



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO

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

## **Informe PFE**

Simón Bruno - 12177

Tutor: Profesor César Aranda

# **ACKNOWLEDGEMENT**

---

I would like to thank everyone who's supported me during this university exchange program and internship period.

To my family for always teaching me to aim as high as possible. Everything they've ever done has been to help me grow. I am also grateful for their material support during this period; without it, I could not have succeeded.

To my friends, both from back home and all the new ones I got to make along the way, for always cheering for me and not letting me give up.

To my boss and colleagues, for trusting in me and my judgement, to find the best possible solution for each assigned task. They all proved to be exemplary in their own way.

To my universities from France and Argentina, for giving me the necessary knowledge and tools to pursue my dreams as a young professional.

# TABLA DE CONTENIDOS

---

<b>Resumen</b>	<b>5</b>
<b>Introducción</b>	<b>6</b>
0.1. Realidad Extendida . . . . .	6
0.2. Ontologías . . . . .	8
0.3. Gaia-X . . . . .	10
<b>1. Estado del Arte</b>	<b>13</b>
<b>2. Trabajo Realizado</b>	<b>15</b>
2.1. Planeación de Desarrollo . . . . .	15
2.2. Visualizador de Ontologías . . . . .	15
2.3. Software de Desarrollo . . . . .	16
2.3.1. OpenXR . . . . .	17
2.3.2. XR Interaction Toolkit . . . . .	18
2.4. Funciones Desarrolladas . . . . .	18
2.4.1. Sistema de Búsqueda . . . . .	18
2.4.2. Sistema de Reconocimiento de Voz . . . . .	21
2.4.3. Menú Jerárquico . . . . .	23
2.4.4. Importación desde archivos Jsonld . . . . .	24
2.4.5. Exportación a archivos Jsonld . . . . .	28
2.4.6. Contenedores de Ontologías . . . . .	29
2.4.7. Modos de distribución de ontologías . . . . .	31
<b>Conclusiones</b>	<b>34</b>
2.5. Trabajo Futuro . . . . .	35
<b>Bibliography</b>	<b>36</b>
<b>Apéndice</b>	<b>38</b>

# ÍNDICE DE FIGURAS

---

1.	Realidad Extendida y sus variantes . . . . .	6
2.	Ontología: Ejemplo 1 . . . . .	9
3.	Ontología: Ejemplo 2 . . . . .	10
4.	Ontología: Ejemplo 3 . . . . .	11
5.	Logo de Gaia-X . . . . .	12
1.1.	Protégé Logo . . . . .	13
2.1.	Planeacion de Desarrollo en Diagrama de Gantt . . . . .	15
2.2.	Escena desarrollada . . . . .	16
2.3.	OpenXR logo . . . . .	17
2.4.	Interaccioes del Toolkit . . . . .	19
2.5.	Búsqueda de clases y resultados . . . . .	20
2.6.	Diagrama de Paquetes del Sistema de Búsqueda . . . . .	21
2.7.	Modelos de entrenamiento de Whisper AI . . . . .	24
2.8.	Ilustración gráfica de funcionamiento . . . . .	25
2.9.	Import/Export process . . . . .	26
2.10.	Conversión de clase y conexión a formato interno . . . . .	27
2.11.	Objeto Json del archivo de posiciones . . . . .	28
2.12.	Diagrama de clases para los contenedores . . . . .	30
2.13.	Ontologías y sus contenedores . . . . .	31
2.14.	Cálculo de proyección para los modos de distribución . . . . .	33

# 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 mas adecuada a las necesidades.

En este informe voy a mostrar los resultados del proyecto realizado durante mi periodo 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.

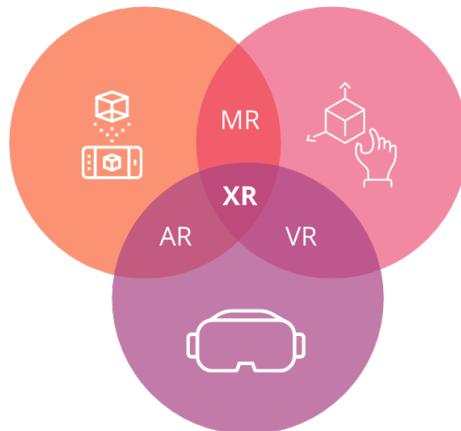


Figura 1: Realidad Extendida y sus variantes

## 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.<sup>o</sup> 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 Pokémon 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.

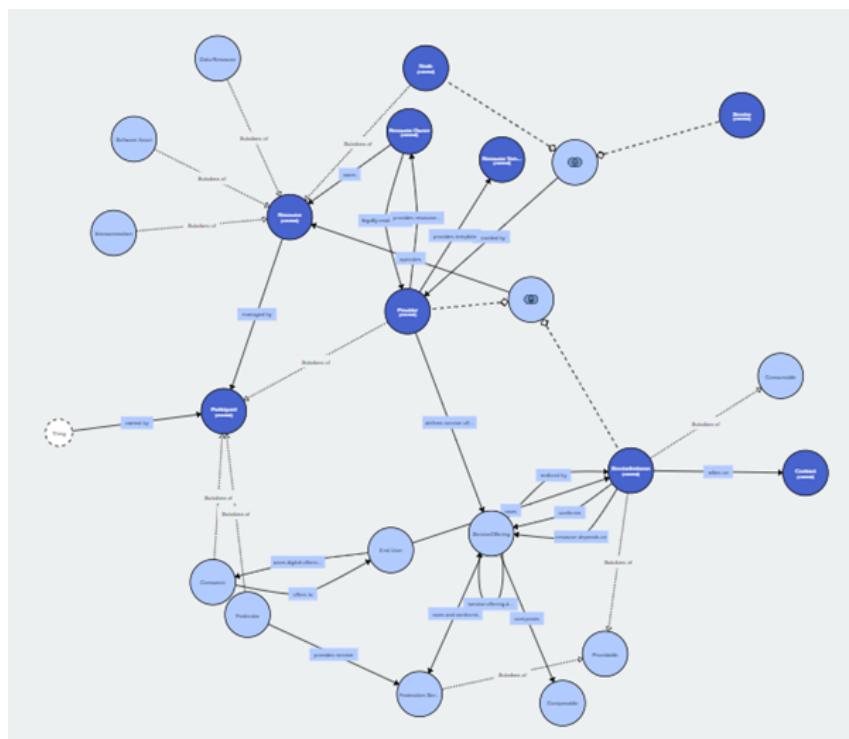


Figura 2: Ontología: Ejemplo 1

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.

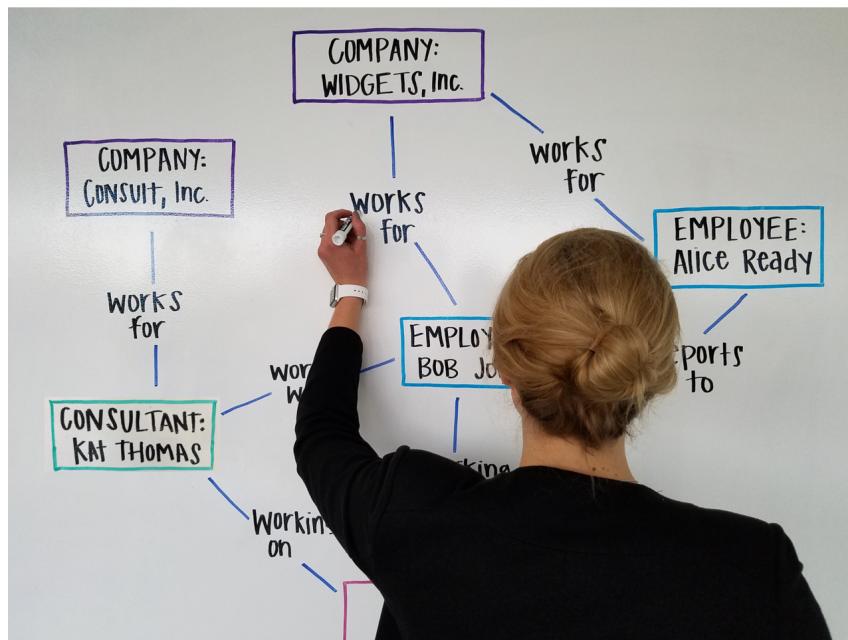


Figura 3: Ontología: Ejemplo 2

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 mayoría de la información y las aplicaciones son todavía inaccesibles e intercambiables. Gaia-X desarrollará una estructura de trabajo que le permita a la gente tomar decisiones

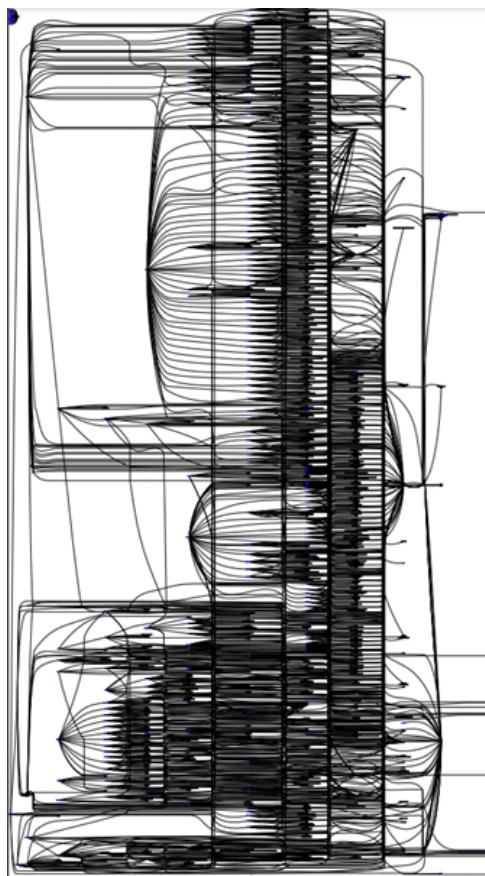


Figura 4: Ontología: Ejemplo 3

informadas a la hora de intercambiar información.

¿Por qué están relacionados este proyecto con 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

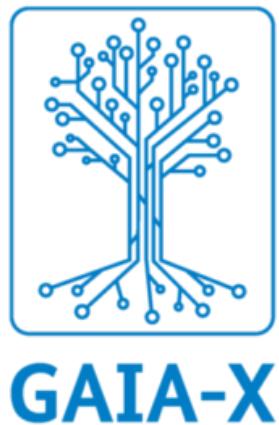


Figura 5: Logo de Gaia-X

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 Protege 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

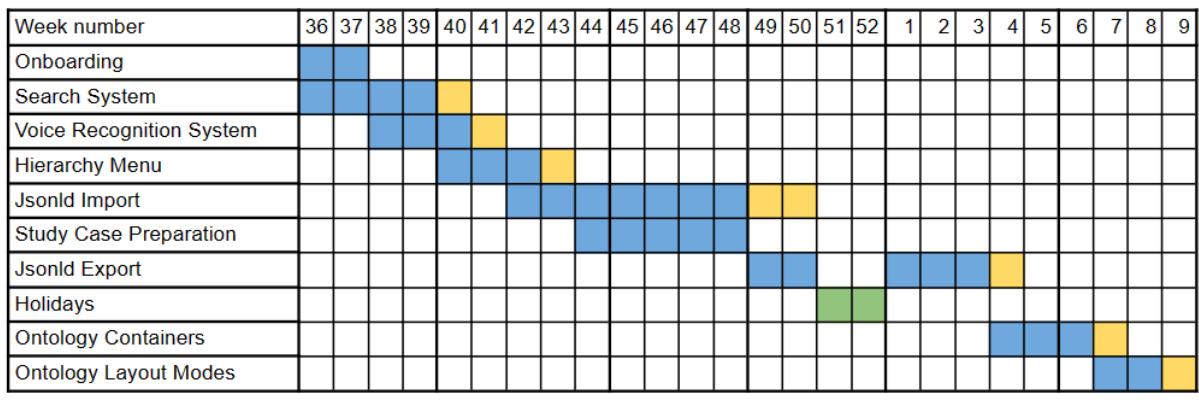


Figura 2.1: Planeación de Desarrollo en Diagrama de Gantt

## 2.2. Visualizador de Ontologías

Esta aplicación tiene objetivos muy concretos. En primer lugar, integrarse al sistema y formato creado por Gaia-X, por lo tanto, debemos incorporar un método para leer los archivos provenientes de este proyecto. Segundo, nos enfocaremos en tareas 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.

