



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO

# **VISUALIZACIÓN Y MANIPULACIÓN DE ONTOLOGÍAS EN REALIDAD VIRTUAL**

## **Informe PFE**

Simón Bruno - 12177

Tutor: Profesor César Aranda

# **AGRADECIMIENTOS**

---

Me gustaría dar las gracias a todos los que me han apoyado durante este programa de intercambio universitario y el periodo de prácticas.

A mi familia por enseñarme siempre a apuntar lo más alto posible. Todo lo que han hecho ha sido para ayudarme a crecer. También les agradezco su apoyo material durante este periodo; sin él, no podría haberlo conseguido.

A mis amigos, tanto los de casa como los nuevos que he ido conociendo por el camino, por animarme y aconsejarme siempre desde cualquier parte del mundo.

A mi supervisor y mis compañeros, por confiar en mí y en mi criterio para encontrar la mejor solución posible para cada tarea asignada. Todos ellos demostraron ser ejemplares a su manera.

A mis universidades de Francia y Argentina, por darme los conocimientos y las herramientas necesarias para perseguir mis sueños como joven profesional.

A mi profesor tutor de Proyecto Final de Estudios, por aconsejarme y acompañarme en este proceso.

# TABLA DE CONTENIDOS

---

<b>Resumen</b>	<b>6</b>
<b>Introducción</b>	<b>7</b>
0.1. Realidad Extendida . . . . .	7
0.2. Ontologías . . . . .	9
0.3. Gaia-X . . . . .	11
<b>1. Estado del Arte</b>	<b>14</b>
<b>2. Trabajo Realizado</b>	<b>16</b>
2.1. Planeación de Desarrollo . . . . .	16
2.2. Software de Desarrollo . . . . .	17
2.2.1. OpenXR . . . . .	17
2.2.2. XR Interaction Toolkit . . . . .	18
2.3. Visualizador de Ontologías . . . . .	18
2.4. Herramientas pre-existentes . . . . .	22
2.4.1. Filtrar por conexiones . . . . .	23
2.4.2. Resaltar burbuja . . . . .	23
2.4.3. Panel de información . . . . .	24
2.4.4. Herramientas de edición . . . . .	25
2.5. Funciones Desarrolladas . . . . .	25
2.5.1. Sistema de Búsqueda . . . . .	25
2.5.2. Sistema de Reconocimiento de Voz . . . . .	29
2.5.3. Menú Jerárquico . . . . .	35
2.5.4. Importación desde archivos Jsonld . . . . .	36
2.5.5. Exportación a archivos Jsonld . . . . .	43
2.5.6. Contenedores de Ontologías . . . . .	44
2.5.7. Modos de distribución de ontologías . . . . .	48

---

**TABLA DE CONTENIDOS**

<b>Conclusiones</b>	<b>51</b>
2.6. Trabajo Futuro . . . . .	52
<b>Bibliografía</b>	<b>53</b>

# ÍNDICE DE FIGURAS

---

1.	Realidad Extendida y sus variantes . . . . .	7
2.	Ontología: Ejemplo 1 . . . . .	10
3.	Ontología: Ejemplo 2 . . . . .	11
4.	Ontología: Ejemplo 3 . . . . .	12
5.	Logo de Gaia-X . . . . .	13
1.1.	Protégé Logo . . . . .	14
2.1.	Planeacion de Desarrollo en Diagrama de Gantt . . . . .	16
2.2.	OpenXR logo . . . . .	18
2.3.	Prefabs de conectores y burbuja . . . . .	19
2.4.	Escena desarrollada . . . . .	21
2.5.	Menú y puntero de los controladores . . . . .	21
2.6.	Interaccioes del Toolkit . . . . .	22
2.7.	Filtrar por conexiones . . . . .	23
2.8.	Resaltar burbuja . . . . .	24
2.9.	Panel de información . . . . .	24
2.10.	Herramientas de edición . . . . .	25
2.11.	Búsqueda de clases y resultados . . . . .	28
2.12.	Modelos de entrenamiento de Whisper AI . . . . .	33
2.13.	Diagrama de Paquetes del Sistema de Búsqueda . . . . .	35
2.14.	Ilustración gráfica de funcionamiento . . . . .	36
2.15.	Diagrama de secuencia de importación de ontologías . . . . .	37
2.16.	Conversión de clase y conexión a formato interno . . . . .	40
2.17.	Objeto Json del archivo de posiciones . . . . .	42
2.18.	Ontologías y sus contenedores . . . . .	44
2.19.	Diagrama de clases para los contenedores . . . . .	45
2.20.	Diagrama de secuencia para la creación de clases y containers . . . . .	46
2.21.	Cálculo de proyección para los modos de distribución . . . . .	50

# RESUMEN

---

Este documento consiste en el informe de mi Proyecto Final de Estudios realizado durante mi período de prácticas en el instituto Fraunhofer IPK en Berlín, Alemania. Mi período en este instituto fue de 6 meses y aquí expondré los resultados de mi trabajo.

En el ámbito científico, trabajar con ontologías es parte del día a día y los software para hacerlo no cuentan con suficientes herramientas. En este contexto, Gaia-X busca desarrollar un estándar de información a nivel europeo para asegurarse la portabilidad y gobernanza. Para brindarles una opción extra a la hora de manipular las ontologías de su proyecto, desarrollamos una aplicación en Realidad Virtual con el objetivo de simplificar algunas tareas. Aquí intentaré explicar lo más claramente posible cuál fue mi trabajo dentro del proyecto y el resultado de estos 6 meses.

El proyecto continúa una vez finalizadas mis prácticas, pero la simulación es estable y sigue en marcha, con mucho trabajo por hacer. En este informe mostraré una parte del proceso de desarrollo de las funciones que me asignaron. El trabajo futuro incluye planificar y desarrollar más funciones, así como probar y pulir la experiencia del usuario.

# INTRODUCCIÓN

---

La velocidad a la que evoluciona la tecnología es cada vez mayor, por lo que necesitamos seguir desarrollando software y crear aplicaciones para aprovechar al máximo las herramientas de que disponemos. Una de estas tecnologías es la Realidad Extendida (XR - *Extended Reality*) con todos sus niveles de interacción y, para cada tarea, necesitamos encontrar la respuesta más adecuada a las necesidades.

En este informe voy a mostrar los resultados del proyecto realizado durante mi período de práctica profesional dentro del instituto Fraunhofer IPK en Berlín, Alemania. Para ello, voy a exponer los grandes bloques conceptuales que componen este proyecto, el estado del arte actual, el trabajo realizado, ejemplos de código, esquemas de datos y conclusiones personales.

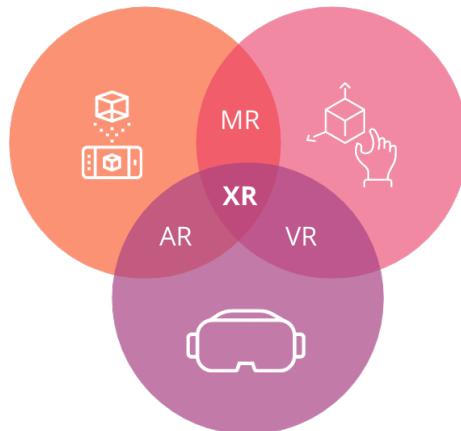


Figura 1: Realidad Extendida y sus variantes

## 0.1. Realidad Extendida

RX es un término emergente que engloba todas las tecnologías inmersivas. Las que ya existen -realidad aumentada (RA), realidad virtual (RV) y realidad mixta (RM)- y las que están por surgir. Todas las tecnologías inmersivas amplían la realidad que experimentamos

mezclando los mundos virtual y “real” o creando una experiencia totalmente inmersiva.  
**Reciente investigación sostiene :**

**Cuándo se espera que la RX se generalice?**

Más del 60 % de los accionistas de empresas de productos RX consideran que se volverá masiva en los próximos 5 años. [1]

Los diferentes tipos de Realidad Extendida son:

■ **Realidad Aumentada (RA)**

En la realidad aumentada, la información y los objetos virtuales se superponen al mundo real. Esta experiencia realza el mundo real con detalles digitales como imágenes, texto y animación. Se puede acceder a la experiencia a través de pantallas, tablets y smartphones. Esto significa que los usuarios no están aislados del mundo real y pueden seguir interactuando y viendo lo que ocurre delante de ellos. Los ejemplos más conocidos de RA son el juego Pokemon GO, que superpone criaturas digitales al mundo real, o los filtros de Snapchat, que colocan en la cabeza objetos digitales como sombreros o gafas.

■ **Realidad Virtual (RV)**

A diferencia de la realidad aumentada, en una experiencia de realidad virtual los usuarios se sumergen por completo en un entorno digital simulado. Las personas deben ponerse un casco de realidad virtual o una pantalla montada en la cabeza para obtener una visión de 360 grados de un mundo artificial que engaña a su cerebro haciéndole creer que está, por ejemplo, caminando sobre la luna, nadando bajo el océano o adentrándose en cualquier nuevo mundo que hayan creado los desarrolladores de la RV.

■ **Realidad Mixta (RM)**

En la realidad mixta, los objetos digitales y los del mundo real coexisten y pueden interactuar entre sí en tiempo real. Es la última tecnología de inmersión y a veces se denomina realidad híbrida. Requiere un casco de realidad mixta y mucha más potencia de procesamiento que la RV o la RA. Apple Vision Pro es un gran ejemplo que, por ejemplo, permite colocar objetos digitales en la habitación en la que uno

se encuentra y brinda la posibilidad de girarlos o interactuar con el objeto digital de cualquier forma posible.

## 0.2. Ontologías

Para entender el contexto del proyecto necesitamos dar explicar algunos puntos sobre el proyecto Gaia-X y definir qué son las ontologías y cómo se relacionan estos conceptos. En esta sección primero veremos qué es una ontología, para qué sirven, algunos ejemplos y por qué son importantes para Gaia-X, que será explicado en la sección siguiente.

Una ontología es una descripción de una estructura de datos: clases, propiedades y relaciones en un dominio de conocimiento. Su objetivo es servir de base para las instancias de los grafos de conocimiento, garantizando la coherencia de los datos y la comprensión del modelo [2]. Esta representación de la información es importante para la ciencia por los motivos siguientes:

- **Organización del conocimiento:**

Las ontologías ayudan a organizar el conocimiento de manera estructurada y coherente, lo que facilita su comprensión y utilización por parte de los científicos y otros usuarios.

- **Interoperabilidad:**

Al proporcionar una estructura común para la representación del conocimiento, las ontologías promueven la interoperabilidad entre diferentes sistemas y fuentes de datos en la ciencia. Esto es crucial en campos donde se necesita combinar información de diversas fuentes para obtener una comprensión más completa.

- **Consistencia y precisión:**

Al definir términos y relaciones de manera precisa y clara, las ontologías ayudan a garantizar la consistencia y la precisión en la comunicación y el intercambio de información científica. Esto es especialmente importante en disciplinas donde la ambigüedad en la terminología puede llevar a malentendidos o errores.

- **Reutilización del conocimiento:**

Las ontologías proporcionan un marco reutilizable para representar el conocimiento en un dominio específico. Esto permite a los científicos construir sobre el tra-

jo existente y evitar la redundancia en la creación de modelos conceptuales para diferentes aplicaciones.

- Facilitación del razonamiento automático:

Las ontologías son utilizadas por sistemas de inteligencia artificial y razonamiento automático para inferir nuevos conocimientos a partir de la información existente. Al definir explícitamente las relaciones entre entidades, las ontologías permiten que estos sistemas realicen inferencias lógicas y respondan a consultas de manera más efectiva.

Para ilustrar esta idea, podemos ver los ejemplos siguientes.

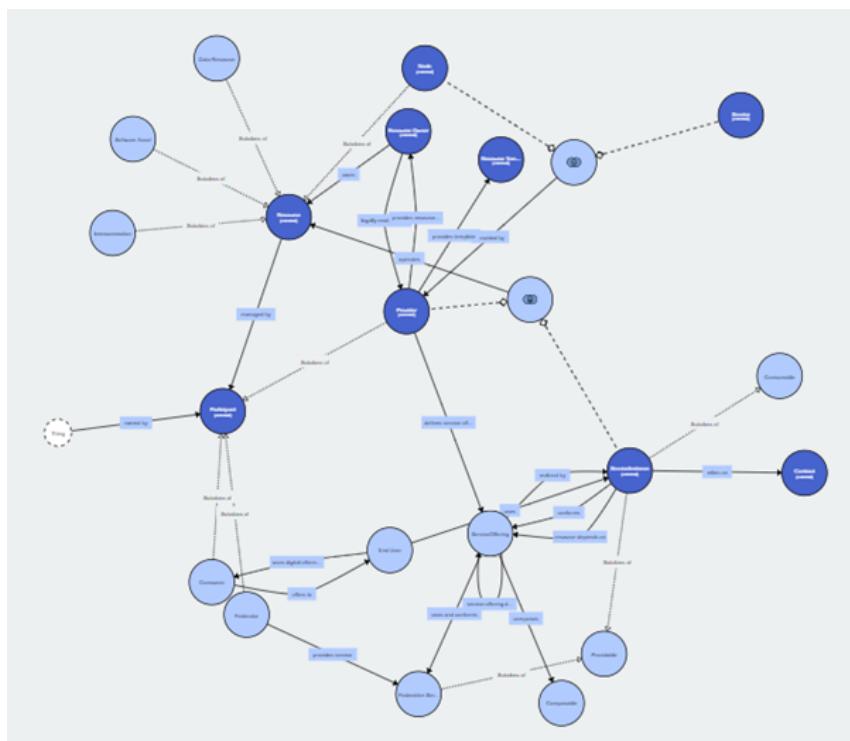


Figura 2: Ontología: Ejemplo 1

En la Figura 2, podemos ver una estructura de datos en la cual cada concepto está representado por un círculo, y entre ellos están conectados por diferentes tipos de líneas, para diferenciar las relaciones.

