UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Voice-Assisted Computer Accesibility
Documentación Técnica

Presentado por Víctor Manuel Martinez García
en Universidad de Burgos — 9 de junio de 2025
Tutor: Pedro Luis Sanchez Ortega

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Este plan detalla los aspectos clave en la planificación, gestión y viabilidad del desarrollo del software **Voice-Assisted Computer Accessibility (VACA)**.

El proyecto es una solución accesible y funcional que facilita el uso de sistemas informáticos a personas con movilidad reducida, introduciendo tecnologias de voz, visión artifical e inteligencia artifical.

Este documento incluye una planificacion temporal asi como un analisis de la viabilidad economica y legal.

A.2. Planificación temporal

La planificación del proyecto se realizó siguiendo una metodología ágil, dividiendo el trabajo en sprints que cubren diferentes fases. El proyecto abarca desde febrero hasta junio, organizándose en las siguientes etapas:

1. **Fase de investigación (febrero)**: Revisión del estado del arte de CUAs, estudio de modelos STT/TTS, aprendizaje sobre frameworks para el uso de LLMs, herramientas para visión artifical.

2. **Desarrollo del back-end (marzo-abril)**: Integración de LLMs, modelos de ASR, modelos de visión, y arquitectura entre todas las partes a un agente principal.

3. **Desarrollo del front-end (abril-mayo)**: Busqueda de una herramienta OpenSource para el desarrollo de la interfaz gráfica.

   Uso y aprendizaje de Tkinter y posterior paso a CTkinter integrando back-end y front-end.

4. **Pruebas y validación (mayo)**: Evaluación funcional de las herramientas, pruebas de prompts de voz, tiempos de respuesta de los modelos de inferencia, pruebas del software en local/VM.

5. **Documentación (mayo-junio)**: elaboración de una memoria del proyecto en conjunto con los anexos del mismo.

A.3. Estudio de viabilidad

Viabilidad económica

El proyecto se ha desarrollado utilizando principalmente herramientas y recursos gratuitos o de código abierto, lo cual ha permitido minimizar los costes asociados. A continuación, se detallan los principales aspectos económicos:

- **Herramientas utilizadas:** Python, CTkinter, OpenCV, PyAutoGUI, LangChain, Whisper, COQUI-TTS, Docker y FastAPI. Todas ellas son gratuitas y de código abierto.
- **Recursos computacionales:** Durante el desarrollo se utilizó hardware local, y en fases avanzadas, modelos pesados como YOLOv8 y FlorenceV2 fueron desplegados en un servidor del ITCL equipado con una **GPU NVIDIA A30 de 24GB**, optimizando así los tiempos de inferencia.
- **Modelos LLM:** Se emplearon tanto modelos de código abierto (FlorenceV2, YOLOv8, Whisper, COQUI) como APIs comerciales (Claude y GPT-4) que implicaron costes variables.

A continuación, se desglosan los costes estimados si el sistema se llevase a producción real, considerando tanto infraestructura como licencias, consumo energético y otros factores de operación continua.

Elemento	**Coste estimado**	**Frecuencia**
Servidor con GPU NVIDIA A30 (24GB VRAM)	6.000 €	Único
Consumo eléctrico servidor (8h/día)	14,40 €	Mensual
Conexión a internet (fibra 1 Gbps)	40 €	Mensual
Mantenimiento de infraestructura (ITCL)	600 €	Anual

Tabla A.1: Coste estimado de infraestructura y recursos computacionales

Servicio/API	**Coste estimado**	**Frecuencia**
OpenAI GPT-4 (100.000 tokens/mes aprox.)	20 €	Mensual
Anthropic Claude (acceso de desarrollador)	15 €	Mensual

Tabla A.2: Costes asociados al uso de APIs comerciales de modelos LLM

Elemento	**Coste estimado**	**Frecuencia**
Salario programador (5 meses)	7.500 €	Proyecto
Hardware usuario final (PC + mic + auriculares)	500 €	Único

Tabla A.3: Otros costes estimados para desarrollo y operación

Alternativa económica basada en ejecución local:

Como alternativa al mantenimiento continuo de un servidor con GPU dedicada, se considera viable la ejecución del sistema en local utilizando un ordenador personal de gama media-alta con una GPU dedicada. Esto permitiría realizar inferencias de modelos como YOLOv8 o Florence sin necesidad de recurrir a infraestructura externa, reduciendo así los costes mensuales de operación.

A continuación, se muestra una configuración de hardware recomendada para lograr tiempos de inferencia aceptables ejecutando el sistema en local:

Elemento	**Coste estimado**
PC Case	80 €
Placa Base (MSI B550M PRO-VDH WIFI	111 €
RAM DDR4 3200Mhz 32GB	60 €
Procesador AM4 (Ryzen 7 5800X)	180 €
Refrigeración	80 €
PSU 750W	100€
GPU 6GB VRAM (RTX 3050)	210€
Disco Duro	100 €

Tabla A.4: Coste de un ordenador que pueda usar el modelo en local sin servidores

Resumen económico aproximado del proyecto en fase de producción:

- **Coste inicial de la infraestructura:** Incluye la adquisición de un servidor con GPU (NVIDIA A30), asi como el hardware minimo necesario para un usuario final, el coste estimado asciende a un valor de entre **6000 - 8000 €**.
- **Coste operativo mensual:** Este se comprende del consumo electrico del servidor, conexión a internet, uso de APIs comerciales de OpenAI y Anthropic. Esto tendria un coste aproximado de **80-100€ mensuales**
- **Coste de desarrollo humano:** Estimado de un único programador trabajando durante 5 meses, con una media salarial de 1500€ mensuales. **Total: 7500€**

Considerando el uso intensivo de herramientas open-source, la reutilización de infraestructura preexistente en el ITCL y la posibilidad de sustituir APIs de pago por modelos open-source (cambiando APIs de Claude y OpenAI), el proyecto presenta una **viabilidad económica alta**. Puede ser escalado o adaptado a distintos entornos sin incurrir en costes prohibitivos, lo que lo convierte en una solución sostenible tanto a corto como a largo plazo.

Viabilidad legal

El proyecto cumple con las normativas legales vigentes en los siguientes aspectos:

- **Licencias de software:** Todas las herramientas utilizadas (Python, bibliotecas de IA, modelos open-source) se encuentran bajo licencias compatibles con su uso, modificación y distribución (MIT, Apache 2.0, GNU).

- **Protección de datos personales:** El sistema no almacena información personal ni sensible. Las grabaciones de voz son procesadas de forma local o en servidores seguros del ITCL, sin ser enviadas a terceros, ademas de que los contenedores de los modelos son destruidos tras un tiempo de inactividad sin dejar ningun tipo de dato en el servidor.

- **Accesibilidad:** El software está diseñado específicamente para cumplir con principios de accesibilidad digital, lo que alinea con las directrices de la *Ley General de Discapacidad*.

- **Código abierto:** El código desarrollado puede ser compartido, auditado o ampliado por terceros, fomentando la transparencia, colaboración y reutilización del software en contextos sociales o institucionales.

Requisito Legal	Aplicación en VACA	Cumplimiento
Accesibilidad universal	Control por voz sin necesidad de dispositivos físicos	Sí
Diseño para todos (diseño universal)	Accesible sin adaptaciones adicionales	Sí
Acceso a las TICs	Compatible con lectores de pantalla y salida TTS	Sí
Solución no discriminatoria	Uso gratuito, sin hardware costoso	Sí
Accesibilidad desde el diseño	Interfaz visual adaptada desde fase inicial	Sí

Tabla A.5: Cumplimiento del proyecto VACA con la Ley General de Discapacidad (España)

Amortización del proyecto

Teniendo en cuenta los costes descritos previamente, se estima que el coste total de desarrollo y puesta en marcha del sistema **VACA** ronda los **8.000 a 10.000 €**, incluyendo infraestructura, desarrollo, y coste de APIs comerciales en el primer año.

Es importante destacar que parte del hardware utilizado, especialmente el servidor con GPU proporcionado por el ITCL, no está dedicado exclusivamente a este proyecto. Dicho servidor alberga modelos que también se utilizan en otras investigaciones y aplicaciones, lo que permite distribuir parte del coste de infraestructura entre varios desarrollos. Esta reutilización de recursos mejora significativamente la rentabilidad global del sistema.

Aunque el código fuente de VACA se encuentra alojado públicamente en GitHub como **software libre**, se podria contemplar su distribución mediante versiones comerciales mantenidas y actualizadas, orientadas a usuarios con movilidad reducida. Estas versiones podrían incluir soporte técnico y actualizaciones periódicas.

Resumen estimado de amortización:

- **Coste inicial total:** 10.000 € aprox. (con parte de infraestructura compartida)
- **Coste de mantenimiento durante 5 años:** 600 € × 5 = 3.000 €
- **Coste total acumulado estimado para VACA: 13.000 €**, aunque parte del hardware puede considerarse amortizado parcialmente por su uso compartido.

