Análisis Lineal del Código: `gui_manager.py`

Este archivo, `gui_manager.py`, es el responsable de crear, gestionar y actualizar todos los elementos visuales de la aplicación. Actúa como el centro de control para el usuario, mostrando datos en tiempo real y proporcionando las herramientas para interactuar con el sistema.

El script comienza importando las librerías necesarias. `import tkinter as tk` y sus componentes `ttk`, `messagebox`, y `simpledialog` son la base para construir la interfaz. También se importa `serial.tools.list_ports` para buscar puertos de Arduino, `threading` para manejar las referencias a los hilos de fondo, y `time` para cualquier pausa necesaria.

A continuación, se realiza una importación crucial de los módulos propios del proyecto, como `constants`, `arduino_comms`, `state_machine_logic`, `db_manager` y `reporting_logging`. Esta importación se hace dentro de un bloque `try...except` para manejar posibles errores si un módulo no se encuentra. En caso de error, se definen unas clases y funciones "stub" o de reemplazo. Esto es una técnica de programación defensiva que permite que el archivo `gui_manager.py` al menos se cargue sin errores fatales, aunque con funcionalidad limitada, lo cual es útil para el desarrollo y la depuración.

La parte central del archivo es la clase `InterfazGrafica`, que hereda de `tk.Tk`, la clase principal de una ventana de Tkinter. Dentro del constructor `__init__`, se inicializan las variables de la clase. Se preparan `self.hilo_listener_ref` y `self.hilo_estados_ref` para guardar las referencias a los hilos de comunicación y de lógica. Se crea una variable especial de Tkinter, `self.uid_escaneado_para_formulario = tk.StringVar()`, que vinculará el valor de un campo de texto con una variable, permitiendo que se actualice automáticamente.

Se configura la ventana principal con un título usando `self.title(...)` y un tamaño inicial con `self.geometry(...)`. Una línea muy importante es `self.protocol("WM_DELETE_WINDOW", self.al_cerrar_ventana)`, que intercepta el evento de cierre de la ventana (al hacer clic en la 'X') y lo redirige a la función personalizada `self.al_cerrar_ventana` para asegurar un apagado limpio.

La estructura visual se organiza en pestañas usando un `ttk.Notebook`. Se crean tres pestañas principales: "Panel Principal", "Gestión de Usuarios" y "Reportes Diarios". Cada una de estas pestañas se construye llamando a funciones dedicadas como `crear_widgets_tab_principal()`, `crear_widgets_tab_gestion_usuarios()`, etc. Esto mantiene el código organizado y legible.

La función `crear_widgets_conexion` define la barra superior donde el usuario puede refrescar, seleccionar un puerto COM de una lista desplegable (`ttk.Combobox`) y conectar o desconectar el Arduino con un botón.

En `crear_widgets_tab_principal`, se añaden etiquetas (`ttk.Label`) para mostrar en tiempo real el estado del sistema, el modo de validación actual, y los valores de todos los sensores (distancias, interruptores, RFID) que provienen del Arduino.

La función `crear_widgets_tab_gestion_usuarios` es la más compleja. Construye un formulario completo para registrar o editar usuarios, con campos de texto (`ttk.Entry`) para el nombre, DNI, área, etc. Incluye un botón para "Escanear UID" que activa un modo especial para capturar el ID de

una tarjeta RFID directamente desde el lector. También muestra una tabla (`ttk.Treeview`) que lista todos los usuarios de la base de datos, con botones para editar o borrar al usuario seleccionado.

La pestaña de reportes, creada en `crear_widgets_tab_reportes_diarios`, muestra un resumen de los accesos del día y contiene dos tablas: una para los accesos exitosos y otra para los intentos fallidos, permitiendo al administrador revisar la actividad diaria.

Una vez creados todos los widgets, el constructor llama a `self.actualizar_gui_periodicamente()`, una función que se ejecutará repetidamente cada 200 milisegundos para refrescar los datos en pantalla. También llama a `self.habilitar_deshabilitar_gui_por_conexion(False)` para deshabilitar la mayoría de los controles al inicio, ya que el sistema está desconectado.

La lógica de la aplicación se implementa a través de varias funciones de acción. Por ejemplo, `accion_conectar_desconectar` es la función que se ejecuta al pulsar el botón "Conectar". Si no está conectado, obtiene el puerto seleccionado y llama a `arduino_comms.conectar_a_arduino()`. Si la conexión es exitosa, pone en marcha los hilos de fondo para la comunicación (`hilo_listener_arduino`) y la máquina de estados (`hilo_maquina_estados`), y habilita los controles de la GUI. Si se pulsa para desconectar, detiene los hilos y cierra la conexión.

La función `iniciar_escaneo_rfid_para_formulario` activa una bandera y envía un comando al Arduino para que espere una tarjeta. Cuando el Arduino envía un UID, la función `actualizar_gui_periodicamente` lo detecta y llama a `_procesar_rfid_llegado_para_formulario`, que valida si ese UID ya existe en la base de datos y lo asigna al campo del formulario.

Para la gestión de usuarios, funciones como `accion_guardar_usuario_formulario`, `accion_editar_usuario_lista` y `accion_borrar_usuario_lista` se comunican con el módulo `db_manager` para realizar operaciones de crear, leer, actualizar y borrar (CRUD) en la base de datos. Antes de guardar, se llama a `validar_formulario_usuario` para asegurarse de que los datos ingresados son correctos y no existen duplicados.

La función `actualizar_gui_periodicamente` es el corazón dinámico de la interfaz. Cada 200 milisegundos, obtiene los datos más recientes de los módulos `arduino_comms` y `state_machine_logic` y actualiza las etiquetas correspondientes. También llama a `actualizar_reportes_en_gui` para refrescar las tablas de logs.

Finalmente, la función `al_cerrar_ventana` gestiona el cierre ordenado. Si el sistema está conectado, primero ejecuta la lógica de desconexión. Luego, invoca al módulo `reporting_logging` para generar un reporte final del día y, por último, cierra la ventana de la aplicación con `self.destroy()`.