En la Figura 3, vemos como una empresa puede ser representada a través de ontologías al representar sus componentes como clases.

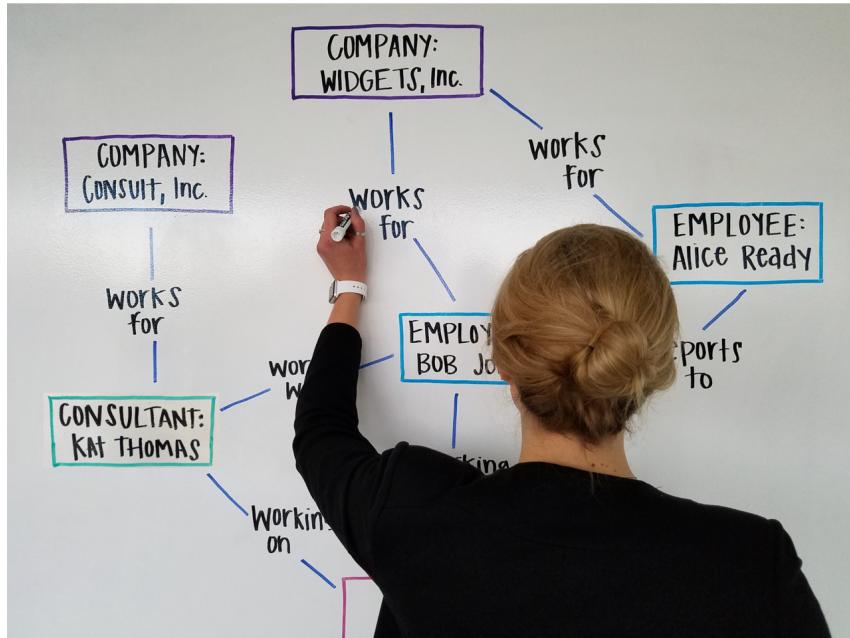


Figura 3: Ontología: Ejemplo 2

Para introducir de a poco la problemática que queremos enfrentar, podemos ver la Figura 4, que muestra qué tan compleja se puede ver una ontología si la cantidad de clases y relaciones incrementa demasiado. Esta idea la iremos expandiendo un poco más en otras secciones.

### 0.3. Gaia-X

Gaia-X es una iniciativa que desarrolla, sobre la base de los valores europeos, una gobernanza digital que puede aplicarse a cualquier pila tecnológica existente de nube para obtener transparencia, controlabilidad, portabilidad e interoperabilidad entre datos y servicios. [3]

Hoy en día, muchos negocios de pequeña y mediana dimensión desarrollan interfaces individualmente para cada cliente para el intercambio de información y la interoperabilidad de soluciones. Esto cuesta tiempo y dinero. Gaia-X busca proveer mecanismos comunes de intercambio de información que adhieran a las necesidades comunes de confianza.

En Europa, la adopción de tecnologías de nube es solo del 26 %. Esto implica que la mayor parte de la información y de las aplicaciones son todavía inaccesibles e inintercambiables. Gaia-X desarrollará una estructura de trabajo que le permita a la gente tomar

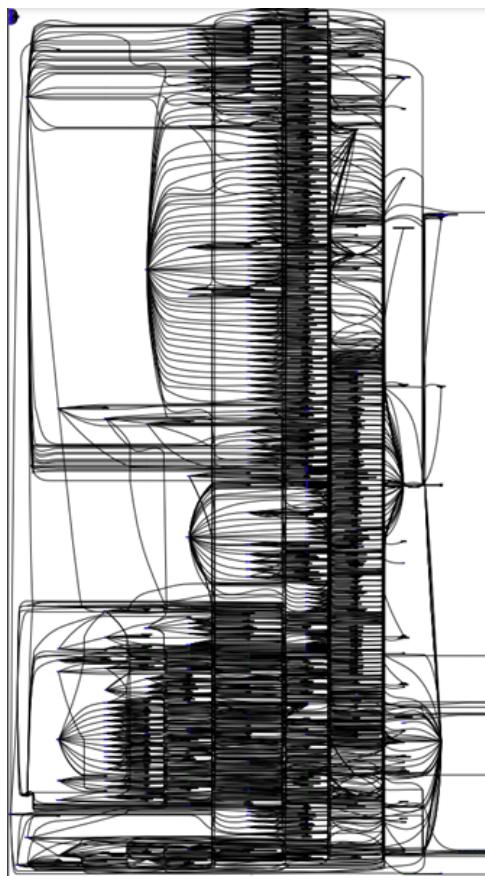


Figura 4: Ontología: Ejemplo 3

decisiones informadas a la hora de intercambiar información.

¿Por qué están relacionados este proyecto y las ontologías? La descripción de entidades ha sido siempre uno de los objetivos de la informática. En primer lugar, porque ayuda a comprender la entidad descrita, y después porque su análisis sintáctico permite aplicar reglas y construir funcionalidades.

El objetivo de la ontología es ser abierta y compartida: describe lo que son las cosas en general, no en un contexto específico. De este modo se simplifica el proceso de diseño de nuevas ontologías, ya que se puede recurrir a las existentes, y también se facilita la interacción con otras organizaciones, porque se utilizan las mismas ontologías.

Por último, y quizás el punto más crucial, la ontología puede ser analizada fácilmente por una máquina, lo que facilita el razonamiento. [4]

El desarrollo del proyecto Gaia-X naturalmente va a depender de las ontologías y los softwares requeridos, por esta razón tenemos que asegurarnos de contar con herramientas

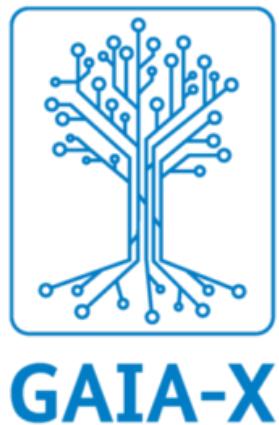


Figura 5: Logo de Gaia-X

adecuadas.

# ESTADO DEL ARTE

---

Organizar información a través de ontologías es muy común en diferentes ámbitos científicos como la biomedicina, informática, filosofía, etc. Esto ha dado lugar a numerosas opciones de software dedicado a representar dichas ontologías [5]. Estos editores se encargan de tareas como creación, edición, visualización, exploración, *debugging*, etc. Por estas razones, el desarrollo de una aplicación para trabajar con ontologías no es una tarea sencilla, y requiere de mucho estudio y planeamiento, ya que todas tienen falencias y puntos débiles en distintas etapas del trabajo [6].

Todas las herramientas que podemos encontrar tienen la característica común de trabajar solo en el espacio bidimensional, como la inmensa mayoría de programas hoy en día. Por esta razón, si queremos expandir el horizonte de la tecnología, debemos empezar a probar qué tan útil es la Realidad Extendida en diferentes ámbitos. En este caso, vamos a tomar como referencia el software más utilizado con fines de visualización, llamado Protégé.



Figura 1.1: Protégé Logo

Es un editor de ontologías gratis y open-source, que provee un esquema de trabajo para construir sistemas inteligentes y se caracteriza por brindar una útil interfaz de visualización, a diferencia de otros softwares de ontologías. Sin embargo, aprender a operar esta herramienta no es fácil y requiere bastante tiempo y dedicación [7].

El principal problema de este tipo de programas, es el rendimiento limitado en grandes ontologías. Aunque Protégé es adecuado para la mayoría de las necesidades de modelado de ontologías, puede experimentar limitaciones de rendimiento al trabajar con ontologías

---

muy grandes o complejas. En tales casos, los usuarios pueden encontrar que el rendimiento de la herramienta se ralentiza o que experimentan problemas de memoria. Sumado al hecho de que compartir una ontología a través de imágenes o capturas de pantalla en un reporte, complica demasiado la interpretación y no se puede apreciar correctamente lo que se está representando.

# TRABAJO REALIZADO

---

En este capítulo intentaré explicar y mostrar los aspectos generales de las funciones que llegué a desarrollar durante mis prácticas. Proporcionaré diagramas de datos y algunas imágenes que ayuden a entender la idea que hay detrás de muchas soluciones a los problemas a los que nos enfrentamos como equipo.

La información real del proyecto permanecerá oculta por cuestiones de confidencialidad, pero no afectará a la comprensión del trabajo.

## 2.1. Planeación de Desarrollo

Mi trabajo se llevó a cabo durante 6 meses de trabajo en el instituto. A modo ilustrativo del tiempo requerido para realizar las tareas asignadas, podemos ver el siguiente diagrama de Gantt.

Cada herramienta desarrollada requiere una adecuada documentación dentro del GitLab del proyecto para poder mantener registro del progreso, de los métodos de programación utilizados y cualquier explicación que pueda ayudar a entender rápidamente qué se realizó.

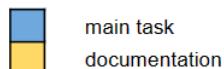
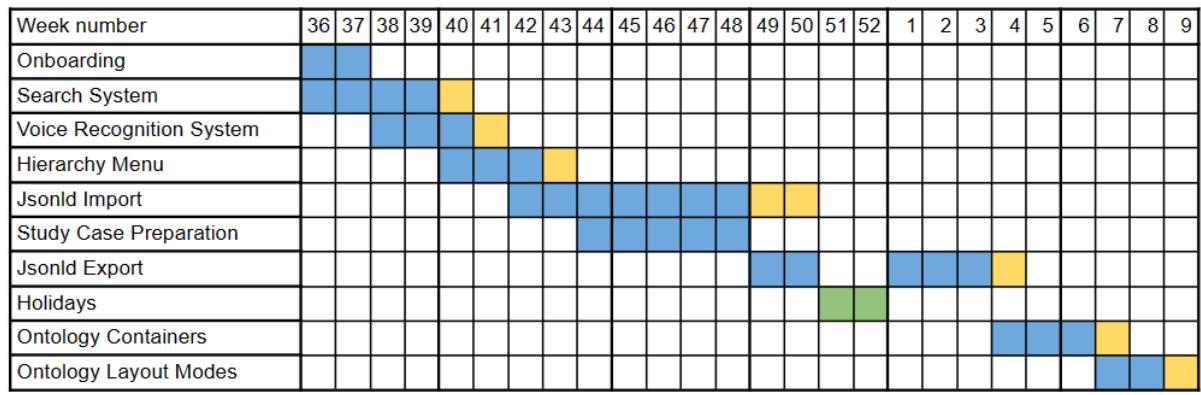


Figura 2.1: Planeacion de Desarrollo en Diagrama de Gantt

## 2.2. Software de Desarrollo

El proyecto se desarrolló con Unity (versión 2021.3.23f1). Se trata de una plataforma de desarrollo de juegos potente y versátil que ha revolucionado la forma de crear juegos, desde proyectos independientes hasta grandes éxitos de taquilla. Con Unity, los desarrolladores pueden diseñar, crear e implantar juegos de alta calidad en multitud de plataformas, como dispositivos móviles, consolas y PC. Su interfaz fácil de usar y su amplia biblioteca de recursos lo hacen accesible tanto para los profesionales experimentados como para los recién llegados al desarrollo de juegos.

Uno de los aspectos más atractivos de Unity es su capacidad para facilitar la creación rápida de prototipos y la iteración. Los desarrolladores pueden dar vida rápidamente a sus ideas, probar mecánicas de juego y perfeccionar sus creaciones en tiempo real. Esta agilidad no sólo acelera el proceso de desarrollo, sino que también permite una mayor creatividad y experimentación.

Además, Unity ofrece un vasto ecosistema de activos, plugins y apoyo de la comunidad, lo que permite a los desarrolladores ampliar la funcionalidad de la plataforma y colaborar con otros creadores. Desde activos y scripts prefabricados hasta técnicas avanzadas de renderizado y algoritmos de inteligencia artificial, Unity proporciona una gran cantidad de recursos para agilizar el desarrollo y mejorar la calidad de los juegos.

### 2.2.1. OpenXR

Para conectar Unity con el casco de realidad virtual, optamos por OpenXR, un estándar abierto que pretende agilizar el desarrollo de aplicaciones de realidad virtual proporcionando una API común para interactuar con el hardware de realidad virtual. Desarrollado por Khronos Group, un consorcio industrial centrado en el desarrollo de estándares abiertos para gráficos y computación paralela, OpenXR pretende abordar la fragmentación del ecosistema de realidad virtual y realidad aumentada permitiendo a los desarrolladores crear aplicaciones compatibles con una amplia gama de dispositivos, plataformas y entornos de ejecución. [8]

Proporciona un conjunto de API estándar para funciones como el renderizado, el manejo de entradas, el seguimiento y la interacción, lo que permite a los desarrolladores acceder y controlar las capacidades de los dispositivos de hardware de realidad virtual y realidad aumentada de forma coherente e independiente de la plataforma. Esto fue de enorme relevancia, debido a que en nuestro departamento disponíamos de diferentes



Figura 2.2: OpenXR logo

headsets para desarrollo y presentaciones, como el HTC Vive Pro, HTC Focus y Meta Quest 2 y 3, por lo que la estandarización fue crucial para la mejora consistente en nuestro proyecto.

### 2.2.2. XR Interaction Toolkit

Otro elemento central de este proyecto fue el uso del *XR Interaction Toolkit* de Unity, que permite todo tipo de interacciones entre el jugador y el entorno.

El paquete XR Interaction Toolkit es un sistema de interacción de alto nivel basado en componentes para crear experiencias de realidad virtual y realidad aumentada. Proporciona un marco que hace que las interacciones 3D y de interfaz de usuario estén disponibles a partir de eventos de entrada de Unity. El núcleo de este sistema es un conjunto de componentes base *Interactor* e *Interactable*, y un *Interaction Manager* que une estos dos tipos de componentes. [9]

Luego daré algunos ejemplos de interacciones posibles gracias a este kit de desarrollo.