Si el sistema fuese licenciado a **50 usuarios** durante esos cinco años con una tarifa de **5 € mensuales** por usuario (modelo de suscripción), se obtendría un ingreso de:

$$50 \text{ usuarios} \times 5 \text{ €/mes} \times 12 \text{ meses} \times 5 \text{ años} = \mathbf{15.000\ €}$$

Este escenario permitiría no solo amortizar completamente la inversión realizada, sino también generar un pequeño margen económico. En caso de alcanzar una mayor base de usuarios o integrar el sistema en contextos institucionales o sanitarios, el retorno económico sería aún más favorable.

Adicionalmente, el sistema sustituye soluciones comerciales mucho más costosas, y reduce significativamente la necesidad de hardware especializado, haciendo su adopción más accesible y sostenible a nivel económico y social.

Conclusión

La amortización del proyecto es viable incluso en contextos de adopción moderada. El uso compartido de infraestructura, el enfoque open-source y la orientación social del sistema refuerzan su sostenibilidad técnica y económica en el medio y largo plazo.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En esta sección se presentaran los requisitos de la aplicación **Voice Assisted Computer Accesibility (VACA)** abordando los objetivos generales como especificos del proyecto.

También se proporcionara una especificación detallada de los requisitos a traves de tablas de casos de uso, complementadas con su respectivos diagramas para mejor comprensión.

B.2. Objetivos generales

El objetivo principal del proyecto **VACA** es conseguir una mejora en la accesibilidad de usuarios con movilidad reducida a sistemas informáticos.

- **Crear una herramienta accesible para sistemas informaticos:** Facilitar a usuarios con movilidad reducida el uso de sistemas informáticos.
- **Eliminar la interaccion fisica con teclado o raton:** Utilizando un agente que se encargue de estas tareas.
- **Fomentar la empleabilidad:** Promover y crear puestos de trabajo que requieran el uso de sistemas informaticos para personas con movilidad reducida gracias al uso de **VACA**

- **Optimizar costes:** Crear una herramienta de coste reducido comparado a soluciones actuales a este mismo dilema, sin ningún tipo de repercusión fisica.
- **Promover el uso de herramientas inteligentes:** Mediante los beneficios que puede dar este para las personas como demuestra el proyecto **VACA**

B.3. Catálogo de requisitos

Requisitos Funcionales

- **RF-01 El sistema debe interpretar prompts de voz del usuario:** La aplicación ha de ser capaz de detectar y usar el microfono del entorno en el que se encuentra y entender al usuario.
- **RF-02 Comprensión del entorno:** La aplicación ha de ser consciente de lo que se encuentra en el entorno gráfico.
- **RF-03 Capacidad de realizar acciones:** La aplicación ha de ser capaz de realizar acciones dentro del sistema del usuario para cumplir el prompt dado por el usuario.
- **RF-04 Proporción de respuestas por voz:** La aplicacion ha de ser capaz de poder convertir texto en habla y devolverla al usuario por una salida periferica.
- **RF-05 Interfaz accesible:** Se ha de contar con una interfaz sencilla y facil de entender para el usuario con distintos tipos de outputs.
- **RF-06 Pensamiento:** La aplicación ha de ser capaz de transmitir sus acciones a realizar o realizandose.
- **RF-07 Memoria:** La aplicación ha de ser capaz de tener una memoria cuando se ejecute.
- **RF-08 Abortar/Reiniciar:** La aplicacion ha de ser capaz de poder ser abortada o reiniciada en caso de deteccion de comportamientos anómalos.
- **RF-09 Input adicional:** La aplicación a mayores de recibir prompts mediante voz ha de ser capaz tambien de funcionar con prompts escritas a mano.

- **RF-10 Ejecutable en entornos locales:** La aplicación ha de ser capaz de ejecutarse en cualquier dispositivo con entorno windows 10.

Requisitos no funcionales

- **RNF-01 Rendimiento:** Los tiempos de inferencia de imagenes han de ser bajos para que la aplicación pueda ser mas fluida.

- **RNF-02 Usabilidad:** La aplicación debe poder funcionar correctamente con sistemas con VRAM de 4GB.

- **RNF-03 Privacidad:** Los datos personales del usuario no seran guardados en ningun sistema que no sea el local propio del usuario.

- **RNF-04 Escalabilidad:** La aplicación es capaz de escalar progresivamente hacia arriba segun se desarrollen mas los modelos implementados en el mismo.

- **RNF-05 Mantenimiento:** La aplicacion es facil de mantener dado que los modelos mas "pesados"se encuentran alojados en los servidores del ITCL, y el proyecto en si se encuentra correctamente estructurado para su mantenimiento.

- **RNF-06 Disponibilidad:** La aplicacion siempre estara disponible mientras los servidores del ITCL esten disponibles y el contenedor con los modelos pesados funcional.

B.4. Especificación de requisitos

Actores

Actor-ID	A01
Nombre:	**Usuario**
Versión	1.0
Autor	
Descripción	Usuario que utiliza la aplicación **VACA** para poder hacer que este gestione su sistema.
Tipo	Usuario
Objetivo	Poder usar un sistema operativo.
Responsabilidades	▪ Insertar prompts mediante voz o teclado. ▪ Confirmar o cancelar los prompts recogidos por el asistente. ▪ Consultar resultado del asistente. ▪ Consultar pensamientos del asistente. ▪ Reiniciar o abortar el asistente.
Relaciones con casos de uso	CU-01,CU-01.1, CU-01.1.2, CU-05, CU-05.1, CU-05.2

Tabla B.1: A01 - Usuario

Actor-ID	A02
Nombre:	**ITCL YOLO-Florence**
Versión	1.0
Autor	Víctor Manuel Martínez García
Descripción	Contenedor que aloja el modelo de Yolo-Florence
Tipo	Sistema
Objetivo	Retornar lo que vea o pida el asistense sobre el entorno gráfico.
Responsabilidades	▪ Obtener una captura de pantalla del sistema. ▪ Generar una imagen que el LLM pueda entender. ▪ Retornar un mensaje con el contexto de la interfaz gráfica.
Relaciones con casos de uso	CU-02, CU-02.3

Tabla B.2: A02 - ITCL Yolo-FLorence

Actor-ID	A03
Nombre:	**ITCL Whisper-Coqui**
Versión	1.0
Autor	Víctor Manuel Martínez García
Descripción	Contenedor que aloja el modelo de Whisper-Coqui
Tipo	Sistema
Objetivo	Retornar audios transcritos a texto o texto transformado en audio.
Responsabilidades	CU-01.1, CU-03.3CU-04, CU-04.2 ▪ Recibir un audio o texto. ▪ Devolver audio bytes para la generacion de un `.wav`. ▪ Retornar un texto obtenido de la inferencia de un audio.
Relaciones con casos de uso	

Tabla B.3: A03 - ITCL Whisper-Coqui

Actor-ID	A04
Nombre:	**Computer Use Agent**
Versión	1.0
Autor	Víctor Manuel Martínez García
Descripción	Agente que se encarga de la intercomunicacion de todo el sistema.
Tipo	Sistema
Objetivo	Gestionar prompts del usuario llamando a las herramientas necesarias en cada momento.
Responsabilidades	CU-01, CU-02, CU-02.1, CU-02.2, CU-02.3, CU-03, CU-03.2, CU-03.3, CU-04, CU-4.1, CU-04.03, CU-06.1, CU-06.3 ▪ Gestionar los prompts del usuario. ▪ Comunicacion con las herramientas. ▪ Retorno de pensamiento y resultados para el usuario. ▪ Reproducir audio TTS del resultado final.
Relaciones con casos de uso	

Tabla B.4: A04 - Computer Use Agent

Actor-ID	A05
Nombre:	**OpenAI / Anthropic**
Versión	1.0
Autor	Víctor Manuel Martínez García
Descripción	LLM proporcionado por OpenAI o Anthropic dependiendo a que API se llame.
Tipo	Sistema
Objetivo	Actuara como el cerebro del agente.
Responsabilidades	▪ Recibir prompt del agente o agentes. ▪ Retornar una respuesta a el prompt del agente. ▪ Entender imagenes.
Relaciones con casos de uso	CU-03.1

Tabla B.5: A05 - OpenAI/Anthropic

Actor-ID	A06
Nombre:	**Browser-Use**
Versión	1.0
Autor	Víctor Manuel Martínez García
Descripción	Agente que se encargara de realizar busquedas por internet.
Tipo	Sistema
Objetivo	Buscar en internet información pedida por el CUA.
Responsabilidades	▪ Recibir prompt de CUA ▪ Navegar por internet ▪ Retornar una respuesta a el prompt del CUA.
Relaciones con casos de uso	CU-06, CU-06.2

Tabla B.6: A06 - Browser-Use

Casos de uso

CU-01	**Enviar prompt al sistema**
Versión	1.0
Autor	Victor Manuel Martinez García
Requisitos asociados	RF-01, RF-09, RF-06, RF08
Descripción	El usuario emite un prompt que será interpretado y procesado por el sistema.
Precondición	El sistema está en funcionamiento y esperando una orden.
Acciones	1. CU-01.1 Capturar prompt por voz. 2. CU-01.2 Capturar prompt escrita. 3. CU-01.3 Enviar prompt al agente CUA.
Postcondición	El agente recibe y empieza a procesar el prompt.
Excepciones	Error en micrófono, texto no válido, orden no comprendida.
Importancia	Alta

Tabla B.7: CU-01 Enviar prompt al sistema.