La escena desarrollada se ve como en la Figura 2.2. Aquí podemos observar, desde el punto de vista del usuario, como se enfrentaría a un grupo de clases provenientes de 4 ontologías diferentes.

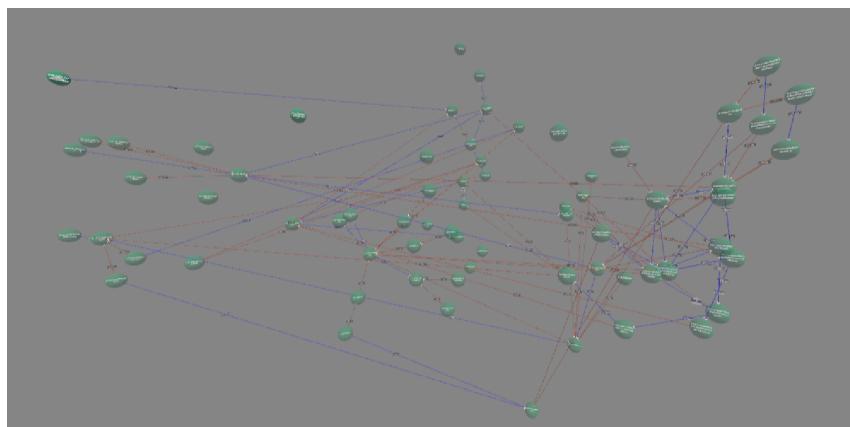


Figura 2.2: Escena desarrollada

Cada clase está representada por una instancia de la clase *Bubble* y se conecta a otras clases a través de dos tipos de conexiones diferentes:

- 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)

### **2.3. 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.

### **2.3.1. 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]



Figura 2.3: OpenXR logo

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

### 2.3.2. 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]

Si observamos el conjunto de Figuras 2.4, vemos algunos ejemplos de interacciones que podemos utilizar gracias a esta kit de herramientas. Con el puntero raycast, manteniendo el click presionado mientras apuntamos a una burbuja, podemos desplazarla en el espacio como queramos. Ese mismo puntero nos permite teletransportarnos a una burbuja deseada si movemos el joystick hacia adelante al apuntarle. También podemos interactuar con elementos de UI como botones e imágenes.

## 2.4. Funciones Desarrolladas

### 2.4.1. 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 cualquiera de estas funcionalidades, necesitábamos asociar cada resultado de búsqueda a la burbuja

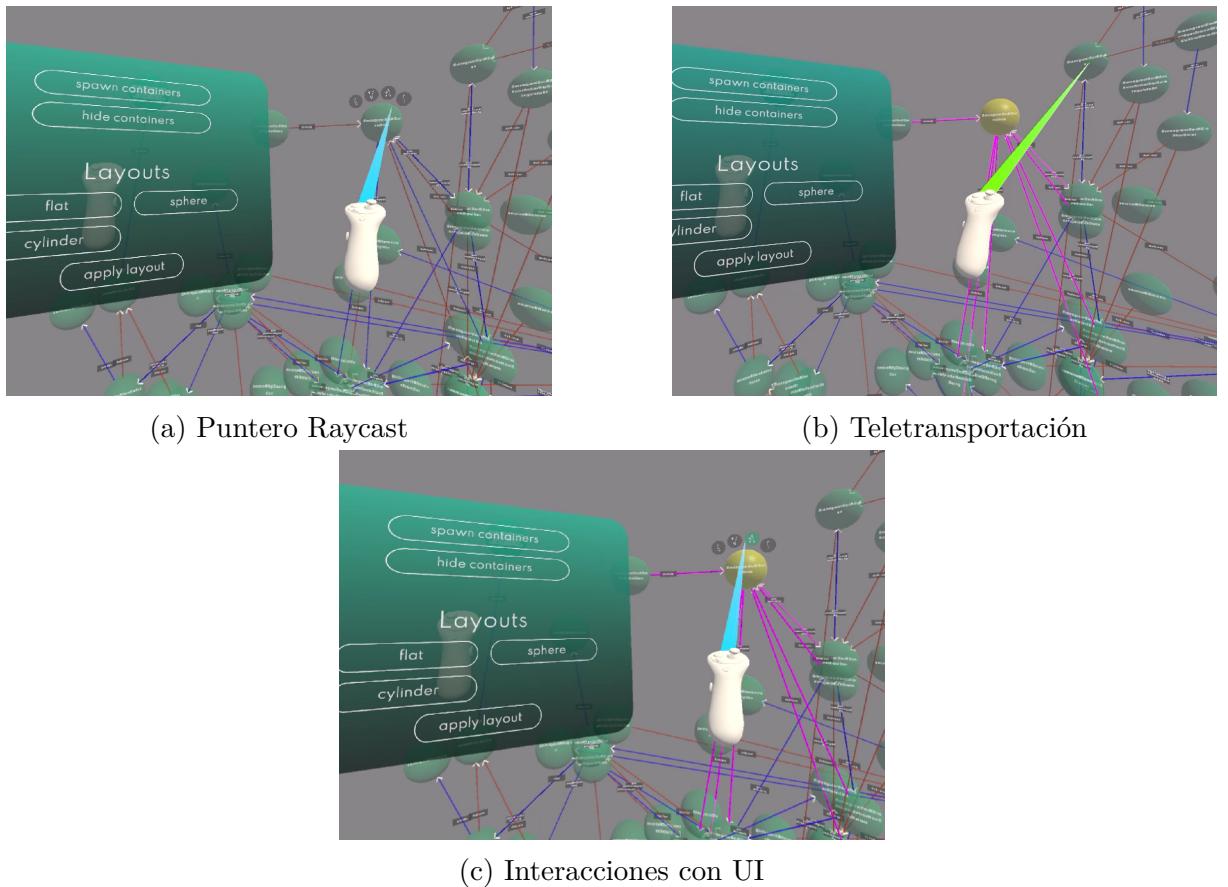


Figura 2.4: Interacciones del Toolkit

correcta, para que podamos encontrar la referencia a los métodos de la burbuja. Resaltar es sólo cambiar el color del material mientras que la teletransportación reposiciona al jugador y lo orienta hacia la burbuja en cuestión.

El proceso de búsqueda es manejado por el *Search Manager* y su componente principal *SearchScript.cs*. Está conectado al campo de entrada de búsqueda y llama al método *Search()* cada vez que hay un cambio en el texto de entrada. Dicho script contiene una lista *List<GameObject>Elements* que contiene todas las clases de la escena. Esta es llenada al crear las burbujas para las clases extraídas de un archivo, pero eso lo veremos en otra sección.

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.

Los métodos 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 sea teletransportar o resaltar. También resalta el texto del botón que coincide con la búsqueda.
- 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.

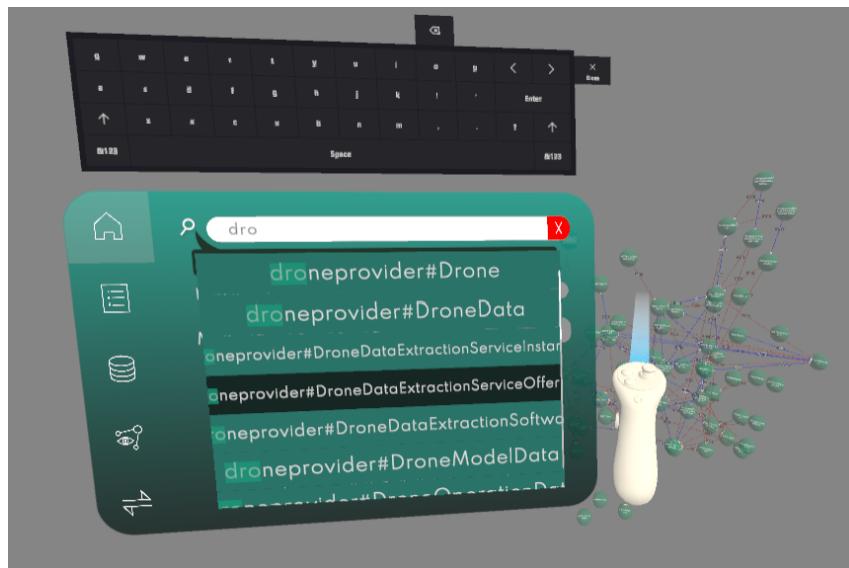


Figura 2.5: Búsqueda de clases y resultados

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. Esta es una forma muy intuitiva y frecuente de añadir un menú en las simulaciones de RV.

Añadimos un teclado que obtuvimos de un proyecto de código abierto en Github y funcionó perfectamente desde el principio. [10]

La figura 2.6 ilustra un poco el funcionamiento del proceso de búsqueda. Incluye el Reconocimiento de Voz explicado en la sección siguiente.

