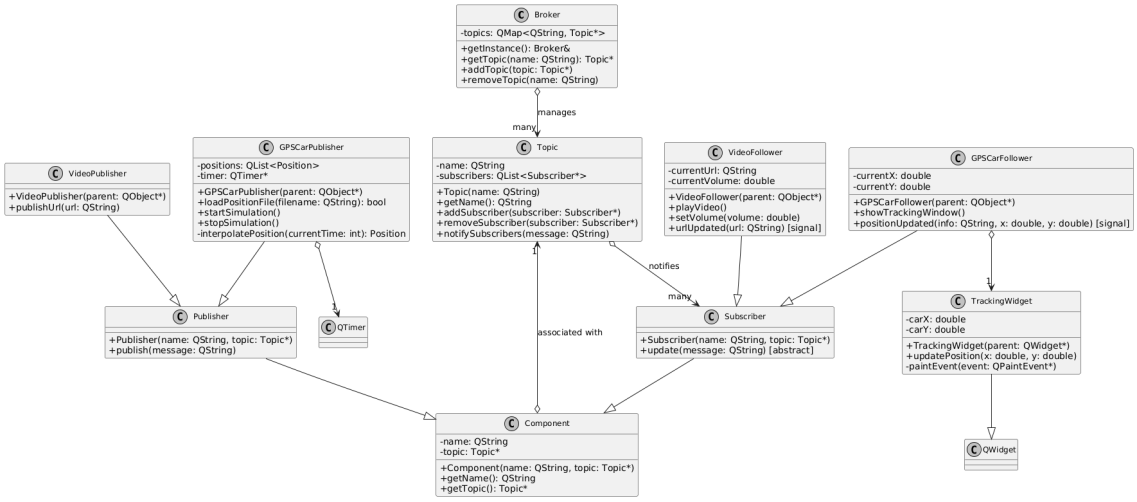
Documentación Tarea 3 ELO-329: Simulador Gráfico del Patrón Publicador-Suscriptor

Grupo 3

La implementación cumple con todos los requisitos de la etapa final (Etapa 4) descritos en el enunciado, incluyendo la funcionalidad adicional para ajustar el volumen de reproducción de videos mediante un `QSlider`.

Diagrama UML

El diagrama de clases UML correspondiente a la última etapa desarrollada se encuentra a continuación:



Explicación de la Solución

La solución implementa un simulador gráfico basado en el patrón Publicador-Suscriptor, con una interfaz gráfica dividida en dos secciones: la izquierda para los publicadores (Video y GPS) y la derecha para los suscriptores (Video y GPS). A continuación, se describe la interacción entre las clases principales durante la ejecución del programa:

1. **Inicialización del Sistema (`MainWindow`)**:
    - La clase `MainWindow` actúa como el núcleo de la aplicación, inicializando los publicadores (`VideoPublisher`, `GPSCarPublisher`) y suscriptores (`VideoFollower`, `GPSCarFollower`). Configura la interfaz gráfica utilizando `ui_mainwindow.h` (generado por Qt Designer) y conecta señales y slots para manejar eventos de usuario, como la entrada de URLs, la carga de archivos GPS y el control de la simulación.
2. **Gestión del Patrón Publicador-Suscriptor (`Broker`, `Topic`, `Component`, `Publisher`, `Subscriber`)**:

- El `Broker` (patrón Singleton) gestiona los tópicos (`Topic`) y asegura que cada tópico sea único. Los publicadores (`Publisher`) y suscriptores (`Subscriber`) heredan de `Component`, que asocia cada instancia a un tópico específico.
- Cuando un publicador (`VideoPublisher` o `GPSCarPublisher`) publica un mensaje mediante el método `publish`, el `Topic` asociado notifica a todos los suscriptores suscritos (`VideoFollower` o `GPSCarFollower`) invocando su método `update`.