CU-01.1	**Capturar prompt por voz**
Versión	1.0
Autor	Victor Manuel Martinez García
Requisitos asociados	RF-01
Descripción	El sistema activa el micrófono y detecta voz del usuario.
Precondición	Micrófono habilitado.
Acciones	1. Activar escucha de voz. 2. Registrar la entrada en formato de audio `.wav`. 3. CU-01.1.1 Transcribir el audio. 4. CU-01.1.2 Confirmar o cancelar prompt
Postcondición	Audio disponible para transcripción.
Excepciones	Micrófono no disponible o sin permisos.
Importancia	Alta

Tabla B.8: CU-01.1 Capturar prompt por voz.

CU-01.1.1	**Transcribir audio**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-01
Descripción	Se transforma la entrada de voz en texto utilizando ASR.
Precondición	Se ha capturado audio del usuario.
Acciones	1. Enviar audio al modelo Whisper. 2. Obtener y almacenar texto resultante.
Postcondición	Texto disponible para validación.
Excepciones	Ruido, errores de transcripción.
Importancia	Alta

Tabla B.9: CU-01.1.1 Transcribir audio.

CU-01.1.2	**Confirmar o cancelar prompt**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-01, RF-05
Descripción	El usuario decide si quiere enviar el prompt transcrito o escrito.
Precondición	Prompt ya transcrito o escrito.
Acciones	1. Mostrar el prompt al usuario en pantalla. 2. Esperar validación o rechazo mediante voz.
Postcondición	Prompt validado o descartado.
Excepciones	No hay confirmación.
Importancia	Alta

Tabla B.10: CU-01.1.2 Confirmar o cancelar prompt.

CU-01.2	**Capturar orden escrita**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-05,RF-09
Descripción	El sistema registra un prompt escrito manualmente por el usuario.
Precondición	El usuario tiene acceso a un teclado.
Acciones	1. Capturar texto introducido.
Postcondición	Prompt textual disponible para procesamiento.
Excepciones	Entrada vacía o inválida.
Importancia	Media

Tabla B.11: CU-01.2 Capturar orden escrita.

CU-01.3	**Enviar prompt al agente CUA**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-06
Descripción	Se comunica el prompt confirmado al Computer Use Agent para su ejecución.
Precondición	Prompt validado.
Acciones	1. Encapsular prompt. 2. Enviar al CUA.
Postcondición	CUA inicia procesamiento.
Excepciones	Falla en envío o tiempo de espera.
Importancia	Alta

Tabla B.12: CU-01.3 Enviar prompt al agente CUA.

CU-02	**Interpretar entorno gráfico**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-02
Descripción	El sistema analiza el entorno gráfico para obtener contexto visual.
Precondición	Prompt del usuario requiere contexto visual.
Acciones	1. CU-02.1 Capturar imagen del sistema. 2. CU-02.2 Enviar imagen a YOLO-Florence. 3. CU-02.3 Obtener descripción del entorno visual.
Postcondición	Contexto visual entregado al agente para razonar.
Excepciones	Captura fallida, error en el contenedor de visión.
Importancia	Alta

Tabla B.13: CU-02 Interpretar entorno gráfico.

CU-02.1	**Capturar imagen del sistema**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-02
Descripción	Se realiza una captura de pantalla del entorno gráfico.
Precondición	CUA requiere información visual.
Acciones	1. Ejecutar captura automática. 2. Guardar imagen en memoria temporal.
Postcondición	Imagen lista para ser analizada.
Excepciones	Fallo en permisos o captura nula.
Importancia	Alta

Tabla B.14: CU-02.1 Capturar imagen del sistema.

CU-02.2	**Enviar imagen a YOLO-Florence**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-02
Descripción	Se envía la imagen capturada al contenedor YOLO-Florence.
Precondición	Imagen capturada correctamente.
Acciones	1. Formatear imagen a base64. 2. Enviar petición POST al contenedor.
Postcondición	Imagen en procesamiento.
Excepciones	Error HTTP, contenedor inactivo.
Importancia	Alta

Tabla B.15: CU-02.2 Enviar imagen a YOLO-Florence.

CU-02.3	**Obtener descripción del entorno visual**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-02, RF-07
Descripción	Recibe el análisis semántico del entorno gráfico y lo pasa al LLM.
Precondición	YOLO-Florence ha respondido con éxito.
Acciones	1. Interpretar respuesta. 2. Añadir información al contexto del agente.
Postcondición	Entorno visual contextualizado.
Excepciones	Respuesta nula o sin comunicación al servidor.
Importancia	Alta

Tabla B.16: CU-02.3 Obtener descripción del entorno visual.

CU-03	**Realizar acciones en el sistema**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-01,RF-03
Descripción	El sistema lleva a cabo las acciones requeridas por el usuario en el sistema operativo.
Precondición	Prompt ya ha sido interpretado y validado.
Acciones	1. CU-03.1 Interpretar intención del usuario. 2. CU-03.2 Ejecutar acción solicitada. 3. CU-03.3 Confirmar ejecución al usuario.
Postcondición	Acción realizada y notificada.
Excepciones	Error al ejecutar comandos, permisos insuficientes.
Importancia	Alta

Tabla B.17: CU-03 Realizar acciones en el sistema.

CU-03.1	**Interpretar intención del usuario**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-03
Descripción	El sistema analiza el contenido del prompt para identificar la acción a ejecutar.
Precondición	Prompt recibido por el CUA.
Acciones	1. Enviar prompt al LLM. 2. Recibir razonamiento e instrucción.
Postcondición	Acción identificada.
Excepciones	Prompt ambiguo o no comprendido.
Importancia	Alta

Tabla B.18: CU-03.1 Interpretar intención del usuario.

CU-03.2	**Ejecutar acción solicitada**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-03
Descripción	Se lleva a cabo la acción dentro del sistema operativo.
Precondición	Instrucción de acción clara y válida.
Acciones	1. Invocar herramienta correspondiente (abrir app, escribir, mover cursor, etc.). 2. Ejecutar la orden recibida.
Postcondición	Acción completada o fallida.
Excepciones	Error del sistema, permisos, acción no implementada.
Importancia	Alta

Tabla B.19: CU-03.2 Ejecutar acción solicitada.

CU-03.3	**Confirmar ejecución al usuario**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-06
Descripción	El sistema notifica al usuario si la acción se realizó correctamente.
Precondición	Acción finalizada.
Acciones	1. Generar mensaje con el resultado. 2. Mostrar y/o vocalizar el mensaje al usuario.
Postcondición	Usuario informado.
Excepciones	Error en la interfaz o en el TTS.
Importancia	Alta

Tabla B.20: CU-03.3 Confirmar ejecución al usuario.

CU-04	**Responder al usuario por voz**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-04, RF-06
Descripción	El sistema convierte la respuesta en texto del asistente a formato de voz y la reproduce para el usuario.
Precondición	El asistente ha generado una respuesta final para el usuario.
Acciones	1. CU-04.1 Generar respuesta textual. 2. CU-04.2 Transformar texto en audio. 3. CU-04.3 Reproducir respuesta por altavoz.
Postcondición	El usuario recibe una respuesta verbal clara.
Excepciones	Fallo en la síntesis o reproducción de audio.
Importancia	Alta

Tabla B.21: CU-04 Responder al usuario por voz.

CU-04.1	**Generar respuesta textual**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-06
Descripción	El asistente crea una respuesta en texto a partir del análisis del prompt.
Precondición	Prompt interpretado correctamente.
Acciones	1. Elaborar texto de respuesta. 2. Validar el contenido semántico. 3. Guardar en memoria la respuesta. 4. CU-5.1 Mostrar texto en GUI
Postcondición	Texto listo para ser vocalizado.
Excepciones	Fallo en la generación por parte del LLM.
Importancia	Alta

Tabla B.22: CU-04.1 Generar respuesta textual.

CU-04.2	**Transformar texto en audio**
Versión	1.0
Autor	Víctor Manuel Martínez Garcia
Requisitos asociados	RF-04
Descripción	Se sintetiza la voz a partir del texto generado usando TTS.
Precondición	Texto válido generado por el asistente.
Acciones	1. Enviar texto al modelo TTS. 2. Recibir archivo de audio.
Postcondición	Audio generado correctamente.
Excepciones	Error de red, respuesta vacía, error del modelo.
Importancia	Alta

Tabla B.23: CU-04.2 Transformar texto en audio.