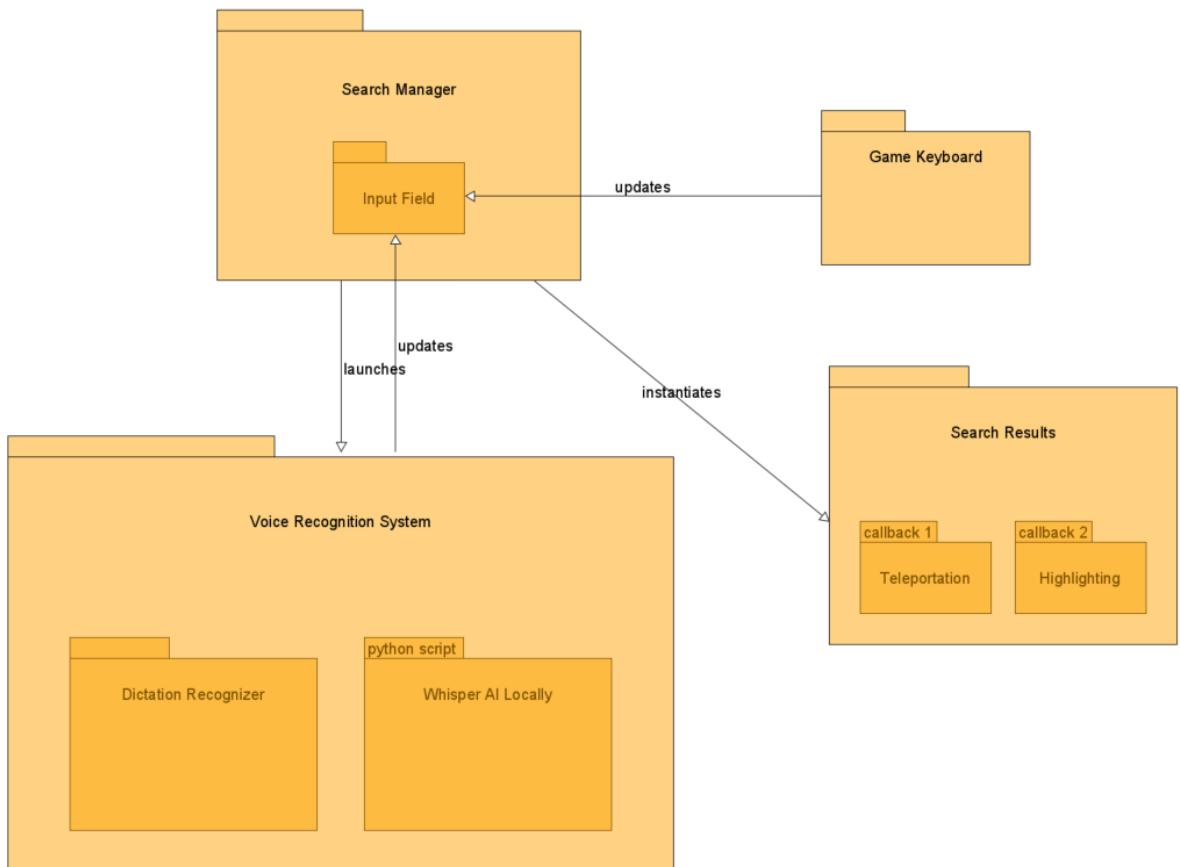


Figura 2.6: Diagrama de Paquetes del Sistema de Búsqueda

## 2.4.2. Sistema de Reconocimiento de Voz

Para complementar el Sistema de Búsqueda, pensamos que sería buena idea implementar un Sistema de Reconocimiento de Voz 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 de pago.

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 Reconocedor de Palabras Clave 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 Reconocedor de Dictados  
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 y trata de determinar qué frase fue  
pronunciada. [11]

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.

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  
registeraría 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. Lo que era real-  
mente interesante era que podíamos seleccionar el tamaño del modelo de entrenamiento,  
por lo que podíamos optimizar los recursos u optar por un mejor reconocimiento si lo  
deseábamos. Ver Figura 2.7.

### 2.4.3. 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

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 2.7: Modelos de entrenamiento de Whisper AI

la interfaz del sistema de carpetas del Explorador de Windows. Vemos una carpeta y, al hacer clic 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 clic 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.

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/teleportado asociadas a ellas. Todo esto cambió cuando empezamos a importar ontologías desde ficheros jsonld y el concepto de objeto raíz dejó de ser

#### 2.4.4. Importación desde archivos Jsonld

Como he mencionado antes, esta simulación depende del Ontology Manager para cargar y guardar ontologías en archivos json, pero queremos ser capaces de leer información de Gaia-X y mostrarla en la escena para operar con ella. Después, deberíamos poder guardarla en un archivo similar y visualizarla en cualquier herramienta 2D y ver las modificaciones que hemos hecho. Para esto necesitamos un Parser jsonld, para interpretar la información de Gaia-X y crear un archivo json que sea legible desde nuestro proyecto. Para ello hemos utilizado el paquete Newtonsoft Json Unity que nos ha facilitado mucho la tarea. [13]

El proceso de conversión es el resultado de un cuidadoso estudio por lo que necesita

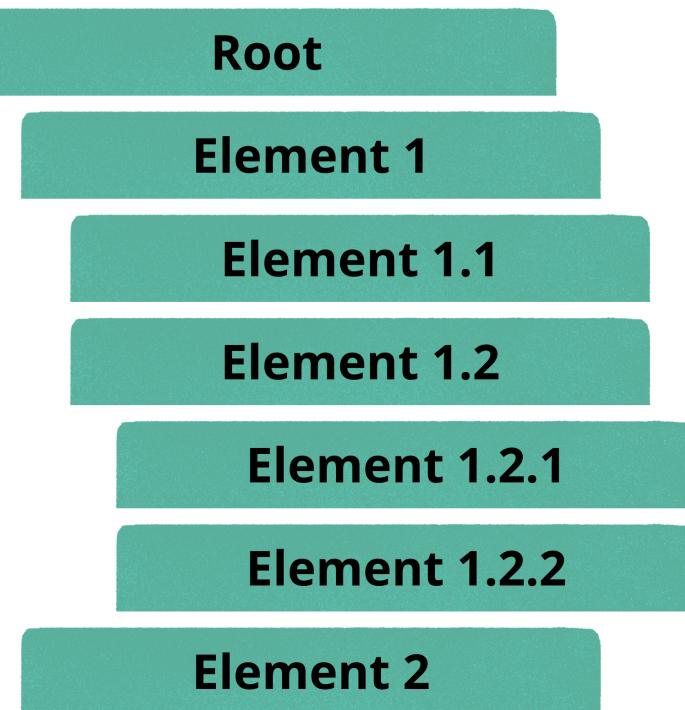


Figura 2.8: Ilustración gráfica de funcionamiento

una explicación clara. Primero descargamos el archivo JsonLD de la página web de la ontología Gaia-X en el estándar que han creado. Esta información luego pasa por dos procesos dentro de la escena Unity. El primero es un Parser para traducir el contenido y los datos a un formato estructurado con los valores que necesitamos para crear el objeto 3D en la escena. Esto incluye:

- ID numérico único del juego
- Tipo de burbuja ( **Class** / Propiedad / Instancia )
- Nombre de la burbuja
- Posición 3D ( importada de otro archivo )
- Burbujas Poseídas
- Burbujas poseídas
- Heredado de Burbujas

- Heredado de Burbujas

Esta es toda la información que utilizamos para crear una Burbuja en la escena, por lo que es importante que no olvidemos ninguna de ellas en el proceso de parseo. Este primer proceso se lleva a cabo en el momento de Inicio de la simulación pero esto puede ser cambiado a otro momento, como un menú de botones para importar ontologías específicas. De esta forma creamos un nuevo fichero que puede ser elegido para importar desde el Menú de Mano en la escena.

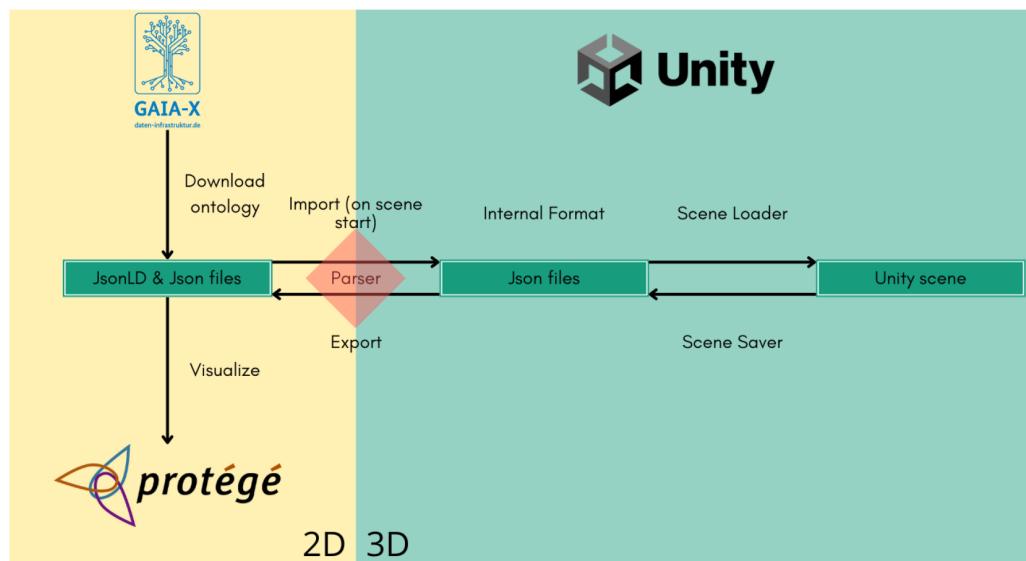


Figura 2.9: Import/Export process diagram

La forma en que filtramos la información es con una lista de condiciones if que comprobarán la existencia de diferentes claves o propiedades json que nos guían y nos dicen qué información coger y cuál dejar, ya que el fichero jsonld contiene mucha información que no vamos a utilizar en nuestra escena, pero que necesitamos conservar para cuando exportemos la escena no perder ningún dato en el proceso.

A partir de los archivos jsonld que hemos descargado, buscamos en primer lugar objetos de dos tipos diferentes: Clases (`.owl#Class`) y Conexiones (`.owl#ObjectProperty`). Analizaremos la información y almacenaremos las dos versiones de cada tipo en cuatro listas diferentes más una lista extra para almacenar toda la información no analizada que necesitaremos para el proceso de exportación. Con esa información, completaremos las listas de nuestro archivojson con las burbujas de propiedad y las subclases.

Para explicar cómo funciona la herramienta de importación, será más fácil si utilizamos un ejemplo de la lista de clases convertidas. También podemos ver el código 2.2 del apéndice.



Figura 2.10: Conversión de clase y conexión a formato interno

Cuando encontramos una clase, tenemos que deshacernos de la 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 de la última '/'. El '#' separará la etiqueta de la ontología a la izquierda del nombre de la clase a la derecha. Repetimos esta operación de pasar por cada objeto json de la lista hasta que hayamos guardado todos los objetos de tipo clase.

Después de analizar todas las clases, podemos empezar a buscar conexiones, que se especifican como tipo "Propiedad de objeto." con la clave "subClassOf" dentro del objeto de clase. Cada propiedad de objeto contiene información sobre el id de la conexión, el dominio (origen), el rango (destino) y una etiqueta. Nuestra tarea consiste en buscar el id específico en la lista de objetos y luego buscar y añadir los ids en ambos extremos de la conexión en nuestro formato.

Así, por ejemplo, la conexión mostrada en la Figura 2.9 tiene el id de la conexión con dominio y rango. En este caso concreto, lo leeríamos así : "La clase Service Offering de

la ontología de servicios produce la clase ApiDescription de la ontología de servicios". La url subClassOf de la misma figura nos indicará de qué clase estamos heredando en este momento. En este caso, ApiDescription hereda de Consumable. 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 .°wnedByBubblesList con el id del origen de la conexión ( range ) y la etiqueta que hemos parseado a partir del id de la conexión. En el otro extremo de la conexión, añadiremos el id 1007 en el "hasBubblesIDsList". La conexión subClass funciona de forma similar. Nuestra ApiDescription hereda de la burbuja 1000, que se añade a la inheritedBubblesList".

