Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención Computación

AppSpace: Publicación de un repositorio de metadatos sobre aplicaciones móviles en Espacios de Datos

Autor:
Alfonso Cabrero de Diego

Tutores:
Mercedes Martínez González
Alejandro Pérez de La Fuente

Resumen

Los espacios de datos son una tecnología en pleno auge, capaz de integrar fuentes de datos heterogéneas e independientes. Los espacios de datos no almacenan la información, sino que abordan la integración de los metadatos que permiten descubrir las fuentes y acceder a sus contenidos. Cada espacio de datos establece su propio marco de gobernanza, orientado a crear un entorno seguro y promover la confianza entre los participantes. Además, cada participante tiene la libertad de establecer las condiciones que considere mas adecuadas para acceder a sus datos. En este contexto, los espacios de datos son capaces de amplificar las ventajas de compartir información, y benefician no solo a sus usuarios, sino a toda la sociedad.

En este Trabajo de Fin de Grado se utilizan espacios de datos para compartir un repositorio de metadatos sobre aplicaciones móviles. Para ello, se han diseñado los metadatos que protegen, describen, y permiten descubrir este repositorio. Como tecnología base, se han utilizado componentes de EDC e INESData, dos proyectos pioneros en su desarrollo.

Palabras clave: Espacios de datos, integración, privacidad, ontología, EDC, INESData.

Abstract

Data spaces are a rapidly emerging technology capable of integrating heterogeneous and independent data sources. Data spaces do not store information themselves, but rather focus on the integration of metadata that enable the discovery of data sources and posterior access to their content. Each data space defines its own governance framework, aimed at creating a secure environment and fostering trust among participants. Moreover, each participant has the freedom to set the conditions they consider most appropriate for accessing their data. In this context, data spaces amplify the advantages of information sharing and benefit not only their users but society as a whole.

In this Bachelor's Thesis, data spaces are used to share a metadata repository about mobile applications. To that end, metadata have been designed to protect, describe, and enable the discovery of this repository. As foundational technology, components from EDC and INESData have been employed, two pioneering projects in this field.

Keywords: Data spaces, integration, privacy, ontology, EDC, INESData.

Agradecimientos

En primer lugar, quiero agradecer profundamente a mi familia por su apoyo incondicional durante todo este tiempo. Vuestros consejos, ánimo y confianza han sido esenciales en este camino.

También quiero expresar mi sincero agradecimiento a mis tutores, Alejandro y Mercedes, por su dedicación, guía y valiosas aportaciones.

Y también quiero darles las gracias a mis amigos y compañeros, por su compañía, por su motivación, y por ser una fuente inagotable de inspiración y aprendizaje.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

Los datos se han convertido en el nuevo protagonista de la economía moderna. Cada vez son más las empresas, instituciones y organismos públicos que reconocen el valor estratégico de los datos como herramienta clave para mejorar la toma de decisiones, optimizar procesos y desarrollar servicios más eficientes y personalizados.

Los sistemas de recomendación, el análisis predictivo, el *Business Intelligence* o la automatización de procesos mediante inteligencia artificial dependen de la calidad y de la cantidad de los datos disponibles para ofrecer resultados efectivos y precisos. Además, todos estos sistemas requieren de una infraestructura tecnológica sólida como centros de datos o redes de comunicaciones de alta capacidad. En este contexto, la Comisión Europea estima que la contribución de la economía del dato aportará al PIB de la Unión más de 630.000 millones de euros en 2025 [1].

Pero a pesar del gran tamaño del mercado, la industria se enfrenta todavía a importantes retos. Uno de ellos es la integración de datos procedentes de múltiples fuentes, formatos y dominios, especialmente en entornos donde participan diferentes organizaciones. Esto es relevante porque los intercambios de datos promueven el descubrimiento de nuevo conocimiento, fomentan la innovación y facilitan la colaboración, amplificando así el impacto positivo que los datos pueden generar dentro y fuera de una única organización [2].

Los espacios de datos surgen para hacer frente a este problema, creando un entorno que permite compartir información de forma controlada, segura e interoperable. Se basan en principios como la soberanía del dato, la descentralización y la trazabilidad de la información, alineados con la estrategia de datos de la Unión Europea.

Como tecnología emergente, los espacios de datos aún se encuentran en proceso de maduración y estandarización. Esto significa que queda mucho trabajo por delante antes de que puedan ser adoptados de forma generalizada. Son una solución innovadora que seguirá evolucionando, pero que se espera acabe convirtiéndose en la base de la nueva economía digital, donde la confianza, la colaboración y la soberanía sean pilares fundamentales.

1.2. Motivación

En este trabajo utilizamos espacios de datos para compartir un repositorio desarrollado y mantenido por el Proyecto App Privacy Impact (App-PI) de la Universidad de Valladolid [3]. Estos datos evalúan el impacto de las aplicaciones móviles sobre la privacidad de los usuarios, un derecho básico que resulta vital proteger.

Este repositorio tiene el nombre de App-PIMD (App Privacy Impact MetaData). De momento se pueden consultar sus datos a través de su API[1], o para usuarios menos técnicos, a través del servicio web APK Falcon[2]. Este trabajo se suma a estas dos soluciones, contribuyendo a dar a conocer esta información para que los usuarios de las aplicaciones puedan tomar decisiones informadas y los desarrolladores adopten prácticas más respetuosas con la privacidad.

Las virtudes de los espacios de datos permiten compartirlos con mayor confianza y seguridad, ya que es posible establecer políticas de acceso y uso que limiten ciertos usos o que permitan el acceso solo a cierto tipo de usuarios. Además, establecen mecanismos para garantizar la aplicación y cumplimiento de estas políticas. Esto permitiría por ejemplo ampliar la oferta de datos del repositorio bajo las condiciones que se consideren más adecuadas, fomentando la colaboración y facilitando la creación de soluciones basadas en datos compartidos.

1.3. Objetivos

El principal objetivo del trabajo es **compartir el repositorio App-PIMD utilizando espacios de datos**. La forma de hacer esto es participando con un proveedor en algún espacio de datos con la temática adecuada[3]. En nuestro caso esta temática podría ser la seguridad, o la privacidad. Por desgracia, no hemos encontrado ningún espacio de datos compatible, por lo que trataremos de **implementar un proveedor aislado** que pueda servir de referencia para trabajos posteriores. Para nuestro proveedor utilizaremos componentes de INESData, una incubadora de espacios de datos pionera en España.

Además, gracias a los créditos para estudiantes de Google Cloud proporcionados por la tutora, un objetivo secundario del trabajo será desplegar el espacio de datos en esta plataforma y aprovechar para aprender sobre esta tecnología.

Resumiendo, nos planteamos los siguientes objetivos más concretos:

- **Investigar** la tecnología de los **espacios de datos**.
- **Compartir el repositorio** App-PIMD a través de un proveedor.
- **Crear un espacio de datos local** (AppSpace) que permita probar el proveedor.
- **Desplegar** el espacio de datos **en la Nube de Google**.

[1] Enlace a la documentación de la API de App-PIMD: https://app-pi.infor.uva.es/docs.
[2] Enlace a APK Falcon: https://apkfalcon.inf.uva.es/.
[3] Los espacios de datos se organizan en torno a sectores temáticos como la salud, la movilidad o la agricultura.

