Desk-chan T3

Описание программы

Приложение “Desk Chan” является кроссплатформенным программным обеспечением с открытым исходным кодом, представляющим собой интерфейс интеллектуального персонального помощника с визуализацией в виде анимированного персонажа, ориентировочно девушки.
Приложение имеет возможность двусторонне взаимодействовать с операционной системой и другими приложениями на устройстве, а именно посылать им запросы и реагировать на их действия или состояния.
Пользователь может контактировать с приложением через голосовой и текстовый интерфейс, а также через устройства ввода.

Технологии разработки

Ядро приложения написано на Java версии 1.8.
Визуальный интерфейс написан на кроссплатформенной библиотеке JavaFX, но все модули программы обособлены от конкретной реализации визуального интерфейса, за счёт чего в любой момент можно выбрать любой другой механизм отображения без редактирования остальных модулей.
Используемая система сборки: Gradle
Используемая система контроля версий: Git
Проект является свободным программным обеспечением, из-за чего нет ни возможности, ни желания использовать в нём платные решения в любом виде.

Основные принципы, декларируемые при разработке

- Кроссплатформенность
- Свободная лицензия
- Максимальная настраиваемость всех компонентов
- Сохранность данных пользователя, запрет на их тихий сбор и отправку
- Локальность выполнения всех операций, в расчётах задействованы только устройства пользователя
- Открытый исходный код, максимальное взаимодействие с сообществом дабы донести до них все аспекты работы программы
- Лёгкость модификации программы, поддержка многих языков программирования для создания плагинов

Отличия от ПО подобного плана

Каждый современный персональный помощник либо разрабатывается как полноценная личность со строго заданным поведением, либо как обезличенная вопросо-ответная система. Пользователь не может задавать её приоритеты, не может менять её характер и визуальное представление (если оно вообще существует). Большинство подобных систем имеют только платную версию и закрытый исходный код, а если у них есть в наличии API, то оно имеет весьма ограниченный функционал. DeskChan - принципиально другая разработка. Она специально создаётся такой, чтобы работать на любой платформе, без наличия сети, содержать только те функции, которые захочет сам пользователь. Её настройка должна быть проста и невероятно вариативна. DeskChan разрабатывается в тесной связи с большим сообществом и каждый может внести посильный вклад, разработав модуль для программы с помощью любых удобных ему средств, стоит только прочитать несколько страниц спецификации работы приложения.
Главный приоритет в разработке - создание конструктора персонажа. Любому пользователю, даже без специального технического образования и знания программирования, обязаны быть предоставлены все средства для создания своего действительно персонального помощника, с определёнными внешностью, лексиконом, навыками, поведением.

Минимальные системные требования

Все распространённые операционные системы версий выпуска от 2000 года
Поддержка Java 1.8
CPU: не ниже Intel Core 2 Duo, AMD Athlon X2
GPU: любая видеокарта с 256 MB видеопамяти, поддерживающая OpenGL.
RAM: 200 MB
HDD: 100 MB
Периферия: клавиатура, мышь

Рекомендуемые системные требования

Все распространённые операционные системы версий выпуска от 2008 года
Поддержка Java 1.8
CPU: не ниже Intel Core 2 Duo, AMD Athlon X2
GPU: любая видеокарта с 512 MB видеопамяти, поддерживающая OpenGL.
RAM: 1 GB
HDD: 300 MB
Периферия: клавиатура, мышь, микрофон

Требования к надежности и безопасности

Приложение как персональный агент должно иметь возможность работать без конечного срока функционирования и быть постоянно свободным от нагрузки, чтобы вовремя реагировать на запросы пользователя.
Приложение будет собирать и хранить большинство запросов, полученных приложением от пользователя, в целях оптимизации работы. Поскольку все расчёты программы выполняются исключительно на устройствах пользователя, их отправка куда-либо считается нецелесообразной.
Все данные пользователя хранятся исключительно на устройствах пользователя и не распространяются никак (кроме как между устройствами самого пользователя в локальной сети, если это разрешено настройками), кроме его личной отправкой этих данных не через внутренние средства приложения, либо через взаимодействие с другими приложениями на устройстве пользователя, которые собирают свою статистику и приложение на этот процесс повлиять не в состоянии.
Пользователь должен иметь опциональную возможность зашифровать приватные данные своим ключом/паролем. Также, если какие-то данные достаточно хранить в виде хеша или какого-нибудь токена, то следует предпочитать хранение в данном формате вместо хранения в виде открытого текста.
Все плагины, написанные сообществом, могут пройти сертификацию главными разработчиками, для чего они обязаны иметь открытый исходный код и распространяться бесплатно. Сертификация проводится в целях обеспечения безопасности данных и устройств пользователя. При подключении плагинов, не прошедших сертификацию, приложение может предупреждать пользователя об опасности использования не сертифицированных плагинов. Однако приложение никогда не будет блокировать установку не одобренных плагинов. Тут можно посмотреть на модель Android - установка неизвестных приложений возможна без ограничений после принятия пользователем всех рисков.