Por ahora, aunque no tenemos un algoritmo de posicionamiento para colocar las burbujas en la escena, necesitamos algún tipo de valor de posición inicial. Para resolver esto parcialmente, podemos generar un archivo de posición 2D para cada ontology con una herramienta en línea llamada WebVowl, y leer los valores de la misma. Estos están conectados a la burbuja correspondiente a través de una dirección IRI, por lo que tenemos que buscar la clase con la misma dirección. Escalaremos los valores y añadiremos la tercera dimensión de la escena para que cada clase se sitúe en algún lugar de la escena y no se solapen todas las burbujas. Ver Figura 2.11.

```
{
  "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 2.11: Objeto Json del archivo de posiciones

#### 2.4.5. 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.

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.

#### **2.4.6. Contenedores de Ontologías**

La idea de esta característica es tener un contenedor para cada ontología de manera que podamos diferenciarlas fácilmente. Cada uno de ellos también debe ser interactivo para que pueda agarrarlos y moverlos en la escena, y al hacerlo, todas las clases contenidas se moverán de la misma manera. 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.

Este diagrama de clases nos muestra cómo están conectados los scripts en el código. *OntologyManager* se encarga de instanciar los prefabs y llenar las listas correspondientes de cada uno de los subgestores.

Cada vez que creamos una nueva burbuja debemos comprobar si la ontología a la que pertenece ya existe y crearla en caso contrario. Cada nueva clase está contenida en un objeto con el nombre de la ontología y este a su vez está vinculado a de mayor jerarquía. Este último tiene 2 componentes de script: *Ontologies.cs* y *ContainersManager.cs*. Cada vez que tenemos que instanciar una nueva ontología, tenemos que añadirla a la lista de ontologías en *Ontologies.cs* y luego instanciar el contenedor y añadirlo a la lista *ontologyContainers* en el *ContainersManager.cs*

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

*OntologyContainer.cs* tiene el control del propio contenedor, de forma que cada vez

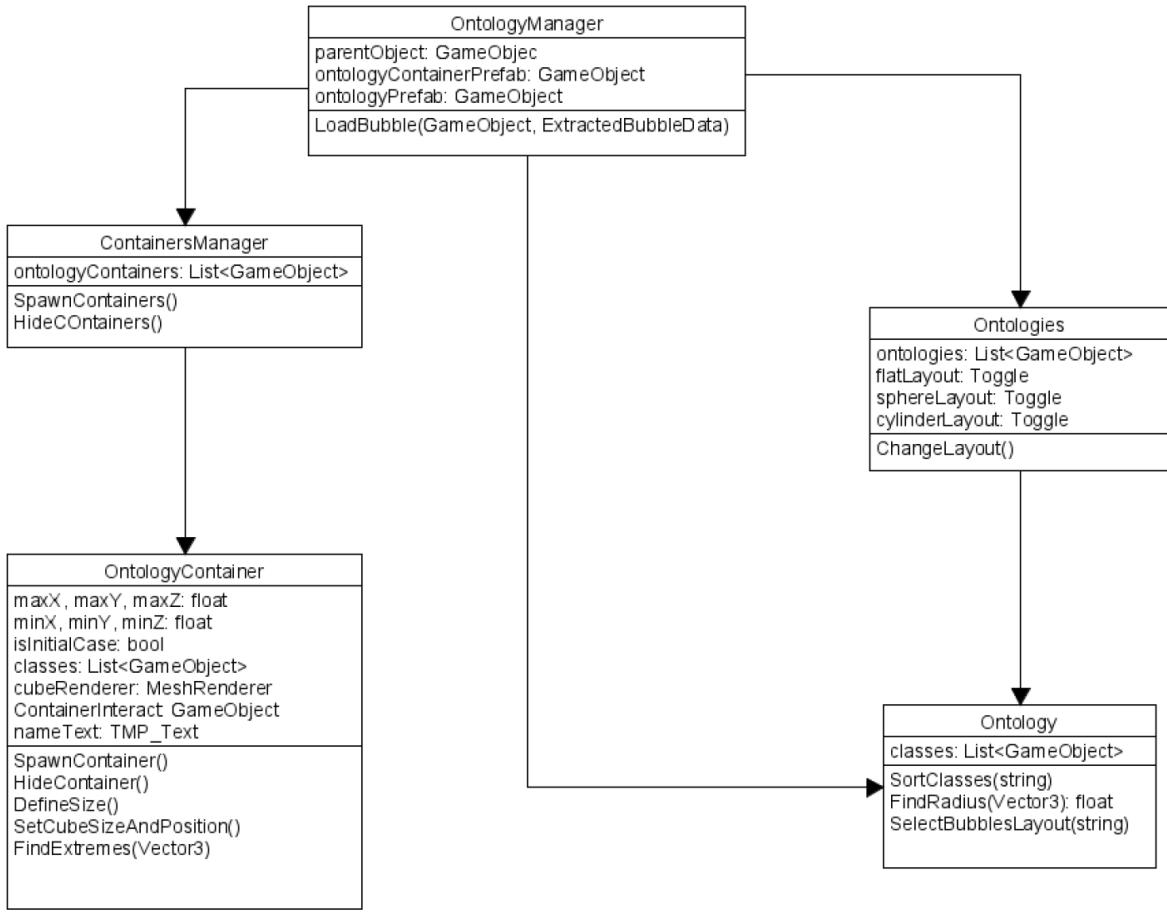


Figura 2.12: Diagrama de clases para los contenedores

que decidamos generarlos, recalculará el tamaño de todas las burbujas. Los métodos son bastante sencillos de entender, pero el proceso es el siguiente: primero echamos un vistazo a todas las clases dentro de una ontología y encontramos los extremos de la misma (min y max de los 3 ejes). Con eso podemos saber el tamaño del propio contenedor y escalar el prefabricado para que se ajuste a la ontología. Ver código 2.3 del apéndice.

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.

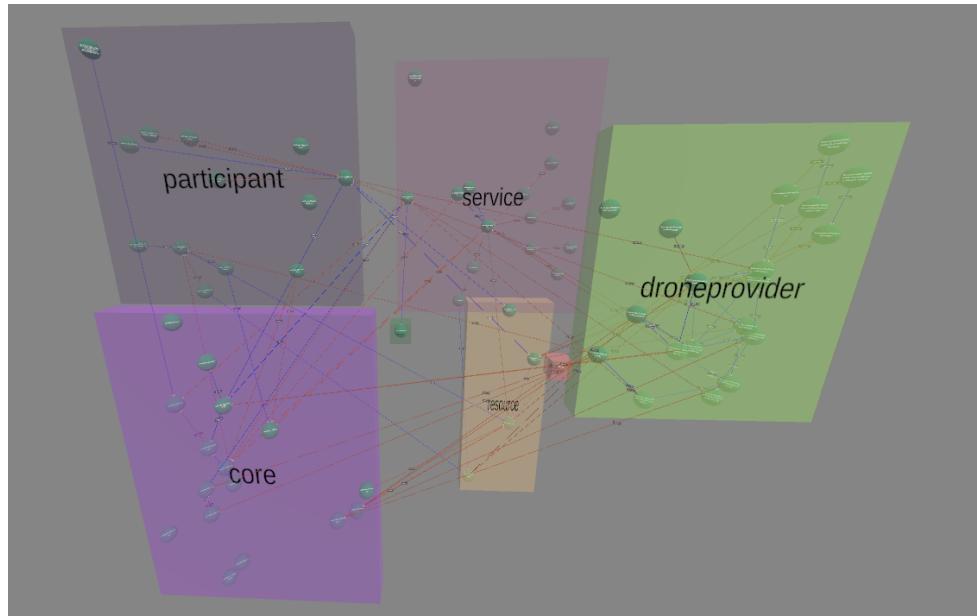


Figura 2.13: Ontologías y sus contenedores

#### 2.4.7. 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 del que estábamos tan orgullosos. Para superarlo, hablamos de utilizar formas geométricas básicas para describir cada ontología. Calculariamos la dimensión de la forma deseada y luego proyectaríamos cada clase según sus valores x e y. Ver código 2.4 del apéndice.

En el caso de la Figura 2.14, tomamos la ontología y sus clases, calculamos la distancia x entre las burbujas más extremas, la utilizamos para crear un radio de cilindro y, a continuación, colocamos cada burbuja en el valor z de este cilindro según las otras dos coordenadas a modo de proyección.

Esta misma lógica se puede aplicar con cualquier forma deseada. También implementé una esférica pero no alcancé a hacerla funcionar correctamente antes del final de la pasantía. Si además añadiéramos rotaciones a las interacciones de los contenedores, podríamos buscar una forma ontológica concreta para colocarlos todos de la forma deseada y crear un layout mucho más agradable al usuario.

También podemos mencionar que, aunque estas distribuciones son puramente geométricas, también podríamos intentar dar con un parámetro que queramos optimizar y

buscar un modelo de IA adecuado para ello. Por ejemplo, podría ser muy útil reducir la longitud total de las conexiones en la escena, de modo que no tengamos tantas líneas en la escena interfiriendo con los efectos visuales.

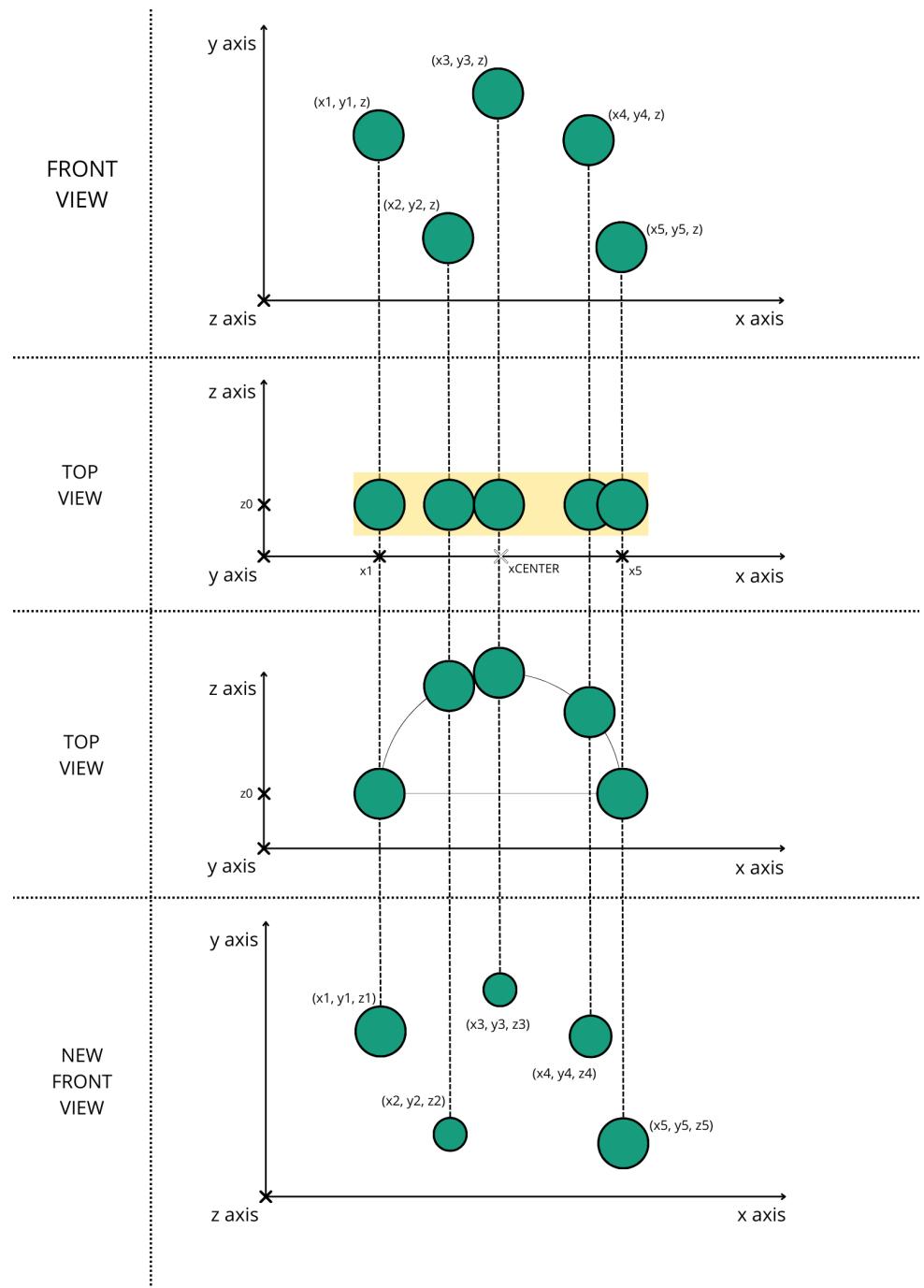


Figura 2.14: Cálculo de proyección para los modos de distribución

# CONCLUSIONES

---

El objetivo principal de estas prácticas era mejorar la herramienta 3D para visualizar ontología, con el fin de tener una alternativa a los softwares 2D tradicionales y evitar algunas limitaciones que normalmente tienen. La primera fase estuvo orientada a filtrar las clases de la escena y desarrollar características para encontrar una clase lo más rápido posible. En noviembre, realizamos un estudio con distintos participantes del instituto para recabar más información y comprobar el progreso actual de nuestro proyecto, y muchos de ellos afirmaron que la herramienta de búsqueda era increíblemente útil, y que el software de gestión ontológica no siempre dispone de una.

En el desarrollo de RV, 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 más a menudo que nunca. 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.

La documentación es una parte crucial de cualquier proyecto, ya que ayuda a los nuevos miembros del equipo y mantiene un control de versiones limpio. Dedicar tiempo a crear diagramas de clases, hacer capturas de pantalla y aclarar las conexiones de los inspectores entre los objetos del juego es la clave para progreso claro. Una semana después de incorporar cada característica a la versión principal de la escena, se dedicó a documentar

---

el trabajo realizado para esa cuestión.

## 2.5. Trabajo Futuro

El proyecto está lejos de estar terminado. Ahora que podemos importar y exportar información de los estándares de Gaia-X, podríamos pensar en añadir herramientas de edición a la escena. Crear y borrar burbujas sería el primer paso, pero luego también tendríamos 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 podríamos 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 RV 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] U. Documentation, *Newtonsoft Json Unity Package*. dirección: <https://docs.unity3d.com/Packages/com.unity.nuget.newtonsoft-json@3.2/manual/index.html>.
  - [14] U. Documentation, *Struct TeleportRequest*. dirección: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/api/UnityEngine.XR.Interaction.Toolkit.TeleportRequest.html>.
  - [15] E. Board, *Corporate Social Responsibility Report 2023*. dirección: <https://www.fraunhofer.de/s/ePaper/CSR%20Report/2023/index.html#0>.
  - [16] A. Graßl, *Sustainability*. dirección: <https://www.fraunhofer.de/en/about-fraunhofer/corporate-responsibility/governance/sustainability.html>.
  - [17] F. C. S. Department, *Guiding Principles of the Fraunhofer-Gesellschaft*. dirección: <https://www.fraunhofer.de/en/about-fraunhofer/corporate-responsibility/governance/guiding-principles.html>.

# APÉNDICE

---