1.4. Estructura de la memoria

Iniciamos todos los capítulos resumiendo su contenido, explicando cómo se relacionan con el resto del documento y conectando cada una de sus secciones. Con esta estructura buscamos facilitar al lector la búsqueda de información y la comprensión global del trabajo.

Esta memoria se divide en 7 capítulos con los siguientes contenidos:

Capítulo 1 - Introducción: Presentación del trabajo, contexto y relevancia.

Capítulo 2 - Planificación del trabajo, metodología, estimaciones de tiempo, riesgos y costes.

Capítulo 3 - Espacios de Datos: qué son y cómo funcionan. Contexto e iniciativas a nivel internacional, estándares. Ejemplo de un proyecto exitoso.

Capítulo 4 - Requisitos y análisis de un proveedor de un espacio de datos siguiendo los estándares de los espacios de datos internacionales (IDS, por sus siglas en inglés).

Capítulo 5 - Diseño del proveedor y de los metadatos necesarios, utilizando los componentes de Eclipse Dataspace Components (EDC) e INESData.

Capítulo 6 - Implementación, despliegue y pruebas: Implementación de AppSpace, proceso, tecnologías utilizadas y pruebas realizadas.

Capítulo 7 - Conclusiones y trabajo futuro: Resumen de los resultados del trabajo, análisis de las limitaciones, partes inacabadas y mejoras propuestas.

Bibliografía Referencias bibliográficas.

Apéndice A - Herramientas utilizadas: Listado de todas las herramientas de software utilizadas para el trabajo.

Capítulo 2

Planificación

La planificación es una parte esencial de cualquier proyecto, y nos permite anticipar problemas, asignar recursos de manera eficiente y cumplir con los plazos establecidos. En este capítulo describimos el proceso que hemos seguido para planificar este trabajo.

Empezamos describiendo la metodología que hemos utilizado en la Sección 2.1, que se basa en un **modelo incremental**, y que nos permitirá avanzar de forma estructurada y flexible. En la Sección 2.2, enumeramos y describimos todos los **recursos** que necesitaremos para el proyecto, incluyendo un desglose de sus costes.

La Sección 2.3 aborda la **gestión del trabajo**. Aquí definimos los incrementos del proyecto, los cuales dividimos en tareas específicas. Enumeramos estas tareas en una lista en la que damos más detalles sobre cada una. También las representamos en la Figura 2.2: Estructura de División del Trabajo (EDT), donde se pueden visualizar de forma clara y organizada. Después en la Sección 2.4, **estimamos la duración** de cada una de ellas y elaboramos el **calendario del proyecto**. Lo representamos en forma de un Diagrama de Gantt en la Figura 2.3: Cronograma del Proyecto.

La Sección 2.5 se centra en la **gestión de los riesgos**, donde analizamos los posibles problemas que podrían afectar al proyecto. Aquí los identificamos, los clasificamos y establecemos estrategias para minimizar su impacto. En la Sección 2.6, calculamos una **estimación el coste** total del proyecto utilizando valores representativos en un entorno profesional. El principal coste del proyecto es con diferencia el tiempo de trabajo del autor y de los tutores.

Para terminar el capítulo, en la Sección 2.7, realizamos un seguimiento al proyecto y a la planificación. En esta sección describimos las problemáticas con las que nos hemos encontrado, y como las hemos abordado. Hemos tenido que modificar levemente el calendario del proyecto, y ajustar también el alcance. Por último, también mostramos la evolución del progreso del trabajo, y comparamos las estimaciones de tiempo de cada tarea con su duración real.

2.1. Metodología

La metodología que hemos seguido en este trabajo es la de un **modelo incremental**. Este es un tipo de proceso iterativo que permite adaptar el diseño del sistema a medida que se desarrolla. Esta metodología se basa en **fragmentar la funcionalidad** del sistema, y de implementarla de manera gradual en incrementos (o iteraciones). En cada incremento se desarrolla una funcionalidad nueva o se mejora alguna ya existente [5]. De esta manera, se dispone de un sistema funcional, el cual es mejorado con cada incremento.

Hemos elegido esta metodología para este trabajo porque la tecnología que pretendemos usar está actualmente en desarrollo[1]. Esto implica un riesgo mucho más elevado de encontrarnos con problemas durante el proyecto, por ejemplo, partes incompletas, errores, o falta de documentación. Todo ello podría provocar retrasos, o incluso la imposibilidad de implementar alguna funcionalidad. Es aquí cuando entran en juego las **ventajas del modelo incremental**:

- Permite **modificar el diseño del proveedor** a medida que se avanza con el proyecto.
- Proporciona **flexibilidad** en caso de que algún incremento no se desarrolle con éxito. Simplemente se puede pasar al siguiente.
- Permite **aprender cada tecnología con más en detalle**, ya que se le dedica un incremento a cada una[2].
- Permite tener un **proveedor operativo desde el primer incremento**, por tanto, aunque el resto de los incrementos fracasen, el objetivo principal del trabajo, que es compartir el repositorio de datos, se habrá cumplido.

Al final de cada incremento, hemos organizado una reunión para mostrar el sistema y la documentación a los tutores. En este punto también se pueden revaluar los requisitos y modificarse de ser necesario.

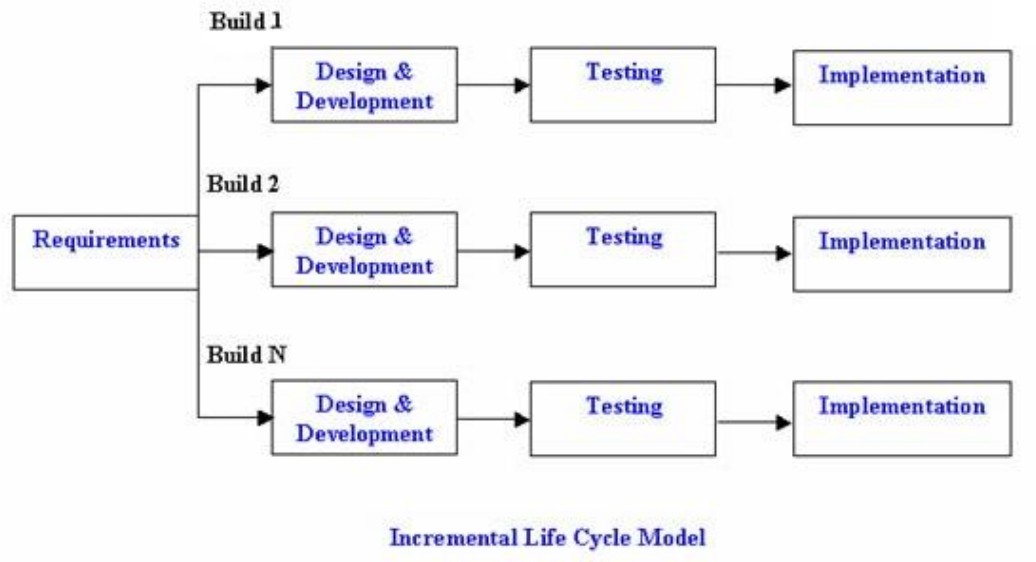


Figura 2.1: Diagrama del modelo incremental [5]

2.2. Gestión de los recursos

Para realizar este proyecto se necesitarán una amplia variedad de recursos, que se pueden clasificar en diferentes categorías: **mano de obra, materiales/herramientas, energía, dinero, espacio o tiempo**. Dentro de la categoría de **herramientas** cabe destacar los recursos de software. Estos se de detallan de forma exhaustiva en el Apéndice A: Herramientas utilizadas. Como ninguno de estos recursos supone un coste explícito y no tienen restricciones de disponibilidad, se han omitido de la tabla por simplicidad.

