ISSN: 1692-7257 - Volumen 2 – Número 46 - 2025

RCTA
Revista Colombiana de Tecnologías de Avanzada
UNIPAMPLONA

Digital Object Identifier: 10.24054/rcta.v2i46.4126

Sistema de modelado UML controlado por voz para estudiantes con discapacidad visual en la formación en Ingeniería de Software

Voice Driven UML Modeling System for Visually Impaired Students in Software Engineering Education

PhD. Carlos Henriquez Miranda[1], Ing. Malak Andres Sanchez Cataño[1], PhD. German Sanchez-Torres[1]

[1] ***Universidad del Magdalena,*** *Facultad de Ingeniería, Grupo de Investigación y Desarrollo en Sistemas y Computación, Santa Marta, Magdalena, Colombia.*

Correspondencia: *chenriquezm@unimagdalena.edu.co*

Recibido: *25 marzo 2025.* ***Aceptado:*** *29 julio 2025.* ***Publicado:*** *06 agosto 2025.*

Cómo citar: *C. Henriquez Miranda, M. A. Sanchez Cataño, y G. Sanchez-Torres, «Sistema de modelado UML controlado por voz para estudiantes con discapacidad visual en la formación en Ingeniería de Software», RCTA, vol. 2, n.º 46, pp. 171–180, ago. 2025. Recuperado de https://ojs.unipamplona.edu.co/index.php/rcta/article/view/4126*

Resumen: La discapacidad visual limita el acceso de los estudiantes a entornos de modelado orientado a objetos que dependen de interfaces gráficas. El avance en las tecnologías Text-to-Speech (TTS) y Speech-to-Text (STT) permite pensar en mecanismos para compensar esta barrera, pero su adopción en plataformas educativas carece de directrices basadas en evidencia. Este trabajo se orienta hacia diseñar, implementar y validar un sistema de modelado UML controlado por voz que permita a estudiantes con discapacidad visual crear, editar y consultar diagramas de clases y de casos de uso. Se siguió una metodología de cuatro fases: (i) levantamiento de requisitos; (ii) diseño arquitectónico modular con validación gramatical y uso de LLMs; (iii) implementación en Python 3.11, FastAPI y servicios de STT/TTS en la nube; (iv) evaluación técnica empleando métricas de precisión de reconocimiento, latencia y tiempos de tarea. El prototipo ejecutó 20 comandos críticos con una precisión de reconocimiento del 97 % y mantuvo coherencia sintáctica total en los modelos UML generados. El tiempo medio para completar tareas de creación, edición y navegación fue de 4,23 s, 6,78 s y 5,24 s, respectivamente. La latencia promedio de TTS (1 468 ms) superó el objetivo de 500 ms, identificando al módulo PLN como principal cuello de botella. El sistema demuestra viabilidad técnica y sigue lineamientos de accesibilidad definidos (WCAG 2.2). Las mejoras futuras se centrarán en reducir la latencia TTS, ampliar el repertorio de comandos y realizar evaluaciones de usabilidad (SUS) a gran escala.

Palabras clave: accesibilidad, síntesis de voz, modelado UML.

Abstract: Visual impairment limits students' access to object-oriented modeling environments that rely on graphical interfaces. Advances in Text-to-Speech (TTS) and Speech-to-Text (STT) technologies make it possible to consider mechanisms to compensate for this barrier, yet their adoption in educational platforms lacks evidence-based guidelines.

This work aims to design, implement, and validate a voice-controlled UML modeling system that enables visually impaired students to create, edit, and query class and use-case diagrams. A four-phase methodology was followed: (i) requirements elicitation; (ii) modular architectural design with grammatical validation and the use of large language models (LLMs); (iii) implementation in Python 3.11, FastAPI, and cloud-based STT/TTS services; and (iv) technical evaluation using recognition accuracy, latency, and task-time metrics. The prototype executed 20 critical commands with a recognition accuracy of 97 % and maintained full syntactic coherence in the UML models generated. The average times to complete creation, editing, and navigation tasks were 4.23 s, 6.78 s, and 5.24 s, respectively. The average TTS latency (1,468 ms) exceeded the 500 ms target, identifying the NLP module as the main bottleneck. The system demonstrates technical feasibility and adheres to the defined accessibility guidelines (WCAG 2.2). Future improvements will focus on reducing TTS latency, expanding the command repertoire, and conducting large-scale usability evaluations (SUS).

Keywords: accessibility, speech synthesis, UML modeling.

1. INTRODUCCIÓN

La discapacidad visual constituye una barrera significativa para la inclusión educativa y el aprendizaje autónomo en la actualidad. De acuerdo con la Organización Mundial de la Salud, más de mil millones de personas presentan algún grado de deficiencia visual [1], lo que restringe su acceso a textos impresos, interfaces gráficas y entornos pedagógicos convencionales.