```
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
9      int flagFoundObjects;
10
11     // get input text
12     searchText = SearchInputField.GetComponent<TMP_InputField
13         >().text;
14     int searchTextLength = searchText.Length;
15     foreach (GameObject element in Elements)
16     {
17         if (element.name.Length >= searchTextLength &&
18             searchTextLength >= 1) // if the text input is
19             longer than the element name and consists of at
20             least 1 character
21         {
22             string elementName = element.name.ToLower();
23             if (elementName.Contains(searchText.ToLower()))
24             {
25                 matchingElements.Add(element);
26             }
27         }
28     }
29
30     int flag = matchingElements.Count > 0 ? 1 : 2;
31     flagFoundObjects = searchTextLength >= 1 ? flag : 0;
32     CreateButtons();
33 }
```

---

```

26     UpdateResultsButton(flagFoundObjects);
27 }
28
29 public void DeleteButtons()
30 {
31     buttonsToDelete = GameObject.FindGameObjectsWithTag("TeleportButton");
32     for (int i = 0; i < buttonsToDelete.Length; i++)
33     {
34         Destroy(buttonsToDelete[i]);
35     }
36 }
37
38 private void CreateButtons()
39 {
40     for (int i = 0; i < matchingElements.Count; i++)
41     {
42         // Button instantiation and placement in the scene
43         GameObject result = Instantiate(PossibleResultButton,
44                                         buttonsParent.transform);
45         result.tag = "TeleportButton";
46
47         // finding start Index for matching text
48         string elementLowerName = matchingElements[i].name.
49             ToLower();
50         int startIndex = elementLowerName.IndexOf(SearchText.
51             ToLower());
52         TextMeshProUGUI buttonText = result.
53             GetComponentInChildren<TextMeshProUGUI>();
54         if (startIndex != -1)
55         {
56             // highlighting matching text
57             string highlightedText = matchingElements[i].name
58                 .Insert(startIndex + SearchText.Length, "</mark>");

```

```
54         highlightedText = highlightedText.Insert(
55             startIndex, "<mark=#179C7Daa>");
56         buttonText.text = highlightedText;
57     }
58     else
59     {
60         buttonText.text = matchingElements[i].name;
61     }
62
63     // buttonText.fontSize = 10f;
64
65     // Setup of all the information inside the button to
66     // connect it to the correct bubble and to manage the
67     // interaction with the corresponding UI.
68     TeleportOnSearch buttonArguments = result.
69         GetComponent<TeleportOnSearch>();
70     buttonArguments.BubbleTeleportAnchor =
71         matchingElements[i].GetComponent<Bubble>().
72         VRBubbleTeleport;
73     buttonArguments.player = GameObject.
74         FindGameObjectWithTag("Player");
75     buttonArguments.BubblePosition = matchingElements[i].
76         transform.position;
77     buttonArguments.ConfirmationCanvas =
78         ConfirmTeleportCanvas;
79     buttonArguments.SearchInputField = SearchInputField;
80     buttonArguments.BubbleName = matchingElements[i].name
81         ;
82     buttonArguments.BubbleMenuManager = matchingElements[
83         i].GetComponent<Bubble>().BubbleMenuManager;
84     if (HighlightBubble && !TeleportToBubble)
85     {
86         result.GetComponent<Button>().onClick.AddListener
87             (buttonArguments.HighlightBubble);
88     }
89
90 }
```

```
78         else if (!HighlightBubble && TeleportToBubble)
79     {
80         result.GetComponent<Button>().onClick.AddListener
81             (buttonArguments.ShowConfirmationCanvas);
82     }
83 }
84
85
86 public void ResetSearch(bool tryAgain = false)
87 {
88     SearchInputField.GetComponent<TMP_InputField>().text = "";
89     statusButton.gameObject.SetActive(false);
90     if (tryAgain)
91     {
92         if (DictationEngine.gameObject.activeSelf)
93         {
94             // case 3 leads to restarting the voice
95             recognition
96             DictationEngine.GetComponent<DictationEngine>().
97                 StartDictationEngine();
98         }
99     }
100 }
101
102 public void UpdateResultsButton(int flagFoundObjects)
103 {
104     TextMeshProUGUI statusButtonText = statusButton.
105         GetComponentInChildren<TextMeshProUGUI>();
106     //statusButton.onClick.AddListener(() => ResetSearch(
107         false));
108     switch (flagFoundObjects)
109     {
110         case 0: // no input in the field
111             statusButton.gameObject.SetActive(false);
```

---

```

108         break;
109     case 1: // results found
110         break;
111     case 2: // input detected, no results found
112         statusButton.gameObject.SetActive(true);
113         statusButtonText.text = "No results found for '"
114             + SearchText;
115         break;
116     case 3: // timeout exceeded, try again
117         statusButton.gameObject.SetActive(true);
118         statusButtonText.text = "No input detected - Try
119             again";
120         break;
121     case 4: // voice recognition activated, waiting for
122         input
123         statusButton.gameObject.SetActive(true);
124         statusButtonText.text = "Listening for input";
125         break;
126     case 5:
127         statusButton.gameObject.SetActive(true);
128         GetComponent<CountdownScript>().StartCountdown();
129         break;
130     case 6:
131         statusButton.gameObject.SetActive(true);
132         statusButtonText.text = "Processing information";
133         break;
134     default:
135         break;
136     }
137 }
138 }
```

Código 2.1: SearchScript.cs

1	void Start()
2	{
3	ParseManager();

---

```

4    }
5
6    private void ParseManager()
7    {
8        ReadFiles();
9        ReadMyOntology();
10       ParseGaiax();
11       ParseMyOntology();
12       SaveJson(ontology,positions);
13    }
14
15    private void ParseGaiax()
16    {
17        FindSubclasses("http://www.w3.org/2000/01/rdf-schema#"
18                      "subClassOf");
19        FindConnections("http://www.w3.org/2000/01/rdf-schema#"
20                        "domain",
21                        "http://www.w3.org/2000/01/rdf-schema#"
22                        "range",
23                        "http://www.w3.org/2000/01/rdf-schema#"
24                        "label");
25        FindComments("http://www.w3.org/2000/01/rdf-schema#"
26                      "comment");
27    }
28
29    private void ParseMyOntology()
30    {
31        FindSubclasses("rdfs:subClassOf");
32    }
33
34    private void ReadFiles()
35    {
36        IDcounter = firstID;
37        HashSet<string> encounteredIDs = new HashSet<string>();
38        for(int j = 0; j < jsonLDFFilePaths.Count; j++)
39        {
40            string folder = "JSON";

```

```
35     string fullPath = Path.Combine(Application.dataPath ,
36                                     folder, jsonLDFilePaths[j]);
37
38     string rawJsonLdData = File.ReadAllText(fullPath);
39     string jsonLdData = string.Copy(rawJsonLdData);
40
41     fullPath = Path.Combine(Application.dataPath, folder,
42                            jsonFilePaths[j]);
43     string rawData = File.ReadAllText(fullPath);
44     string jsonData = string.Copy(rawData);
45
46     JArray jsonLdArray = JsonConvert.DeserializeObject<
47                           JArray>(jsonLdData);
48
49     JObject jsonObject = JsonConvert.DeserializeObject<
50                           JObject>(jsonData);
51
52     JArray classAttributesArray = jsonObject["
53                                         classAttribute"] as JArray;
54
55     foreach (JObject jsonLdObject in jsonLdArray)
56     {
57         JArray typeArray = jsonLdObject["@type"] as
58                               JArray;
59
60         ImportedBubbleInformation newBubbleData = new
61                                         ImportedBubbleInformation();
62         ImportedPositionInformation newPositionData = new
63                                         ImportedPositionInformation();
64
65         string type = typeArray[0].Value<string>();
66         string id = jsonLdObject["@id"].Value<string>();
67
68         // we are looking for objects with only one type
69         // so they can be a class or an object property
70         // connecting classes only
```

---

```

62         // and also they have to belong to the specific
63         gaia-x ontologies
64         if (type.Contains("owl#Class") && id.Contains(
65             "http://w3id.org/gaia-x/"))
66         {
67             if (!encounteredIDs.Contains(id)) // Check if
68                 the ID has not been encountered before.
69             {
70                 encounteredIDs.Add(id);
71                 string[] parts = id.Split('/');
72                     separates the url in the iri base url
73                     and the ontology+class
74                 string result = parts[parts.Length - 1];
75                     // result = ontology+class
76
76                 parts = result.Split('#');
77                 string tag = parts[0]; // tag = ontology
78
78                 newBubbleData.elementName = result;
79                     // we'll only display the class name in
80                     the bubble but the class can be
81                     repeated in different ontologies, so we
82                     still need to specify which one it is
83                 newBubbleData.elementType = "Class";
84                 newBubbleData.elementID = IDcounter.
85                     ToString();
86                 newBubbleData.ontologyTag = tag;
87                 newPositionData.bubbleID = newBubbleData.
88                     elementID;
89
90
91                 float[] pos = FindPosByIRIValue(
92                     classAttributesArray, id);
93                 newPositionData.posX = pos[0];
94                 newPositionData.posY = pos[1];
95
96                 classObjects.Add(jsonLdObject);

```

---