## 2.3. Visualizador de Ontologías

Esta aplicación tiene objetivos y estructura muy concretos. En primer lugar, desarrollar herramientas utilizando el formato utilizado por Gaia-X, por lo que debemos incorporar un método para leer los archivos provenientes de este proyecto y para funcionar en conjunto con las otras opciones de software actuales. Segundo, enfocarnos en tareas estrictamente de visualización de ontologías y no tanto de creación o *debugging*. Este último punto nos determina el tipo de herramientas a priorizar, como pueden ser métodos de filtrado de clases, búsqueda, comparación de propiedades, manipulación de la apariencia de las ontologías, etc. Más adelante, una vez tengamos este punto cubierto, veremos qué podemos desarrollar para la edición de una ontología. Mientras tanto, debemos asegurarnos de poder incorporar nuestra aplicación al conjunto de software disponible y así utilizarla en

conjunto con las opciones dedicadas a otras tareas.

Antes de explicar las herramientas que desarrollé, es conveniente mostrar cómo se compone la escena y cuáles son los *game objects* que van a estar interactuando.

En nuestro proyecto, decidimos representar una clase utilizando una burbuja. Esta se conecta a otras semejantes a través de flechas de dos colores según el tipo de conexión:

- Herencia: en el que se especifica que una clase es subclase de otra de mayor jerarquía. (conexión naranja)
- Relación específica: una conexión que incluye una etiqueta de texto para especificar de qué manera se conectan dichas clases. (conexión azul)

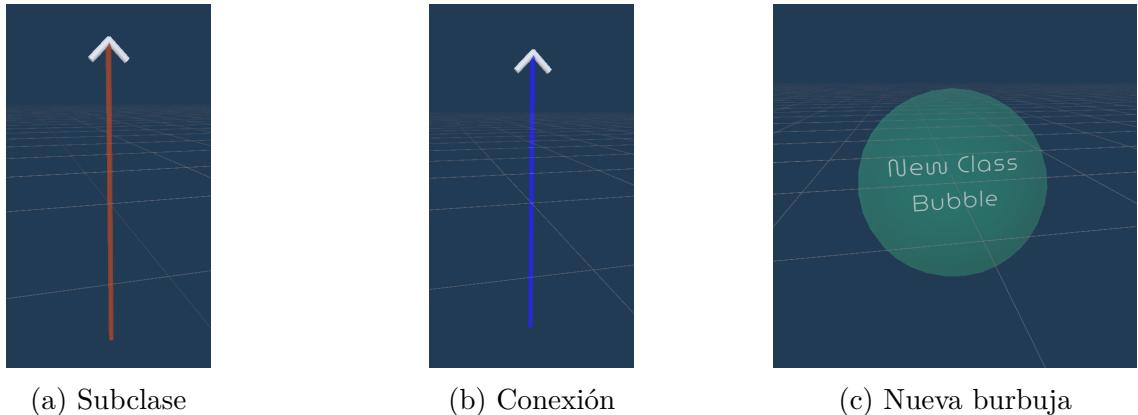


Figura 2.3: Prefabs de conectores y burbuja

Un objeto burbuja que representa una clase tiene una serie de variables y listas que le dan identidad única y le brindan la información necesaria para construir el mapa de conexiones.

1. UniqueID: id numérico que representa al objeto dentro de la simulación.
2. Nombre de la clase.
3. Nombre de la ontología a la que pertenece.
4. Lista de burbujas con destino de conexión a la burbuja actual (`ownedByBubblesList`).
5. Lista de burbujas con origen de conexión en la burbuja actual (`hasBubblesIDsList`).

6. Lista de burbujas de las que hereda la burbuja actual (inheritedBubblesList).
7. Lista de burbujas a las que hereda la burbuja actual (inheritingBubblesIDsList).
8. Lista de comentarios o propiedades de la clase (elementPropertyValues).

Los primeros 3 ítems son necesarios para construir la clase, es decir, deben contener valores. Las listas pueden estar vacías, lo cual significa que la clase no tiene conexiones.

```

1   [Header("Bubble Relations")]
2   [Tooltip("The bubble from which this bubble inherits.")]
3   public List<Bubble> inheritedBubblesList = new();
4
5   [Tooltip("List of bubbles directly connected to this bubble.")]
6   )
7   [SerializeField]
8   private List<Bubble> directlyConnectedBubblesList = new();
9
10  [Tooltip("List of bubbles that inherit from this bubble.")]
11  public List<Bubble> inheritingBubblesList = new();
12
13  [Tooltip("List of bubbles that this bubble connects to")]
14  public List<Bubble> hasBubblesList = new();
15
16  public NetworkList<ulong> hasBubblesIDList;
17
18  [Tooltip("List of labels for the connections to the bubbles
19      that this bubble has.")]
20  public List<string> connectionLabels = new();
21
22  [Tooltip("List of comments/properties of each bubble.")]
23  public List<string> propertyValues = new();
24
25  [Tooltip("List of bubbles that connect to this bubble.")]
26  public List<Bubble> ownedByBubblesList = new();

```

Código 2.1: Información que contiene cada Burbuja

La escena desarrollada se ve como en la Figura 2.4. Aquí podemos observar, desde el punto de vista del usuario, como se enfrentaría a un grupo de clases provenientes de 4 ontologías diferentes.

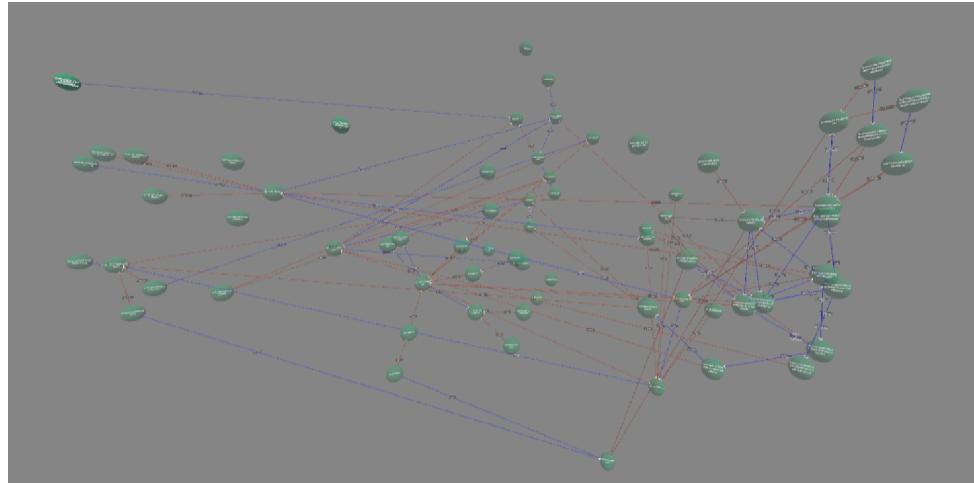


Figura 2.4: Escena desarrollada

El usuario cuenta con 2 controles de mano para relacionarse con la simulación. La mano izquierda está vinculada a un menú con paneles de información y botones para lanzar tareas. La mano derecha, en cambio, proyecta un puntero láser llamado *Raycast* que realiza cálculos de intersección con los objetos presentes en la escena (Ver Figura 2.5). Así, el usuario puede trabajar con su mano derecha interactuando con las burbujas, o posicionar el menú con la mano izquierda y seleccionar con el puntero.



Figura 2.5: Menú y puntero de los controladores

El raycast del controlador de la mano derecha puede generar interacciones de diferente tipo según la combinación de botones que se utilice al reconocer un objeto intersectado por el rayo. Con el puntero raycast, manteniendo el botón del dedo índice (normalmente llamado *Trigger*) presionado mientras apuntamos a una burbuja, podemos moverla en el espacio como queramos. Ese mismo puntero también nos permite teletransportarnos a dicha burbuja si movemos el joystick hacia adelante mientras apuntarmos en vez de presionar el trigger. Las interacciones con el menú se realizan con el mismo puntero y seleccionando los botones que queramos utilizando el mismo trigger.

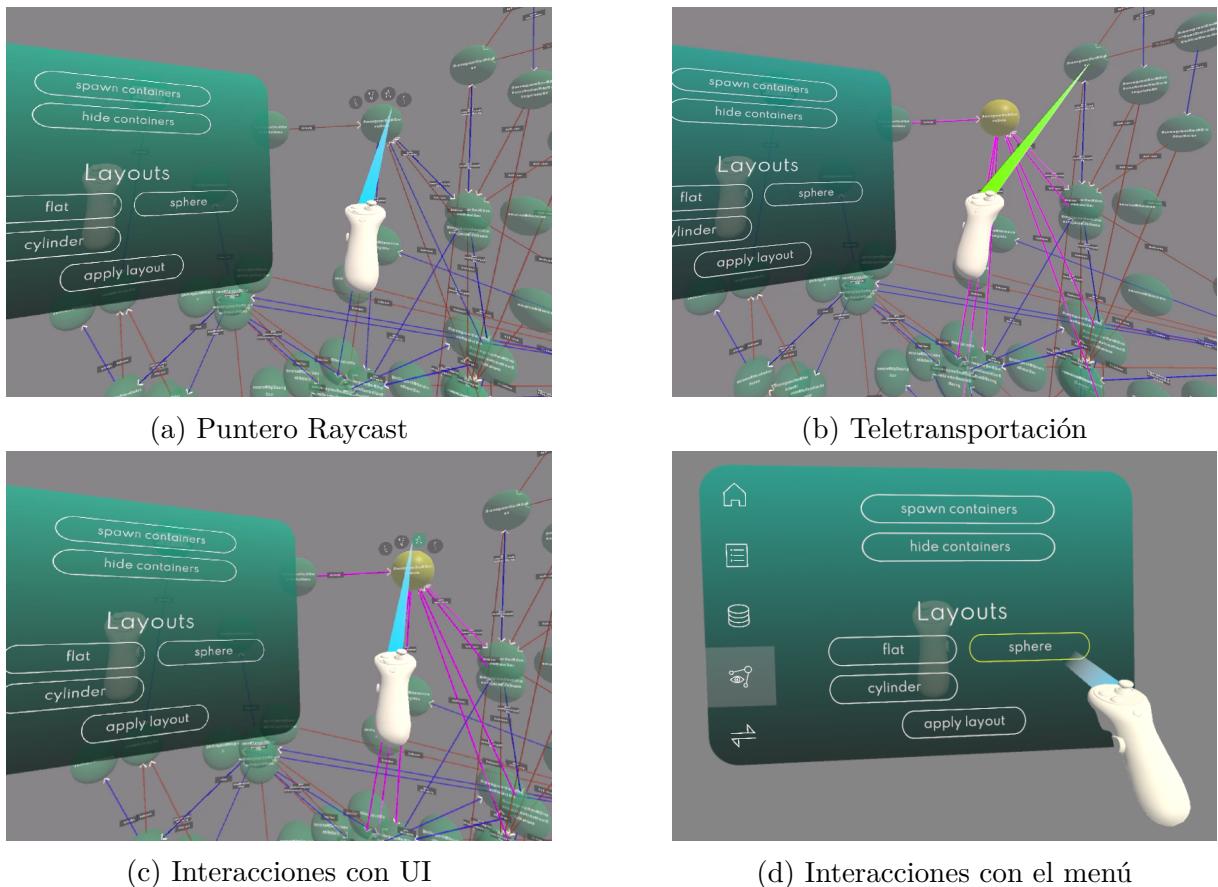


Figura 2.6: Interacciones del Toolkit

## 2.4. Herramientas pre-existentes

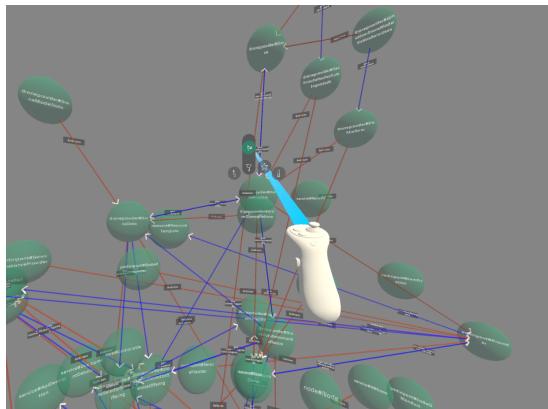
En esta sección voy a mostrar las herramientas que ya existían en la escena en el momento de mi comienzo en el proyecto. Es importante explicarlas rápidamente ya que

parte de mi tarea fue integrar mis funciones a lo ya existente, al igual que solucionar los posibles errores que surgieran de aumentar las interacciones entre los códigos.

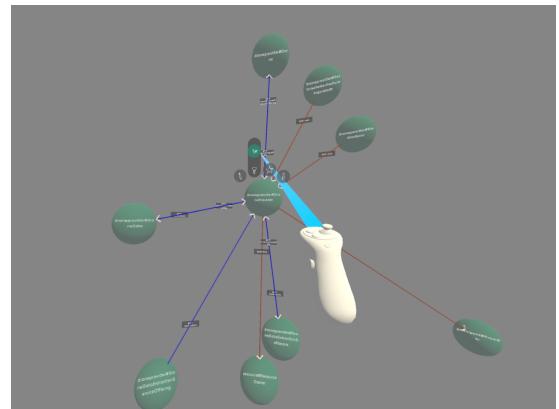
Al apuntarle a una burbuja, aparece sobre ella un menú con distintos botones asociados a las funciones que veremos en las siguientes páginas.