En este escenario, las tecnologías de asistencia auditiva, y en particular la síntesis de voz (Text-to-Speech, TTS), emergen como un recurso clave para convertir contenidos textuales en información audible, favoreciendo la accesibilidad cognitiva y la autonomía de los estudiantes con discapacidad visual. La creciente adopción de plataformas de educación en línea y entornos de aprendizaje remoto aumenta la necesidad de mecanismos de retroalimentación vocal interactiva—lectura de textos, cuestionarios y exámenes—para garantizar la igualdad de oportunidades formativas.

Aunque la retroalimentación por voz está bien establecida en aplicaciones de navegación para movilidad independiente, su integración en plataformas educativas requiere un análisis exhaustivo de las dinámicas de interacción, la sinergia multisensorial y la medición de los resultados pedagógicos. Además, marcos regulatorios como las *Web Content Accessibility Guidelines* (WCAG) [2], exigen no solo la incorporación de funciones TTS, sino su coherencia en flujos pedagógicos e interfaces interactivas; sin embargo, carecen de directrices basadas en evidencia empírica sobre su aplicación eficaz.

Las investigaciones recientes sobre tecnologías interactivas para usuarios con discapacidad visual muestran avances técnicos y de interacción valiosa, pero son débiles en establecer un enfoque integral que articule diseños, arquitecturas y métricas de evaluación.

Además, en algunos contextos específicos desarrollos en esta dirección significativos son: en movilidad, los dispositivos portátiles combinan retroalimentación auditiva con sensores de ultrasonido e infrarrojos, con buenos resultados pero con debilidades en estudios sobre su seguridad en entornos reales [1].

En el ámbito de la educación musical remota, las plataformas de *Networked Music Performance* priorizan la baja latencia y la alta fidelidad sonora para estudiantes con discapacidades múltiples, pero omiten esquemas TTS que faciliten retroalimentación textual en tiempo real [3]. Asimismo, en el aprendizaje de texturas virtuales se han integrado estímulos vibrotáctiles y síntesis de voz para la clasificación de materiales, mostrando potencial para el reconocimiento no visual pero limitándose a dominios no textuales [4].

A pesar de los aportes señalados, aún persisten debilidades. No existen comparaciones exhaustivas de criterios de usabilidad ni evaluaciones de la efectividad en el rendimiento de aprendizaje. Sin un marco comparativo, Los desarrolladores y los profesores, carecen de criterios objetivos para elegir y ajustar tecnologías TTS que optimicen tanto la experiencia del usuario como los resultados académicos [5].

En este contexto, el presente trabajo propone un sistema de modelado UML activado por voz, diseñado específicamente para estudiantes con discapacidad visual en entornos de educación en ingeniería de software. A diferencia de enfoques generales basados únicamente en lectura de contenido, esta propuesta se orienta hacia una interacción bidireccional mediante comandos verbales que permiten crear, editar y consultar elementos de diagramas UML. La iniciativa responde a la necesidad de integrar tecnologías TTS y STT en entornos de aprendizaje, donde la retroalimentación vocal no solo convierte texto a audio, sino que participa activamente en la navegación y construcción del conocimiento técnico. Para ello, se desarrolló una metodología estructurada en cuatro fases que incluyen desde el levantamiento de requisitos hasta la validación técnica.

Este documento se estructura de la siguiente forma: en la sección 2 se describen trabajos en tres líneas diferentes. En la sección 3 se describe la metodología usada para la construcción del prototipo. En la sección 4, 5 y 6, se presentan los resultados, la discusión y las conclusiones, respectivamente.

2. ANTECEDENTES TECNOLÓGICOS

Los desarrollos recientes en tecnologías de asistencia para personas con discapacidad visual se agrupan en tres líneas complementarias.

2.1 Tecnologías de asistencia no educativas

Los dispositivos portátiles de orientación y movilidad para personas con baja visión fusionan lecturas de cámaras RGB-D/estéreo y ultrasonido, emitiendo retroalimentación auditiva o háptica, con gran precisión pero débiles en métricas de seguridad estandarizadas y validaciones en entornos dinámicos [1].

Otros trabajos exploran pantallas vibrotáctiles donde una *Convolutional Neural Network* (CNN) suaviza contornos y los traduce a patrones direccionales que aceleran la exploración [6], y *displays Braille* MEMS 16×16 acoplados a entornos háptico-auditivos que, con modelos Seq2Seq + WaveNet, reconocen caracteres con 97% de precisión en 0,5 s [7].

Las guías robóticas portátiles entrenadas mediante redes *human path prediction network* (HPPN) para estimación de trayectoria, usando una estrategia *covariance matrix adaptation evolution strategy* (CMA-ES) alcanzan trayectorias fiables con 1 507 episodios reales [8].

En interacción espacio-táctil, MapIO [9] combina mapas físicos, síntesis de voz y diálogo basado en LLM, mejorando la precisión de respuestas *System Usability Scale* (SUS), a costa de mayor latencia de inferencia. La *Force-Feedback Tablet* (20×20 cm, <1 kg) emplea un *flat thumbstick* para generar efectos hápticos (fricción, bordes, atracción) y reduce un 44 % el tiempo de exploración guiada, con pruebas limitadas a laboratorio [10]. Finalmente, el sistema VIS4ION integra *edge computing* 5G sub-6 GHz/mmWave para procesar visión de alta resolución con una latencia global <100 ms, aunque falta evaluar consumo energético y ergonomía en escenarios reales [11].