CU-04.3	**Reproducir respuesta por altavoz**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-04
Descripción	El sistema reproduce el audio resultante por la salida de sonido del dispositivo.
Precondición	Audio generado correctamente.
Acciones	1. Reproducir audio por altavoz. 2. Finalizar emisión.
Postcondición	El usuario ha escuchado la respuesta.
Excepciones	Fallo de hardware, volumen nulo.
Importancia	Alta

Tabla B.24: CU-04.3 Reproducir respuesta por altavoz.

CU-05	**Mostrar interfaz accesible**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-05, RF-06
Descripción	El sistema proporciona una interfaz visual clara y comprensible para el usuario.
Precondición	El sistema está ejecutándose y en espera de interacción.
Acciones	1. CU-05.1 Presentar resultado del asistente. 2. CU-05.2 Mostrar pensamiento y razonamiento.
Postcondición	El usuario puede interpretar y actuar sobre la información.
Excepciones	Error en el renderizado, datos incompletos.
Importancia	Media

Tabla B.25: CU-05 Mostrar interfaz accesible.

CU-05.1	**Presentar resultado del asistente**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-05
Descripción	Se muestra el resultado final del razonamiento del asistente al usuario.
Precondición	El asistente ha finalizado su tarea.
Acciones	1. Mostrar resultado textual o gráfico. 2. Permitir al usuario leer o interactuar con la salida.
Postcondición	El usuario ve la respuesta generada.
Excepciones	Interfaz no responde, error de diseño.
Importancia	Media

Tabla B.26: CU-05.1 Presentar resultado del asistente.

CU-05.2	**Mostrar pensamientos**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-06
Descripción	El sistema muestra los pasos o razonamientos internos que ha seguido el asistente.
Precondición	El asistente ha generado una trazabilidad lógica.
Acciones	1. Mostrar la cadena de razonamiento. 2. Ofrecer seguimiento del proceso paso a paso.
Postcondición	El usuario comprende cómo se ha llegado al resultado.
Excepciones	Datos parciales o razonamiento inconsistente.
Importancia	Media

Tabla B.27: CU-05.2 Mostrar pensamientos

CU-06	**Buscar información en internet**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-03, RF-06
Descripción	El sistema consulta información externa en internet cuando el agente lo considera necesario.
Precondición	El CUA determina que necesita apoyo externo para responder.
Acciones	1. CU-06.1 Enviar prompt de búsqueda a Browser-Use. 2. CU-06.2 Realizar navegación y recuperación de datos. 3. CU-06.3 Devolver información estructurada al CUA.
Postcondición	Información externa recibida por el agente.
Excepciones	Timeout, error en navegación, resultados no encontrados.
Importancia	Media

Tabla B.28: CU-06 Buscar información en internet.

CU-06.1	**Enviar prompt de búsqueda a Browser-Use**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-03
Descripción	El CUA emite un prompt a Browser-Use para consultar información externa.
Precondición	Prompt analizado y marcado como necesidad de búsqueda.
Acciones	1. Formatear la consulta. 2. Enviar la petición al agente Browser-Use.
Postcondición	Petición correctamente transmitida.
Excepciones	Fallo de comunicación o formato inválido.
Importancia	Media

Tabla B.29: CU-06.1 Enviar prompt de búsqueda a Browser-Use.

CU-06.2	**Realizar navegación y recuperación de datos**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-03
Descripción	Browser-Use accede a internet, realiza la búsqueda y filtra resultados.
Precondición	Prompt de búsqueda recibido correctamente.
Acciones	1. Ejecutar motor de búsqueda o navegación. 2. Analizar y extraer contenido útil.
Postcondición	Información externa lista para envío.
Excepciones	Web inaccesible, datos irrelevantes, timeout.
Importancia	Media

Tabla B.30: CU-06.2 Realizar navegación y recuperación de datos.

CU-06.3	**Devolver información estructurada al CUA**
Versión	1.0
Autor	Víctor Manuel Martínez García
Requisitos asociados	RF-06
Descripción	Browser-Use devuelve los resultados procesados al agente para ser utilizados en la respuesta final.
Precondición	Información obtenida correctamente.
Acciones	1. Estructurar respuesta (resumen, URL, etc.). 2. Enviar respuesta al agente CUA.
Postcondición	Información integrada al razonamiento del asistente.
Excepciones	Error en el parseo de datos o en la transmisión.
Importancia	Media

Tabla B.31: CU-06.3 Devolver información estructurada al CUA.

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado detallaremos la arquitectura del proyecto centrandonos en aspectos fundamentales para el desarrollo de **VACA**. Se describira como se organizan los datos que lo componen, su diseño procedimental y la interconexión entre los objetos que conforman el sistema, con el objetivo de dar una vision técnica y detallada que justifique las decisiones que he adoptado durante la fase de desarrollo.

C.2. Diseño de datos

La aplicación **VACA** gestiona principalmente estos tipos de datos:

- **Prompts de voz(Input):** capturados por un microfono porteriormente tratados y transformados a un formato `.wav`
- **Prompts en text(Input):** tratados directamente saltandose el paso de tratamiento de voz.
- **Representación textual gráfica:** tratado por los modelos YOLO y FLORENCE, capturados directamente del entorno gráfico del usuario.
- **Generación de logs:** Para mantenimiento del programa en caso de fallos guardados directamente en el sistema local del usuario.

La gran mayoria de datos se almacenan temporalmente en buffers y no persisten por motivos de privacidad, hay otros datos que se guardan en el

propio sistema local para su tratamiento y posterior envio al servidor de ITCL.

Como se menciono con anterioridad el ITCL no guarda ni trata los datos que se reciben desde sus endpoints, ademas de ser datos volatiles, dado que los contenedores se eliminan una vez no esten en funcionamiento.

C.3. Diseño arquitectónico

El proyecto está diseñado para ser modular, facilitando la sustitución de componentes (por ejemplo, usar otro modelo STT o TTS) sin alterar el núcleo del sistema.

A continuación detallare el UML del sistema:

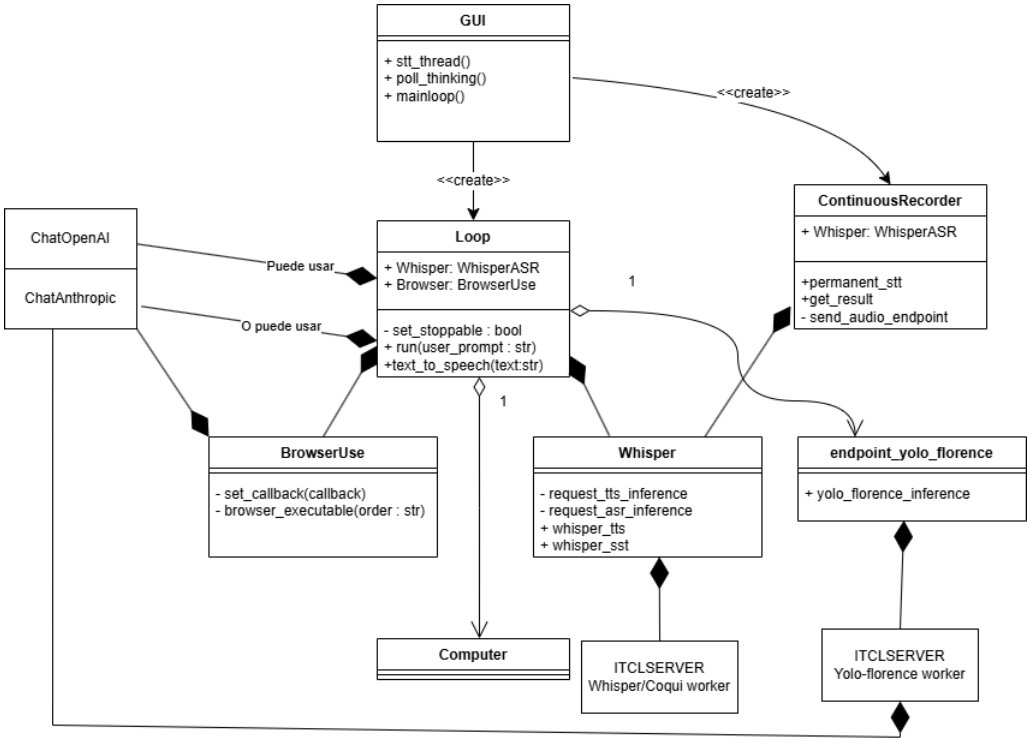

Figura C.1: Diagrama de clases UML

Explicación de clases

- **GUI:** Representa la interfaz principal del programa, crea el Loop y hace uso de `ContinuousRecorder` para realizar inferencia continua sobre el audio recibido.

- **Loop:** La clase central del agente. `Loop` está compuesta por `BrowserUse` y `Whisper`, ya que necesita de ambas para cumplir su funcionalidad principal.

  También incorpora una composición con las librerías de LangChain (`ChatOpenAI` o `ChatAnthropic`), siendo estas imprescindibles para el funcionamiento del sistema, aunque solo una puede estar activa en tiempo de ejecución.