[1] La iniciativa de INESData pretende crear una incubadora de espacios de datos en España [4]. El proyecto se inicia en 2023 y su desarrollo todavía continúa. En la Subsección 3.4.2: INESData proporcionamos información adicional.

[2] Como se explica con más detalle en la Subsección 3.4.2: INESData (y en la Subsección 5.3.4: Componentes), el conector de INESData se basa en el *framework* de Eclipse Dataspace Components (EDC). Por tanto al utilizar EDC en un primer incremento, estaremos mucho mejor preparados para resolver errores cuando utilicemos el conector de INESData.

En cuanto al **dinero**, el presupuesto para este proyecto es de 50€. La mayoría de los costes que se reflejan en la tabla son o bien ficticios (por depreciación, de oportunidad), o bien soportados por terceros (créditos para estudiantes de Google Cloud). Los únicos costes reales son los de **energía**, que son relativamente pequeños.

RECURSO	DESCRIPCIÓN	DISPONIBILIDAD	COSTE[3]
Alumno	Mano de obra del alumno	5h al día	18,23€/h
Tutores	Mano de obra de los tutores	5h al mes	42,77€/h
Portátil 1	Para investigación, redacción y uso del proveedor	Siempre	0,12€/h
Portátil 2	Infraestructura de despliegue en local	Siempre	0,13€/h
Google Cloud	Infraestructura de despliegue en la nube	72,11€ en total	0,23€/h
Lugar de trabajo	Vivienda del alumno	Siempre	1,20€/h
Lugar de reunión	Sala de la Escuela	Reservar antes	-
Energía eléctrica	Para los ordenadores portátiles	Siempre	0,2015€/kWh
Calefacción	Para el lugar de trabajo	Siempre	0,2015€/h
Tiempo	Límite orientativo	300h en total	-
Dinero	Presupuesto del proyecto	50€ en total	-

Tabla 2.1: Recursos del proyecto

2.3. Gestión del trabajo

Como se explica en la Sección 2.1: Metodología, se seguirá un **plan de trabajo basado en incrementos**. De esta manera, dividiremos el trabajo cuatro incrementos, además de uno inicial y otro final. El desglose de las tareas se puede observar en la lista a continuación y en la Figura 2.2: Estructura de División del Trabajo (EDT), elaborada con la herramienta draw.io [6].

Incremento inicial Agrupa las tareas que se han realizado antes de planificar el proyecto. Las dos primeras tareas se han realizado de forma paralela, seguidas por la planificación del trabajo. Las tareas del incremento inicial son las siguientes:

- Definición del proyecto: objetivos y alcance.
- Investigación inicial: búsqueda de tecnologías disponibles, posibilidades que ofrecen, contexto internacional sobre Espacios de Datos.
- Planificación.
- Memoria: Capítulo 1: Introducción (Objetivos y Estructura de la memoria), Capítulo 2: Planificación (excepto Seguimiento del proyecto) y Capítulo 3: Espacios de Datos (¿Qué son los espacios de datos?).

Primer incremento Se desarrolla un primer proveedor utilizando los componentes de Eclipse Dataspace Components (EDC). EDC tiene una documentación más extensa y completa por lo podremos aprender la base de la próxima tecnología que utilizaremos: INESData. Además, se diseñará el proveedor, junto con una ontología para compartir los datos.

[3]Para el detalle del cálculo de los costes ver Sección 2.6: Gestión de los costes.

- Investigación: Espacios de Datos, *framework* de EDC, componentes y funcionamiento.
- Análisis de requisitos del proveedor y diseño del proveedor.
- Diseño de la ontología para compartir los datos.
- Creación y despliegue de un Espacio de Datos en local utilizando los componentes de EDC. *Warehouse* en local.
- Memoria: Capítulo 3: Espacios de Datos (resto del capítulo), Capítulo 4: Requisitos y análisis, Capítulo 5: Diseño (excepto INESData), y Capítulo 6: Implementación, despliegue y pruebas (EDC).

Segundo incremento Se cambia la tecnología del proveedor para utilizar los componentes de INESData. Se adapta el diseño del sistema y se reutiliza el resto de elementos del incremento anterior.

- Investigación: INESData, componentes y extensiones.
- Diseño del proveedor.
- Creación y despliegue de un Espacio de Datos en local utilizando los componentes de INESData. *Warehouse* en local.
- Memoria: Capítulo 5: Diseño (INESData), y Capítulo 6: Implementación, despliegue y pruebas (segundo incremento).

Tercer incremento Se añade la interfaz gráfica al proveedor.

- Investigación: Componente de interfaz gráfica de INESData.
- Adaptación del diseño con interfaz gráfica.
- Despliegue del Espacio de Datos donde el proveedor tiene interfaz gráfica.
- Memoria: Capítulo 5: Diseño (tercer incremento), y Capítulo 6: Implementación, despliegue y pruebas (tercer incremento).

Cuarto incremento Se despliega el sistema en la infraestructura de la Nube de Google.

- Investigación: Computación en la Nube, Nube de Google en específico.
- Adaptación del diseño a despliegue en la nube.
- Despliegue en la nube tanto del proveedor como del Warehouse.
- Memoria: Capítulo 5: Diseño (cuarto incremento), y Capítulo 6: Implementación, despliegue y pruebas (cuarto incremento).

Incremento final Se redactan los capítulos que faltan, los apéndices y se revisa el resto. Revisión a la planificación.

- Resumen, Abstract.
- Capítulo 1: Introducción (Motivación, Contexto).
- Capítulo 2: Planificación (Seguimiento del proyecto).
- Capítulo 7: Conclusiones y trabajo futuro.
- Apéndice A: Herramientas utilizadas.

Al final de cada incremento Se hacen después de todos los incrementos, a medida que avanza el proyecto.

- Pruebas.
- Reuniones de seguimiento.