2.2 Interfaces táctiles y hápticas educativas

Se han propuesto y usado las gráficas táctiles (TG), que trasladan imágenes a relieve mediante pantallas dinámicas, actuadores vibrotáctiles y retroalimentación de fuerza para representar estructuras y curvas matemáticas [12].

En música, los retos se centran en la latencia. Plataformas de baja latencia como LOLA, JackTrip, SoundJack, JamKazam, SonoBus o FarPla, apuntan a latencias muy bajas, similares a las presenciales ($L \leq 30$ ms), pero requieren hardware dedicado y aún carecen de interfaces completamente accesibles [3]. De forma similar, un motor háptico multimodal que combina Informer y VGG16 genera señales de alta fidelidad en 45–57 ms con 93,33% de precisión al unir vibración y TTS [4].

En educación y rehabilitación, el *display Hyperbraille* [13] (30×32 taxels, 5 Hz) reporta mejoras del 33–68 % en reconocimiento de formas y del 41–61 % en memoria espacial. Herramientas como TAURIS [12] permiten explorar gráficos preetiquetados con narración contextual, superando al Braille tradicional y a los lectores de pantalla. Una taxonomía reciente mapea las soluciones de acceso y creación de contenido matemático, destacando la prevalencia de TTS y estándares como MathML y EPUB3 [14]. En la misma línea, GraficiAccessibili [15] emplea sonificación rítmica y vibración para representar funciones en tablets Android, logrando identificación correcta tras breve entrenamiento, aunque limitada a funciones sencillas. Las interfaces tangibles, p. ej., Tac-Trace [16], usan tokens 3D

rastreados por visión Trackmate para proveer audio en árabe y seguimiento docente.

Por último, la aplicación de vibraciones asimétricas en bastones blancos reduce significativamente el *Root Mean Square Error* (RMSE) del ancho de barrido durante el aprendizaje de la técnica de toque, posibilitando entrenamiento remoto y sin contacto físico con el especialista [17]. Persisten retos comunes: sincronizar háptica-audio-video, reducir ancho de banda táctil, estandarizar métricas de usabilidad y validar la transferencia de habilidades en contextos reales.

2.3 Currículo multimodal y herramientas de evaluación

Esta línea investiga entornos basados en instrucción que sincronizan audio, tacto y visualización con retroalimentación automatizada, diálogo estructurado y analíticas de aprendizaje. Sus marcos suelen basarse en ciclos investigación basada en diseño y soportan TTS/STT y lectores de pantalla para maximizar accesibilidad y personalización.

En educación superior, la evaluación entre pares digital (APD) habilita que los estudiantes valoren el trabajo de sus colegas mientras desarrollan pensamiento crítico [18].

Los marcos accesibles de investigación basada en diseño integran audio-háptica, por ejemplo, Eyeland alcanzó una satisfacción de 91.7% y mejoró el rendimiento tanto de usuarios con discapacidad visual como videntes [19].

Un sistema multilingüe de escritura accesible, basado en Random Forest (f1 = 99,8%) y un umbral de confianza > 70 %, demostró la viabilidad de incorporar evaluación automática y auditiva en APD [20]. Un sistema de exploración de grafos de naturaleza conversacional que se basó en un diálogo guiado por reglas para describir autómatas finitos superó la representación HTML estándar en un test A/B ($p < 0,05$), mientras que LLM genéricos no mantuvieron consistencia [21].

3. METODOLOGÍA

La investigación se estructuró en cuatro fases secuenciales, cada una orientada a un objetivo específico (ver Fig 1):

- En la Fase 1 se realizó el levantamiento de requisitos y la definición de escenarios de uso para estudiantes con discapacidad visual en actividades de modelado *Unified Modeling Language* (UML). A partir de entrevistas con docentes y del análisis de planes de estudio, se identificaron los artefactos UML prioritarios y los flujos de trabajo que debía soportar el prototipo.
- La Fase 2 estuvo dedicada al diseño arquitectónico del sistema. Se definió una solución modular capaz de recibir comandos por voz o texto, procesar su significado mediante reglas de gramática y servicios de Procesamiento de Lenguaje Natural (PLN), y gestionar la generación y persistencia de diagramas en formato JSON. Se establecieron protocolos de comunicación, patrones de integración y criterios de accesibilidad basados en WCAG 2.2.
- Durante la Fase 3 se implementó e integró el prototipo sobre Python 3.11. Se emplearon FastAPI para el servicio web, SQLite como almacén ligero, y adaptadores independientes para los servicios de STT, TTS y LLM. La entrega continua y las pruebas automáticas aseguraron la calidad del código y la estabilidad de las interfaces REST y WebSocket.
- En la Fase 4 se programó la evaluación del sistema con usuarios reales. Se diseñó un piloto con estudiantes con discapacidad visual para medir rendimiento, precisión en el reconocimiento de comandos y usabilidad mediante la escala SUS. Los datos obtenidos en esta fase guiarán la afinación de los componentes y los *prompts* de *Natural Lenguage Understanding* (NLU).