3. **Publicador y Suscriptor de Videos (`VideoPublisher`, `VideoFollower`)**:
   - `VideoPublisher` toma una URL ingresada por el usuario a través de `QLineEdit` en `MainWindow` y la publica al tópico "video" mediante el método `publishUrl`.
   - `VideoFollower` recibe el mensaje (URL) a través de su clase auxiliar `VideoSubscriber`, que invoca `onMessageReceived` para actualizar la URL actual y emitir una señal `urlUpdated`. Esta señal actualiza el texto de un `QPushButton` en `MainWindow`. Al presionar el botón, `VideoFollower::playVideo` crea una ventana con un `QVideoWidget` y un `QMediaPlayer` para reproducir el video, junto con un `QSlider` para controlar el volumen mediante un `QAudioOutput`.
4. **Publicador y Suscriptor de Posiciones GPS (`GPSCarPublisher`, `GPSCarFollower`)**:
   - `GPSCarPublisher` utiliza `QFileDialog` para cargar un archivo de texto con posiciones GPS en el formato `<tiempo> <x> <y>`. Un `QTimer` dispara cada segundo el método `updatePosition`, que interpola linealmente las posiciones intermedias y publica un mensaje con el formato "Tiempo: X, X: Y, Y: Z" al tópico "GPS".
   - `GPSCarFollower` recibe los mensajes a través de su clase auxiliar `GPSSubscriber`, que invoca `onMessageReceived` para procesar el mensaje usando una expresión regular (`QRegularExpression`). La posición se actualiza en `TrackingWidget`, que dibuja un círculo rojo en las coordenadas correspondientes mediante `paintEvent`. Un `QLabel` muestra la información de tiempo y coordenadas en la parte inferior de la ventana.
5. **Interfaz Gráfica y Flujo de Ejecución**:
   - La interfaz gráfica en `MainWindow` organiza los widgets en dos columnas: publicadores a la izquierda (`QLineEdit` para URLs, botones para cargar archivo GPS y controlar la simulación) y suscriptores a la derecha (`QPushButton` para el video, botón para mostrar la ventana de seguimiento GPS).
   - Las señales y slots de Qt aseguran que las interacciones del usuario desencadenen las acciones correspondientes en los publicadores y suscriptores.
   - `TrackingWidget` personaliza la visualización del móvil GPS con una cuadrícula de fondo y un círculo rojo que se mueve suavemente según las posiciones interpoladas.

Dificultades Encontradas y Soluciones Implementadas

1. **Dificultad: Interpolación Lineal en `GPSCarPublisher`**

- **Descripción**: Implementar la interpolación lineal para generar posiciones intermedias cada segundo a partir de un archivo con intervalos de tiempo mayores fue un desafío, ya que requería calcular correctamente las coordenadas intermedias basadas en el tiempo actual.
- **Solución**: Se implementó el método `interpolatePosition` en `GPSCarPublisher`, que calcula la fracción de tiempo entre dos puntos consecutivos del archivo y aplica una interpolación lineal para las coordenadas `x` e `y`. Se probaron múltiples casos con archivos de entrada para asegurar que la interpolación fuera precisa y que el móvil se moviera suavemente en la simulación.

2. **Dificultad: Sincronización de Actualizaciones Gráficas en `TrackingWidget`**
   - **Descripción**: El círculo en `TrackingWidget` no se actualizaba correctamente al recibir nuevas posiciones, lo que provocaba un retraso o parpadeo en la visualización del móvil.
   - **Solución**: Se aseguró que el método `updatePosition` de `TrackingWidget` llamara a `update()` para forzar el repintado del widget tras cada actualización de posición. Además, se aplicaron límites (`qBound`) para mantener el círculo dentro de los márgenes de la ventana, evitando errores visuales. Se optimizó el `paintEvent` utilizando `QPainter::Antialiasing` para mejorar la calidad visual del círculo.
3. **Dificultad: Gestión del Volumen en `VideoFollower`**
   - **Descripción**: Integrar el `QSlider` para controlar el volumen del video presentó problemas, ya que el `QAudioOutput` no se actualizaba correctamente tras la creación dinámica del reproductor de video.
   - **Solución**: Se implementó una señal (`audioOutputCreated`) en `VideoFollower` para notificar a `MainWindow` cuando se crea un nuevo `QAudioOutput`. Esto permitió conectar el `QSlider` al volumen del `QAudioOutput` activo, asegurando que los cambios en el slider se reflejaran inmediatamente en el volumen del video. Se estableció un valor inicial de volumen (80%) para una experiencia de usuario consistente.

Información Adicional

- **Ejecución**: El programa se ejecuta desde QtCreator, utilizando el archivo `main.cpp` como punto de entrada. La interfaz gráfica se configura en `MainWindow`, y los publicadores/suscriptores se inicializan automáticamente al iniciar la aplicación.
- **Archivos de Entrada**: Para la simulación GPS, se recomienda usar archivos de texto con el formato `<tiempo> <x> <y>`. Un archivo de ejemplo compatible se incluye en el repositorio.
- **Extra-crédito**: La funcionalidad del `QSlider` para ajustar el volumen está implementada y documentada en el archivo `README.md` del repositorio.
- **Entrega**: La solución completa, incluyendo el código fuente, el diagrama UML, y esta documentación, se encuentra en el repositorio Git. El repositorio incluye instrucciones detalladas para compilar y ejecutar el proyecto.