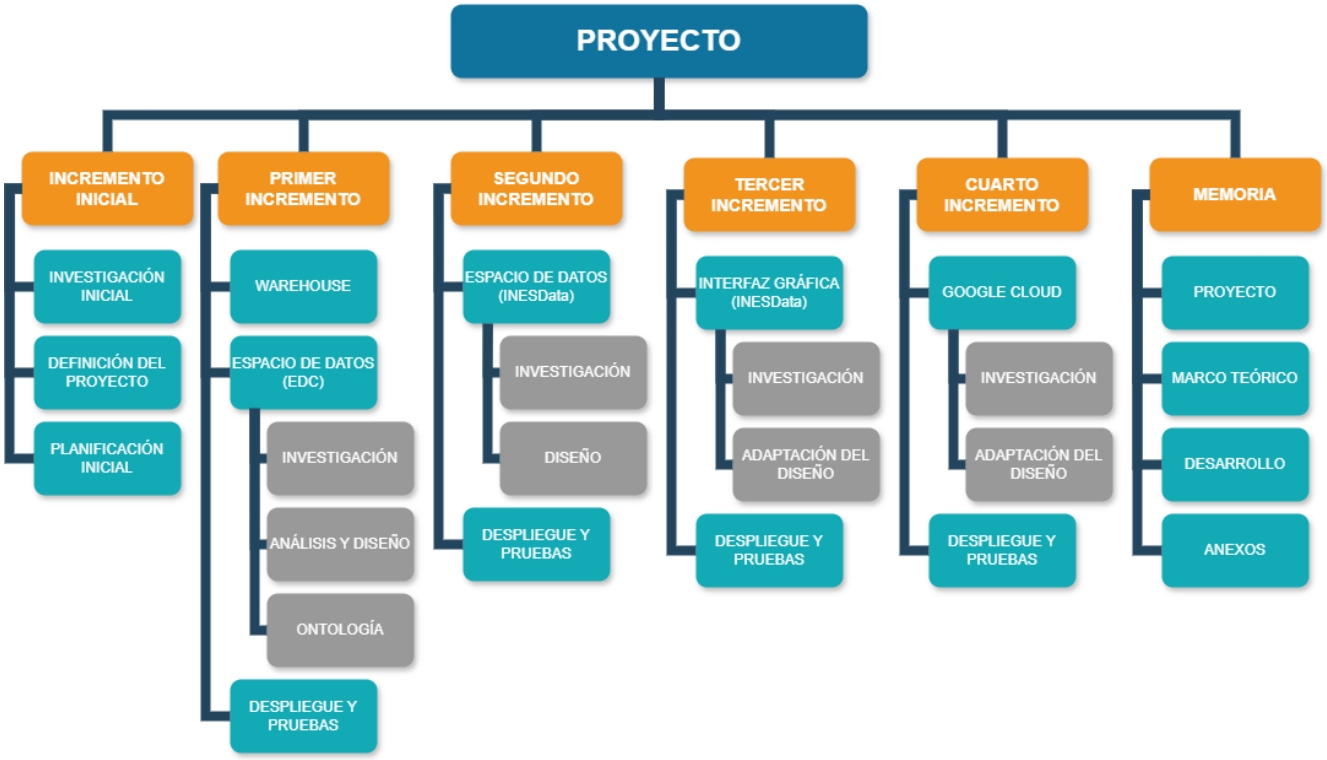


Figura 2.2: Estructura de División del Trabajo (EDT)

2.4. Gestión del tiempo

En esta sección calculamos el **calendario del proyecto**. Para ello, en primer lugar necesitaremos una **estimación del tiempo** que se tardará en realizar cada tarea, y en segundo lugar deberemos **asignar recursos** a cada una. En nuestro trabajo el único recurso que tenemos que administrar es el tiempo de trabajo del autor, ya que el resto están siempre disponibles. Las estimaciones para cada tarea se detallan en la Tabla 2.3.

Teniendo además en cuenta que la fecha en la que se inicia el proyecto es el día 20 de enero de 2025, podemos elaborar la Figura 2.3: Cronograma del Proyecto, que se ha hecho utilizando la versión gratuita de teamgantt.com [7]. En ella se incluyen las dependencias entre tareas, y se muestran cuáles se pueden realizar en paralelo respecto a otras. Del cronograma podemos obtener las siguientes fechas de finalización esperada de cada incremento:

Incremento inicial	3 de febrero
Primer incremento	4 de marzo
Segundo incremento	20 de marzo
Tercer incremento	1 de abril
Cuarto incremento	17 de abril
Incremento final	1 de mayo

Tabla 2.2: Fechas de finalización estimada para cada incremento

INCREMENTO	TAREA	ESTIMACIÓN (h)	TOTAL (h)
Inicial	Definición del proyecto	2	52
	Investigación inicial	15	
	Planificación inicial	15	
	Documentación	20	
Primero	Investigación	20	104
	Análisis y diseño	15	
	Diseño de la ontología	5	
	Creación y despliegue del E.D.	15	
	Pruebas	9	
	Documentación	40	
Segundo	Investigación	15	36
	Diseño	2	
	Creación y despliegue del E.D.	10	
	Pruebas	2	
	Documentación	7	
Tercero	Investigación	8	22
	Adaptación del diseño	1	
	Despliegue del E.D.	3	
	Pruebas	5	
	Documentación	5	
Cuarto	Investigación	10	36
	Adaptación del diseño	1	
	Despliegue en la nube	10	
	Pruebas	10	
	Documentación	5	
Final y otras tareas	Documentación	40	50
	Reuniones de seguimiento	10	
SUMA		**300**	**300**

Tabla 2.3: Estimación de la duración de cada tarea

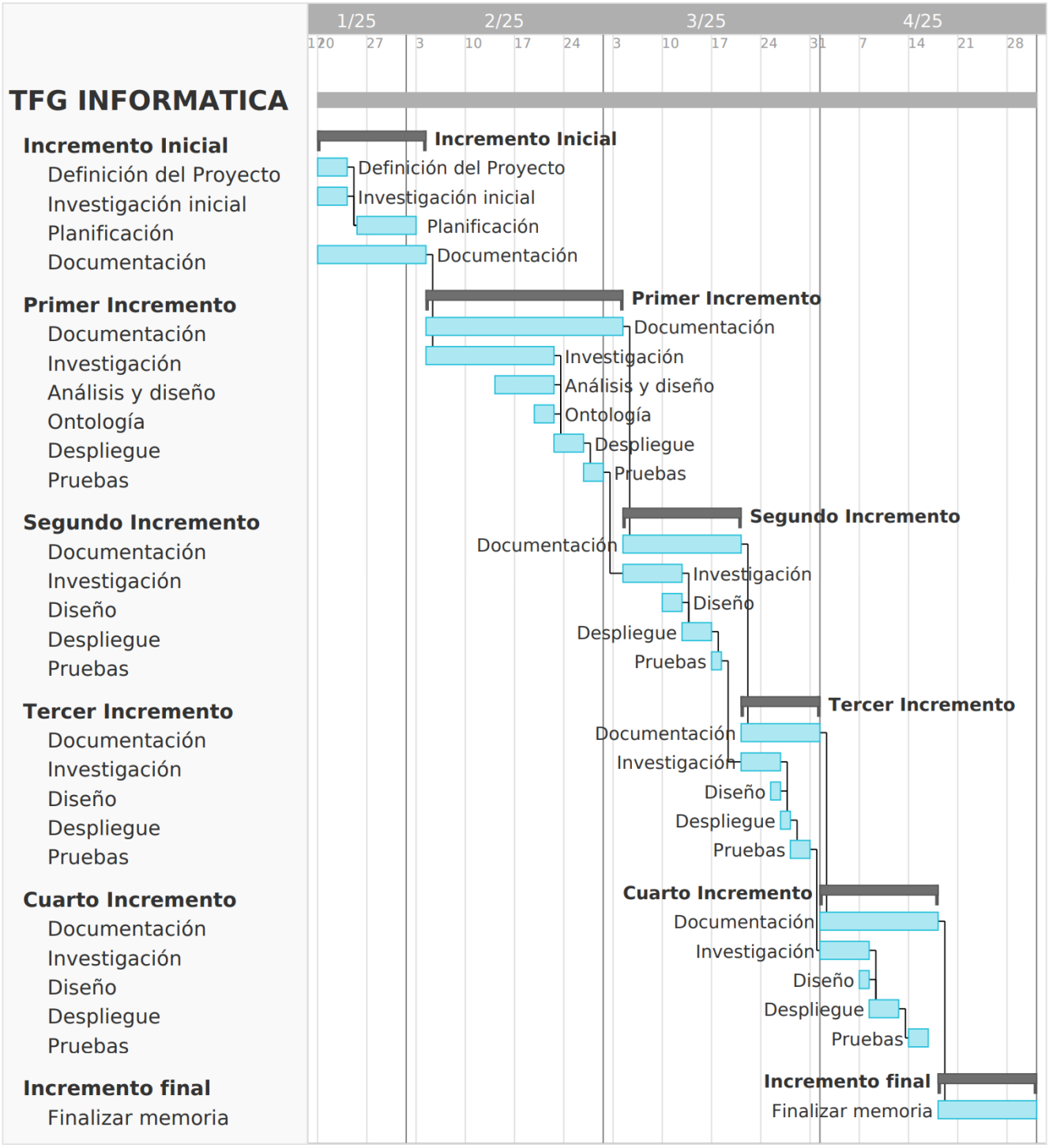


Figura 2.3: Cronograma del Proyecto

2.5. Gestión de los riesgos

Para hacer el plan de riesgos seguiremos la guía de Becker [8]. En la Figura 2.4 podemos observar cada uno de los pasos de este proceso.