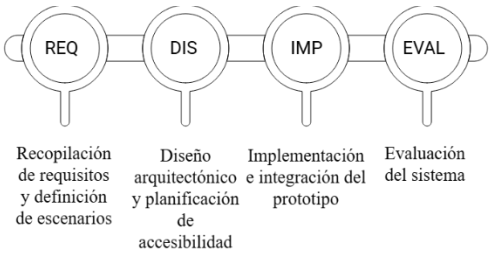

***Fig. 1**. Esquema de la metodología general aplicada.*
***Fuente:** elaboración propia.*

4. RESULTADOS Y DISCUSIONES

4.1 Levantamiento de requisitos y definición de escenarios

Durante esta fase se condujeron tres entrevistas semiestructuradas con docentes de Ingeniería de

Software, complementadas con el análisis de planes de estudio vigentes. De este análisis se concluyó que, en los cursos de modelado UML de pregrado, los diagramas de clases y de casos de uso concentran la mayoría de las actividades prácticas. Esta prevalencia está documentada tanto en estudios de *curriculum mapping* para la enseñanza de Ingeniería de Software [22], [23] como en la propia especificación UML 2.5 [24], que recomienda estos artefactos para representar respectivamente la estructura estática y los requisitos funcionales del sistema.

Sobre estos dos tipos de diagrama se definieron dos escenarios de uso principales: por un lado, la creación de un modelo desde cero durante una sesión práctica, y por otro, la edición incremental de un diagrama ya existente. Estos escenarios permitieron la definición de un grupo inicial de 41 comandos de voz organizados en nueve categorías —siete globales (por ejemplo, iniciar, guardar o deshacer), seis de navegación y consulta, y 28 específicos para operaciones CRUD (*Create*, *Read*, *Update y Remove*) en clases y casos de uso— (ver Tabla 1). El repertorio cubre instrucciones como "crea una clase Cliente", "agrega un atributo nombre a Cliente" o "vincula el actor Usuario con el caso de uso Registrar".

Para validar el *Minimum Viable Product* (MVP), se implementaron 20 de estos comandos (un 50 % del total), centrados exclusivamente en operaciones sobre diagramas de clases y de casos de uso. Este enfoque asegura que la interfaz de voz cubra las tareas críticas identificadas en el levamiento de requisitos, compatible con las recomendaciones de [25] sobre la priorización de artefactos en entornos educativos.

Con base en métricas de tiempo de respuesta y precisión reportadas en la literatura, se fijó como objetivo que cada comando se procese en menos de 2,0 s con una tasa de acierto de al menos el 90 % en el reconocimiento de voz [26]. De igual modo, se definió que el sistema debe mantener un modelo interno coherente, aplicar las reglas sintácticas de UML y exponer la estructura del diagrama en formato JSON para su persistencia y exportación. En general, la especificación establecida (ver Tabla 2):

- Requisitos funcionales:
  - Entrada multimodal (voz/texto)
  - Tiempo de procesamiento ≤ 2,0 s
  - Precisión ≥ 90 %
- No funcionales:
  - Latencia extrema a extremo < 4 s en redes académicas (< 10 Mbps)
  - Conformidad con las *Web Content Accessibility Guidelines* (WCAG 2.2)
  - Navegación y etiquetado *Accessible Rich Internet Applications* (ARIA), garantizando trazabilidad y compatibilidad con las herramientas CASE existentes.

Tabla 1: *Ejemplos de comandos CRUD del prototipo*

Comando	Categoría	Diagrama
crea una clase <Nombre>	CRUD clase	Clases
elimina la clase <Nombre>	CRUD clase	Clases
cambia nombre de clase <Viejo> a <Nuevo>	CRUD clase	Clases
agrega un atributo <Atributo> a <Clase>	CRUD clase	Clases
agrega un método <Método> a <Clase>	CRUD clase	Clases
deshacer la última acción	Control	Ambos
crea un caso de uso <Nombre>	CRUD caso uso	Casos de uso
vincula actor <Actor> con caso <Nombre>	CRUD caso uso	Casos de uso
guarda el diagrama	Global	Ambos
cierra el diagrama	Global	Ambos

Fuente: *elaboración propia*

Tabla 2: *Requisitos definidos*

ID	Requisito	Tipo	Valor objetivo
RF1	Interpretar cada comando en menos de 1,5 s	Funcional	< 1,5 s
RF2	Precisión mínima en reconocimiento de voz	Funcional	≥ 92 %
RF3	Modelo interno consistente con reglas sintácticas de UML	Funcional	Cumplimiento total
RF4	Exponer diagrama en formato JSON para persistencia/exportación	Funcional	JSON válido
RNF 1	Latencia extremo-a-extremo en redes académicas (≤ 10 Mbps)	No funcional	< 4 s
RNF 2	Conformidad con WCAG 2.2 (contraste, ARIA, navegación por teclado)	No funcional	Cumplimiento de todos los criterios
RNF 3	Retroalimentación auditiva con confirmaciones explícitas (TTS)	No funcional	Confirmación tras cada acción

Fuente: *elaboración propia*

4.2 Diseño arquitectónico

El sistema se estructuró con un patrón modular de tres capas —entrada, procesamiento semántico y generación— complementado por un módulo de presentación (ver Figura 2). Cada capa agrupa componentes con responsabilidades bien definidas, lo que facilita la trazabilidad entre requisitos y diseño [27].

