выполняется. После выполнения каждого этапа SE уведомляет RSCore. Во время первого этапа работают модули IPT и IPAT. Во время второго – IST и ISV.
7. IPT Он же Interface Phrase Translator. Вызывается, когда SE ещё не получил скрипт, то есть в самом начале, и отвечает за генерацию скрипта исходя из данных.
8. IPAT, Interface Phrase Additional Translator – вызывается, когда SE ожидает ввода дополнительных данных. Выполняет преобразования данных, однако скрипт не генерирует.
9. IST, Interface Sia Translator – выполняет полученный скрипт, а все данные, которые нужны ему по ходу содержит
10. ISV – Interface Sia Variable.
11. Теперь перейдём к теме режимов SE. Как я уже сказал, после каждого этапа обработки SE возвращает своё состояние в RSCore. Вот список таких состояний. Перечисляю состояния. А теперь давайте вернёмся к логике обработки возврата таких состояний.
12. Состояние о завершении первого этапа называется OEC – On Execution Completed. Это состояние вызывается SE после обработки IPT. Состояние о завершении второго этапа называется OSC – On State Completed. Это состояние вызывается SE после обработки IST.
13. После запуска нового SE RSCore ждёт OEC. Дальше работает такая логика.
14. После отработки скрипта RSCore ждёт OSC, после чего работает такая логика.
15. Теперь давайте обсудим взаимодействие IPT между собой, а также межу RSCore. IPT содержит два поля: DynamicPhrase и Parent. Первое определяет должен ли RSCore удалить IPT после его отработки. Это одноразовый обработчик. Второе поле хранит информацию о IPT- родителе, который создал его в системе. Если удаляется родитель, то удаляется и дочерний IPT. Это всевозможные вопросы к фразе, которые могут возникнуть у пользователя.
16. Соответствующая логика ложится и на RSCore. Если отработавший IPT имел дочерей, то они удаляются вместе с ним, а если он был одноразовым, то удаляется он, и все его дочерние IPT.
17. Отсюда выходит, что IPT может динамически добавлять в систему другие IPT. Выходит диалог ☺
18. Отдельно нужно сказать, что все IPT, IPAT и IST хранятся в специальном модуле SiaLibrary, который обеспечивает к ним удобный доступ.
19. Теперь поговорим о данных, необходимых для работы приложения. Все данные, которые содержатся вне программы, могут изменяться, а это ключ к гибкости системы. В качестве языка хранения данных был выбран XML. Рассмотрим его структуру.
20. Разберём XML более подробно. Рассказываю о каждом узле.
21. Продолжаю рассказывать о каждом узле.
22. Теперь по скрипту. Скрипт это одна из самых важных частей проекта. От скорости и правильности его работы, а также лёгкой программной генерации зависит судьба всего проекта. Пройдёмся по тому, что он должен реализовывать.
23. Теперь давайте посмотрим на общий синтаксис, и что уже реализовано. IST работает вот так…
24. Дальше по скрипту…
25. Теперь давайте глобально разберём структуру проекта. Он делится на три части. Описываю части.
26. Теперь рассмотрим способ взаимодействия всех частей нашего решения.
27. Теперь настало время поговорить о базе данных. Она будет нужна для следующей информации. Описываю для чего.
28. Также нам понадобится и дополнительный софт, который будет работать непосредственно с данными. Это поможет нам оперативно реагировать на ошибки, отзывы и предпочтения пользователей.
29. Теперь настало время поговорить о эмоциональной окраске. Это будет одна из главных фишек нашего проекта. Она будет заниматься весьма простым делом: подгружать свои обработчики, при необходимости менять отношение, и говорить можно ли дальше продолжать выполнять команду.
30. Как мы видим….
31. Помимо этого…
32. План работ таков….
33. Ваши вопросы.