Figura 2.4: PMBOK® Proceso de Gestión de Riesgos [8]

En el primer paso identificamos **dos áreas de posible impacto** de los riesgos: **calendario y producto**. Los posibles impactos en el calendario del proyecto serán retrasos/adelantos, mientras que en el producto será la disminución de sus prestaciones. Clasificaremos el impacto en el calendario como bajo, medio o alto si el retraso del proyecto es menor a un día, menor a una semana o superior a una semana, respectivamente. La clasificación del impacto en el área de producto será siempre alta. Además, clasificaremos la probabilidad de ocurrencia en 4 categorías: muy baja, baja, media y alta, si las probabilidades son <0,01, <0,10, <0,50 y >0,50, respectivamente. En la Figura 2.5 representamos la relación entre el impacto y la probabilidad de suceso de los riesgos.

ÁREA DE IMPACTO / PROBABILIDAD

IMPACTO	Calendario	Producto	Muy baja (< 0,01)	Baja (< 0,10)	Media (< 0,50)	Alta (> 0,50)
Bajo	< 1 día	-	XII	XI	X	VIII
Medio	< 1 semana	-	IX	VII	VI	IV
Alto	> 1 semana	siempre	V	III	II	I

Figura 2.5: Matriz de índices de prioridad de riesgos y tipo de impacto por área

En el segundo paso **identificamos los riesgos** del proyecto. En la Tabla 2.4 describimos brevemente cada riesgo que hemos identificado. En el tercer paso **clasificamos los riesgos**, localizando su posición en la matriz de índices de prioridad de riesgos. Esto lo hemos hecho en la Tabla 2.5, donde podemos observar esta clasificación con los riesgos ordenados por prioridad. Por último, debemos elaborar un plan de respuesta para cada riesgo, que estará basado en dos estrategias: mitigación y contingencia. Este plan se puede ver en la Tabla 2.6.

Riesgo	**Causas o descripción**
Error en la planificación	Estimación de horas incorrecta, objetivos poco claros o riesgos imprevistos
Tecnología insuficientemente madura	Falta de documentación, funcionalidades no completas o presencia de errores
Disponibilidad inferior a la estimada	Enfermedad, necesidades académicas
Desconocimiento de la tecnología	Aprender a usar de forma básica todas las tecnologías necesarias lleva más tiempo del esperado
Modificación del modelo de datos del warehouse	El esquema de datos del warehouse cambia
Créditos de Google Cloud no disponibles	Caducidad, cantidad insuficiente
Modificación de requisitos	Los requisitos del proveedor cambian en una fase tardía del desarrollo
Pérdida de información	El alumno pierde los archivos donde almacena el trabajo

Tabla 2.4: Identificación de riesgos

Riesgo	**Probabilidad**	**Impacto en**		**Índice de prioridad**
		Calendario	**Producto**	
Error en la planificación	Media	Alto	-	II
Tecnología insuficientemente madura	Media	Alto	Alto	II
Disponibilidad inferior a la estimada	Baja	Alto	-	III
Desconocimiento de la tecnología	Alta	Medio	-	IV
Modificación del modelo de datos del warehouse	Muy baja	Alto	-	V
Créditos de Google Cloud no disponibles	Muy baja	Bajo	Alto	V
Modificación de requisitos	Muy baja	Alto	-	V
Pérdida de información	Muy baja	Alto	-	V

Tabla 2.5: Clasificación de riesgos

Riesgo	**Plan de mitigación**	**Plan de contingencia**
Error en la planificación	Revisar la planificación con los tutores, planificar un margen de tiempo para imprevistos	Aumentar la carga de trabajo, retrasar la entrega del proyecto
Tecnología insuficientemente madura	La planificación en incrementos ayuda a no depender en exceso de una sola tecnología	Saltar un incremento, comunicación con los autores de la tecnología
Disponibilidad inferior a la estimada	Llevar una vida saludable, planificar un margen de tiempo para imprevistos, priorizar el trabajo	Aumentar la carga de trabajo tras la recuperación, retrasar la entrega del proyecto
Desconocimiento de la tecnología	Planificar un margen de tiempo para imprevistos	Buscar tecnologías alternativas más simples
Modificación del modelo de datos del warehouse	Comunicación con los tutores para estar prevenido	Aumentar la carga de trabajo, retrasar la entrega del proyecto
Créditos de Google Cloud no disponibles	Consumir los mínimos recursos necesarios, buena planificación	Solicitar unos nuevos créditos, saltar el cuarto incremento
Modificación de requisitos	Realizar un buen análisis	Aumentar la carga de trabajo, retrasar la entrega del proyecto
Pérdida de información	Utilizar un sistema de control de versiones que almacene los datos en la nube, tener copias de seguridad	Aumentar la carga de trabajo, retrasar la entrega del proyecto

Tabla 2.6: Plan de respuesta para cada riesgo

2.6. Gestión de los costes

En esta sección nuestro objetivo será contabilizar y cuantificar todos los costes asociados al proyecto. De esta manera, tendremos a nuestra disposición una estimación del coste agregado del proyecto. Este cálculo se proporciona al final de la sección, en la Tabla 2.8.

Los principales costes que calcularemos no son reales, pero sí que tratan de representar el verdadero coste del proyecto en un entorno profesional real. Siguiendo de esta idea, al trabajo del alumno y de los tutores, se les asociará al coste de oportunidad de estar empleados por cuenta ajena, y al lugar de trabajo del alumno, se le asociará un coste de sustitución de alquilar un espacio de *coworking*.

Mano de obra Para estimar el coste de la mano de obra utilizaremos esta lista de hipótesis. Hemos obtenido los datos de salarios de la web glassdoor.es [9, 10].

- Las horas de trabajo del alumno serán 300.
- Las horas de trabajo de los tutores serán 30.
- El empleo del alumno sería el de Ingeniero de Software Junior, que es remunerado de forma promedio en España con 25.150€ a jornada completa [9].
- El empleo de los tutores sería el de *Tech Lead*, que es remunerado de forma promedio en España con 59.000€ a jornada completa [10].
- El número de horas laborables al año a jornada completa en España cambia según el convenio colectivo, pero es del entorno de 1.800 horas [11].