Требования к стилистике написания кода

Для всех языков

- Табы предпочтительны перед пробелами
- Имена переменных на английском языке, транслит крайне нежелателен
- Осмысленные имена переменных, функций и классов (но без фанатизма, имена на 50 символов тоже неудобны), кроме небольших временных переменных (например, счётчики циклов).
- Рекомендуемая максимальная длина строки - 80-90 символов.
- Следует предпочитать кроссплатформенные функции и библиотеки вместо платформозависимых, если есть такая возможность.
- При отсутствии информации по теме в данном документе следует обращаться к официальным рекомендациям авторов соответствующих языков. Также можно

действовать по аналогии с рекомендациями для Java (подходят для многих Си-подобных языков).

Для Java и Groovy (основаны на стандартном стиле кода Java)

- Имена переменных и функций должны выглядеть так:
  coolFunctionName, coolVariableName
  а не так:
  CoolFunctionName, cool_function_name, CoOlVaRiAbLeNaMe
  То есть первая буква первого слова строчная, первые буквы всех последующих слов заглавные, все слова пишутся слитно без пробелов и подчёркиваний между ними.
- Имена классов и интерфейсов должны выглядеть так:
  MyCoolClass
  а не так:
  myCoolClass, my_cool_class, MY_COOL_CLASS
  То есть требования такие же как и в предыдущем пункте, но первая буква имени класса тоже заглавная.
- Имена констант (обычно это public static final поля с простым типом, а также члены enum) должны выглядеть так:
  SUPER_COOL_VALUE
  а не так:
  super_cool_value, SuperCoolValue
  То есть все буквы заглавные, слова разделены знаками подчёркивания.
- Имена пакетов должны использовать только строчные буквы. Слова должны быть разделены с помощью знаков подчёркивания. Примеры:
  my.super_package.with_cool_classes
  Вообще, для избежания конфликтов рекомендуется использовать в названии пакета название домена, который принадлежит разработчику пакета. Например, если ваш домен eternal-search.com, а ваш пакет называется cool_classes, то полное имя пакета должно иметь вид:
  com.eternal_search.cool_classes.
  Подробнее про это можно прочитать по официальной ссылке:
  https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html
- Следует оборачивать в блок из фигурных скобок {} команду после if, while, for, do, даже если она всего одна. То есть правильно делать:

```
if (a > 10) {
        b = 20;
}
```

  а не:

```
if (a > 10) b = 20;
```

  Аргументация: подобный стиль значительно уменьшает вероятность ошибки типа “добавил ещё одну команду в блок if, а скобки добавить забыл” или “удалил команду из while, а сам while оставил (он вполне бывает полезен и без тела, когда основную работу выполняет условие), в результате условной стала

команда, которая быть не должна”. Подобный стиль принят в очень большом количестве OpenSource-проектов.
- Вы должны ловить все исключения, которые могут возникнуть в более-менее адекватных условиях с помощью блоков try-catch. Надо помнить, что необработанные исключения рушат не только ваш код, но и тот код, который к нему обратился, поэтому последствия могут быть самые произвольные и непредсказуемые. Исключения, которые обозначают штатные ситуации по своей сути (например, отсутствие конфигурационного файла) следует полностью игнорировать, остальное - логгировать штатными средствами DeskChan, если имеется такая возможность.
- В целом весьма неплохо форматирует автоформаттер IDEA с дефолтными настройками, надо только включить оборачивание одиночных команд в ветвлениях и циклах в блоки.

Для Python

- Рекомендуется использовать последний актуальный PEP на эту тему, но при этом использовать табы вместо пробелов.