En la capa de entrada se implementa un *Voice Command Interpreter* capaz de recibir tanto señales de audio (vía servicio STT) como texto escrito. En línea con la *Web Speech API* para navegadores modernos, este componente convierte el dictado en texto bruto y encola las peticiones para su análisis posterior. Para garantizar accesibilidad, todos los elementos de control se etiquetan según las WCAG y las especificaciones ARIA, e incorpora navegación por teclado.

La capa de procesamiento semántico integra dos subcomponentes. Primero, un validador de sintaxis basado en *UML Grammar Rules* (implementadas en esquema JSON) descarta estructuras malformadas y, en su caso, invoca un motor de desambiguación que genera peticiones de aclaración al usuario. Segundo, un servicio de PLN externo —implementado mediante llamadas a un LLM con *function calling*— traduce el texto validado en operaciones CRUD sobre elementos UML. Este enfoque híbrido, que combina gramáticas controladas y modelos de lenguaje, equilibra precisión y flexibilidad [28].

En la capa de generación y persistencia reside el *Diagram Orchestrator*, responsable de coordinar los cambios sobre el modelo interno y de exponerlo en formato JSON a través de una API REST. Para notificaciones en tiempo real (p. ej., actualizaciones de vista) se habilita un canal WebSocket. La persistencia emplea SQLite como almacén ligero, lo que facilita la exportación y la integración con herramientas CASE.

Por último, el módulo de presentación corre en un cliente React que consume la API REST y el canal WebSocket para renderizar diagramas SVG mediante PlantUML. Este cliente reproduce las salidas de TTS para confirmar cada acción y actualiza la interfaz sin recarga, cumpliendo requisitos de operabilidad y robustez [29].

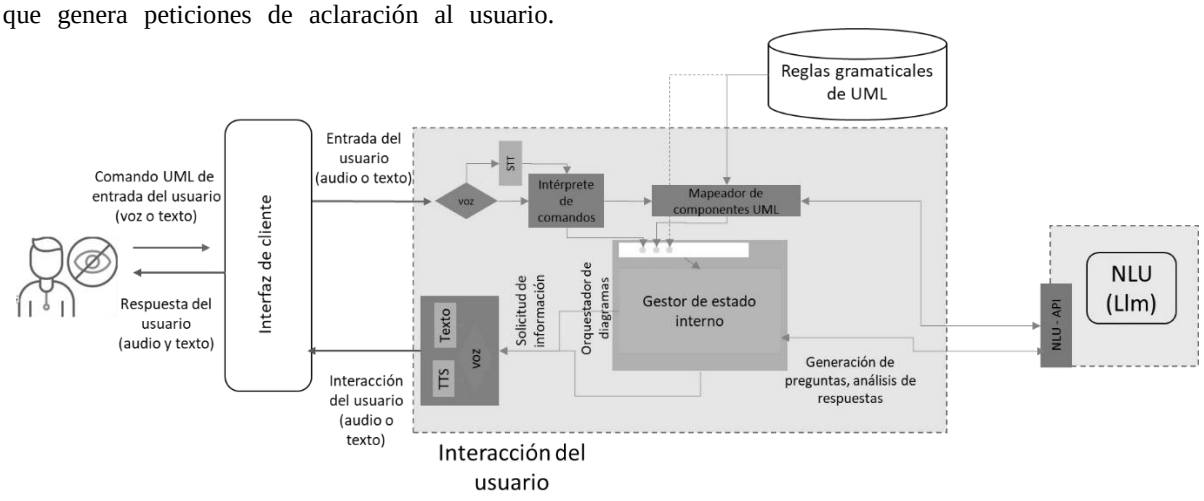

***Fig. 2**. Diagrama a bloques con tres zonas: (i) Entrada accesible —usuario dicta o escribe un comando UML; (ii) Procesamiento semántico —comando pasa por intérprete, mapeo UML y consulta a un LLM validado con reglas UML; (iii) Generación/gestión del modelo —estado interno persistido y respuestas en audio/texto devueltas al usuario.*
***Fuente:** elaboración propia.*

4.3 Implementación e integración

El prototipo se desarrolló íntegramente en Python 3.11 y se organizó en módulos que cubren desde la captura de audio hasta la generación del diagrama final. La aplicación arranca en el script principal, donde se leen las credenciales y parámetros de configuración y se crean las instancias de:

- *LLM Interface*, responsable de la comunicación con el modelo de lenguaje y de gestionar el registro y la invocación de herramientas mediante *function calling*.
- *InternalStateManager*, que mantiene el estado interno de los diagramas, registra cada operación como una herramienta y ejecuta las funciones CRUD y de exportación.