Por tanto, el salario bruto equivalente a las 300h de trabajo del alumno sería de 4.191,67€, a los cuales habría que sumar un 30,48 % de cotizaciones a la Seguridad Social a cargo del empleador [12]. Llegamos por tanto a un coste de oportunidad total de 5.469,29€. De igual manera para los tutores: el salario bruto equivalente a 30h de trabajo sería de 983,33€, y el coste total de 1.283,05€.

CONCEPTO	ALUMNO	TUTORES	SUMA
Coste total	**5.469,29 €**	**1.283,05 €**	**6.752,34 €**
- SS Empresa	1.277,62 €	299,72 €	1.577,34 €
Salario bruto	**4.191,67 €**	**983,33 €**	**5.175,00 €**
- SS Trabajador	271,20 €	63,62 €	334,82 €
- IRPF[4]	593,89 €	240,55 €	834,44 €
Salario neto	**3.326,58 €**	**679,16 €**	**4.005,74 €**

Tabla 2.7: Desglose del coste de la mano de obra

Lugar de trabajo Para estimar el coste de un espacio de trabajo utilizaremos como referencia el alquiler de un espacio de *coworking* en Valladolid. El espacio Leburó ofrece una disponibilidad de 4 horas al día por 90 € mensuales [13]. Suponiendo que el desarrollo del proyecto abarca cuatro meses naturales, el coste final del lugar de trabajo sería de **360€**.

Materiales El coste por materiales es el de los ordenadores portátiles que se utilizarán en el proyecto. Su uso no supone ningún desembolso real pero sí que hará que su valor disminuya, por tanto el coste que calcularemos es un coste por depreciación de valor. Los modelos de ordenador que se

[4]El cálculo del IRPF se hace prorrateando el impuesto que se pagaría cobrando ese sueldo durante todo el mismo año fiscal, a las horas trabajadas.

utilizarán son: Acer Aspire 5 y Lenovo Ideapad 330, que tienen un coste de adquisición de 599€ [14] y 668,85€ [15] en la web pccomponentes.com, respectivamente. Si estimamos unas 5.000 horas de uso útil para cada uno, y el uso es de 255 y 45[5] horas respectivamente, la depreciación de los portátiles durante el proyecto será de 30,55€ y 6,02€, respectivamente. En total: **36,57€**.

Google Cloud Para estimar el coste de la infraestructura, debemos saber en primer lugar qué necesidades tendremos, algo que en este punto todavía desconocemos. Pero aunque la estimación no sea muy precisa, podemos intentar acercarnos lo máximo posible. Como referencia, calcularemos el coste de despliegue del espacio de datos mínimamente viable (MVD, por sus siglas en inglés) de Eclipse Dataspace Components (EDC) [16].

El MVD se puede desplegar en un clúster de kubernetes. La edición estándar de Google Kubernetes Engine nos da un precio de $0,10 por hora [17]. Para desplegar el *warehouse* necesitaremos también un instancia de mysql. La versión más básica que nos ofrece Google Cloud (Edición Enterprise, Zona de pruebas[6], región: europe-west1) tiene un precio de $0,14 por hora [18]. Si la infraestructura se tiene que mantener activa durante 4 días[7], el coste total sería de $23,04. Usando un tipo de cambio aproximado de 1,04 dólares por euro[8], el coste total sería de **22,15€**.

Para hacer el proyecto disponemos de créditos para estudiantes por un valor de $75, con una fecha de caducidad el 26 de septiembre de 2025. Esto significa que estaremos limitados a un uso máximo de la infraestructura de Google Cloud por 13 días.

Energía Si suponemos que hemos alquilado un espacio de *coworking*, el coste de la energía es de **0€**. Desde el lugar de trabajo tenemos acceso a electricidad sin coste adicional y además, está convenientemente calefactado.

Por otra parte, sí que podemos tratar de estimar el coste real en energía del proyecto, ya que en la realidad si que hemos tenido que enfrentar estos costes. Para ello supondremos que ambos portátiles tienen un consumo de 60W, y otros consumos como la iluminación serán de aproximadamente 10W. Como el precio futuro de la electricidad es desconocido, utilizaremos como referencia el precio promedio de la tarifa PVPC 2.0TD durante el mes de enero de 2025, que ha sido de 0,1584€ por kWh [20]. Este precio no incluye los impuestos [21], por lo que debemos sumarle un 5,113 % en concepto de Impuesto Especial a la Electricidad (IEE) [22], y después añadirle un 21 % adicional en concepto de Impuesto sobre el Valor Añadido (IVA) [23]. Esto nos deja con un precio final efectivo por kWh de 0,2015€. Para estimar el consumo en calefacción, supondremos que hemos utilizado un radiador eléctrico con una potencia de 1.000W durante 100 horas. Por tanto, el consumo total de energía será 121 kWh, con un coste estimado de **24,38€**.

Software Todas las herramientas que se utilizarán en este proyecto serán: (1) gratuitas, (2) tienen una versión de prueba gratuita o, (3) tienen licencias para estudiantes. Por tanto el alumno no deberá pagar por su uso.

Sin embargo, y siguiendo la misma idea que con los gastos de mano de obra y de lugar de trabajo, calcularemos el coste que habría tenido el uso de estas herramientas en un entorno no académico. Esto es, el coste de las licencias que hemos incluido en la categoría 3. El único programa dentro de esta categoría es *astah professional*, utilizado para el modelado del sistema y diagramas de diseño. Las licencias individuales tienen un precio de 8,99€ al mes. El coste durante todo el proyecto ascendería a **35,96€**.

[5] El segundo portátil se utiliza para las tareas de creación, despliegue y pruebas de los espacios de datos en los incrementos primero, segundo y tercero. El primero en todas las restantes.

[6] 2 CPUs virtuales, 8 GB de RAM y 10 GB de almacenamiento SSD.

[7] Hemos estimado la duración de las tareas de despliegue y pruebas en el cuarto incremento como 20 horas, que se tardarían 4 días en terminar con una dedicación de 5 horas al día.

[8] El tipo de cambio EUR/USD cerró el día 31 de enero a 1,0393 [19].

RECURSO	COSTES
Alumno	-
Tutores	-
Lugar de trabajo	-
Materiales	36,57 €
Google Cloud	-
Energía	24,38 €
Software	-
SUMA	**60,95 €**

(a) Costes para el alumno

RECURSO	COSTES	TIPO DE COSTE
Alumno	5.469,29 €	De oportunidad
Tutores	1.283,05 €	De oportunidad
Lugar de trabajo	360,00 €	De sustitución
Materiales	36,57 €	Por depreciación
Google Cloud	22,15 €	-
Energía	-	-
Software	35,96 €	-
SUMA	**7.207,02 €**	-

(b) Costes en un escenario ficticio

Tabla 2.8: Suma de todos los costes del proyecto

2.7. Seguimiento del proyecto

En esta sección realizamos un seguimiento del proyecto, evaluando en qué medida hemos cumplido con esta planificación y describiendo las desviaciones con respecto al alcance del proyecto, del cronograma o del trabajo que habíamos estimado.

Durante el primer incremento hemos realizado el único cambio técnico a la planificación, sin perjuicio para el alcance del proyecto. No desplegaremos nuestro propio *warehouse*, sino que utilizaremos la API de App-PIMD para acceder a los datos. Con este cambio simplificamos el trabajo a la vez que conseguimos una mayor calidad de los datos, ya que estarán actualizados. Otra ventaja de este cambio es que no estamos expuestos tan a modificaciones en el modelo de datos del *warehouse*, pasando a depender de la API que debería ser más estable. Del modelo de datos solo dependería la ontología.