Для C/C++

- В целом требования к коду аналогичны требованиям к Java. За исключением того, что в C/C++ константами также могут являться define-ы, а пакеты отсутствуют.
- В C++ рекомендуется использовать enum class вместо enum, если не требуется совместимость с кодом на C.
- При использовании обычных enum имена констант должны иметь префикс, связанный с именем enum, чтобы не вызывать конфликты имён.
- Для заголовочных файловы следует предпочитать директиву #pragma once вместо других вариантов include guard.
- Использование конструкции using namespace std в C++ считается чем-то плохим.
- Функции и переменные (не члены класса), которые не должны использоваться нигде кроме текущего файла следует объявлять static.
- Для классов C++ поля должны быть private или (в особых случаях) protected. Доступ к ним из-вне должен осуществляться с помощью геттеров и сеттеров. Тривиальные геттеры и сеттеры следует инлайнить в объявление класса, чтобы гарантировать оптимизации.
- Тривиальные конструкторы (т. е. те, которые просто инициализируют поля объекта своими аргументами) также допускается инлайнить в объявление класса.
- Сложные функции инлайнить в объявление класса не допускается.

Архитектура системы

DeskChan следует рассматривать как ядро и совокупность модулей. Модули напрямую взаимодействуют исключительно с ядром и предоставляемыми ядром интерфейсами. Между собой же модули общаются посредством так называемых сообщений. Сообщение состоит из тега (строка) и данных (объект). Тег обязан быть непустой строкой. Данные имеют следующие допустимые типы: null, String, Integer, Byte, Short, Long, Float, Double, Boolean, List<Object>, Map<String, Object>. При этом последние два должны содержать в себе только объекты этих же типов. Допускается в особых случаях использовать другие типы, но должно гарантироваться корректное приведение к String с помощью вызова toString, при этом данные должны сохранить осмысленность. Также должно быть возможно задать все те же данные с помощью String (получатель сообщения должен поддерживать помимо собственного класса данных и получение String вместо него).
Всё это можно подытожить одним условием “данные сообщения должно быть возможно сериализовать и десериализовать в JSON без потери данных”.
Каждый плагин может подписываться на сообщения по тегу. Таким образом будет вызван зарегистрированный callback для каждого сообщения с совпадающим тегом. Каждый плагин может подписываться на любое количество сообщений. Каждый плагин может не только подписываться, но и отписываться от тегов сообщений.
Каждый плагин имеет уникальный строковый id (система не даст загрузить два плагина с одинаковым id). Каждый плагин может в любой момент произвести отправку сообщения, указав тег и данные. При этом все получатели помимо тега и данных также получают id отправителя. Считается, что владение данными сообщения после его отправки переходит ядру и после окончания процесса ни у кого не должно остаться ссылки на данные, чтобы их собрал сборщик мусора. Таким образом отправитель сообщения не должен обращаться к отправленному значению после отправки, а получатели не должны модифицировать непосредственно полученный объект. Ядро гарантирует потокобезопасность процесса отправки и получения сообщений, однако зарегистрированный получатель сообщения не должен делать каких-либо предположений на счёт того из какого потока он вызван (соответственно, при доступе к каким-либо внешним данным следует использовать примитивы синхронизации).
На основе данной парадигмы можно строить любые более высокоуровневые абстракции.
Имеется рекомендация для случаев, когда ожидается ответ на сообщение. В этом случае рекомендуется использовать тип Map для данных сообщения, в который поместить как минимум одно значение любого допустимого типа с ключом “seq”. Ответное сообщение следует доставлять на тег, совпадающий с идентификатором плагина отправителя и данными типа Map, содержащих как минимум один ключ “seq” абсолютно совпадающий с тем значением, которое передал отправитель.
Допускается, что некоторые плагины могут располагаться в отдельных процессах или даже быть серверами на других хостах. В этом случае данные сообщения сериализуются в JSON или подобный по возможностям протокол и передаются через пайпы/сокеты получателю. В обратную сторону это работает так же.

Ограничения на типы данных сообщения накладывается также с целью совместимости с другими языками программирования - аналогичные типы данных присутствуют так или иначе в любом языке программирования общего назначения.
Нарушение данного правила приведёт к ограничению функциональности для плагинов на других языков, не поддерживающих прямой доступ к классам Java, что считается крайне нежелательным.
Ядро реализует систему обмена сообщений (отправку, подписку, отписку), учёт и загрузку плагинов, логгирование и небольшое количество вспомогательных функций и интерфейсов. Расширение данного функционала нежелательно и должно согласовываться с архитектором приложения.
Помимо ядра имеется также “ядерный плагин” (CorePlugin), он предоставляет свой интерфейс только через сообщения, при этом реализует различный дополнительный функционал (например, “альтернативы”, “пайпы”). Его присутствие в системе в отличии от остальных плагинов гарантируется.
Подробнее про сообщения, API ядра и некоторых стандартных плагинов можно почитать по ссылке:
https://github.com/deskchanproject/DeskChanJava/blob/master/docs/ru/api.md