  Adicionalmente, mantiene una dependencia con `endpoint-yolo-florence`, que permite la comprensión del contexto visual del entorno. Por último, `Loop` presenta una relación de agregación con el resto de herramientas de `Computer`, como escritura, movimiento de ratón, etc.

- **Whisper:** La clase `Whisper` tiene una composicion hacia `ContinuousRecorder` y `Loop`, ya que ambas dependen de ella para ejecutar funciones vitales. Estas funciones incluyen la transcripción continua del audio (STT) y la generación de archivos `.wav` con las respuestas del agente (TTS).

  Si bien la parte de TTS podría desacoplarse y gestionarse directamente desde `Whisper`, se ha optado por mantener esta responsabilidad dentro de `Loop`, dado que actúa como el núcleo del agente.

  Aunque esta implementación genera una composición fuerte, `Whisper` ha sido diseñada para realizar peticiones a un servidor externo del ITCL. Esto permite una cierta flexibilidad: para cambiar el modelo de inferencia (TTS/STT), solo sería necesario modificar el endpoint correspondiente dentro de `Whisper`, siempre y cuando el nuevo modelo siga utilizando los mismos formatos de entrada y salida.

- **BrowserUse:** Aunque no es estrictamente necesaria para el funcionamiento del agente, ya que algunas herramientas internas de `Computer` pueden cubrir parcialmente sus funciones, se ha optado por incluirla mediante composición dentro de `Loop`, ya que mejora considerablemente la precisión en las tareas de búsqueda.

  Cabe mencionar que `BrowserUse` depende completamente de `ChatAnthropic` para su funcionamiento, por lo que no puede operar sin esta.

- **ContinuousRecorder:** Aunque `GUI` no está compuesta directamente por esta clase ,debido a que existen funciones alternativas (actualmente desactivadas) para la transcripción, `ContinuousRecorder` representa un módulo importante dentro de la interfaz.

  Es responsable de la captura de audio y su conversión continua a texto sin interrupciones. Esta clase depende totalmente de `Whisper`, ya que sin ella no puede realizar las transcripciones.

- **endpoint-yolo-florence:** Al igual que `ContinuousRecorder`, este componente es esencial para ciertas funcionalidades de `Loop`, particularmente para la comprensión del contexto visual.

  No obstante, el agente puede seguir funcionando parcialmente sin él, motivo por el cual se ha modelado como una agregación y no como una composición directa desde `Loop`.

- **Computer:** Incluye herramientas auxiliares utilizadas por `Loop`, tales como el movimiento del ratón, escritura mediante teclado, y otras funcionalidades relacionadas con el control del sistema operativo.

- **ChatOpenAI / ChatAnthropic:** Son las interfaces que permiten interactuar con modelos de lenguaje a través de LangChain. Constituyen una parte fundamental del sistema, ya que sin ellas el agente no podría generar respuestas ni mantener una conversación.

- **ITCLSERVER Whisper / Coqui:** Corresponden a los modelos de inferencia TTS/STT alojados en los servidores del ITCL. Son utilizados por la clase `Whisper` para procesar el audio recibido (transcripción) o generar audio a partir de texto.

- **ITCLSERVER YOLO / Florence:** Contienen los modelos de visión por computador encargados de detectar objetos y generar una representación gráfica del entorno. La lógica actual de estos modelos está acoplada a `ChatAnthropic` para obtener descripciones semánticas, aunque esta parte podría desacoplarse fácilmente y ubicarse directamente dentro de `endpoint-yolo-florence`.

  Esto permitiría que el endpoint devuelva un diccionario de índices, coordenadas, descripciones y una imagen procesada (ƨoleada"), facilitando así una mayor modularidad. Sin embargo, por motivos de simplicidad y comodidad, se ha mantenido la lógica actual en el worker.

Diseño centrado en la clase Loop

Durante el desarrollo del agente, se tomó la decisión de estructurar el sistema de forma que la clase `Loop` actuase como el núcleo central de la aplicación. Esta elección responde a la necesidad de contar con un componente que coordinara de forma ordenada todas las funcionalidades críticas del sistema: procesamiento de lenguaje natural, transformacion de audio a texto, percepción visual del entorno, interacción con herramientas del sistema operativo y uso de fuentes externas como navegadores.

`Loop` se encarga de realizar la ejecución del agente, controlando cuándo y cómo deben actuar los distintos módulos. Por esta razón, se le asignaron relaciones de composición con elementos esenciales como `Whisper` (para STT y TTS), `BrowserUse` (búsqueda en la web), y las librerías de LangChain (`ChatOpenAI` o `ChatAnthropic`), ya que su funcionamiento correcto depende directamente de estos para llevar a cabo su lógica y función principal.

Adicionalmente, `Loop` mantiene relaciones de agregación con componentes más secundarios, como los módulos de visión (`endpoint yolo-florence`) o las herramientas de `Computer` (teclado, ratón, etc.), permitiendo que estas puedan activadas o desactivadas sin comprometer la integridad del sistema.

Aunque esta centralización puede dar lugar a una clase con muchas responsabilidades, lo que en ingeniería del software se conoce como un "God Object", en este caso se justifica plenamente por el contexto del proyecto. Al tratarse de un agente que debe operar de manera autónoma y reactiva, resulta conveniente contar con un único punto de entrada que mantenga el control y secuenciación de los diferentes módulos.

Además, este diseño permite una clara separación entre el núcleo lógico del agente y sus módulos funcionales. Cada clase ha sido diseñada para tener responsabilidades bien delimitadas, permitiendo que su lógica interna se mantenga lo más independiente posible. Esta modularidad facilita la sustitución de componentes individuales (por ejemplo, cambiar el modelo de lenguaje o el motor de transcripción) sin necesidad de alterar el diseño general del sistema.

Conclusion

He priorizado un enfoque que combina centralización lógica en `Loop` con una arquitectura modular, en la que cada componente mantiene un alto grado de cohesión. Esta estructura no solo facilita el mantenimiento y la escalabilidad futura del proyecto, sino que también hace más clara su comprensión y evaluación.

C.4. Diseño procedimental

En este apartado describire la lógica del flujo de trabajo del sistema relacionado con las tareas principales de la aplicación.

1. La clase `GUI` inicia la ejecución del programa y lanza un bucle continuo gestionado por la clase `Loop` para inicializar el CUA.
2. `Loop` activa el contenedor de `Whisper`, que emplea `ContinuousRecorder` para comenzar la captura y transcripción continua de audio.
3. Cuando se detecta un prompt de voz, se valida la entrada y se encapsula como un prompt. En caso de hacer una introducción por teclado simplemente habría que escribir el prompt y enviarlo.
4. El prompt es analizado por el modelo LLM (a través de `ChatOpenAI` o `ChatAnthropic`) para extraer la intención del usuario.
5. Si el LLM determina que necesita información visual, se activa el endpoint `yolo-florence`, que genera una descripción semántica del entorno gráfico.
6. Si la tarea requiere datos externos, `BrowserUse` lanza una búsqueda web y filtra los resultados obtenidos.
7. Una vez recolectada toda la información necesaria (contexto visual, datos externos, historial, etc.), el agente genera una respuesta textual, que se muestra al usuario a través de la interfaz `GUI`.
8. En caso de ser necesario para cumplir la intención del usuario, el agente utilizaría herramientas incorporadas para conseguir el objetivo descrito por el usuario.
9. Opcionalmente, la respuesta puede ser vocalizada. Para ello, `Loop` reutiliza `Whisper` para convertir el texto en audio mediante TTS, el cual se reproduce automáticamente.
10. Finalmente, el agente queda de nuevo en espera de una nueva interacción, reiniciando el ciclo.

A continuación se mostrara el diagrama de secuencia de todas las tareas mencionadas anteriormente.