2.7.1. Riesgos

Durante el proyecto han sucedido los 4 riesgos más probables que habíamos previsto. A continuación contamos los detalles y las consecuencias que han tenido en el proyecto.

- **Error en la planificación**: Hemos tenido que realizar varias tareas que no habíamos previsto:
  - Diseño de los datos (a parte del diseño del proveedor).
  - Diseño e implementación de una extensión al Data Plane para permitir el acceso al *warehouse*, más la investigación previa necesaria.
  - Investigar sobre ontologías (no se conocían lo suficiente).
- **Tecnología insuficientemente madura**: No se dispone de un manual para la interfaz gráfica del conector de INESData, por lo cual no se ha conseguido depurar los fallos que hemos tenido con este componente. Por este motivo no hemos conseguido una funcionalidad completa de la interfaz (más detalles en el Capítulo 6, en la Subsección 6.6.2: Pruebas de sistema).

- **Disponibilidad inferior a la estimada**: Durante el mes de marzo ha habido una menor intensidad de trabajo, del cual una semana corresponde a indisposición médica.

- **Desconocimiento de la tecnología**: Ha hecho falta más tiempo del que habíamos planificado para investigar las tecnologías que necesarias para el trabajo. Además hemos tenido problemas con terraform, ya que adaptar una configuración desde docker compose estaba llevando demasiado tiempo. Para no provocar un retraso adicional del trabajo, hemos mantenido la configuración de docker compose (más detalles en el Capítulo 6, en la Subsección 6.2.2: INESData Dataspace Local Enviroment).

Las tres tareas imprevistas se han realizado durante el primer incremento, que sumadas a la menor intensidad de trabajo durante el mes de marzo y al mayor tiempo necesario para investigar las tecnologías necesarias, han provocado su **retraso por mes y medio**. Esta gran desviación con respecto al plan original ha motivado la decisión de **eliminar el cuarto incremento**.

El cuarto incremento tenía una duración estimada aproximada de dos semanas. Gracias a este ahorro y a una mayor intensidad de trabajo posteriormente, hemos conseguido reducir el retraso total del trabajo a dos semanas en total.

2.7.2. Tiempo

En esta subsección comparamos las estimaciones de la Sección 2.4 con la realidad del proyecto. La Tabla 2.9 muestra estas comparaciones de forma resumida. En esta tabla se pueden ver estas comparaciones para cada incremento con respecto a su duración total y fechas de finalización.

En el resto de la sección, hacemos un seguimiento más detallado de cada incremento, revisando sucesos y tareas concretas. También incluimos la Figura 2.6, que muestra la evolución en el tiempo del número de palabras escritas del trabajo para cada capítulo. Esta imagen visualiza muy claramente el progreso del trabajo, aunque de forma parcial, ya que excluye el esfuerzo de investigación o de implementación. Aun así, ayuda a visualizar el tiempo invertido en cada capítulo, su extensión o el tamaño de cada incremento en relación a los demás.

INCREMENTO	ESTIMACIÓN (h)	REAL (h)	FIN ESTIMADO	FIN REAL
Incremento inicial	52	62	3 de febrero	7 de febrero
Primer incremento	104	182	4 de marzo	20 de abril
Segundo incremento	36	49	20 de marzo	1 de mayo
Tercer incremento	22	23	1 de abril	5 de mayo
Cuarto incremento	36	-	17 de abril	Suprmido
Incremento final	50	41	1 de mayo	15 de mayo
TOTAL	**300**	**357**	**1 de mayo**	**15 de mayo**

Tabla 2.9: Revisión general a las estimaciones de tiempo y fechas de finalización de cada incremento

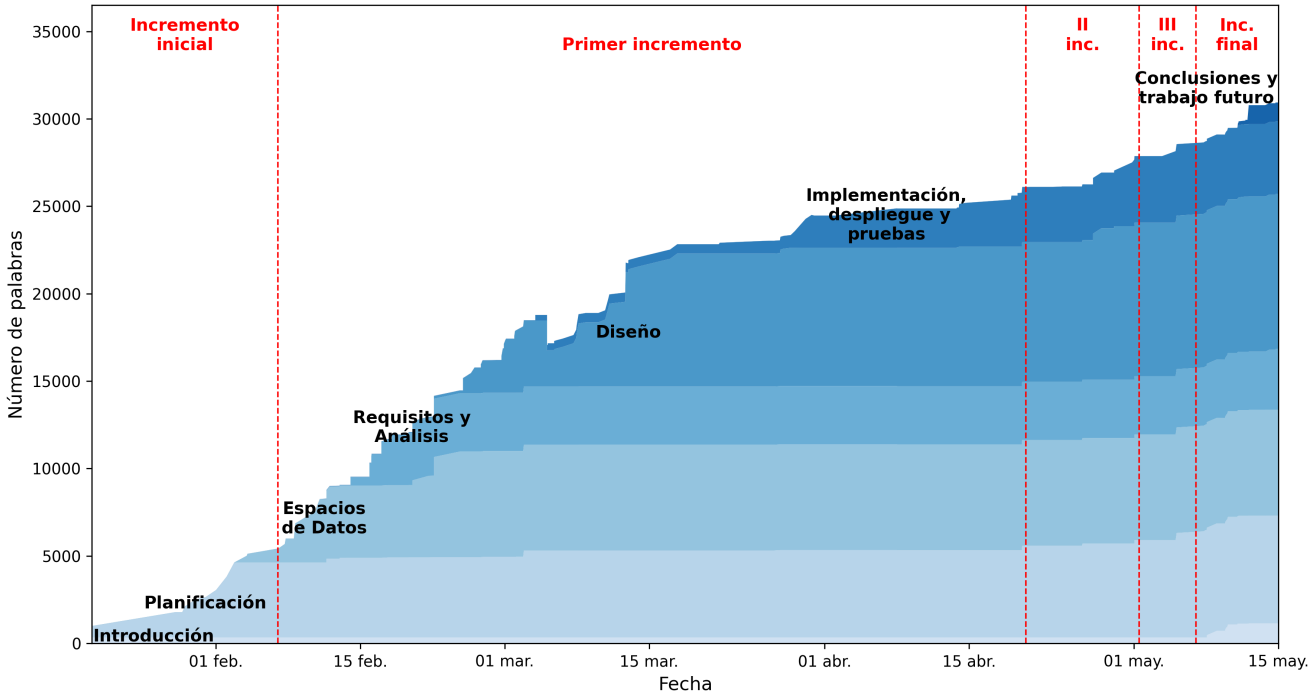


Figura 2.6: Evolución del número de palabras por capítulo

Incremento inicial La investigación de los espacios de datos, de las tecnologías disponibles para el proyecto, y de sus posibilidades han llevado algo más de tiempo de lo esperado. La Tabla 2.10 compara la duración estimada para cada tarea y su duración real.

TAREA	ESTIMACIÓN (h)	REAL (h)
Definición del proyecto	2	2
Investigación inicial	15	25
Planificación inicial	15	15
Documentación	20	20
TOTAL	**52**	**62**

Tabla 2.10: Revisión de las estimaciones de tiempo del incremento inicial

Primer incremento Casi todas las tareas de este incremento han llevado más tiempo del esperado inicialmente, en especial la investigación y el diseño. En la planificación no se había tenido en cuenta suficientemente el diseño de los datos (a parte del proveedor), o la gran cantidad de tecnologías que han sido necesarias y con las cuales no se tenía ninguna experiencia previa.