```

86             ontology.extractedBubbleDatasList.Add(
87                 newBubbleData);
88             positions.extractedPositionDataList.Add(
89                 newPositionData);
90             IDcounter++;
91         }
92     }
93
94     else if (type.Contains("owl#ObjectProperty") &&
95             typeArray.Count == 1)
96     {
97         propertyObjects.Add(jsonLdObject);
98     }
99     else
100    {
101        exportToJson.Add(jsonLdObject);
102    }
103
104 private void FindSubclasses(string key)
105 {
106     for (int i = 0; i < classObjects.Count; i++)
107     {
108
109         JObject classObject = classObjects[i];
110         if (classObject.ContainsKey(key))
111         {
112             JToken subClassToken = classObject[key];
113             if (subClassToken is JArray)
114             {
115                 JArray subClassArray = subClassToken as
116                     JArray;
117                 foreach (JObject subClassObject in
118                     subClassArray)
119             }
120         }
121     }
122 }
```

---

```

117         {
118             ProcessSubclass(subClassObject , i);
119         }
120     }
121     else if (subClassToken is JObject)
122     {
123         JObject subClassObject = subClassToken as
124             JObject;
125         ProcessSubclass(subClassObject , i);
126     }
127 }
128 }
129
130 /// <summary>
131 /// This method will verify if the parent class is from gaiax
132     or droneprovider format and find the right ID
133 /// as well as finding the restrictions for the drone
134     provider connections
135 /// </summary>
136 /// <param name="subClassObject"> the parent class from whom
137     we inherit </param>
138 /// <param name="currentIndex"> index of our bubble </param>
139 private void ProcessSubclass(JObject subClassObject , int
140     currentIndex)
141 {
142     if (subClassObject.ContainsKey("@id"))
143     {
144         string subClassID = subClassObject["@id"].Value<
145             string>();
146
147         string convertedID = ConvertToGaiaxFormat(subClassID)
148             ;
149
150         int subClassIndex = FindIndexOfBubbleById(ontology .
151             extractedBubbleDatasList , convertedID);

```

```

145
146     if (subClassIndex >= 0)
147     {
148         ontology.extractedBubbleDatasList[currentIndex].
149             inheritedBubblesIDsList.Add((subClassIndex+
150                 firstID).ToString());
151         ontology.extractedBubbleDatasList[subClassIndex].
152             inheritingBubblesIDsList.Add((currentIndex+
153                 firstID).ToString());
154     }
155 }
156 else if (subClassObject.ContainsKey("rdf:type")) // DroneProvider ontology specifies relationships between
157 classes inside the "subClassOf" key
158 {
159     JObject type = subClassObject["rdf:type"] as JObject;
160     string id = type["@id"].Value<string>();
161     if(id == "owl:Restriction")
162     {
163         if(subClassObject.ContainsKey("owl:someValuesFrom"))
164         {
165             JObject ownerClass = subClassObject["owl:
166                 someValuesFrom"] as JObject;
167             string ownerClassName = ownerClass["@id"].
168                 Value<string>();
169             string convertedID = ConvertToGaiaxFormat(
170                 ownerClassName);
171
172             int ownerIndex = FindIndexOfBubbleById(
173                 ontology.extractedBubbleDatasList,
174                 convertedID);
175             if(ownerIndex >= 0)
176             {
177                 JObject propertyName = subClassObject["
178                     owl:onProperty"] as JObject;

```

---

```
168     string label = propertyName["@id"].Value<
169         string>();
170     string labelName = label;
171
172     VerifyListsForRepetitionAndAdd(ownerIndex
173         , currentIndex, labelName);
174 }
175
176 // To contemplate the idea of importing "
177 // allValuesFrom" means that we should be able to
178 // export to it as well
179 // To do that, we need to import and store which
180 // case it is we are dealing with
181
182
183 // if(subClassObject.ContainsKey("owl:
184 // allValuesFrom"))
185 // {
186 //     ownerClass = subClassObject["owl:
187 // allValuesFrom"] as JObject;
188 // }
189
190 }
191
192 private string ConvertToGaiaxFormat(string id)
193 {
194     // Check if the ID is in the "gax-" format
195     if (id.StartsWith("gax-"))
196     {
197         // Remove "gax-" part, replace ":" with "#"
198         string convertedID = id.Replace("gax-", "").Replace(":
199             ", "#");
```

```
196
197         return convertedID;
198     }
199     else
200     {
201         string convertedID = id.Replace("0:", "droneprovider#"
202                                         );
203         return convertedID;
204     }
205
206     // If the ID is not in the expected format, return it as
207     // is
208 }
209
210 private void FindConnections(string domain, string range,
211                             string label)
212 {
213     for (int i = 0; i < propertyObjects.Count; i++)
214     {
215         JObject propertyObject = propertyObjects[i];
216         if (propertyObject.ContainsKey(domain) &&
217             propertyObject.ContainsKey(range))
218         {
219             string domainID;
220             string rangeID;
221             JArray domainArray = propertyObject[domain] as
222                 JArray;
223             JArray rangeArray = propertyObject[range] as
224                 JArray;
225
226             JObject domainObject = domainArray[0] as JObject;
227             JObject rangeObject = rangeArray[0] as JObject;
228
229             domainID = domainObject["@id"].Value<string>();
230             rangeID = rangeObject["@id"].Value<string>();
```

---

```

226
227     int domainIndex = FindIndexOfBubbleById(ontology.
228         extractedBubbleDatasList, domainID);
229
230     int rangeIndex = FindIndexOfBubbleById(ontology.
231         extractedBubbleDatasList, rangeID);
232
233     string connectionID = propertyObject["@id"].Value
234         <string>();
235     string[] parts = connectionID.Split("#");
236     string labelName = parts[parts.Length - 1]; // 
237         many object properties have labels for
238         connections but not all of them, it can be
239         better to just use the id text
240
241     VerifyListsForRepetitionAndAdd(domainIndex,
242         rangeIndex, labelName);
243
244 }
245
246 }
247
248 /// <summary>
249 /// we check if there is already an existing connection going
250     from bubble1 to bubble2
251
252 /// if there is one from bubble2 to bubble1, it wont count as
253     an existing connection because the direction is different
254
255 /// </summary>
256
257 /// <param name="domainIndex"></param>
258 /// <param name="rangeIndex"></param>
259 /// <param name="labelName"></param>
260
261
262 private void VerifyListsForRepetitionAndAdd(int domainIndex,
263     int rangeIndex, string labelName)
264 {
265     if(domainIndex >= 0 && rangeIndex >= 0)
266     {

```

---

```

252     int indexInDomainList = ontology.
253         extractedBubbleDatasList[domainIndex].
254         hasBubblesIDsList.IndexOf((rangeIndex+firstID).
255             ToString());
256
257     if(indexInDomainList >= 0) // verification if there
258         is already a connection between these 2 bubbles to
259         just add the tag for the connection we are creating
260     {
261         for(int i = 0; i < ontology.
262             extractedBubbleDatasList[rangeIndex].
263             ownedByBubblesIDsList.Count; i++)
264         {
265             if(ontology.extractedBubbleDatasList[
266                 rangeIndex].ownedByBubblesIDsList[i].
267                 Contains((domainIndex+firstID).ToString()))
268                 // the list of owning bubbles also holds
269                 the label of the connection, so we verify
270                 if the list contains the id
271             {
272                 int indexInRangeList = i;
273                 ontology.extractedBubbleDatasList[
274                     rangeIndex].ownedByBubblesIDsList[
275                         indexInRangeList] += (divider +
276                             labelName); // if the connection
277                             already exists, we just add a divider
278                             and the new label
279                 break;
280             }
281         }
282     }
283
284     else
285     {
286         ontology.extractedBubbleDatasList[rangeIndex].
287             ownedByBubblesIDsList.Add((domainIndex+firstID)
288                 .ToString() + separator + labelName);

```

```
269         ontology.extractedBubbleDatasList[domainIndex].
270             hasBubblesIDsList.Add((rangeIndex+firstID).
271             ToString());
272     }
273 }
274 private int FindIndexOfBubbleById(List<
275     ImportedBubbleInformation> bubbleList, string bubbleId)
276 {
277     string[] parts = bubbleId.Split('/');
278     string result = parts[parts.Length - 1];
279
280     for (int i = 0; i < bubbleList.Count; i++)
281     {
282         if (bubbleList[i].elementName == result)
283         {
284             return i; // Return the index of the bubble with
285                     // the matching ID.
286         }
287     }
288
289     return -1; // Return -1 if the bubble is not found.
290 }
291
292 private float[] FindPosByIRIValue(JArray classAttributesArray,
293     string searchValue)
294 {
295     foreach (JObject classAttributeObject in
296             classAttributesArray)
297     {
298         if(classAttributeObject.ContainsKey("iri"))
299         {
300             string iriObject = classAttributeObject["iri"].
301                 Value<string>();
302             if (iriObject != null && iriObject == searchValue
303             )
304         }
305     }
306 }
```

---

```

297     {
298         JArray posArray = classAttributeObject["pos"]
299             as JArray;
300
301         if (posArray != null)
302         {
303             // scaled values
304             float posX = posArray[0].Value<float>() /
305                 100.0f;
306             float posY = posArray[1].Value<float>() /
307                 100.0f;
308             float[] position = new float[] { posX,
309                 posY};
310             return position;
311         }
312     }
313
314
315     return new float[] { 0.0f, 15.0f }; // default return
316     value. It's the same as the initialized values
317 }
318
319 private void SaveJson(ImportJsonToOntology ontology,
320     ImportJsonToOntologyPositions positions)
321 {
322     string outputPath = "Saves/TestingJsonLD";
323     string ontologyFile = "savedOntology_TestingJsonLD.json";
324     string positionsFile = "savedPositions_TestingJsonLD.json"
325         ";
326     string fullPath = Path.Combine(Application.dataPath,
327         outputPath, ontologyFile);
328
329     string jsonData = JsonConvert.SerializeObject(ontology,
330         Formatting.Indented);
331     File.WriteAllText(fullPath, jsonData);

```

---

```
324
325     fullPath = Path.Combine(Application.dataPath, outputPath,
326                             positionsFile);
327
328     jsonData = JsonConvert.SerializeObject(positions,
329                                             Formatting.Indented);
330
331     File.WriteAllText(fullPath, jsonData);
332 }
333
334 private void ReadMyOntology()
335 {
336     string folder = "JSON";
337     string fileName = "Droneprovider_v001.jsonld";
338     string fullPath = Path.Combine(Application.dataPath,
339                                   folder, fileName);
340
341     string jsonLdData = File.ReadAllText(fullPath);
342
343     fileName = "foaf.json";
344     fullPath = Path.Combine(Application.dataPath, folder,
345                           fileName);
346
347     string jsonData = File.ReadAllText(fullPath);
348
349     JObject jsonObject = JsonConvert.DeserializeObject<
350                               JObject>(jsonData);
351
352     JArray classAttributesArray = jsonObject["classAttribute"]
353                                 as JArray;
354
355     JObject jsonLdObject = JObject.Parse(jsonLdData);
356
357     ontologyContext = jsonLdObject["@context"] as JObject;
358     JArray graph = jsonLdObject["@graph"] as JArray;
359     if (graph != null)
360     {
361         foreach (JObject element in graph)
```

---