- *AudioHandler*, que delega en el módulo STT y módulo TTS la conversión entre audio y texto, y la captura de audio por micrófono.

En el ciclo principal se alterna entre modo de texto y voz; en el caso de voz, *AudioHandler* escucha la señal PCM y la convierte a WAV mediante Pydub antes de invocar el servicio de reconocimiento de Google. El texto resultante se envía a *LLMInterface*, que genera una respuesta estructurada conforme a los esquemas JSON definidos en function_schemas.py. Esta respuesta invoca la herramienta correspondiente en InternalStateManager, que actualiza el modelo interno almacenado en SQLite y devuelve un mensaje de confirmación. A continuación, AudioHandler sintetiza la respuesta en audio y la reproduce al usuario (ver Tabla 3).

La persistencia y la interoperabilidad con herramientas CASE se logran exportando el modelo interno a JSON, que transforma en un script PlantUML y procesa vía línea de comandos para producir un diagrama en PNG, SVG o PDF. La comunicación entre componentes de back-end se realiza con FastAPI (endpoints REST para operaciones discretas) y un servidor WebSocket (para notificaciones en tiempo real), apoyado en HTTPX y AnyIO para operaciones asíncronas.

Para facilitar la extensión y el mantenimiento, cada herramienta del LLM está asociada a un *schema* JSON que valida los parámetros de entrada antes de ejecutar la función. Este patrón garantiza que solo lleguen a ejecución peticiones con la forma esperada, reduciendo errores de deserialización y fortaleciendo la robustez del prototipo (Newman, 2015).

Tabla 3: *Componentes de implementación y tecnología empleada*

Módulo	Función principal	Tecnología
Captura de audio	Escucha eventos de teclado y graba audio en formato bruto	pyaudio, keyboard
Transcripción de audio (STT)	Conversión de audio a texto con API	speech_recognition, Pydub
Síntesis de voz (TTS)	Generación de audio a partir de texto con Google Cloud TTS o gTTS	google-cloud-texttospeech, gTTS, Pydub
Interpretación de comandos	Registro y validación de llamadas function calling	OpenAI API, JSON Schema
Gestión del estado interno	CRUD y exportación de diagramas, orquestación de acciones	SQLite, Python dict
Exportación de diagramas	Conversión de JSON a PlantUML y ejecución de PlantUML por línea de comandos	PlantUML, subprocess
API REST y WebSocket	Exposición de operaciones síncronas y asíncronas	FastAPI, HTTPX, AnyIO
Interfaz de usuario (cliente)	Renderizado dinámico de SVG y reproducción de TTS	React, PlantUML

Fuente: *elaboración propia*

4.4 Validación

La evaluación se centró solo en aspectos técnicos realizando tareas principales: creación de un diagrama desde cero, edición incremental y navegación en un diagrama existente. Para cuantificar la eficacia y la usabilidad del sistema se definieron las siguientes métricas:

- Tiempo en tarea (*Time on Task*): duración transcurrida desde el inicio hasta la finalización de cada tarea (en segundos). Esta métrica permite evaluar la eficiencia del flujo de trabajo y se calcula como:

$$T_{task} = t_{fin} - t_{ini} \quad (1)$$

donde t_{ini} y t_{fin} corresponden a las marcas de tiempo al comenzar y concluir la tarea, respectivamente [29].

- Precisión de reconocimiento (*Recognition Accuracy*): proporción de comandos correctamente interpretados por el sistema, definida como

$$A_{rec} = \frac{N_{correctos}}{N_{totales}} \times 100\% \quad (2)$$

donde $N_{correctos}$ es el número de comandos ejecutados sin error de reconocimiento y $N_{totales}$ el total de comandos emitidos por el usuario.

- Tasa de error (*Error Rate*): complemento de la precisión, calculada como

$$E = 100\% - A_{rec} \quad (3)$$

- Latencia TTS (*TTS Latency*): tiempo transcurrido entre la invocación de la función de síntesis y el inicio de la reproducción de audio, medido en milisegundos.

$$L_{TTS} = t_{iniRep} - t_{iniInv} \quad (4)$$

Donde, t_{iniRep} es el momento de inicio de reproducción y t_{iniInv} el tiempo de inicio d invocación.

Vea la tabla 4, con las métricas de evaluación del sistema.

Tabla 4: *Tabla de Evaluación de Métricas del Sistema*

Métrica	m	ds	Obj
Tiempo creación (s)	4.23s	±1.8	–
Tiempo edición (s)	6.78s	±1.4	–
Tiempo navegación (s)	5.24s	–	–
Precisión rec (%)	97 %	–	≥ 90 %
Tasa de error (%)	3 %	–	≤ 10 %
Latencia TTS (ms)	1468ms	±224	< 500 ms

ds: desviación estándar

Fuente: *elaboración propia*

4.5 Discusión

El análisis de los resultados permite identificar aspectos técnicos relevantes y su implicación en el dominio de usuarios con discapacidad visual:

La precisión de reconocimiento alcanzó un 97 %, superando el umbral mínimo del 90 % establecido en RF2. Este nivel de acierto indica que el motor STT y el motor de PLN, en conjunto, ofrecen una fiabilidad adecuada para la mayoría de las interacciones, reduciendo la necesidad de solicitudes de aclaración al usuario.