#### 2.4.1. Filtrar por conexiones

Esta herramienta es de las mas útiles en nuestro visualizador ya que se encarga de mostrar sólo las burbujas directamente conectadas a la burbuja que seleccionemos. Esto se hace apagando el componente físico de todas las burbujas de la escena excepto la burbuja seleccionada y sus conexiones. Así, podremos visualizar muy rápidamente, filtrando una gran cantidad de objetos, una burbuja deseada y sus relaciones.



(a) Fragmento de escena antes de filtrar



(b) Fragmento de escena después de filtrar

Figura 2.7: Filtrar por conexiones

En múltiples entrevistas y ensayos que hemos hecho con distintos participantes, hemos obtenido excelentes devoluciones sobre esta funcionalidad, incluso con ideas para mejorarla.

#### 2.4.2. Resaltar burbuja

En caso que queramos distinguir una burbuja específica, podemos resaltarla con una herramienta que se encarga de cambiarle el color a su mesh y al de sus conectores. Gracias a esto, podemos identificarla fácilmente dentro de la red de nodos y conexiones por más compleja o compacta que sea.

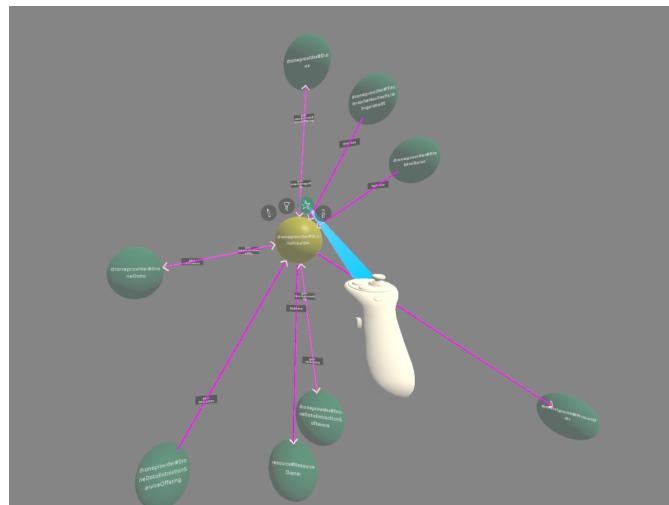


Figura 2.8: Resaltar burbuja

#### 2.4.3. Panel de información

Cada clase posee una lista con comentarios o propiedades que aclaran algún aspecto específico. Toda esta información la mostramos en un panel que pertenece a cada burbuja.

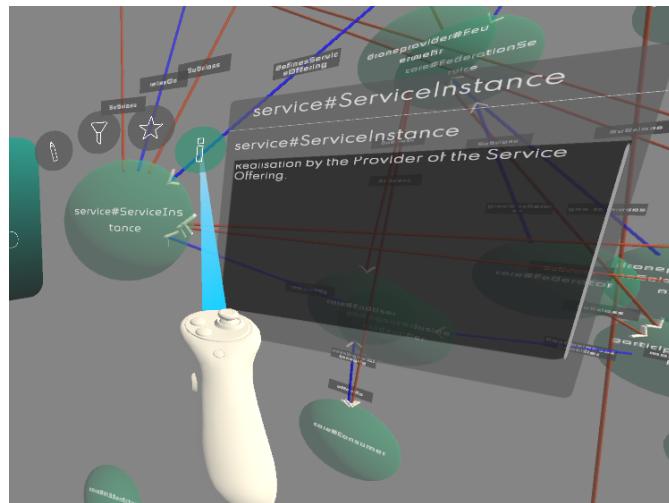


Figura 2.9: Panel de información

## 2.4.4. Herramientas de edición

Si bien la edición excede el objetivo inicial de nuestro visualizador, también decidimos agregar algunas pocas herramientas para probarlas en algunos ensayos con participantes expertos en trabajo con ontologías y así obtener devoluciones que nos ayuden a planificar desarrollo futuro. En primera instancia, creamos los métodos para crear, eliminar y editar una burbuja (nombre y ontología a la que pertenece).

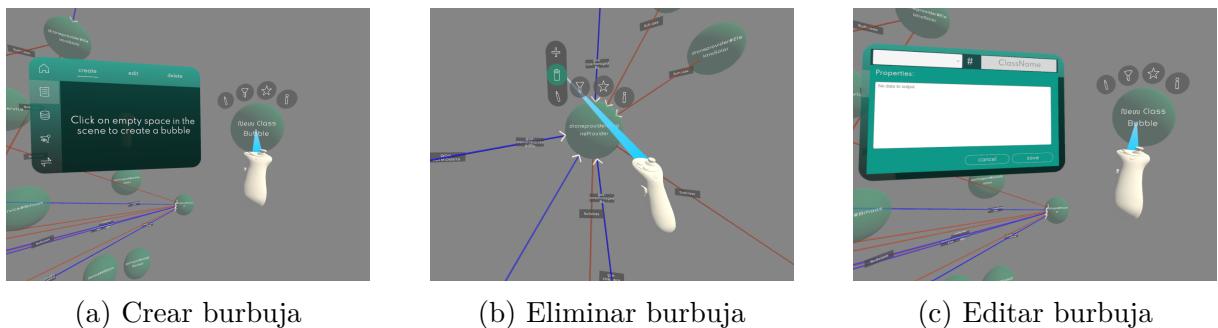


Figura 2.10: Herramientas de edición

## 2.5. Funciones Desarrolladas

### 2.5.1. Sistema de Búsqueda

Implementar un sistema de búsqueda es casi obligatorio a la hora de crear un software de datos e información. Siempre necesitamos comprobar la existencia de clases y en la mayoría de los casos los resultados de búsqueda tienen también una funcionalidad asociada. En nuestro caso optamos por 2 diferentes, que pueden ser habilitadas desde el inspector de Unity: resaltar la burbuja elegida o teletransportarse a ella si se desea. Esto demostró ser extremadamente útil y escala muy bien si la cantidad de clases aumenta.

Para el ingreso de texto de búsqueda, primero optamos por utilizar un teclado virtual importado de Github que acoplamos al menú del controlador [10]. Así podemos seleccionar con el puntero de la mano derecha cada letra que queramos buscar.

El proceso de búsqueda es manejado por el *Search Manager* y su método *Search()* se ejecuta cada vez que se lance la función de callback *On Value Changed* del campo de entrada de texto del buscador. Este objeto contiene una lista *List<GameObject>Elements* que contiene referencias a todas las clases de la escena.

Cuando hay una coincidencia entre el texto de entrada y el nombre de cualquier burbuja, creamos botones dentro de un panel y los referenciamos a su burbuja correspondiente, de modo que podemos llamar a un método específico al pulsar sobre él. Si no hay coincidencia alguna, habilitamos un objeto que muestra un mensaje al usuario, indicándole que no hay resultado para la búsqueda realizada. Para esto utilizamos el booleano *flagFoundObjects* para determinar los diferentes casos posibles de la búsqueda.

Es importante aclarar que cada vez que ingresamos un carácter en el buscador, debemos resetear los resultados anteriores, ya que estamos acotando la cantidad de resultados posibles. Esto significa que al principio del método de callback, primero debemos vaciar las listas de resultados y sus correspondientes botones instanciados. Todo esto lo podemos ver en el siguiente extracto de código.

```
1 public void Search()
2 {
3     // Clearing variables and scene elements
4     DeleteButtons(); // Search() is called every time there
5         is a change in the inputField so for every new search,
6         we need to destroy previous buttons and spawn new ones
7     matchingElements.Clear();
8     int flagFoundObjects;
9
10    // get input text
11    SearchText = SearchInputField.GetComponent<TMP_InputField
12        >().text;
13    int searchTextLength = SearchText.Length;
14    foreach (GameObject element in Elements)
15    {
16        if (element.name.Length >= searchTextLength &&
17            searchTextLength >= 1) // if the text input is
18            longer than the element name and consists of at
19            least 1 character
20        {
21            string elementName = element.name.ToLower();
22            if (elementName.Contains(SearchText.ToLower()))
23            {
24                matchingElements.Add(element);
25            }
26        }
27    }
28 }
```

```

20    }
21}
22
23    int flag = matchingElements.Count > 0 ? 1 : 2;
24    flagFoundObjects = searchTextLength >= 1 ? flag : 0;
25    CreateButtons();
26    UpdateResultsButton(flagFoundObjects);
27}

```

Código 2.2: Método Search()

Los métodos utilizados son los siguientes:

- Search: este es el método de callback cada vez que se modifica el campo de entrada de texto. Se encarga de comparar el texto ingresado con los nombres de las clases de la escena y crear una lista de resultados compuesta por referencias a las burbujas correspondientes.
- DeleteButtons: cada vez que ingresamos una letra, los resultados cambian. Por esta razón, debemos borrar los botones de los resultados de la búsqueda anterior.
- CreateButtons: instancia un botón por cada burbuja que haya dentro de la lista de resultados y realiza las conexiones necesarias para las funciones de callback de cada botón, ya se teletransportar o resaltar.
- ResetSearch: vacía el campo de entrada para cuando se complete una función de callback y así se resetea el sistema.
- UpdateResultsButton: es un sistema de *flag* para determinar el mensaje de estado de búsqueda. Este es transmitido al usuario a través de un cuadro de texto.

Esta herramienta se incorpora al menú de la mano, que se acopla al mando de la mano izquierda en la simulación, e interactuamos con él con el raycast del mando de la mano derecha. En la Figura 2.11 tenemos una captura de imagen del simulador que muestra como sería el resultado de buscar “dro”. Dentro de lo encontrado, resaltamos cualquier parte del texto que coincida con la búsqueda, ya que hemos utilizado el método propio de los strings *Contains()* para identificar coincidencias en cualquier parte del nombre de la clase.

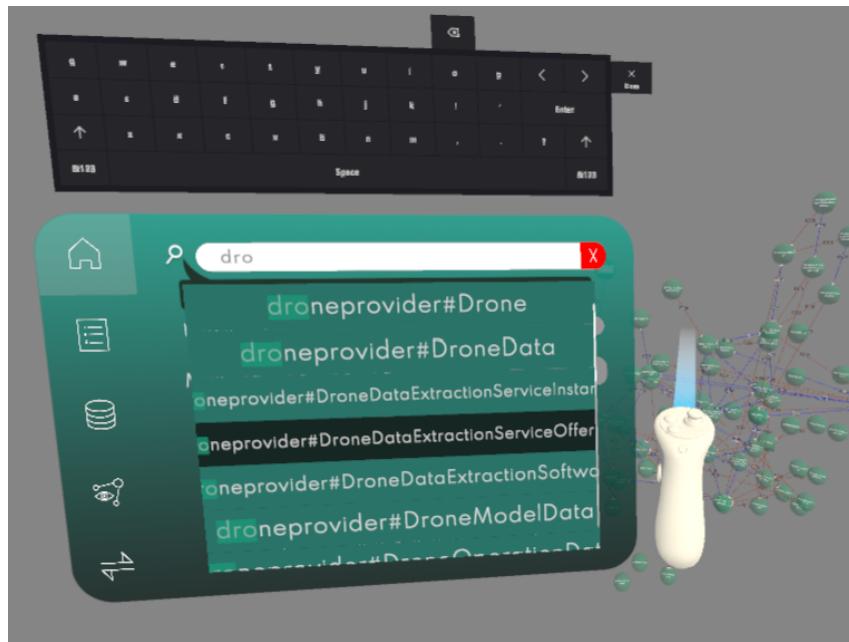


Figura 2.11: Búsqueda de clases y resultados

Al seleccionar alguno de los resultados, lanzaremos la función de callback que hayamos seleccionado en el inspector antes de darle Play al visualizador. Si hemos seleccionado el Resaltador, llamaremos al mismo método que está conectado al botón del menú de cada burbuja para cambiarle el color. Si decidimos utilizar la Teletransportación, debemos brindarle al botón del resultado toda la información correspondiente a la posición de la burbuja para iniciar un *Teleport Request* hacia ella según el *XR Interaction Toolkit*.

```

1  private void Teleport()
2  {
3      SearchInputField.GetComponent<ShowKeyboard>().CloseAll();
4
5
6      if (VRBubbleTeleport == null)
7      {
8          VRBubbleTeleport = GameObject.Find(BubbleName).
9              GetComponentInChildren<VR_TeleportAnchorWithFade>()
10         ;
11     }
12 }
```

```

11|     if (defaultTeleportationTarget == null)
12|     {
13|         defaultTeleportationTarget = VRBubbleTeleport.
14|             gameObject.transform.Find("Default Teleport
15|             Destination").gameObject;
16|
17|     if (teleportationProvider == null)
18|     {
19|         teleportationProvider = FindObjectOfType<
20|             TeleportationProvider>();
21|
22|         TeleportRequest teleportRequest = new TeleportRequest
23|         {
24|             requestTime = Time.time,
25|             destinationPosition = defaultTeleportationTarget.
26|                 transform.position,
27|             destinationRotation = defaultTeleportationTarget.
28|                 transform.rotation
29|         };
30|
31|         StartCoroutine(FadeSequence(teleportRequest));
32|     }
33|
34| }

```

Código 2.3: Rutina de teletransportación

### 2.5.2. Sistema de Reconocimiento de Voz

Para completar el Sistema de Búsqueda y proponer otra manera de ingresar texto al campo de búsqueda, pensamos que sería buena idea implementar un Mecanismo de Reconocimiento de Voz en tiempo real con IA para reducir los tiempos de tecleo. La principal limitación era que para conseguirlo necesitábamos una herramienta offline, de forma que el proyecto no dependiera de la conexión a internet y los datos se mantuvieran seguros y confidenciales. Esto era bastante difícil dado que los mejores sistemas gratuitos de reconocimiento de voz requieren conexión o si no, son servicios pagos.

Después de investigar un poco encontré algunas alternativas, pero necesitaba probarlas y ver cuál funcionaba mejor.