```

354    {
355        ImportedBubbleInformation newBubbleData = new
356            ImportedBubbleInformation();
357        ImportedPositionInformation newPositionData = new
358            ImportedPositionInformation();
359
360        if (element.ContainsKey("rdf:type") && element["rdf:type"] is JObject)
361        {
362            JObject type = element["rdf:type"] as JObject
363            ;
364            string myType = type["@id"].Value<string>();
365            if(myType == "owl:Class")
366            {
367                newBubbleData.elementID = IDcounter.
368                    ToString();
369                newPositionData.bubbleID = newBubbleData.
370                    elementID;
371
372                string name = element["@id"].Value<string
373                    >();
374                string searchName = name.Replace("0:", " "
375                    http://www.ipk.semantics.org/ontologies
376                    /konierik/droneprovider#");
377
378                newBubbleData.elementType = "Class";
379                newBubbleData.ontologyTag = "
380                    droneprovider";
381                newBubbleData.elementName = name.Replace(
382                    "0:", "droneprovider#");
383
384                float[] pos = FindPosByIRIValue(
385                    classAttributesArray, searchName);
386                newPositionData.posX = pos[0];
387                newPositionData.posY = pos[1];

```

---

```

378
379         IDcounter++;
380         classObjects.Add(element);
381         ontology.extractedBubbleDatasList.Add(
382             newBubbleData);
383         positions.extractedPositionDataList.Add(
384             newPositionData);
385     }
386     else if(myType == "owl:Ontology")
387     {
388         myGraph.Add(element);
389     }
390 }
391
392
393 private void FindComments(string key)
394 {
395     for (int i = 0; i < classObjects.Count; i++)
396     {
397         JObject classObject = classObjects[i];
398
399         // string classObjectString = JsonConvert.
400         // SerializeObject(classObject, Formatting.Indented);
401         // ontology.extractedBubbleDatasList[i].
402         elementPropertyValues.Add(classObjectString);
403
404         if (classObject.ContainsKey(key))
405         {
406             JArray commentArray = classObject[key] as JArray;
407             foreach (JObject commentObject in commentArray)
408             {
409                 string comment = commentObject["@value"].
410                 Value<string>();
411                 ontology.extractedBubbleDatasList[i].
```

---

```
        elementPropertyValues.Add(comment);
409    }
410}
411}
412}
413}
414
415    public void ParseToJsonLD(string filePath, string folderPath,
416        string saveFileName)
417    {
418        string jsonData = File.ReadAllText(filePath);
419
420        JObject jsonObject = JsonConvert.DeserializeObject<
421            JObject>(jsonData);
422
423        JArray bubblesDataList = jsonObject["  

424            extractedBubbleDatasList"] as JArray;
425
426        foreach (JObject bubbleData in bubblesDataList)
427        {
428            Dictionary <string, object> dataToAdd = new();
429
430            JArray elementPropertyValuesArray = bubbleData["  

431                elementPropertyValues"] as JArray;
432            JArray subClassArray = bubbleData["  

433                inheritedBubblesIDsList"] as JArray;
434            JArray ownedByBubblesArray = bubbleData["  

435                ownedByBubblesIDsList"] as JArray;

if (bubbleData["ontologyTag"].Value<string>() == "  

droneprovider")
{
    dataToAdd["@id"] = bubbleData["elementName"].  

        Value<string>();
    GenericObject type = new();
```

---

```

436     type.ID = "owl:Class";
437
438     dataToAdd["rdf:type"] = type;
439
440     int amountOfSubclasses = subClassArray.Count +
441         ownedByBubblesArray.Count;
442
443     // in Erik's ontology, if there is more than one
444     // connection, either subclass or property, it
445     // will be added to a list. If there is only one
446     // connection, it will be a simple JObject
447
448     List<object> connections = new();
449     if(subClassArray.Count > 0)
450     {
451         List<string> names = GetNamesFromIDs(
452             subClassArray, bubblesDataList, false);
453         if(amountOfSubclasses > 1)
454         {
455             connections.AddRange(names.Select(name =>
456                 new GenericObject { ID = name }));
457         }
458         else
459         {
460             GenericObject connection = new
461                 GenericObject { ID = names[0] };
462             dataToAdd["rdfs:subClassOf"] = connection
463                 ;
464         }
465     }
466
467     if(ownedByBubblesArray.Count > 0)
468     {
469         List<string> ownerNames = GetNamesFromIDs(
470             ownedByBubblesArray, bubblesDataList, false
471         );

```

---

```
462     for(int i = 0; i < ownedByBubblesArray.Count;
463         i++)
464     {
465         string[] labels = ownedByBubblesArray[i].
466             Value<string>().Split(separator);
467         string[] parts = labels[labels.Length-1].
468             Split(divider);
469         foreach(string separateLabel in parts)
470         {
471             string ownerTag = GetTagFromID(
472                 ownedByBubblesArray[i].Value<string
473                     >(), bubblesDataList);
474             RestrictionObject connection = new
475                 RestrictionObject {
476                     type = new GenericObject { ID =
477                         "owl:
478                             Restriction" },
479                     property = new GenericObject { ID =
480                         separateLabel },
481                     destination = new GenericObject { ID
482                         = ownerNames[i] }
483                 };
484             if(amountOfSubclasses > 1 || parts.
485                 Length > 1) // there is more than
486                 one connection
487             {
488                 connections.Add(connection);
489                 dataToAdd["rdfs:subClassOf"] =
490                     connections;
491             }
492             else
493             {
494                 dataToAdd["rdfs:subClassOf"] =
495                     connection;
496             }
497         }
498     }
499 }
```

---

```

485     }
486
487     JObject jsonDataToAdd = JObject.FromObject(
488         dataToAdd);
489     myGraph.Add(jsonDataToAdd);
490
491 }
492 else
493 {
494     Dictionary <string,object> dataToExport = new();
495     dataToExport["@id"] = "http://w3id.org/gaia-x/" +
496         bubbleData["elementName"].Value<string>();
497
498     List<string> classTypeList = new();
499     classTypeList.Add("http://www.w3.org/2002/07/owl#
500         Class");
501     dataToExport["@type"] = classTypeList;
502
503     if(elementPropertyValuesArray.Count > 0)
504     {
505         List<CommentObject> commentObjects = new();
506         commentObjects.AddRange(
507             elementPropertyValuesArray.Select(comment
508                 => new CommentObject { Value = comment.
509                     ToString() }));
510
511         dataToExport["http://www.w3.org/2000/01/rdf-
512             schema#comment"] = commentObjects;
513     }
514
515     if(subClassArray.Count > 0)
516     {
517         List<GenericObject> subClassObjects = new();
518         List<string> names = GetNamesFromIDs(
519             subClassArray, bubblesDataList, true);

```

---

```

513     subClassObjects.AddRange(names.Select(name =>
514         new GenericObject { ID = name }));
515     dataToExport["http://www.w3.org/2000/01/rdf-schema#subClassOf"] = subClassObjects;
516 }
517
518 if(ownedByBubblesArray.Count > 0)
{
519     List<string> ownerNames = GetNamesFromIDs(
520         ownedByBubblesArray, bubblesDataList, true)
521         ;
522     for(int i = 0; i < ownedByBubblesArray.Count;
523         i++)
524     {
525         // labels are after the separator so we
526         // need to split the string to get them
527         string[] labels = ownedByBubblesArray[i].
528             Value<string>().Split(separator);
529         string[] parts = labels[labels.Length-1].
530             Split(divider);
531         foreach(string separateLabel in parts)
532         {
533             Dictionary<string,object>
534                 objectProperty = new();
535             string ownerTag = GetTagFromID(
536                 ownedByBubblesArray[i].Value<string
537                     >(), bubblesDataList);
538             objectProperty["@id"] = "http://w3id.org/gaia-x/" + ownerTag + "#" +
539                 separateLabel;
540             List<string> objectPropertyTypeList =
541                 new();
542             objectPropertyTypeList.Add("http://
543                 www.w3.org/2002/07/owl#
544                 ObjectProperty");
545             objectProperty["@type"] =

```

---

```

533         objectPropertyTypeList;
534
535     List<DomainRangeObject>
536         domainObjectList = new();
537     DomainRangeObject domainObject = new
538         ();
539     domainObject.ID = ownerNames[i];
540     domainObjectList.Add(domainObject);
541     objectProperty["http://www.w3.org/2000/01/rdf-schema#domain"] =
542         domainObjectList;
543
544     List<DomainRangeObject>
545         rangeObjectList = new();
546     DomainRangeObject rangeObject = new()
547         ;
548     rangeObject.ID = "http://w3id.org/gaia-x/" + bubbleData["elementName"];
549     rangeObjectList.Add(rangeObject);
550     objectProperty["http://www.w3.org/2000/01/rdf-schema#range"] =
551         rangeObjectList;
552     JObject newConnectionToExport =
553         JObject.FromObject(objectProperty);
554     exportToJson.Add(
555         newConnectionToExport);
556     }
557     }
558 }
559
560 JObject newBubbleToExport = JObject.FromObject(
561     dataToExport);
562 exportToJson.Add(newBubbleToExport);
563 }
564 }
```

---

```

555     JObject wrapperObject = new JObject();
556     wrapperObject.Add("@context", ontologyContext);
557     wrapperObject.Add("@graph", myGraph);
558     exportToJson.Add(wrapperObject);

559
560     string fullPath = Path.Combine(folderPath, saveFileName);

561
562     string jsonDataToExport = JsonConvert.SerializeObject(
563         exportToJson, Formatting.Indented);
564     File.WriteAllText(fullPath, jsonDataToExport);
565 }

566 private List<string> GetNamesFromIDs(JArray IdList, JArray
567     bubblesList, bool flag) // true = url ids, false = parsed
568     ids
569 {
570     List<string> names = new();
571     if(flag)
572     {
573         for(int i = 0; i < IdList.Count; i++)
574         {
575             string idAndLabel = IdList[i].Value<string>();
576             string[] parts = idAndLabel.Split(separator);
577             string LookingForID = parts[0];
578             foreach(JObject bubbleData in bubblesList)
579             {
580                 string bubbleID = bubbleData["elementID"].
581                     Value<string>();
582                 if(LookingForID == bubbleID)
583                 {
584                     names.Add("http://w3id.org/gaia-x/" +
585                         bubbleData["elementName"].Value<string
586                         >());
587                     break;
588                 }
589             }
590         }
591     }

```

---

```

585    }
586  }
587  else // this routine is for when we are looking for
588      classes names to add to the drone provider ontology in
589      its own format.
590  // for example a core#consumable class would be "gax-core
591  :consumable", so we get the data and parse it
592  {
593      for(int i = 0; i < IdList.Count; i++)
594      {
595          string idAndLabel = IdList[i].Value<string>();
596          string[] parts = idAndLabel.Split(separator);
597          string LookingForID = parts[0];
598          foreach(JObject bubbleData in bubblesList)
599          {
600              string bubbleID = bubbleData["elementID"].
601                  Value<string>();
602              if(LookingForID == bubbleID)
603              {
604                  if(bubbleData["ontologyTag"].Value<string
605                      >() != "droneprovider")
606                  {
607                      Array.Clear(parts, 0, parts.Length);
608                      // empty array
609                      parts = bubbleData["elementName"].
610                          Value<string>().Split('#');
611                      string result = parts[parts.Length
612                          -1];
613
614                      names.Add("gax-" + bubbleData["
615                          ontologyTag"].Value<string>() + ":" +
616                          result);
617                  }
618                  else
619                  {
620                      names.Add(bubbleData["elementName"] .

```

---