La latencia TTS media de 1468 ms se sitúa por encima del objetivo de 500 ms (RNF3), lo cual no permite garantizar confirmaciones auditivas ágiles. La mayor componente de costo en latencia el módulo PLN.

Los tiempos promedio para las tareas de creación (4,23 s), edición (6,78 s) y navegación (5,24 s) reflejan el flujo completo de interacción —desde el dictado hasta la actualización de la vista—. Aunque estos valores exceden la métrica de procesamiento de comando (< 1,5 s) definida en RF1, es necesario considerar que incluyen operaciones de PLN, persistencia y validación de reglas. Para usuarios con discapacidad visual, minimizar el número de pasos en el flujo de trabajo y optimizar el canal WebSocket puede reducir el tiempo total de tarea.

El cumplimiento total de las reglas sintácticas de UML (RF3) y la generación de JSON válido (RF4) confirman la solidez de la capa semántica y del orquestador de diagramas.

Accesibilidad: la adopción de WCAG 2.2 y ARIA para la navegación y el etiquetado garantiza que el sistema responda a los criterios de accesibilidad esperados. Si bien no se cuantificaron métricas de usabilidad en esta fase, la conformidad técnica sienta las bases para una evaluación de la escala SUS en futuras iteraciones.

5. CONCLUSIONES

La implementación de un modelo interno coherente y la exposición de diagramas en formato JSON aseguran la interoperabilidad con herramientas CASE y la consistencia sintáctica de UML. Aunque los tiempos de tarea globales (4–7 s) superan los umbrales de procesamiento de comando aislado, el modularidad del sistema permite enfoques de optimización centrados en el orquestador y en la reducción de operaciones de cliente.

En términos de aplicación en entornos educativos, el sistema demuestra viabilidad técnica para soportar actividades de modelado UML mediante voz, lo cual podría incrementar la autonomía y eficiencia de estudiantes con discapacidad visual.

Como trabajos futuros se propone: extender el repertorio de comandos CRUD, así como los tipos de diagramas; optimizar el canal WebSocket y la interacción con el módulo PLN en el cual se centra el mayor costo en latencia y realizar la evaluación de usabilidad con la escala SUS y ajustar los prompts de NLU según el feedback de usuarios reales.

REFERENCIAS

[1] A. D. P. D. Santos, A. H. G. Suzuki, F. O. Medola, y A. Vaezipour, «A systematic review of wearable devices for orientation and mobility of adults with visual impairment and blindness», *IEEE Access*, vol. 9, pp. 162306-162324, 2021, doi: 10.1109/ACCESS.2021.3132887.

[2] World Wide Web Consortium, «Web Content Accessibility Guidelines (WCAG) 2.2». diciembre de 2024.

[3] C. Rottondi, M. Sacchetto, L. Severi, y A. Bianco, «Toward an inclusive framework for remote musical education and practices», *IEEE Access*, vol. 12, pp. 173836-173849, 2024, doi: 10.1109/ACCESS.2024.3501414.

[4] D. Chen *et al.*, «Visually impaired people learning virtual textures through multimodal feedback combining vibrotactile and voice», *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol.

33, pp. 453-465, 2025, doi: 10.1109/TNSRE.2025.3528048.

[5] S. Raffoul y L. Jaber, «Text-to-Speech Software and Reading Comprehension: The Impact for Students with Learning Disabilities», *Canadian Journal of Learning and Technology*, vol. 49, n.º 2, Art. n.º 2, nov. 2023, doi: 10.21432/cjlt28296.

[6] D. Chen, J. Liu, L. Tian, X. Hu, y A. Song, «Research on the method of displaying the contour features of image to the visually impaired on the touch screen», *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 29, pp. 2260-2270, 2021, doi: 10.1109/TNSRE.2021.3123394.

[7] D. Chen *et al.*, «Development and evaluation of refreshable braille display and active touch-reading system for digital reading of the visually impaired», *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 32, pp. 934-945, 2024, doi: 10.1109/TNSRE.2024.3363495.

[8] H. -S. Moon y J. Seo, «Sample-efficient training of robotic guide using human path prediction network», *IEEE Access*, vol. 10, pp. 104996-105007, 2022, doi: 10.1109/ACCESS.2022.3210932.

[9] M. Manzoni, S. Mascetti, D. Ahmetovic, R. Crabb, y J. M. Coughlan, «MapIO: a gestural and conversational interface for tactile maps», *IEEE Access*, vol. 13, pp. 84038-84056, 2025, doi: 10.1109/ACCESS.2025.3566286.