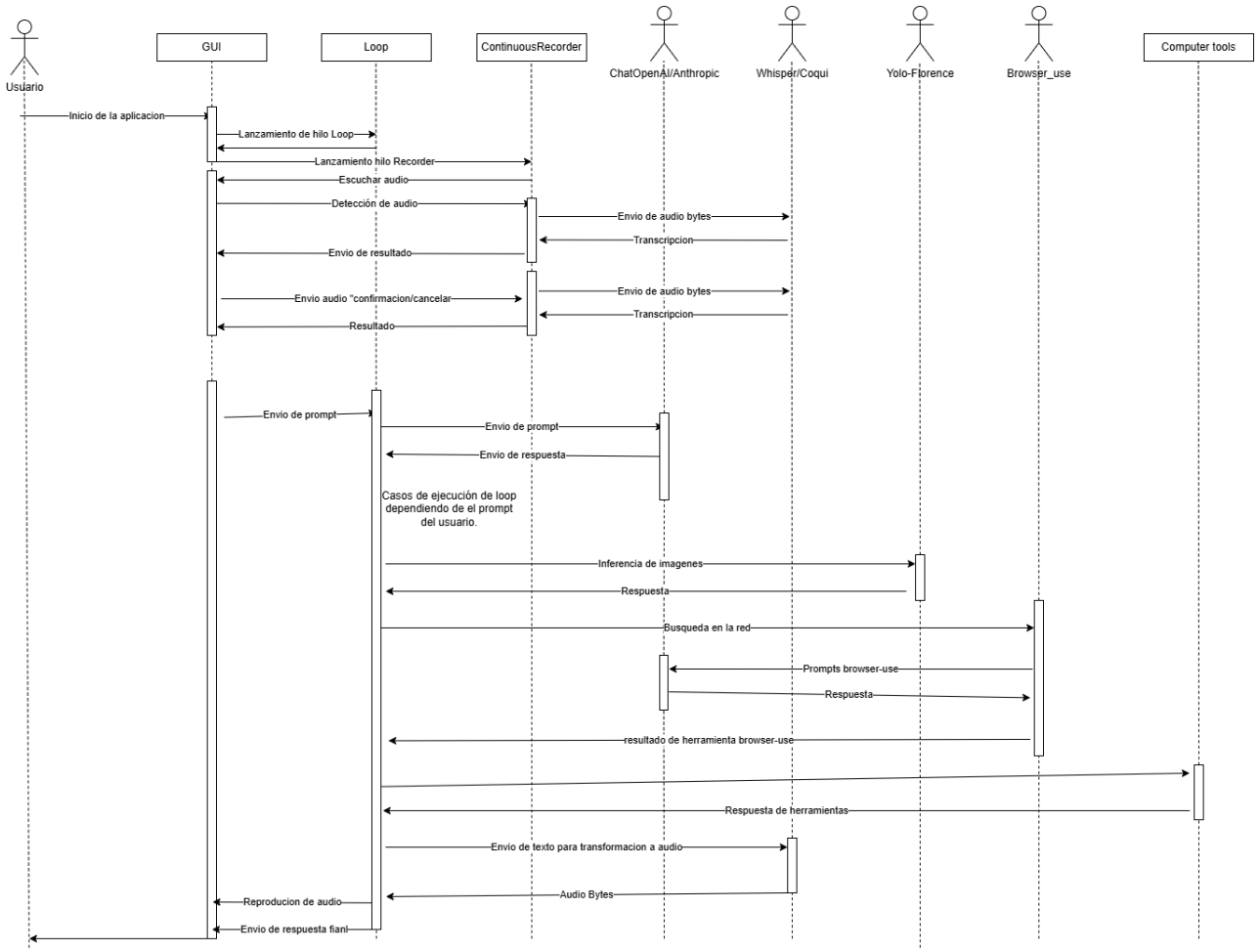

Figura C.2: Diagrama de secuencia de todas las tareas.

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Datos tratados

D.3. Estructura de directorios

- **Raíz del proyecto (VACA/)**
  - `main_loop.py` – Bucle principal del agente
  - `gui.py` – Interfaz gráfica en CTkinter
  - `initializer.py` – Inicialización del entorno
  - `.env`, `pyproject.toml`, `README.md`
- **CUA/** – Módulo central del agente
  - `cfg.py`, `__init__.py`
  - **tools/**
    - `class_whisper.py`, `class_browser_use.py`, `computer.py`
    - `endpoint_yolo_florence.py` – Objetos de los modelos para instanciación y tools varias.
    - `audio_TTS.wav`, `output.wav` – Generacion de TTS y STT

        - **tmpcrops/** – Capturas temporales para detección de elementos visuales
    - **util/** – Funciones de soporte (configuración, rutas, logs)
- **assets/** – Material gráfico del proyecto
    - Diagramas: `CUA FLOW.png`, `GUI_inicial.png`, `resultado_final.png`
    - Pruebas visuales: `ejemplo_original.jpeg`, `ejemplo_yoloed.jpeg`
    - Videos demostrativos: `cua_example1.mp4`, `cua_example2.mp4`
- **tests/** – Suite de pruebas del sistema
    - `test_001_ScreenAssistant.py`, `test_002_Florence.py`, etc.
    - **resources/**
        - **generatedAudio/** – Audios generados en pruebas
        - **generatedimage/** – Imágenes procesadas por los modelos
- **logs/** – Registro de eventos y configuración
    - `CUA.util.cfg_base.log`

D.4. Manual del programador

En esta sección elaborare una "visión"técnica para que cualquier otra persona que desee comprender, mantener o mejorar la aplicación VACA.

A continuación detallare los módulos clave y las dependendias principales del proyecto:

- **Lenguaje de programación:** Python 3.12
- **Gestor de dependencias:** Poetry
- **Frameworks y librerías clave:**
    - `openai, langchain-openai, langchain-anthropic,python-dotenv`
    - `pyautogui, pillow,anthropic,langchain`
    - `opencv-python, screeninfo, ultralytics,safetensors`
    - `transformers,langchain-community,keyboard`
    - `timm,einops,langgraph,logging`

  - `pyaudio,wave,browser-use`
  - `threaded`

- **Estructura modular:**
  - Módulo `CUA/`: lógica del agente, herramientas de audio y visión.
  - `main_loop.py`: orquestador del flujo principal.
  - `gui.py`: interfaz visual del sistema.
  - `initializer.py`: carga inicial para levantar modelos del servidor del ITCL.

Instrucciones para ejecución sin modelos dockerizados

En caso de querer ejecutar la aplicación sin utilizar los modelos dockerizados para `Whisper`, `Coqui`, `Florence` o `YOLO`, se proporciona en el repositorio de GitHub el contenedor Docker correspondiente al modelo `Florence-YOLO`, facilitando así su despliegue local. Para ello, únicamente sería necesario adaptar los endpoints a la configuración local deseada.

Respecto a los modelos `Whisper` y `Coqui`, no se incluyen directamente en el proyecto debido a motivos dep privacidad. Sin embargo, ambos modelos son de código abierto y están disponibles en los siguientes enlaces:

- **Coqui XTTS-v2**[2]
- **OpenAI Whisper Large v3**[4]

Para ejecutar el sistema sin estos modelos, se recomienda desactivar el módulo de TTS (activado por defecto) y también el hilo de transcripción continua (STT) que se lanza desde la clase `GUI`. Esta configuración permitirá el uso parcial del agente sin necesidad de contar con los modelos de voz en funcionamiento, manteniendo otras funcionalidades operativas.

D.5. Compilación, instalación y ejecución del proyecto

En esta sección se describen los pasos necesarios para compilar, instalar y ejecutar correctamente la aplicación **VACA**.

Descarga del programa desde GitHub

El código del proyecto se encuentra disponible en el repositorio personal del alumno. Se recomienda descargar la última versión publicada en la sección de releases del repositorio.[3]

Instalación de los programas necesarios e inicialización

Para la correcta ejecución del proyecto, se deben seguir los siguientes pasos:

1. Instalar un entorno de desarrollo para Python. En este caso se ha utilizado Visual Studio Code, aunque puede utilizarse cualquier otro compatible.
2. Instalar Python en su versión 3.12. [6]
3. Instalar un gestor de entornos virtuales compatible con archivos `pyproject.toml`. En este proyecto se ha utilizado **Poetry**[5].
4. Crear un nuevo entorno virtual y utilizar el archivo `pyproject.toml` del repositorio para importar automáticamente todas las librerías necesarias.
5. Una vez finalizada la instalación de dependencias, se puede ejecutar el archivo `gui.py`, el cual lanza la interfaz principal del programa.

Recomendaciones de uso

Se recomienda encarecidamente utilizar un sistema con doble monitor, ya que la aplicación trabaja principalmente sobre el monitor configurado como principal, y requiere visibilidad total del entorno gráfico para su funcionamiento correcto.

D.6. Pruebas del sistema

El sistema fue sometido a diversas pruebas con el objetivo de garantizar su correcto funcionamiento en distintos contextos de uso.

Dado que **VACA** se basa en el uso de modelos de lenguaje de gran escala (LLM) y controla el entorno operativo del usuario, muchas de las pruebas fueron realizadas de forma manual. Esto se debe a que, por la naturaleza del

sistema, cualquier ejecución fuera de control podría requerir intervención humana para ser detenida o corregida.

Pruebas automatizadas

Se implementó una batería de pruebas unitarias y funcionales que puede encontrarse en la carpeta `test` del repositorio del proyecto. Estas pruebas tienen como objetivo verificar el correcto funcionamiento individual de los distintos módulos que componen el sistema.

Para ejecutar todas las pruebas, basta con utilizar el siguiente comando en el entorno del proyecto:

```
pytests master_test.txt
```

Este comando lanza la ejecución de todos los casos definidos de manera estructurada.

Adicionalmente, se valoró la posibilidad de integrar la aplicación en un entorno de evaluación tipo *benchmark* con el fin de analizar su rendimiento de forma más rigurosa. Sin embargo, debido a que se trata de un agente de recientez que funciona exclusivamente sobre el sistema operativo Windows, no existen muchas opciones disponibles.