Структура файлов DeskChan

Каталог DeskChan содержит три основных подкаталога:

- bin - содержит исполняемые файлы ядра
- plugins - содержит установленные плагины (их исполняемые файлы, ресурсы, исходный код должны находится там). Каждый плагин должен размещаться в отдельном подкаталоге.
- data - содержит временные данные, а также настройки плагинов. Каждому плагину выделяется отдельный одноимённый каталог. Ядро может также создавать некоторые файлы прямо в data (например, чёрный список плагинов, системный лог), однако плагинам это не допускается. Пользователь должен иметь возможность удалить каталог data для сброса к “заводским настройкам”, что не должно повлечь сбои в работе плагинов. Дистрибутив приложения поставляется БЕЗ каталога data. В будущем возможно появление функции синхронизации каталога data между устройствами пользователя (но это требует дополнительного обсуждения).

Требования к поддержке модификации

Приложение должно иметь как можно большие возможности для модификации. Каждая модификация должна иметь возможность сразу быть подключённой во время работы программы и сразу начать работу. Планы на данным момент

- Подключение скомпилированных java-модулей
- Подключение модулей на Groovy (уже есть)
- Подключение модулей на Python (уже есть, в виде сторонней модификации, желательно переписать на JNI и libpython вместо устаревшего jython)
- Подключение модулей на C/C++ (способ реализации такой возможности обсуждается)

- Подключение скриптов на скриптовых языках (возможно, это будет Lua или JS)
- Интерфейс типа простой TCP сокет (обмен данными через JSON, также возможен опциональный бинарный протокол)
- Интерфейс типа WebSockets (для простого доступа к функциям DeskChan из плагинов для браузеров на JavaScript)
- Процедуры командной строки (удобнее всего будет реализовать в виде обёрток над TCP-интерфейсом)
- Инструкции на особом скриптовом языке, по факту представляющий описание отправляемых модулям программы сообщений

Помимо этого программа должна легко подключать ресурсы и хранить их в незашифрованном виде.

Визуализация работы программы

При запуске программы на экране сразу появляется персонаж. Представление персонажа может быть разным в зависимости от потребностей и возможностей пользователя:

Растровая графика

Персонаж отображается через показ на экране неанимированных однотипных растровых спрайтов. Приложение выбирает между спрайтами в зависимости от внутренней конфигурации. Приспособить свои спрайты для программы может любой желающий без изменения внутренних ресурсов и кода приложения.

Векторная графика

Персонаж отображается как стоящая в анфас двумерная девушка, нарисованная в стиле японской анимации. Все её движения программно анимированы. Её пропорции тела, одежда, части головы и цвет каждой из компонент могут быть изменены через внутренний редактор. Добавлять ресурсы в виде одежды и частей головы в векторном формате может любой желающий без изменения внутренних ресурсов и кода приложения. Предпочтительный формат векторной графики - SVG (возможно сохранение дополнительных данных через нестандартные теги и атрибуты с определёнными префиксами, что допускается стандартом).

3D-графика

Персонаж отображается на экране в виде 3D-модели. Её движения задаются ручной анимацией, созданной в стороннем ПО. Добавлять 3D-модели и анимацию для них может любой желающий без изменения внутренних ресурсов и кода приложения.

Помимо персонажа на экране, конечно же по желанию пользователя, отображается интерфейс уведомлений со стороны приложения через текст на естественном языке. В качестве основных инструментов для этого могут использоваться "облачко" или чат.

Пользователь может вызвать и свернуть окно ввода, чтобы взаимодействовать с программой.
Каждый элемент, визуализирующий работу программы, можно масштабировать и перемещать в границах рабочей области операционной системы.

Аудио-интерфейс

Помимо визуализации между пользователем и приложением организован аудио-интерфейс: пользователь может озвучивать в микрофон запросы на естественном языке, а в обратную сторону получать синтезированный голос.
Пользователь начать запись с микрофона по горячим клавишам, через меню или обратившись по заданному имени или выражению к программе. Всю текстовую отчётность программа может проговаривать через синтезатор голоса.
Для распознавания голоса используется программный комплекс Pocketsphinx, который обеспечивает тонкую настройку словарей и может работать без подключения к сети.
Для синтеза голоса планируется написать собственный синтезатор, который позволит не вручную настраивать тембр и высоту голоса. Конкретные технологии реализации синтеза голоса обсуждаются.