■ **Servicios fuera de línea:**

- **DeepSpeech:** Motor Speech-To-Text de código abierto, que utiliza un modelo entrenado mediante técnicas de aprendizaje automático basado en el trabajo de investigación Deep Speech de Baidu. El proyecto no ha recibido actualizaciones desde 2020 y no hay soporte para las últimas versiones de Python, por lo que para usarlo es necesario utilizar una herramienta como win-pyenv para gestionar versiones antiguas. Siguiendo el único tutorial online, conseguí instalarlo pero el reconocimiento no fue bueno, alrededor de un 20 % y un 30 % de efectividad.
- **Vosk:** También es un kit de herramientas offline pero con soporte en diferentes idiomas. En su página web principal ofrecen un ejemplo de Unity, pero los resultados no fueron buenos y no se reconoció ninguna palabra.
- **Unity Keyword Recognizer:** Haciendo uso del servicio de reconocimiento de voz de Windows, podemos reconocer palabras y compararlas con comandos de la lista para activar eventos específicos. Inconveniente: hay que añadir manualmente cada comando a la lista y la llamada de retorno correspondiente.
- **WhisperAI:** Normalmente WhisperAI requiere una API y pagas cada vez que lo usas, pero también puedes instalarlo localmente usando la librería pip para Python.
- **Undertone:** Un activo de pago en Unity Store que también funciona sin conexión con WhisperAI.

■ **Servicios en línea:**

- **Unity Dictation Recognizer:** Funciona igual que el Keyword Recognizer pero para identificar frases aleatorias necesitamos conexión a internet. Funciona bastante bien y reconoce los idiomas añadidos en la configuración de Windows del PC.
- **Hugging Face:** Es una IA online gratuita que proporciona una API con Unity y se encarga del reconocimiento de voz.

- **Recognissimo:** Es un activo de pago de Unity que también se encarga del reconocimiento de voz. Técnicamente está basado en el sistema Vosk.

Habiéndolo consultado con mi equipo y mi supervisor, decidimos que implementaría-  
mos la versión offline de WhisperAI y también probaríamos el Dictation Recognizer de  
Unity, dado que depende de Unity y Windows y usar la conexión a internet para eso era  
aceptable. El sistema de búsqueda funciona de la misma manera, sólo estamos cambiando  
el método de entrada.

DictationRecognizer escucha la entrada de voz durante un período determinado y  
devuelve el resultado del reconocimiento, si es que hay [11]. En caso de superarse el tiempo  
límite de escucha o si se corta la conexión, actualizaremos el valor del flag correspondiente  
para comunicárselo al usuario con el cuadro de mensaje del buscador.

Los usuarios pueden registrarse y escuchar los eventos de hipótesis y frase completada.  
Los métodos Start() y Stop() activan y desactivan respectivamente el reconocimiento del  
dictado. Una vez que se ha terminado con el reconocedor, debe ser desecharo utilizando  
el método Dispose() para liberar los recursos que utiliza. Liberará estos recursos automá-  
ticamente durante la recolección de basura con un coste adicional de rendimiento si no se  
liberan antes.

```

1  public void StartDictationEngine()
2  {
3      textToPrint = "";
4      if (isListening == false) // quick verification there are
5          no other current dictation sessions.
6      {
7          isListening = true;
8          dictationRecognizer = new DictationRecognizer();
9          dictationRecognizer.AutoSilenceTimeoutSeconds = 5;
10
11         dictationRecognizer.DictationHypothesis +=
12             DictationRecognizer_OnDictationHypothesis;
13         dictationRecognizer.DictationResult +=
14             DictationRecognizer_OnDictationResult;
15         dictationRecognizer.DictationComplete +=
16             DictationRecognizer_OnDictationComplete;
17         dictationRecognizer.DictationError +=
18             DictationRecognizer_OnDictationError;

```

```

14
15         dictationRecognizer.Start();
16
17         flagFoundObjects = 4; // Listening for input
18         SearchManager.GetComponent<SearchScript>().
19             UpdateResultsButton(flagFoundObjects);
20     }
21 }
22
23 public void CloseDictationEngine()
24 {
25     Debug.Log("Result: " + textToPrint);
26     inputField.text = textToPrint;
27     flagFoundObjects = 0;
28     isListening = false;
29     if (dictationRecognizer != null)
30     {
31         dictationRecognizer.DictationHypothesis ==
32             DictationRecognizer_OnDictationHypothesis;
33         dictationRecognizer.DictationComplete ==
34             DictationRecognizer_OnDictationComplete;
35         dictationRecognizer.DictationResult ==
36             DictationRecognizer_OnDictationResult;
37         dictationRecognizer.DictationError ==
38             DictationRecognizer_OnDictationError;
39
40         if (dictationRecognizer.Status == SpeechSystemStatus.
41             Running)
42             dictationRecognizer.Stop();
43
44         dictationRecognizer.Dispose();
45     }
46 }
```

Código 2.4: Start y Stop Dictation Recognizer

Por otro lado, la versión offline de WhisperAI fue muy interesante de usar ya que sólo puede leer archivos de audio y no voz en tiempo real [12]. Esto añadía un nuevo paso en el proceso de reconocimiento ya que primero teníamos que grabar la voz y luego interpretarla con la librería instalada. Desarrollé un script en Python que era llamado desde Unity cuando queríamos buscar algo por voz, y primero registraría la voz en un archivo, luego reconocería el texto en él y lo enviaría de vuelta a Unity abriendo un proceso e imprimiendo en él. Unity estaría escuchando cualquier dato de entrada del proceso abierto después de llamar al script de Python. Dado que ahora contamos con un lapso de tiempo de grabación de sonido y luego un tiempo de reconocimiento, fue necesario aumentar los valores posibles del flag que utilizamos para modificar el mensaje del cuadro de texto de estado de búsqueda. El mecanismo desarrollado indica al usuario través de un temporizador en qué momento hablar y que espere el procesamiento del audio cuando termine de grabar.

Para controlar el tiempo de procesamiento de audio, podemos seleccionar el tamaño del modelo de WhisperAI en inglés, por lo que podemos optimizar recursos u optar por un mejor reconocimiento según se prefiera.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	<code>tiny.en</code>	<code>tiny</code>	~1 GB	~32x
base	74 M	<code>base.en</code>	<code>base</code>	~1 GB	~16x
small	244 M	<code>small.en</code>	<code>small</code>	~2 GB	~6x
medium	769 M	<code>medium.en</code>	<code>medium</code>	~5 GB	~2x
large	1550 M	N/A	<code>large</code>	~10 GB	1x

Figura 2.12: Modelos de entrenamiento de Whisper AI

El código siguiente muestra la rutina de registro de audio junto con la interpretación con WhisperAI y la apertura del proceso de comunicación.

```

1 # Initialize the audio stream
2 audio = pyaudio.PyAudio()
3
4 # Open a new audio stream for recording
5 stream = audio.open(format=FORMAT, channels=CHANNELS,
6                      rate=RATE, input=True,
7                      frames_per_buffer=1024)

```

```

8
9 frames = []
10
11 # Record audio data in chunks and save to frames list
12 for _ in range(0, int(RATE / 1024 * RECORD_SECONDS)):
13     data = stream.read(1024)
14     frames.append(data)
15
16 # Close and terminate the audio stream
17 stream.stop_stream()
18 stream.close()
19 audio.terminate()
20
21 # Save the recorded audio to a WAV file
22 with wave.open(output_file_path, 'wb') as wf:
23     wf.setnchannels(CHANNELS)
24     wf.setsampwidth(audio.get_sample_size(FORMAT))
25     wf.setframerate(RATE)
26     wf.writeframes(b''.join(frames))
27
28 # Run Whisper ASR processing on the recorded audio
29 whisper_command = f"whisper {output_file_path} --model base --
30   language English"
31 process = subprocess.Popen(whisper_command, stdout=subprocess.
32   PIPE, stderr=subprocess.PIPE, shell=True)
33 stdout, stderr = process.communicate()
34
35 # Print the output from Whisper ASR
36 whisper_output = stdout.decode()

```

Código 2.5: Registro de Audio y reconocimiento con WhisperAI

La incorporación del Reconocimiento de Voz nos deja un esquema de funcionamiento como el siguiente. El SearchManager recibe texto por teclado o por voz y actualiza los resultados de la búsqueda.

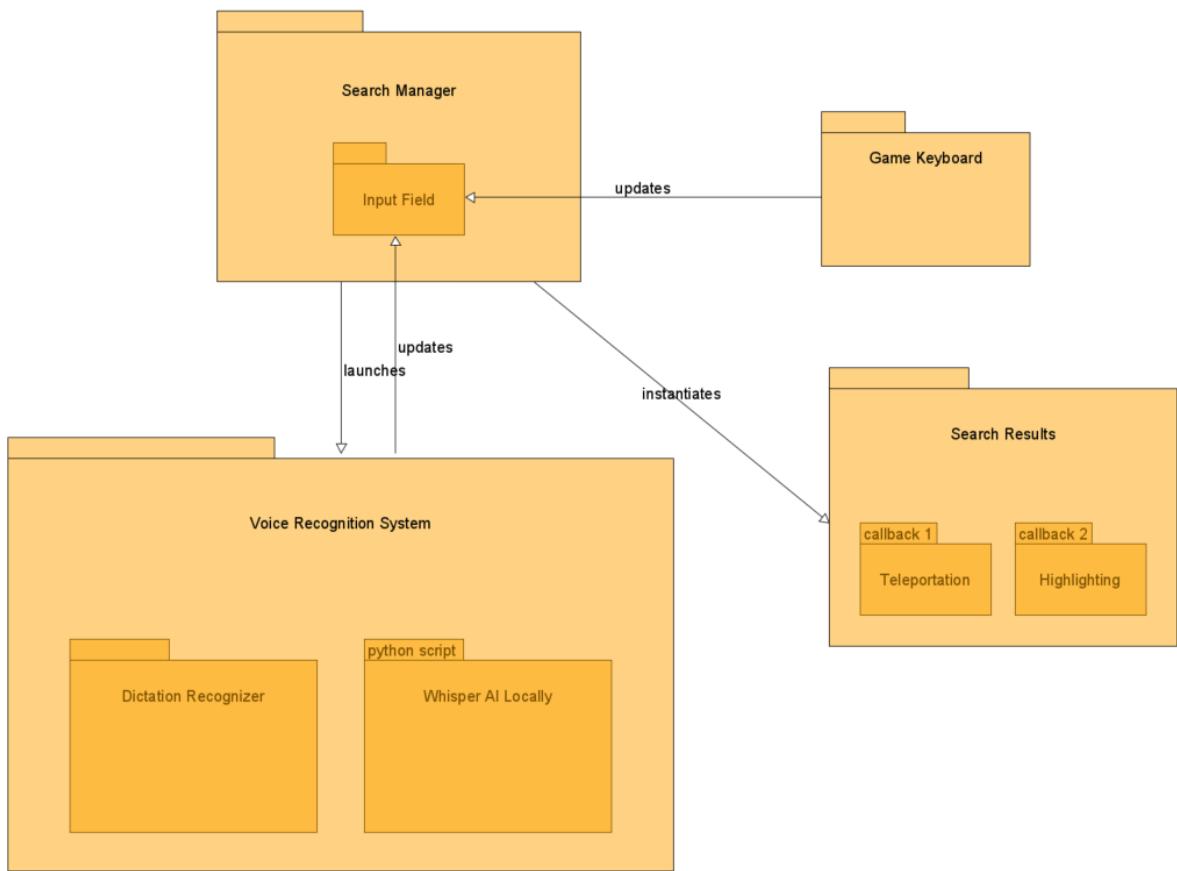


Figura 2.13: Diagrama de Paquetes del Sistema de Búsqueda

### 2.5.3. Menú Jerárquico

Esta característica fue desarrollada al principio de mi pasantía y en ese momento las ontologías eran diferentes a las que estamos usando ahora, por lo que esta desactualizada y requiere ser reestructurada para ser agregada al formato actual del proyecto.

La idea era tener una pestaña en el menú de mano donde pudiéramos ver rápidamente qué clases había en la escena según su nivel en la ontología. La inspiración para esto fue la interfaz del sistema de carpetas del Explorador de Windows. Vemos una carpeta y, al hacer click en ella, expandimos el contenido que hay dentro y vemos las subcarpetas. Así podemos navegar por nuestro sistema de datos y explorar todo lo que hay dentro. Esto era muy similar a nuestra antigua estructura ontológica, ya que teníamos un objeto Raíz y a partir de él empezábamos a conectar clases, de forma que cada una de ellas estaba conectada de alguna manera a ese objeto raíz. Esto significaba que podíamos hacer click

y expandir una clase, para ver qué clases derivaban de ella, según su nivel en la ontología. En la Figura 2.8, podemos ver una ilustración de como sería este funcionamiento.