La única alternativa identificada fue el entorno de evaluación **Windows Agent Arena** [1]. No obstante, su integración se descartó en esta fase del proyecto debido a la falta de tiempo y la complejidad técnica para su puesta en marcha, quedando como posible mejora a implementar en el futuro.

Apéndice E

Documentación de usuario

E.1. Introducción

Esta sección tiene como objetivo proporcionar a los usuarios finales una guía clara y concisa sobre el uso de la aplicación **VACA** (Voice Assisted Computer Accessibility). Se detallan los requisitos necesarios, el proceso de instalación, y un manual de uso básico para facilitar su adopción.

E.2. Requisitos de usuarios

Requisitos del sistema

Para garantizar el correcto funcionamiento de la aplicación **VACA**, es necesario cumplir con los siguientes requisitos mínimos, diferenciados entre componentes de software y hardware.

Requisitos de software

- **Sistema operativo:** Microsoft Windows 10 o superior.
- **Versión de Python:** Python 3.12 instalado correctamente en el sistema.
- **Dependencias:** Todas las bibliotecas necesarias especificadas en el archivo `pyproject.toml`, las cuales deben instalarse mediante un gestor compatible como **Poetry**.

Requisitos de hardware

- **Micrófono:** Necesario para utilizar las funcionalidades de reconocimiento de voz (STT).
- **Altavoces o salida de audio:** Requeridos para escuchar las respuestas generadas por el sistema mediante síntesis de voz (TTS).
- **Pantallas:** Se recomienda disponer de una configuración de doble monitor. Idealmente, la pantalla principal debe estar situada en el centro y la secundaria a la derecha. Aunque no es estrictamente obligatorio, esta disposición facilita la interacción con el sistema, permitiendo que el usuario visualice el funcionamiento de **VACA** en la pantalla secundaria, mientras el programa opera sobre la principal.

E.3. Instalación

El procedimiento de instalación está pensado para usuarios con conocimientos básicos de informática. Los pasos a seguir son:

1. Descargar la última versión de la aplicación desde el repositorio oficial en GitHub: https://github.com/VictorManuelMG/TFGGII_VACA. [3]
2. Instalar **Python 3.12** desde https://www.python.org/downloads/release/python-3120/.[6]
3. Instalar **Poetry** como gestor de entornos virtuales, siguiendo la documentación oficial: https://python-poetry.org/docs/.[5]
4. Definir en `.env.example` las APIs necesarias.
5. Abrir una terminal en la carpeta del proyecto y ejecutar el comando `poetry install` para instalar las dependencias.
6. Iniciar la aplicación ejecutando el archivo `gui.py` con el comando `poetry run python gui.py`.

E.4. Manual del usuario

El uso de la aplicación **VACA** ha sido diseñado para ser intuitivo. Todas las operaciones se realizan desde la interfaz gráfica integrada que se muestra a continuación:

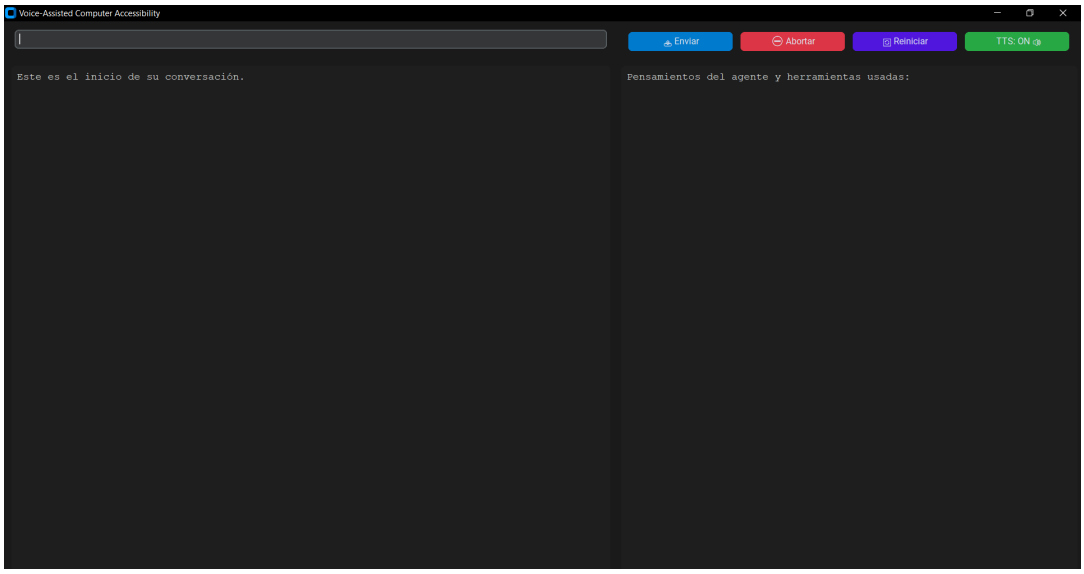

Figura E.1: Interfaz de usuario.

Como puede observarse, la interfaz cuenta con un cuadro de entrada de texto y varios botones con distintas acciones situados en los laterales. El sistema admite entrada de comandos tanto por texto como por voz.

Entrada por voz

Para utilizar esta funcionalidad, basta con hablar al micrófono tras iniciar el programa. Una vez que el comando de voz ha sido capturado y confirmado, el sistema procede automáticamente a procesar la petición del usuario.

Como se muestra en la imagen anterior, tras procesar el prompt, el sistema devuelve una respuesta visual (y opcionalmente también por voz, si el TTS está activado).

Visualización de pensamientos del agente

Además de los resultados visibles de cada acción, el sistema muestra en tiempo real los **pensamientos del agente**, es decir, los razonamientos internos que realiza al ejecutar determinadas tareas, como la búsqueda de información o el análisis del entorno.

Esta funcionalidad permite al usuario comprender mejor qué está haciendo el sistema en cada momento y seguir su lógica de actuación.

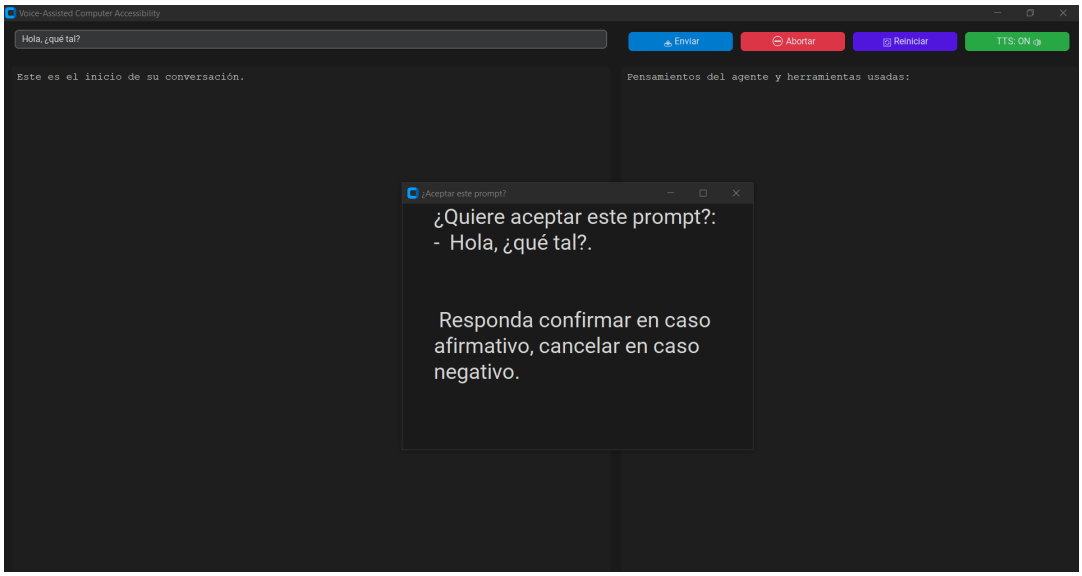

Figura E.2: Entrada de un prompt mediante voz.