Взаимодействие с другими приложениями

Приложение должно иметь возможность взаимодействовать с другими приложениями. Поскольку многие приложения не имеют возможность получать и обрабатывать сторонние запросы, не связанные с событиями операционной системы, как один из вариантов предполагается создание пользовательских макросов действий устройств ввода.
Для всех остальных случаев смотреть пункт "Требования к поддержке модификаций".

Персонализация

Вся конфигурация приложения, за исключением стандартных значений, должна храниться в незашифрованном виде в отдельных файлах конфигурации, чтобы каждый пользователь мог настроить любой компонент системы. Для этих же целей пользователю предоставлен интерфейс настроек программы.
Пользователь может включать и отключать любые ресурсы программы, а так же создавать свои.

Команды

Приложение как персональный агент может выполнять определённые команды. Каждая команда или набор команд должны быть выполнены в отдельном модуле. При подключении они отправляют составленную согласно определённому стандарту текстовую информацию о том, как использовать эти команды. При отправке запроса к команде она полностью свободна в своих действиях. Модуль команд, как и любой

другой модуль, может оперировать собственными ресурсами, иметь собственную конфигурацию и общаться с любыми встроенными в приложении или сторонними модулями.
Приложение само решает, когда и при каких условиях команда будет вызвана, а также собирать статистику о частоте её вызовов.
Команда при получении запроса от приложения получает все данные о том, каким образом была вызвана программа, включая стороннюю информацию в запросе пользователя, которая может быть использована как аргументы.
Вызов команд может быть организован через визуальный интерфейс, горячие клавиши или запросы на естественном языке.

Стандартные команды

Данные команды будут распространяться в базовой комплектации, их создание в приоритете

- Органайзер: события, напоминания, будильник, ...
- Работа с файловой системой: поиск, удаление, запуск файлов
- Поиск в интернете
- Работа с популярными веб-сервисами: погода, пробки, афиша, почта, ...
- Работа с операционной системой: администрирование, чистка, слежение за показателями компонент устройства, ...
- Макросы устройств ввода
- Чтение и отправка сообщений в мессенджерах, соц.сетях

Собеседник

Помимо выполнения команд и слежением за устройством приложение может имитировать собеседника. Для этого используется тот же самый интерфейс, что и для вызова команд: голосовой или текстовый.
Приложение старается имитировать человека, для этого программа обладает следующими характеристиками:

- Возможность оценивать высказывания пользователя
- Наличие внутренней конфигурации, имитирующая характер и эмоциональное состояние
- Наличие собственной базы данных, с которой согласуются все реплики приложения
- Наличие логико-семантического аппарата, с помощью которого программа может оперировать смысловыми объектами и составлять их в логические цепочки
- Возможность видоизменять текст для склонения его к определённому лексикону
- Наличие пакетов интересов и пользовательских сценариев, которые можно подключать и отключать в любое время
- Нечувствительность к регистру, ошибкам, порядку слов, синонимам и жаргонизмам.

Во всём остальном механизм беседы будет аналогичен типовой архитектуре чат-ботов.
Последовательность действий приложения, с помощью которой создаётся ощущение определённости характера приложения, более приоритетна, чем схожесть речи приложения с человеческой.

Описание релизной версии

В релизную версию входят:

- Ядро, полностью реализующее описанный в главе “Архитектура системы” функционал. - 70%
- Визуальный интерфейс, одинаково работающий на всех распространённых десктопных ОС, описанных в главе “Рекомендованные системные требования”. Интерфейс должен реализовать как минимум 2 из 3 возможных выводов персонажа на экран в главе “Визуализация”, а также предоставлять возможность создания многофункциональных окон для всех модулей приложения. - 30%
- Как минимум половина списка различных способов модификации приложения без изменения исходного кода, представленных в главе “Требования к поддержке модификаций”. - 40%
- Система характера. - 40%
- Полностью готовый аудио-интерфейс. - 0%
- Функция собеседника, содержащая как минимум 3 характеристики, описанных в главе “Собеседник”. - 0%
- Интерфейс вызова команд, полностью реализующий функционал из главы “Команды”. Все описанные модули команд в главе “Стандартные команды”. - 0%
- Ресурсы для маскота. - 2%
- Текстовые ресурсы системы характеров как минимум в 3 тысячи единиц. - 3%
- Графические ресурсы для векторной графики в количестве 100 единиц. - 0%
- Пользовательские паки персонажей в количестве 20 единиц. - 5%
- Сайт с форумом, разделом статей по работе программы, разделом для модификаций. - 5%
- Локализация на английский язык. - 0%