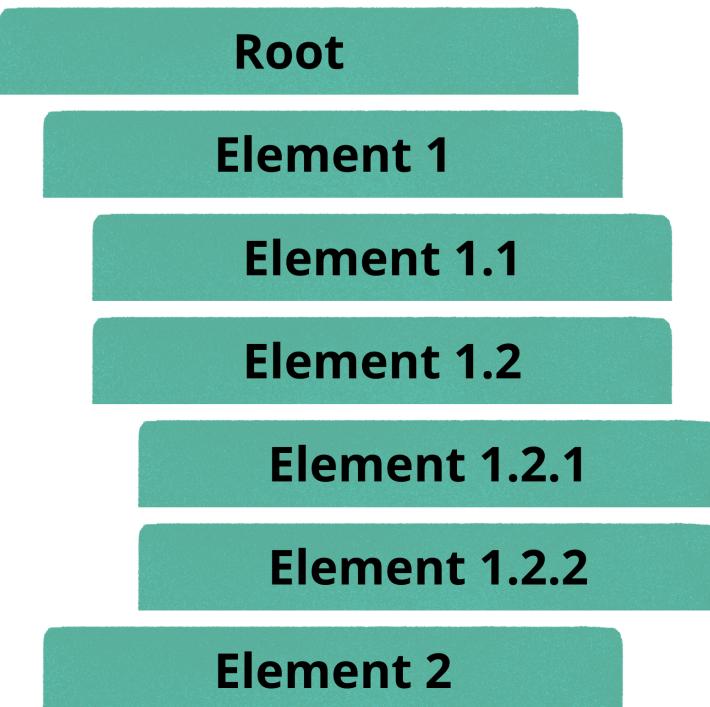


Figura 2.14: Ilustración gráfica de funcionamiento

La herramienta estaba implementada y funcionaba correctamente, y cada botón del menú también hacía referencia a las burbujas correspondientes, por lo que también teníamos funcionalidades de resaltado/teletransportación asociadas a ellas. Todo esto cambió cuando empezamos a importar ontologías desde ficheros jsonld y el concepto de objeto raíz dejó de existir. Ahora contamos con múltiples ontologías en simultáneo y hay varias burbujas que no heredan de ninguna otra.

#### 2.5.4. Importación desde archivos Jsonld

Dado que no poseemos muchas herramientas de creación y edición de ontologías, salvo las pocas que he explicado, debemos tener una forma de importar datos de ontologías que hayan sido creadas por softwares externos. Luego de consultar con el equipo de Gaia-X, empezamos a planear el desarrollo de una interfaz de importación de archivos de formato .jsonld. El proceso de conversión es el resultado de un cuidadoso estudio por lo que necesita

una explicación clara.

Dentro de nuestro simulador en Unity desarrollamos un sistema para guardar y cargar la información de todas las burbujas instanciadas en ese momento. Esto se realiza guardando en un archivo json la información básica de cada burbuja, como vimos en la sección 2.3, junto con los 3 valores (x,y,z) de la posición. Esto nos va a permitir cargar la información y reconstruir las ontologías en sesiones posteriores del visualizador. Podemos mover objetos y guardar la posición, sabiendo que el trabajo queda registrado y no se pierde el progreso realizado.

Lo que nos falta es la conversión de datos entre los archivos descargados directamente de Gaia-X a nuestro tipo de archivo json que podemos leer y cargar en la escena. Para eso desarrollamos un objeto que al inicio de la escena, lee los archivos jsonld que le indiquemos y busca los datos necesarios para conformar un único archivo json que tendrá todas las clases que existan dentro de los archivos del ecosistema Gaia-X. La información que no haya sido utilizada para la importación, es conservada para el proceso de exportación cuando queramos recrear los archivos jsonld originales.

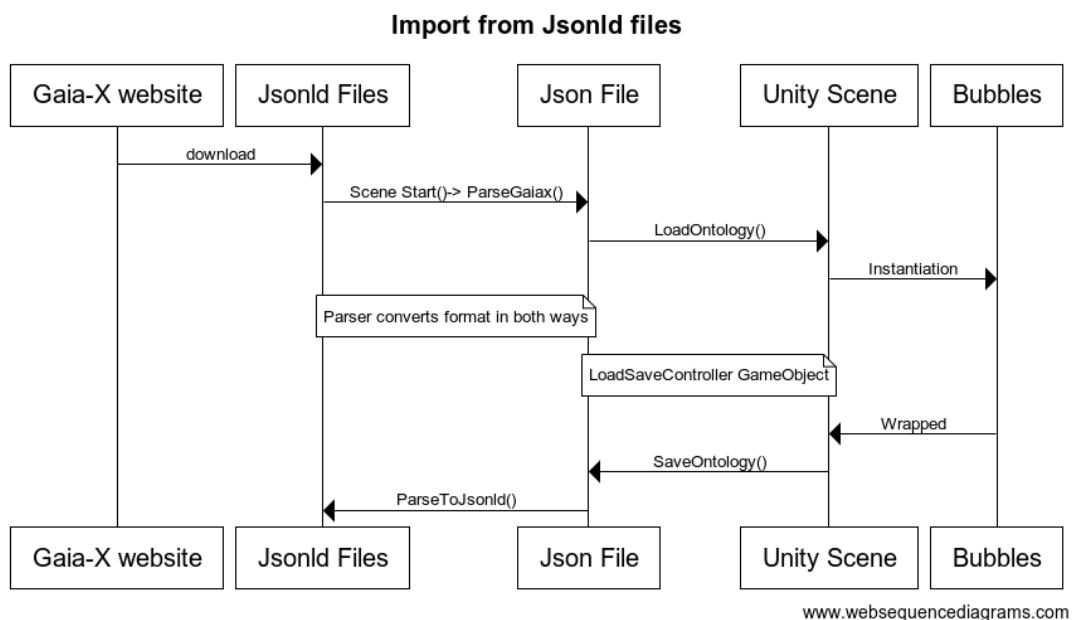


Figura 2.15: Diagrama de secuencia de importación de ontologías

La forma en que filtramos la información es con una secuencia de verificaciones de condición que comprueban la existencia de diferentes claves o propiedades json que nos guían y nos dicen qué información tomar y cuál dejar, ya que el fichero jsonld contiene

mucho material que no vamos a utilizar en nuestra escena, pero que necesitamos conservar para cuando exportemos la escena.

A partir de los archivos jsonld que hemos descargado, buscamos en primer lugar objetos del tipo *Class*, los cuales poseen el valor (“owl#Class”) dentro de la propiedad *type*. Con estos objetos podremos crear una lista con las burbujas a importar, asignarles un ID numérico, determinar a qué ontología pertenecen y un valor de posición, utilizando un archivo complementario. Estos datos son los que le dan identidad a cada burbuja individual, sin ninguna especificación de relaciones o conexiones con otras burbujas.

Los archivos que describen ontologías están enfocados en la información relacionada a las clases presentes junto con sus conexiones, por lo que no contiene indicaciones sobre la intérpretación gráfica de la estructura. Esto significa que nosotros mismos debemos encontrar una manera de posicionar las burbujas dentro de la escena. Una forma sería desarrollar un algoritmo que determine valores de posición y busque optimizarlos según algún criterio como longitud de las conexiones, distribución de los objetos, espacio, etc. La otra opción consiste en utilizar una herramienta externa que se encarga de resolver este problema pero en dos dimensiones. Lo que hacemos es, inmediatamente después de obtener los archivos de Gaia-X, cargarlos en este software y descargar el archivo con la información de la distribución gráfica. En el momento de instanciar clases en nuestro visualizador, accedemos a este último archivo y obtenemos los valores *x* e *y* de cada clase y asignamos un valor *z* por defecto para darle la misma profundidad a todas las burbujas. Esto nos da como resultado una representación plana de la ontología dentro de nuestra escena 3D, pero ya hemos solucionado 2 de 3 valores de posición. El último lo resolveremos más tarde.

```

1      if (type.Contains("owl#Class") && id.Contains("http://w3id.org/gaia-x/"))
2      {
3          if (!encounteredIDs.Contains(id)) // Check if
4              the ID has not been encountered before.
5          {
6              encounteredIDs.Add(id);
7              string[] parts = id.Split('/');
8              separates the url in the iri base url
9              and the ontology+class
10             string result = parts[parts.Length - 1];
11             // result = ontology+class

```

```

8
9         parts = result.Split('#');
10        string tag   = parts[0]; // tag = ontology
11
12        newBubbleData.elementName = result;
13        // we'll only display the class name in
14        // the bubble but the class can be
15        // repeated in different ontologies, so we
16        // still need to specify which one it is
17        newBubbleData.elementType = "Class";
18        newBubbleData.elementID = IDcounter.
19                      ToString();
20        newPositionData.bubbleID = newBubbleData.
21                      elementID;
22
23        float[] pos = FindPosByIRIValue(
24                      classAttributesArray, id);
25        newPositionData posX = pos[0];
26        newPositionData posY = pos[1];
27
28        classObjects.Add(jsonLdObject);
29        ontology.extractedBubbleDatasList.Add(
30                      newBubbleData);
31        positions.extractedPositionDataList.Add(
32                      newPositionData);
33        IDcounter++;
34    }
35}

```

Código 2.6: Búsqueda de información para la creación de clases

Una vez tengamos una lista con todas las clases de los archivos a importar, podemos empezar a buscar conexiones entre ellas. Esto se hace buscando dentro de los mismos archivos, pero en este caso queremos encontrar objetos del tipo “*owl#ObjectProperty*” y todos aquellas clases que posean la propiedad “*owl#subClassOf*”, para encontrar ambos tipos de conexiones. Cada vez que encontrremos cualquiera de estos casos, obtendremos

de nuestra lista de clases las burbujas pertinentes e iremos completando la información faltante sobre sus conexiones.

Para entender mejor cómo funciona la herramienta de importación, será más fácil si utilizamos un ejemplo de conversión.



Figura 2.16: Conversión de clase y conexión a formato interno

Gaia-X utiliza un estándar de información con identificadores web permanentes [13] para mantener un nivel de comprensión, especificación y gobernanza en los datos. Esto se logra construyendo sobre la comprensibilidad universal asegurada por los estándares del *World Wide Web Consortium* (W3C) [14]. Esto hace que gran parte del texto dentro de los archivos a importar sólo sirva dentro de este esquema de información, pero no es necesario utilizarlo en nuestra escena. Por esto, nuestro código se encarga principalmente de filtrar texto y conservar únicamente las porciones de *string* que necesitemos.

Cuando encontramos una clase, tenemos que deshacernos del URL en el id y quedarnos con el nombre real de la clase y la ontología a la que pertenece y asignarle un ID de objeto de juego único. Para ello, nos fijamos en el id y el texto después del último separador “/”. El carácter # separará la etiqueta de la ontología a la izquierda y el nombre de la clase a la derecha. Repetimos esta operación por cada objeto json de la lista hasta que hayamos guardado todos los objetos de tipo clase.

En el caso de las conexiones de subclase, simplemente tenemos que observar si las clases poseen una propiedad que indique la herencia (*owl#subClassOf*). Este campo nos señalará directamente la burbuja faltante para completar las listas de ambos extremos de la conexión. Por otro lado, cada propiedad de objeto contiene información sobre el id de la conexión, el origen (dominio) y el destino (rango). Nuestra tarea consiste en buscar las burbujas especificadas en los extremos de la propiedad de objeto y agregarse mutuamente en las listas de conexión.

Así, por ejemplo, la conexión mostrada en la Figura 2.16 tiene el id de la conexión con dominio y rango. En este caso concreto, lo leeríamos así : "La clase *Service Offering* de la ontología *service* **produce** la clase *ApiDescription* de la ontología *service*". El URL subClassOf de la otra subfigura de la izquierda nos indicará de qué clase estamos heredando en este momento. En este caso, " *ApiDescription* de la ontología *service* **hereda** de la clase *Consumable* de la ontología *core*". Lo que hacemos ahora es añadir estas conexiones en las listas correspondientes con nuestro propio formato. La conexión de propiedad objeto se añade en la lista "ownedByBubblesList" con el id del origen (dominio) y su etiqueta. En el otro extremo, añadiremos el id del destino o rango (1007 en este caso) en la lista "hasBubblesIDsList". La conexión subClass funciona de forma similar. Nuestra *ApiDescription* hereda de la burbuja 1000, que se añade a la "inheritedBubblesList".

```
1     private void FindConnections(string domain, string range,
2         string label)
3     {
4         for (int i = 0; i < propertyObjects.Count; i++)
5         {
6             JObject propertyObject = propertyObjects[i];
7             if (propertyObject.ContainsKey(domain) &&
8                 propertyObject.ContainsKey(range))
9             {
10                 string domainID;
11                 string rangeID;
12                 JArray domainArray = propertyObject[domain] as
13                     JArray;
14                 JArray rangeArray = propertyObject[range] as
15                     JArray;
16
17                 JObject domainObject = domainArray[0] as JObject;
```

```

14     JObject rangeObject = rangeArray[0] as JObject;
15
16     domainID = domainObject["@id"].Value<string>();
17     rangeID = rangeObject["@id"].Value<string>();
18
19     int domainIndex = FindIndexOfBubbleById(ontology.
20         extractedBubbleDatasList, domainID);
21     int rangeIndex = FindIndexOfBubbleById(ontology.
22         extractedBubbleDatasList, rangeID);
23
24     string connectionID = propertyObject["@id"].Value
25         <string>();
26     string[] parts = connectionID.Split("#");
27     string labelName = parts[parts.Length - 1];
28
29     VerifyListsForRepetitionAndAdd(domainIndex,
30         rangeIndex, labelName);
31
32 }
33 }
```

Código 2.7: Rutina de búsqueda de conexiones

```
{
  "id": "8",
  "iri": "http://w3id.org/gaia-x/service#ApiDescription",
  "baseIri": "http://w3id.org/gaia-x/service",
  "label": {
    "IRI-based": "ApiDescription",
    "en": "API description"
  },
  "pos": [
    420.03,
    497.54
  ]
},
```

Figura 2.17: Objeto Json del archivo de posiciones

La figura 2.17 muestra un objeto json de posición. Lo único que debemos hacer es

buscar el id (url) de la clase en cuestión dentro del campo “iri” de los objetos json del archivo y recuperar los valores de la propiedad "pos".