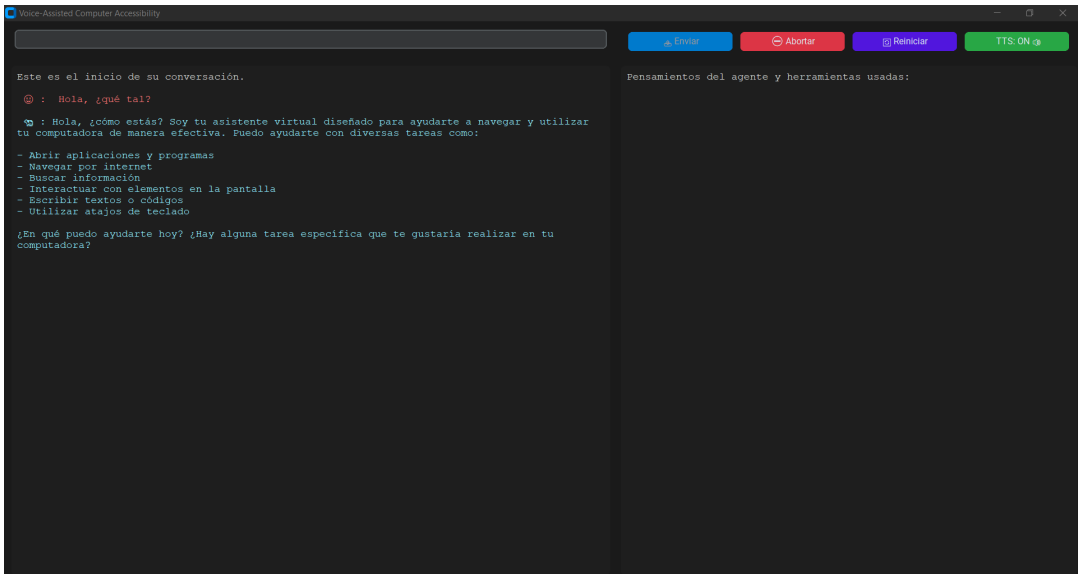

Figura E.3: Respuesta generada tras la entrada por voz.

Recomendaciones de uso

Se recomienda encarecidamente utilizar un micrófono de buena calidad para garantizar una correcta detección de voz. En caso contrario, podrían

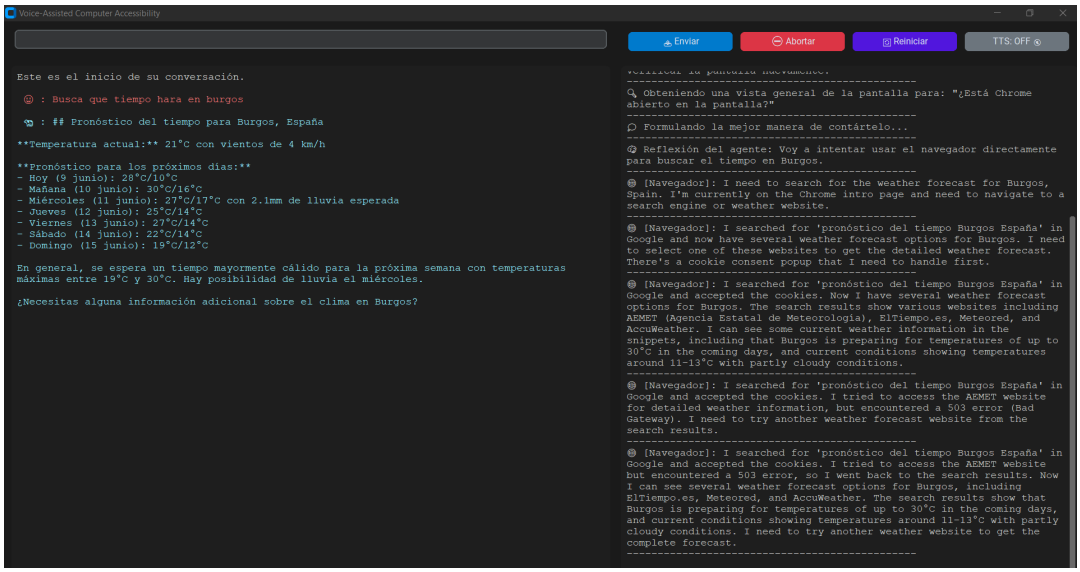

Figura E.4: Ejemplo de razonamiento mostrado durante una búsqueda en internet.

producirse errores de reconocimiento. Actualmente, la interfaz no incluye opciones de configuración avanzada del audio, por lo que ajustes como la sensibilidad del micrófono, la ganancia o la amplificación deberán realizarse directamente desde la configuración del sistema operativo.

En futuras versiones se contempla incluir un apartado de configuración accesible desde la propia interfaz para facilitar esta tarea al usuario.

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

Este anexo tiene como objetivo hacer una reflexión personal sobre cómo se han abordado temas de sostenibilidad a lo largo del desarrollo de este Trabajo de Fin de Grado (TFG). A través del proyecto **VACA** (Voice Assisted Computer Accessibility), he podido trabajar no solo con herramientas técnicas, sino también con ideas relacionadas con la accesibilidad, la inclusión, el uso responsable de la tecnología y el impacto que puede tener un software en el entorno y en las personas.

F.2. Contribución del TFG a la inclusión social

Desde el principio, uno de los objetivos principales del proyecto fue crear una herramienta que pudiera ayudar a personas con movilidad reducida a interactuar con su ordenador de forma más sencilla. **VACA** permite controlar el equipo mediante comandos de voz, algo que puede marcar la diferencia para muchos usuarios que no pueden utilizar el teclado o el ratón con normalidad.

Esto conecta directamente con la idea de reducir desigualdades, tal como plantea el Objetivo de Desarrollo Sostenible (ODS) número 10 de la ONU. Me parece importante que desde la informática se puedan proponer soluciones

que tengan un impacto positivo en la vida de la gente, especialmente en colectivos que suelen quedar fuera del foco tecnológico.

F.3. Sostenibilidad tecnológica y uso de software libre

Durante todo el desarrollo he apostado por el uso de herramientas y modelos de código abierto como Whisper, Coqui o Florence. Además de ser gratuitos, estos recursos están disponibles para toda la comunidad, lo que permite que otras personas puedan aprovecharlos, mejorarlos o adaptarlos a sus propios proyectos.

Trabajar con software libre me ha hecho más consciente del valor de compartir el conocimiento, y también de cómo esto ayuda a reducir barreras económicas. No depender de soluciones cerradas o de pago también puede ser una forma de hacer la tecnología más accesible para más gente, y eso encaja perfectamente con la idea de sostenibilidad social y económica.

F.4. Impacto ambiental y decisiones técnicas

Aunque muchas veces no lo pensemos, el software también tiene un impacto ambiental. Elegir si un modelo se ejecuta en local o en la nube puede influir en el consumo energético. En mi caso, he intentado que la mayoría de procesos puedan hacerse de forma local, sin necesidad de servidores externos. Esto no solo hace que el programa sea más rápido y autónomo, sino que también evita depender de infraestructuras que consumen muchos recursos.

También he intentado que el sistema no esté constantemente ejecutando procesos pesados si no son necesarios, lo que ayuda a optimizar el rendimiento y reducir el gasto de energía.

F.5. Competencias de sostenibilidad adquiridas

Durante el proyecto he aprendido muchas cosas, no solo técnicas, sino también relacionadas con la sostenibilidad. Por ejemplo, ahora tengo más claro que pensar en la accesibilidad no es solo un añadido, sino algo que debería formar parte del diseño desde el principio.

También he desarrollado una mirada más crítica a la hora de tomar decisiones técnicas. Ya que no se trata solo de incluir modelos sin gestión propia, sino también de que implicaría usar dicho modelo, si es abierto, si deja huella en internet, etc.

Por último, he podido ver cómo la informática puede ser una herramienta útil para cambiar cosas en el mundo real. A veces damos por hecho que los proyectos técnicos solo sirven para resolver problemas funcionales, pero también pueden tener un impacto social muy potente.

F.6. Conclusión

Este TFG me ha ayudado a tomar conciencia de cómo el desarrollo de software puede (y debería) hacerse teniendo en cuenta aspectos éticos y sostenibles. A lo largo del trabajo he intentado aplicar este enfoque en lo que estaba a mi alcance, y me ha motivado a seguir haciéndolo en el futuro.

Mi intención es seguir trabajando con esta mentalidad en los proyectos que vengan, combinando lo técnico con lo humano. Creo que la informática tiene un gran potencial para mejorar la vida de las personas, y que es responsabilidad nuestra como desarrolladores usar ese potencial de forma responsable y consciente.

En definitiva, este proyecto ha sido mucho más que una práctica técnica: me ha permitido abrir los ojos a un enfoque más completo y comprometido con la sociedad.

Bibliografía

[1] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows Agent Arena: Evaluating Multi-Modal OS Agents at Scale. *arXiv preprint arXiv:2409.08264*, 2024. Submitted 12 Sep 2024; revised 13 Sep 2024.

[2] Coqui. XTTS-v2 — Hugging Face. https://huggingface.co/coqui/XTTS-v2, 2024. [Accessed 3-06-2025].

[3] Víctor Manuel Martínez García. VACA - voice assisted computer accessibility. https://github.com/VictorManuelMG/TFGGII_VACA, 2025. [Accessed 3-06-2025].

[4] OpenAI. Whisper-large-v3 — Hugging Face. https://huggingface.co/openai/whisper-large-v3, 2024. [Accessed 3-06-2025].

[5] Python Poetry Project. Poetry documentation — python-poetry.org. https://python-poetry.org/docs/, 2024. [Accessed 3-06-2025].

[6] Python Software Foundation. Python 3.12.0 release — python.org. https://www.python.org/downloads/release/python-3120/, 2023. [Accessed 3-06-2025].