```

                                Value<string>());
611                         }
612                         break;
613                     }
614                 }
615             }
616         }
617
618
619     return names;
620 }
621
622 private string GetTagFromID(string idAndLabel, JArray
623 bubblesList)
624 {
625     // the id from the ownedByBubblesIDsList has also a
626     // separator and the connection label
627     string[] parts = idAndLabel.Split(separator);
628     string LookingForID = parts[0];
629     string tag = "";
630     foreach(JObject bubbleData in bubblesList)
631     {
632         string bubbleID = bubbleData["elementID"].Value<
633             string>();
634         if(LookingForID == bubbleID)
635         {
636             tag = bubbleData["ontologyTag"].Value<string>();
637             break;
638         }
639     }
}

```

Código 2.2: Parser.cs

```

1 public class OntologyContainer : MonoBehaviour

```

```
2 {
3     private float maxX, maxY, maxZ;
4     private float minX, minY, minZ;
5     private bool isInitialCase = true;
6     private List<GameObject> classes = new();
7     private MeshRenderer cubeRenderer = new();
8     [SerializeField] private GameObject ContainerInteract;
9     [SerializeField] private TMP_Text nameText;
10    private void Start()
11    {
12        classes = gameObject.transform.parent.GetComponent<
13            Ontology>().classes;
14        ContainerInteract.GetComponent<VR_ContainerInteract>().
15            classes = classes;
16        cubeRenderer = gameObject.GetComponent<MeshRenderer>();
17        nameText.text = transform.parent.name;
18    }
19
20    public void SpawnContainer()
21    {
22        isInitialCase = true;
23        DefineSize();
24        ContainerInteract.SetActive(true);
25        nameText.gameObject.SetActive(true);
26    }
27
28    public void HideContainer()
29    {
30        cubeRenderer.enabled = false;
31        ContainerInteract.SetActive(false);
32        nameText.gameObject.SetActive(false);
33    }
34
35    private void DefineSize()
36    {
37        for (int i = 0; i < classes.Count; i++)
```

```
36    {
37        FindExtremes(classes[i].transform.position);
38    }
39
40    SetCubeSizeAndPosition();
41}
42
43 private void SetCubeSizeAndPosition()
44 {
45     // Calculate size
46     float sizeX = maxX - minX + 2.0f;
47     float sizeY = maxY - minY + 2.0f;
48     float sizeZ = maxZ - minZ + 2.0f;
49
50     // Calculate center position
51     float centerX = (minX + maxX) / 2f;
52     float centerY = (minY + maxY) / 2f;
53     float centerZ = (minZ + maxZ) / 2f;
54
55     // Set the cube's size and position
56     gameObject.transform.localScale = new Vector3(sizeX,
57         sizeY, sizeZ);
58     Vector3 newPosition = new Vector3(centerX, centerY,
59         centerZ);
60     gameObject.GetComponent<Multiplayer_MoveObjects>().
61         Mutliplayer_UpdatePosition(newPosition); // spawn the
62         container
63
64         cubeRenderer.enabled = true;
65
66         cubeRenderer.material.color = new Color(Random.value,
67             Random.value, Random.value, 0.4f);
68    }
69
70    private void FindExtremes(Vector3 bubblePosition)
71    {
```

---

```

67     if(bubblePosition.x < minX || isInitialCase)
68     {
69         minX = bubblePosition.x;
70     }
71     if (bubblePosition.y < minY || isInitialCase)
72     {
73         minY = bubblePosition.y;
74     }
75     if (bubblePosition.z < minZ || isInitialCase)
76     {
77         minZ = bubblePosition.z;
78     }

79
80     if (bubblePosition.x > maxX || isInitialCase)
81     {
82         maxX = bubblePosition.x;
83     }
84     if (bubblePosition.y > maxY || isInitialCase)
85     {
86         maxY = bubblePosition.y;
87     }
88     if (bubblePosition.z > maxZ || isInitialCase)
89     {
90         maxZ = bubblePosition.z;
91     }

92
93     isInitialCase = false;
94 }
95 }
```

Código 2.3: OntologyContainer.cs

1	/* CURRENT STATUS
2	
3	This feature works but has some bugs and miscalculations in the
	sphere shape.
4	I will include proper documentation explaining what is going on

---

```
    and how to fix it.

5
6 All the geometrical calculations for the sphere shape are made
7     with the hypothesis of starting from
8 a flat layout, so either we switch to "flat" every time we switch
9     layouts, or we generalize the calculation of nodes
10 for both shapes.
11 */
12
13 public class Ontology : MonoBehaviour
14 {
15     /// <summary>
16     /// each ontology empty game object holds a list of all the
17     /// classes it owns
18     /// </summary>
19     public List<GameObject> classes = new();
20     private float zDefaultPosition = 0.0f; // verify in Parser.cs
21         the default value in the ImportedPositionInformation class
22
23     private void SortClasses(string axis)
24     {
25         switch(axis)
26         {
27             case "x":
28                 classes.Sort((a, b) => a.transform.position.x.
29                     CompareTo(b.transform.position.x));
30                 break;
31             case "y":
32                 classes.Sort((a, b) => a.transform.position.y.
33                     CompareTo(b.transform.position.y));
34                 break;
35             case "z":
36                 classes.Sort((a, b) => a.transform.position.z.
37                     CompareTo(b.transform.position.z));
38                 break;
39             default:
```

---

```
33         break;
34     }
35 }
36 }
37
38 private float FindRadius(Vector3 center)
39 {
40     float maxDistance = 0.0f;
41     foreach(GameObject classObject in classes)
42     {
43         float distance = Vector3.Distance(classObject.
44             transform.position, center);
45         if(distance > maxDistance)
46         {
47             maxDistance = distance;
48         }
49     }
50     return maxDistance;
51 }
52
53 public void SelectBubblesLayout(string mode)
54 {
55     switch (mode)
56     {
57         case "cylinder":
58
59             /* radius will be calculated with the list in
60              crecent order according to x position,
61              so we need to sort first by z and then do x. Plus
62              the first and last element wont be moved and
63              will serve as
64              opposites in the side of the cylinder
65
66             SortClasses("z");
67             float zClosestValue = classes[0].transform.
```

---

```

                position.z;
65        SortClasses("x");
66        float radius = (classes[classes.Count - 1].
67                        transform.position.x - classes[0].transform.
68                        position.x) / 2.0f;
69        float cylinderCenter = classes[0].transform.
70                        position.x + radius;
71        if (radius > 0) // this means that the ontology
72                        has more than 1 element
73        {
74            for (int i = 0; i < classes.Count; i++)
75            {
76                float acosArgument = (classes[i].
77                                transform.position.x - cylinderCenter)
78                                / radius;
79                if (Mathf.Abs(acosArgument) > 1)
80                {
81                    Debug.LogWarning("cosine out of order
82                                with a value of " + acosArgument);
83                    if (acosArgument > 1) acosArgument =
84                        1;
85                    else if (acosArgument < -1)
86                        acosArgument = -1;
87                }
88                float angle = Mathf.Acos(acosArgument);
89                float zNewPosition = zClosestValue +
90                    Mathf.Sin(angle) * radius;
91                Vector3 newPosition = new Vector3(classes
92                                [i].transform.position.x, classes[i].
93                                transform.position.y, zNewPosition);
94                classes[i].GetComponent<
95                                Multiplayer_MoveObjects>().
96                                Multiplayer_VR_MoveBubble(newPosition);
97            }
98        }
99        break;

```

---

```

86
87     /* the sphere layout will look at the classes of the
88      ontology and look for the (x,y,z) coordinates
89      * of the center by dividing by 2 the difference
90      between the first and last element of the classes
91      list
92      * ( ordered according to each axis ). With this
93      center, we'll look for the furthest bubble of the
94      ontology and use that distance as the sphere
95      radius. With that sphere equation and the (x,y)
96      coordinates of all bubbles, we'll look for the
97      third one.
98      * Note : there are 2 possibilites for each pair of x
99      y values because of the sqrt. Take into
100     consideration */
101
102     case "sphere":
103
104         /* the sphere doesnt matter so much which axis we
105          use to sort the classes, as all of them will
106          be calculated */
107         SortClasses("z");
108         zClosestValue = classes[0].transform.position.z;
109         SortClasses("x");
110         float xRadius = (classes[classes.Count - 1].
111                         transform.position.x - classes[0].transform.
112                         position.x) / 2.0f;
113         float centerX = classes[0].transform.position.x +
114             xRadius;
115         SortClasses("y");
116         float yRadius = (classes[classes.Count - 1].
117                         transform.position.y - classes[0].transform.
118                         position.y) / 2.0f;
119         float centerY = classes[0].transform.position.y +
120             yRadius;

```

```
106     Vector3 sphereCenter = new Vector3(centerX,
107         centerY, zClosestValue);
108
109     radius = FindRadius(sphereCenter);
110
111     if (radius > 0)
112     {
113         for(int i = 0; i < classes.Count; i++)
114         {
115             float xDistance = classes[i].transform.
116                 position.x - centerX;
117             float yDistance = classes[i].transform.
118                 position.y - centerY;
119
119             float zSquared = Mathf.Pow(radius, 2.0f)
120                 - Mathf.Pow(xDistance, 2.0f) - Mathf.
121                 Pow(yDistance, 2.0f);
122
123             if (zSquared >= 0) // Check if zSquared
124                 is non-negative
125             {
126                 float zNewPosition = sphereCenter.z +
127                     Mathf.Sqrt(zSquared);
128
128                 Vector3 newPosition = new Vector3(
129                     classes[i].transform.position.x,
130                     classes[i].transform.position.y,
131                     zNewPosition);
132
132                 classes[i].GetComponent<
133                     Multiplayer_MoveObjects>().
134                     Multiplayer_VR_MoveBubble(
135                     newPosition);
136             }
137             else
138             {
139                 //Debug.LogWarning("zSquared is
```

---

```
negative for object: " + classes[i]
].name);
129 //Debug.LogWarning("xDistance: " +
xDistance + ", yDistance: " +
yDistance);
130 //Debug.LogWarning("Radius: " +
radius);
131 //Debug.LogWarning("Zsquared = " +
zSquared);

132
133 // after analysis and debug, I found
// that errors in calculation are
// because of floating point and
// approximation, so the
134 // square root returns a value very
// very close to 0 but negative, so we
// cant assign it

135
136 Vector3 newPosition = new Vector3(
    classes[i].transform.position.x,
    classes[i].transform.position.y,
    0.0f);
137 classes[i].GetComponent<
    Multiplayer_MoveObjects>().
    Multiplayer_VR_MoveBubble(
    newPosition);
138 }
139 }
140 }
141 break;

142
143 case "flat": // initial flat layout
144 SortClasses("z");
145 zClosestValue = classes[0].transform.position.z;
146 foreach (GameObject obj in classes)
147 {
```

```
148     Vector3 newPosition = new Vector3(obj.  
149         transform.position.x, obj.transform.  
150         position.y, zClosestValue);  
151     obj.GetComponent<Multiplayer_MoveObjects>().  
152         Multiplayer_VR_MoveBubble(newPosition);  
153     }  
154     break;  
155 }  
156 }  
157 }
```

Código 2.4: Parser.cs