### 2.5.5. Exportación a archivos Jsonld

El proceso de Exportación es muy similar, ya que leerá toda la información actual de cada burbuja en la escena y la guardará en un archivo json. Nuestro Parser tomará cada elemento y creará la clase Gaia-X correspondiente en un archivo jsonld. La lógica sigue siendo la misma, crear todas las clases y una vez que hayamos terminado, crear todas las conexiones. La única diferencia radica en que nuestro formato de guardar/cargar siempre especifica los mismos campos de propiedad dentro de cada objeto json, incluso si no contienen información. En cambio, los archivos de Gaia-X sólo especifican las propiedades que tengan un valor asociado. Para cumplir con el estándar, es necesario utilizar un Diccionario y especificar las propiedades y valores a través de las *keys* del mismo. En el siguiente fragmento de código podemos ver esta implementación, donde *dataToExport* es el nombre del diccionario que utilizamos.

```

1      if (subClassArray.Count > 0)
2      {
3          List<GenericObject> subClassObjects = new();
4          List<string> names = GetNamesFromIDs(
5              subClassArray, bubblesDataList, true);
6          subClassObjects.AddRange(names.Select(name =>
7              new GenericObject { ID = name }));
8          dataToExport["http://www.w3.org/2000/01/rdf-
9          schema#subClassOf"] = subClassObjects;
10     }

```

Código 2.8: Ejemplo de creación de llave en el Diccionario a exportar

Esta herramienta de Importación/Exportación es crucial para el proyecto. Sin ella, no tendríamos la interoperabilidad necesaria. Pero para ello, debemos estar seguros de que no estamos perdiendo información en ningún paso o dirección del proceso. Esto significa que tenemos que comprobar todas las listas con frecuencia y buscar anomalías y datos

irregulares.

Una vez que terminemos las principales tareas de visualización, podremos enfocarnos en tareas de edición y creación por lo que las ontologías sufrirán modificaciones dentro de nuestra aplicación y si esta herramienta funciona correctamente, deberíamos ver los resultados en los archivos exportados.

### 2.5.6. Contenedores de Ontologías

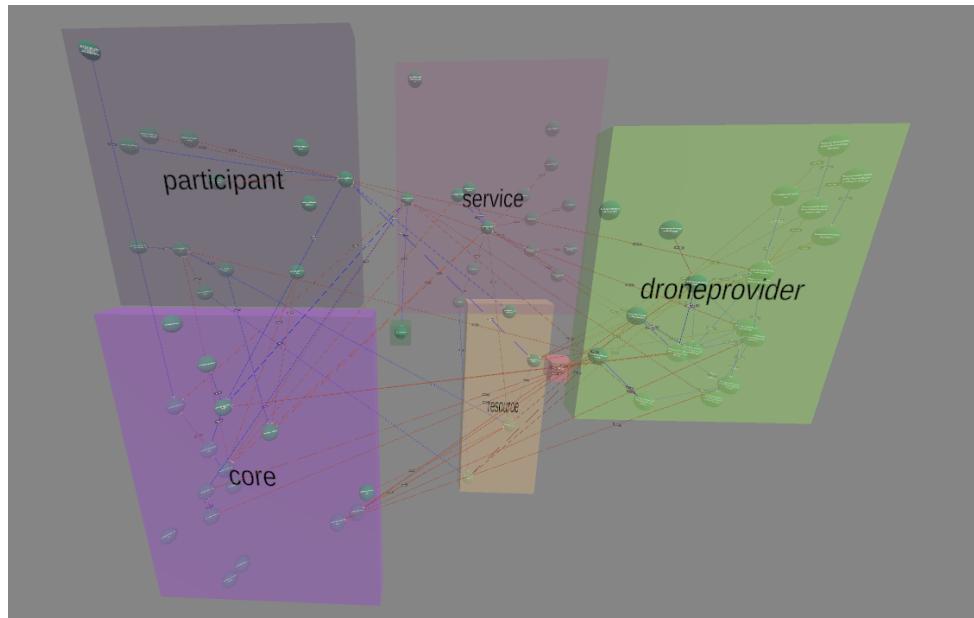


Figura 2.18: Ontologías y sus contenedores

La idea de esta característica es delimitar gráficamente cada ontología de manera que podamos diferenciarlas fácilmente. Además, cada contenedor debe ser interactivo para que uno pueda seleccionarlos y moverlos en el espacio, y al hacerlo, todas las clases contenidas se moveran dentro del bloque. Esto creará una nueva capa de interacción en nuestra simulación, permitiéndonos interactuar tanto con burbujas como con ontologías completas. A partir de aquí, podemos añadir nuevas características que también se aplican a todas las clases de una ontología.

El diagrama de clases de la Figura 2.19 nos muestra cómo están conectados los scripts en el código. *OntologyManager* se encarga de instanciar los prefabs y rellenar las listas correspondientes de cada uno de los subgestores. Entonces:

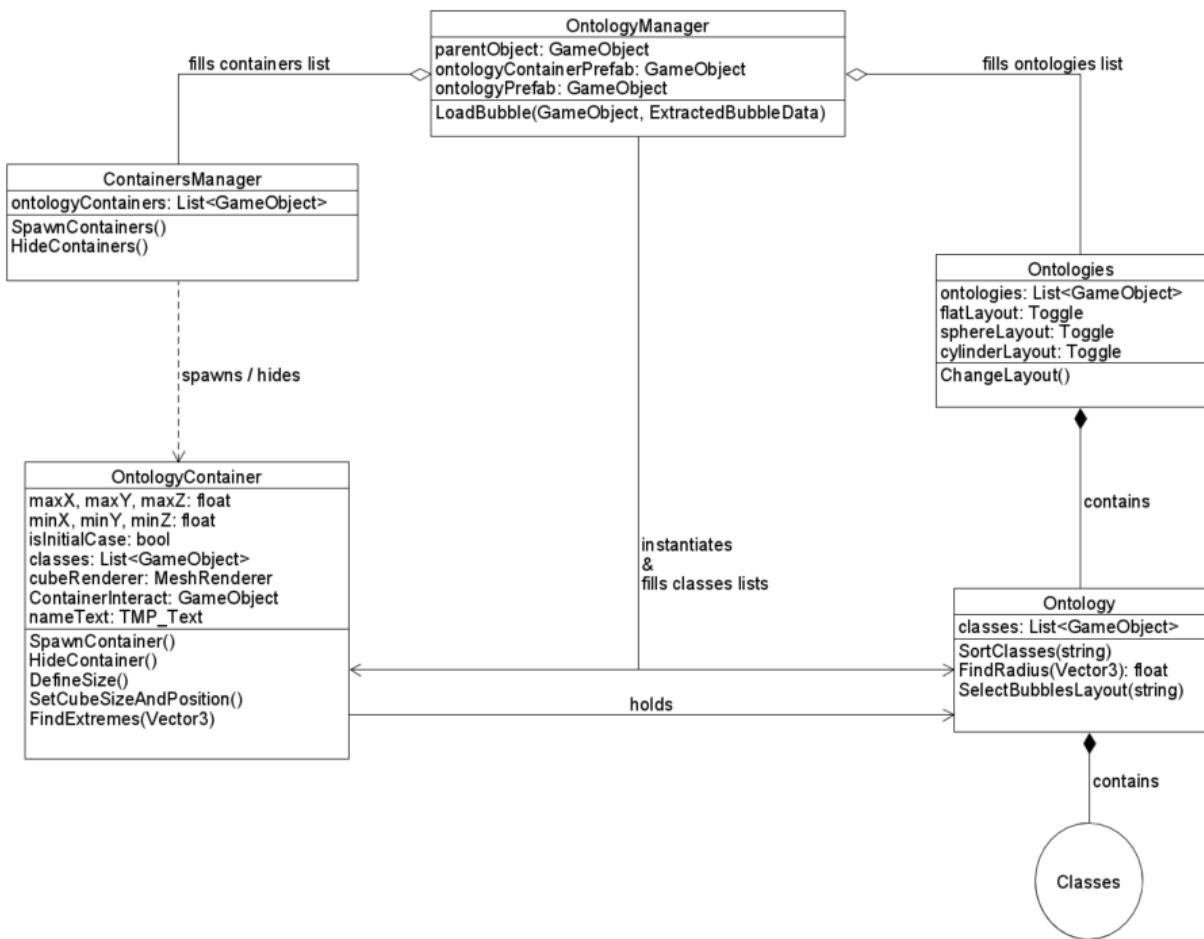


Figura 2.19: Diagrama de clases para los contenedores

- Todas las burbujas de una misma ontología están contenidas dentro del objeto padre *Ontology*.
- Todas las ontologías están contenidas en un objeto padre mayor *Ontologies*.
- Cada ontología tiene asociado un Container que va a encuadrar todas las burbujas correspondientes.
- El ContainersManager tiene referencias al Container de cada ontología.

Cada vez que creamos una nueva burbuja, debemos verificar si existe la ontología a la que pertenece. Si ya ha sido creada, tenemos que agregarla a la lista de burbujas. En caso contrario, debemos crear la ontología, asociarle un container, y agregar todos estos

nuevos objetos a las listas correspondientes según la jerarquía. Podemos ver el siguiente fragmento de código para entender mejor los comandos. Esta secuencia la podemos ver en la Figura 2.20 más claramente.

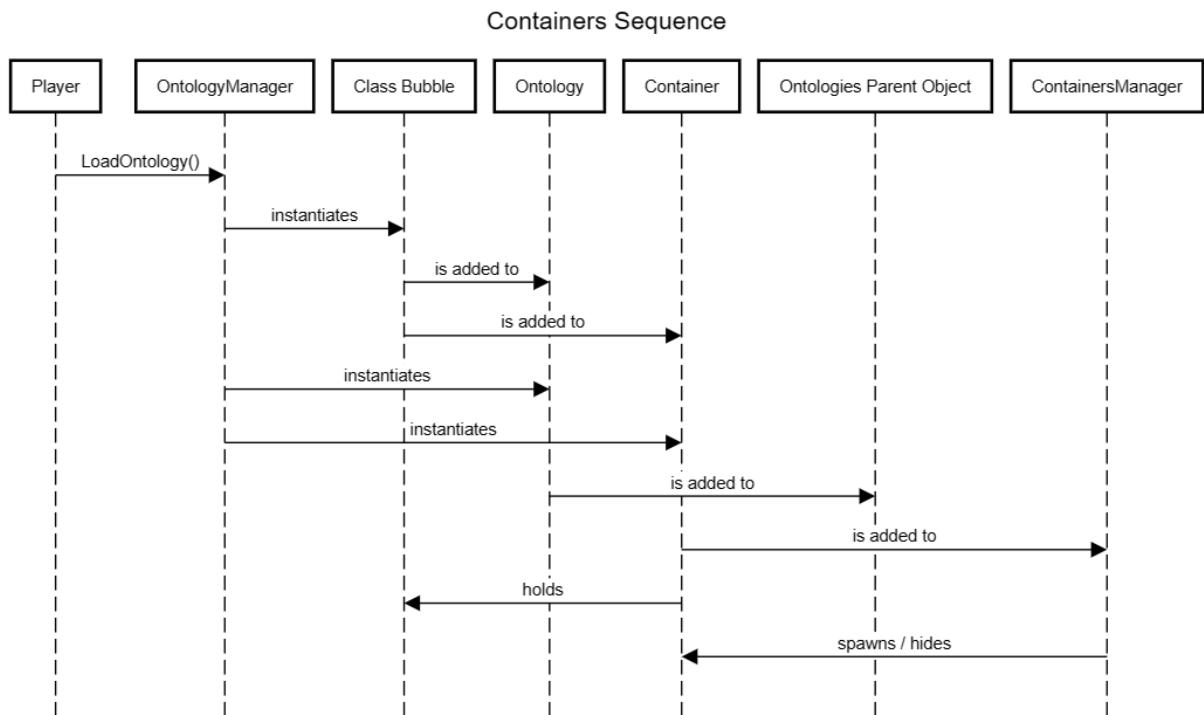


Figura 2.20: Diagrama de secuencia para la creación de clases y containers

```

1 List<GameObject> ontologies = parentObject.GetComponent<
2     Ontologies>().ontologies;
3 GameObject foundOntology = ontologies.FirstOrDefault(obj
4         => obj.name == data.ontologyTag);
5
6 if (foundOntology != null)
7 {
8     bubbleObject.transform.SetParent(foundOntology.
9         transform);
10    foundOntology.GetComponent<Ontology>().classes.Add(
11        bubbleObject);
12 }
13 else
14 {
15 }
  
```

```

11 // instantiating
12 GameObject newOntology = Instantiate(ontologyPrefab);
13 newOntology.name = data.ontologyTag;
14 GameObject ontologyContainer = Instantiate(
15     ontologyContainerPrefab);
16
17 // adding to lists
18 parentObject.GetComponent<ContainersManager>().
19     ontologyContainers.Add(ontologyContainer);
20 parentObject.GetComponent<Ontologies>().ontologies.
21     Add(newOntology);
22 newOntology.GetComponent<Ontology>().classes.Add(
23     bubbleObject);
24
25 // spawning network objects
26 ontologyContainer.GetComponent<NetworkObject>().Spawn
27     ();
28 newOntology.GetComponent<NetworkObject>().Spawn();
29
30 // assigning parents
31 ontologyContainer.transform.SetParent(newOntology.
32     transform);
33 newOntology.transform.parent = parentObject.transform
34     ;
35 bubbleObject.transform.SetParent(newOntology.
36     transform);
37
38 }

```

Código 2.9: Crear burbujas, ontologías y containers

ContainersManager.cs tiene el control de la funcionalidad, por lo que es el componente que necesitamos para los métodos de callback.

OntologyContainer.cs pertenece al objeto container y tiene los métodos de control respectivos para calcular su tamaño, posición y mostrarlo o esconderlo según sea requerido.

Cada contenedor también tiene un script VRInteractable para que podamos moverlo con el raycast de nuestro controlador. Lo bueno de esto es que cada clase contenida en él también se moverá en el mismo vector de desplazamiento y magnitud, por lo que todo

se comportará como un bloque. Con esto conseguimos la capa de interacción mencionada anteriormente.