[10] S. L. Gay, E. Pissaloux, K. Romeo, y N. -T. Truong, «F2T: a novel force-feedback haptic architecture delivering 2D data to visually impaired people», *IEEE Access*, vol. 9, pp. 94901-94911, 2021, doi: 10.1109/ACCESS.2021.3091441.

[11] Z. Yuan *et al.*, «Network-aware 5G edge computing for object detection: Augmenting wearables to "see" more, farther and faster», *IEEE Access*, vol. 10, pp. 29612-29632, 2022, doi: 10.1109/ACCESS.2022.3157876.

[12] M. Zeinullin y M. Hersh, «Tactile audio responsive intelligent system», *IEEE Access*, vol. 10, pp. 122074-122091, 2022, doi: 10.1109/ACCESS.2022.3223099.

[13] F. Leo, E. Cocchi, y L. Brayda, «The effect of programmable tactile displays on spatial learning skills in children and adolescents of different visual disability», *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 25, n.º 7, pp. 861-872, jul. 2017, doi: 10.1109/TNSRE.2016.2619742.

[14] P. Mejía, L. C. Martini, F. Grijalva, J. C. Larco, y J. C. Rodríguez, «A survey on mathematical software tools for visually impaired persons: a practical perspective», *IEEE Access*, vol. 9, pp. 66929-66947, 2021, doi: 10.1109/ACCESS.2021.3076306.

[15] S. Gatto, O. Gaggi, L. Grosset, y L. G. N. Fovino, «Accessible mathematics: Representation of functions through sound and touch», *IEEE Access*, vol. 12, pp. 121552-121569, 2024, doi: 10.1109/ACCESS.2024.3448509.

[16] R. Jafri, S. M. M. Althbiti, N. A. A. Alattas, A. A. A. Albraiki, y S. H. A. Almuhawwis, «Tac-trace: a tangible user interface-based solution for teaching shape concepts to visually impaired children», *IEEE Access*, vol. 10, pp. 131153-131165, 2022, doi: 10.1109/ACCESS.2022.3228455.

[17] T. Tanabe, K. Nunokawa, K. Doi, y S. Ino, «Training system for white cane technique using illusory pulling cues induced by asymmetric vibrations», *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 30, pp. 305-313, 2022, doi: 10.1109/TNSRE.2022.3148770.

[18] G. V. Helden, V. Van Der Werf, G. N. Saunders-Smits, y M. M. Specht, «The use of digital peer assessment in higher education—an umbrella review of literature», *IEEE Access*, vol. 11, pp. 22948-22960, 2023, doi: 10.1109/ACCESS.2023.3252914.

[19] K. Villalba *et al.*, «Eyeland: a visually-impaired accessible english learning application using a design-based research framework», *IEEE Access*, vol. 12, pp. 142275-142290, 2024, doi: 10.1109/ACCESS.2024.3444741.

[20] M. N. Islam *et al.*, «A multilingual handwriting learning system for visually impaired people», *IEEE Access*, vol. 12, pp. 10521-10534, 2024, doi: 10.1109/ACCESS.2024.3353781.

[21] P. F. Balestrucci, E. Di Nuovo, M. Sanguinetti, L. Anselma, C. Bernareggi, y A. Mazzei, «An educational dialogue system for visually impaired people», *IEEE Access*, vol. 12, pp. 150502-150519, 2024, doi: 10.1109/ACCESS.2024.3479883.

[22] T. C. Lethbridge, S. E. Sim, y J. Singer, «Studying Software Engineers: Data Collection Techniques for Software Field Studies», *Empir Software Eng*, vol. 10, n.º 3, pp. 311-341, jul. 2005, doi: 10.1007/s10664-005-1290-x.

[23] O. Cico, L. Jaccheri, A. Nguyen-Duc, y H. Zhang, «Exploring the intersection between software industry and Software Engineering education - A systematic mapping of

Software Engineering Trends», *Journal of Systems and Software*, vol. 172, p. 110736, feb. 2021, doi: 10.1016/j.jss.2020.110736.

[24] Object Management Group, «OMG Unified Modeling Language (OMG UML), Version 2.5.1», Object Management Group (OMG), Specification formal/17-12-05, dic. 2017. [En línea]. Disponible en: https://www.omg.org/spec/UML/2.5.1

[25] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3.ª ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[26] T.-S. Nguyen, S. Stueker, y A. Waibel, «Super-Human Performance in Online Low-latency Recognition of Conversational Speech», 26 de julio de 2021, *arXiv*: arXiv:2010.03449. doi: 10.48550/arXiv.2010.03449.

[27] «Software Architecture in Practice, 3rd Edition». Accedido: 27 de julio de 2025. [En línea]. Disponible en: https://www.sei.cmu.edu/library/software-architecture-in-practice-third-edition/

[28] S. Geng, M. Josifoski, M. Peyrard, y R. West, «Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning», 18 de enero de 2024, *arXiv*: arXiv:2305.13971. doi: 10.48550/arXiv.2305.13971.

[29] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994.