### 2.5.7. Modos de distribución de ontologías

Esta característica pretende complementar el proceso de importación, ya que estamos asignando un valor de tercera dimensión al vector de posición para instanciar la burbuja, pero todas tienen el mismo, por lo que estamos perdiendo el efecto 3D tan necesario. Para superarlo, vamos a utilizar formas geométricas básicas para describir cada ontología. La idea consiste en determinar un objeto 3D con una forma específica, que englobe y contenga a las burbujas de cada ontología. Luego, utilizando la función del cuerpo que hayamos elegido, buscamos el valor  $z$  para cada par  $x$  e  $y$  de cada burbuja. Esto va a generar una proyección de la ontología completa segun una curva deseada.

```

1      case "cylinder":
2
3          /* radius will be calculed with the list in
4             crecent order according to x position,
5             so we need to sort first by z and then do x. Plus
6             the first and last element wont be moved and
7             will serve as
8             opposites in the side of the cylinder
9             */
10            SortClasses("z");
11            float zClosestValue = classes[0].transform.
12                position.z;
13            SortClasses("x");
14            float radius = (classes[classes.Count - 1].
15                transform.position.x - classes[0].transform.
16                position.x) / 2.0f;
17            float cylinderCenter = classes[0].transform.
18                position.x + radius;
19            if (radius > 0) // this means that the ontology
20                has more than 1 element
21            {
22                for (int i = 0; i < classes.Count; i++)
23                {

```

```

16         float acosArgument = (classes[i].
17             transform.position.x - cylinderCenter)
18             / radius;
19         if(Mathf.Abs(acosArgument) > 1)
20         {
21             Debug.LogWarning("cosine out of order
22                 with a value of " + acosArgument);
23             if (acosArgument > 1) acosArgument =
24                 1;
25             else if (acosArgument < -1)
26                 acosArgument = -1;
27         }
28         float angle = Mathf.Acos(acosArgument);
29         float zNewPosition = zClosestValue +
30             Mathf.Sin(angle) * radius;
31         Vector3 newPosition = new Vector3(classes
32             [i].transform.position.x, classes[i].
33             transform.position.y, zNewPosition);
34         classes[i].GetComponent<
35             Multiplayer_MoveObjects>().
36             Multiplayer_VR_MoveBubble(newPosition);
37     }
38 }
39 break;

```

Código 2.10: Ejemplo de forma cilíndrica

En la Figura 2.21, vemos un esquema de cómo funciona el ejemplo anterior. Tomamos las burbujas de una ontología, calculamos la distancia x entre las más extremas y la utilizamos para crear un radio de cilindro. A continuación, colocamos cada burbuja en el valor z de este cilindro según las otras dos coordenadas a modo de proyección.

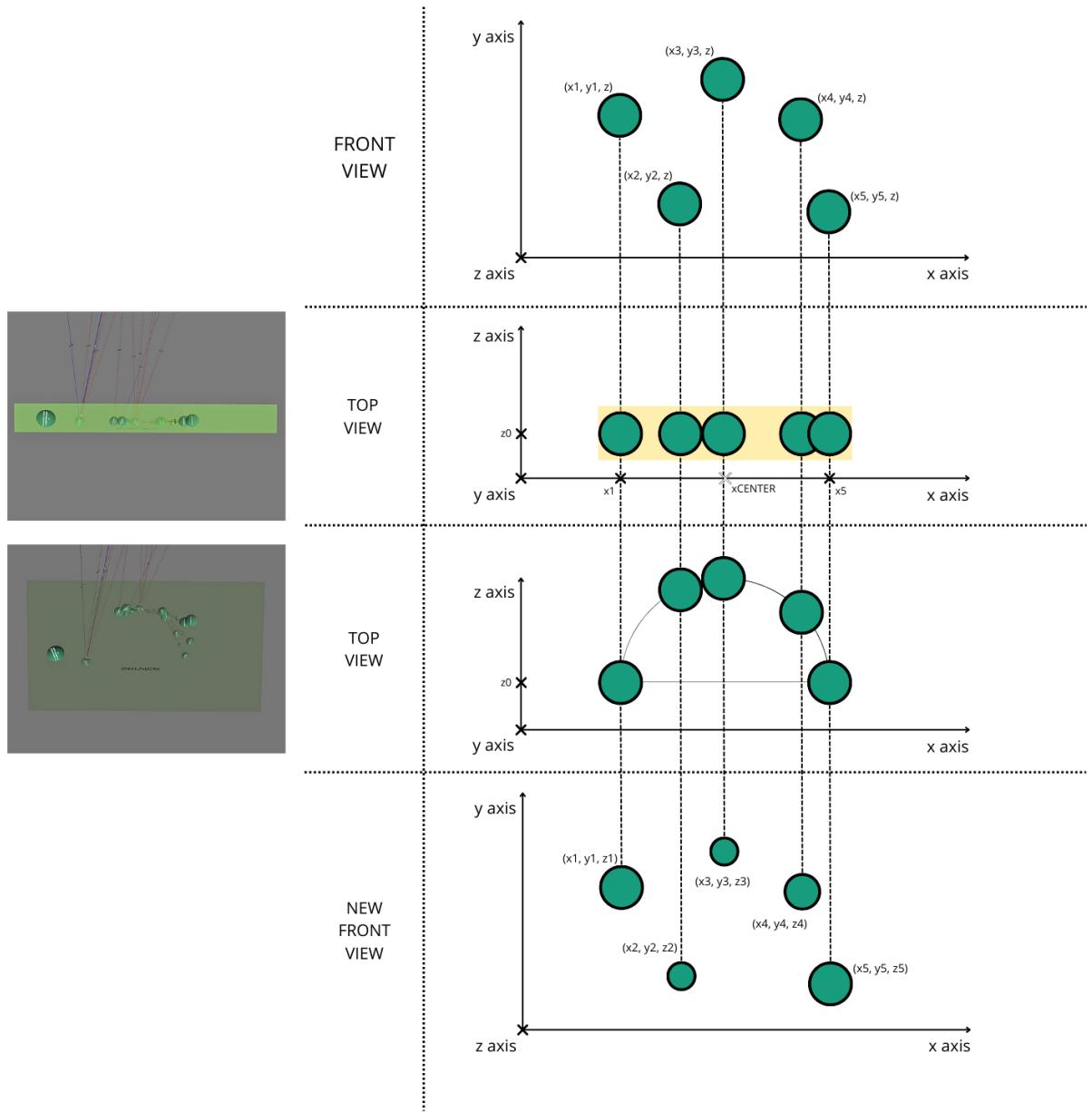


Figura 2.21: Cálculo de proyección para los modos de distribución

# CONCLUSIONES

---

El objetivo principal de estas prácticas era mejorar la herramienta 3D para visualizar ontologías, con el fin de tener una alternativa a los softwares 2D tradicionales y evitar algunas limitaciones que normalmente tienen.

La primera fase de mi contribución al proyecto estuvo orientada a filtrar las clases de la escena y desarrollar features para encontrar una clase lo más rápido posible. El sistema de búsqueda recibió en repetidas ocasiones muy buenas devoluciones de parte de mi supervisor, colegas y miembros del equipo de Gaia-X. Cada nueva herramienta fue pensada y planificada para encajar lo más fácilmente posible con el resto del proyecto. Así, logré agregar exitosamente una manera de leer archivos y construir una escena completa e interactiva respetando la información original. También conseguí desarrollar una manera de separar gráficamente las diferentes ontologías importadas y añadir niveles de interacción para darle dinamismo al visualizador.

En el desarrollo de Realidad Virtual, cada herramienta tiene que probarse y analizarse desde diferentes perspectivas, ya que la mayor parte del tiempo nos ocupamos de la experiencia del usuario. Tenemos que implementar las cosas de forma que sean intuitivas para el usuario y eso es algo que olvidamos con cierta frecuencia. Teníamos un miembro del equipo que se centraba exclusivamente en los prefabricados de interfaz de usuario, y teníamos que mantener actualizaciones constantes en nuestras ramas de trabajo con las versiones más recientes de los elementos de menú o interfaz de la escena.

También tuvimos que prestar mucha atención a la optimización para que la escena se ejecutara con la mayor fluidez posible y no rompiera la sensación de inmersión del usuario. Las simulaciones de RV dependen más del nivel de interacción entre el usuario y la escena que del realismo, como mucha gente puede creer, así que todo tiene que funcionar de una manera muy específica para satisfacer al usuario. Se realizan pruebas con frecuencia para intentar encontrar fallos y posibles agujeros en el código.

Trabajar en equipo es todo un reto y exige desarrollar una gran capacidad de comunicación. Ser capaz de expresar lo que hay que hacer o sugerir nuevas ideas depende en gran medida de la forma en que exponemos lo que pensamos.

Considero que luego de mi paso por el instituto y mi tiempo de trabajo, logré agudizar

---

mi comprensión de las implicaciones de un proyecto de Realidad Virtual, mejorar mis habilidades de programación, pulir mis herramientas de comunicación efectiva y practicar conocimientos adquiridos durante mi carrera en Ingeniería Mecatrónica.

## 2.6. Trabajo Futuro

El proyecto está lejos de estar terminado. Ahora que es posible importar y exportar información de los estándares de Gaia-X, se podría pensar en añadir herramientas de edición a la escena. Crear y borrar burbujas es el primer paso, pero luego también habría que añadir conexiones. Las clases deberían poder trasladarse de una ontología a otra, o incluso podrían crearse otras nuevas.

Como esta simulación forma parte del contexto del proyecto Gaia-X, también sería interesante llevar a cabo más pruebas y recoger ideas de en qué sería importante centrarse a continuación. Algunas personas han sugerido añadir una función multijugador, de modo que podamos utilizar la simulación tanto como una aplicación de coworking o como un tour de presentación. Esto significa que una persona interactúa con las ontologías y el resto observa lo que ocurre.

Se podría añadir otro intérprete para importar ontologías desde un formato de archivo diferente, como turtle. La idea es casi la misma que la que ya está implementada pero podría añadir mucha versatilidad al proyecto.

Como podemos ver, hay muchas cosas que se pueden hacer para seguir mejorando la simulación. La Realidad Virtual es una herramienta muy útil, pero aún no conocemos todo su potencial. Tenemos que seguir investigando, desarrollando y probando para encontrar nuevas formas de explorar la información.

# BIBLIOGRAFÍA

---

- [1] J. Scribani, *What is Extended Reality (XR)?*, 2019. dirección: <https://www.visualcapitalist.com/extended-reality-xr/>.
- [2] Oxford Semantic Technologies, *What is an Ontology?* Dirección: <https://www.oxfordsemantic.tech/faqs/what-is-an-ontology#:~:text=An%20ontology%20is%20a%20description,understanding%20of%20the%20data%20model..>
- [3] Gaia-X, *What is Gaia-X?* Dirección: <https://gaia-x.eu/>.
- [4] J. Vanwambeke, *The Role of Ontologies in Gaia-X*, 2023. dirección: <https://gaia-x.eu/news-press/the-role-of-ontologies-in-gaia-x/>.
- [5] W. Wiki, *Ontology Editors*. dirección: [https://www.w3.org/wiki/Ontology\\_editors](https://www.w3.org/wiki/Ontology_editors).
- [6] M. Vigo, «Overcoming the pitfalls of ontology authoring: Strategies and implications for tool design», *International Journal of Human-Computer Studies*, vol. 72, n.º 12, págs. 835-845, 2014.
- [7] N. F. Noy y D. L. McGuinness, *A Guide to Creating Your First Ontology*. dirección: [https://protege.stanford.edu/publications/ontology\\_development/ontology101.pdf](https://protege.stanford.edu/publications/ontology_development/ontology101.pdf).
- [8] Khronos.org, *Unifying Reality*. dirección: <https://www.khronos.org/openxr/>.
- [9] Unity Documentation, *XR Interaction Toolkit*. dirección: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.5/manual/index.html>.
- [10] Ayfel, *Virtual Keyboard Documentation*. dirección: <https://github.com/Ayfel/MRTK-Keyboard>.
- [11] U. Documentation, *Dictation Recognizer*. dirección: <https://docs.unity3d.com/ScriptReference/Windows.Speech.DictationRecognizer.html>.
- [12] OpenAI, *Whisper AI Documentation*. dirección: <https://github.com/openai/whisper>.

- 
- [13] Gaia-X, *Self-Description of Resources, Service Offerings and Participants within Gaia-X Ecosystems*. dirección: [https://protege.stanford.edu/publications/ontology\\_development/ontology101.pdf](https://protege.stanford.edu/publications/ontology_development/ontology101.pdf).
  - [14] W. Documentation, *The Self-Describing Web*. dirección: <https://www.w3.org/2001/tag/doc/selfDescribingDocuments.html>.
  - [15] U. Documentation, *Struct TeleportRequest*. dirección: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/api/UnityEngine.XR.Interaction.Toolkit.TeleportRequest.html>.
  - [16] U. Documentation, *Newtonsoft Json Unity Package*. dirección: <https://docs.unity3d.com/Packages/com.unity.nuget.newtonsoft-json@3.2/manual/index.html>.
  - [17] E. Board, *Corporate Social Responsibility Report 2023*. dirección: <https://www.fraunhofer.de/s/ePaper/CSR%20Report/2023/index.html#0>.
  - [18] A. Graßl, *Sustainability*. dirección: <https://www.fraunhofer.de/en/about-fraunhofer/corporate-responsibility/governance/sustainability.html>.
  - [19] F. C. S. Department, *Guiding Principles of the Fraunhofer-Gesellschaft*. dirección: <https://www.fraunhofer.de/en/about-fraunhofer/corporate-responsibility/governance/guiding-principles.html>.