Las tareas de documentación también han llevado más tiempo, ya que durante este incremento se ha redactado la mayor parte de la memoria (esto se puede apreciar de forma muy visual en la Figura 2.6).

Las diferencias entre el tiempo estimado para cada tarea y el tiempo real se pueden ver en la Tabla 2.11. Desglosamos el tiempo que ha llevado cada capítulo, que se han ido haciendo de forma aproximadamente secuencial. Además, desagregamos las tareas de investigación y redacción, permitiendo así saber con más detalle el tiempo invertido en cada capítulo.

Otros motivos que han provocado el retraso de este incremento, a parte del mayor tiempo requerido, han sido una menor intensidad de trabajo durante el mes de marzo, incluyendo una semana por indisposición médica.

TAREA	ESTIMACIÓN (h)	REAL (h)
Investigación	20	51
Análisis y diseño	15	29
Diseño de la ontología	5	8
Creación y despliegue del E.D.	15	15
Pruebas	9	15
Documentación	40	64
TOTAL	**104**	**182**

(a) Desglose por tarea

CAPÍTULO	ESTIMACIÓN (h)	REAL (h)		
		Tarea	Investigación	Documentación
Espacios de Datos	-	-	22	20
Requisitos y análisis	10	14	3	6
Diseño	10	23	21	27
Implementación y pruebas	24	30	5	11
TOTAL	**44**	**67**	**51**	**64**

(b) Desglose por capítulo

Tabla 2.11: Revisión de las estimaciones de tiempo del primer incremento

Segundo incremento La exhaustiva preparación del incremento anterior y su gran parecido con este, han hecho que la investigación haya llevado bastante menos tiempo de lo planificado. Sin embargo, este parecido no se ha mantenido para la implementación.

Hemos tenido más dificultades de las esperadas para poder desplegar el espacio de datos con los componentes de INESData e integrarlo en el resto del repositorio. Una de estas dificultades es la ya mencionada con terraform en la revisión de los riesgos (Subsección 2.7.1).

El resto de tareas han ido en línea de lo planificado. La Tabla 2.12 cuantifica las diferencias de duración para cada tarea.

TAREA	ESTIMACIÓN (h)	REAL (h)
Investigación	15	6
Diseño	2	1
Creación y despliegue del E.D.	10	30
Pruebas	2	1
Documentación	7	11
TOTAL	**36**	**49**

Tabla 2.12: Revisión de las estimaciones de tiempo del segundo incremento

Tercer incremento Debido a la falta de documentación para la interfaz del conector, la investigación ha llevado menos tiempo y el despliegue más tiempo de lo planificado. Esta cuestión ya la hemos mencionado en la revisión de los riesgos (Subsección 2.7.1), y sobre la que damos más información en la Subsección 6.6.2: Pruebas de sistema.

TAREA	ESTIMACIÓN (h)	REAL (h)
Investigación	8	4
Diseño	1	0
Creación y despliegue del E.D.	3	13
Pruebas	5	3
Documentación	5	3
TOTAL	**22**	**23**

Tabla 2.13: Revisión de las estimaciones de tiempo del tercer incremento

Cuarto incremento Suprimido.

Incremento final Las horas dedicadas a la documentación (revisión general, introducción, conclusiones, abstract, anexo) han sido 31, frente a las 40 planificadas. Las reuniones de seguimiento con los tutores han tomado aproximadamente 10h, las planificadas inicialmente.

Capítulo 3

Espacios de Datos

Los espacios de datos son una idea que se empezó a desarrollar hace más de 20 años, pero no ha sido hasta ahora que ha empezado a ganar protagonismo. Con el impulso que se le está dando desde las instituciones europeas, se espera que en los próximos años sea una tecnología que gane mucha adopción y se convierta en un estándar para el intercambio de datos a nivel internacional.

En este capítulo explicamos en qué consiste esta tecnología, cómo funciona, cuáles son sus ventajas y principios fundamentales y hacemos una recopilación de algunas de las iniciativas más importantes a nivel internacional y a nivel europeo en particular.

Las dos **iniciativas de estandarización** en las que nos centraremos son la International Data Spaces Association (IDSA, Sección 3.2) y Gaia-X (Sección 3.3), que son reconocidas a nivel internacional como las más relevantes. Además, entraremos en detalle en dos de los artefactos técnicos que elabora la IDSA como son el IDS RAM (Subsección 3.2.1) y el *Dataspace Protocol* (Subsección 3.2.2). Utilizaremos estos dos documentos en el Capítulo 4: Requisitos y análisis para elaborar las especificaciones técnicas de nuestro proveedor.

Ninguna de estas dos organizaciones proporciona infraestructura o código directamente, sino que son otras organizaciones las que implementan componentes a la medida de sus necesidades y casos de uso. Estos proyectos se deben adherir a algún estándar como los de IDSA o Gaia-X para ganar la confianza de la comunidad. Veremos también el contexto de las dos **iniciativas de implementación** (Sección 3.4) que usaremos en este trabajo: Eclipse Dataspace Components (EDC, Subsección 3.4.1) e INESData (Subsección 3.4.2).

Contaremos brevemente además, algunas **iniciativas institucionales** (Sección 3.5) de apoyo a esta tecnología, en concreto las impulsadas por la Unión Europea (Subsección 3.5.1) y por España (Subsección 3.5.2).

Por último, veremos como la aplicación de esta tecnología en el sector de la movilidad (Sección 3.6: Ejemplo: movilidad) puede traer beneficios tan evidentes como una mejora significativa en la seguridad vial o una menor congestión del tráfico, aplicaciones que no serían posibles sin utilizar espacios de datos.

3.1. ¿Qué son los espacios de datos?

Los espacios de datos son el siguiente paso en la evolución de los **sistemas de gestión de datos** y surgen para solucionar algunos de los problemas que tiene la **integración de datos a gran escala**. Los espacios de datos no aspiran a obtener una integración total en esquemas, sino a ofrecer, en lo posible, una **integración semántica** [24, 25, 26].

Los espacios de datos **no almacenan los datos**, y se limitan a ofrecer una infraestructura que permite la interacción eficiente y segura entre diversas fuentes. Esta conexión facilita la **interoperabilidad**, permitiendo el intercambio de datos sin necesidad de centralizarlos en un único repositorio [2]. De esta forma, cada fuente de datos conserva su autonomía, mientras que los participantes pueden acceder y utilizar la información según las **políticas de acceso y uso** establecidas, siempre dentro de un marco que garantiza la **privacidad, seguridad y soberanía de los datos**.

Para lograr una mayor **interoperabilidad entre las fuentes de datos**, es imprescindible el uso de herramientas como **vocabularios**[1]. No obstante, los espacios de datos permiten la coexistencia entre las fuentes, sin importar cómo de integradas estén, y proporcionan funcionalidades básicas para todas ellas [25, 26]. Una virtud muy importante de los espacios de datos es que **reducen de manera muy significativa los costes iniciales de integración** entre una gran cantidad de fuentes, esta relación se puede observar en la Figura 3.1. Los espacios de datos pueden utilizar técnicas ya conocidas como el emparejamiento automático de esquemas para realizar consultas. De esta manera, se puede **posponer o incluso prescindir de la integración total de esquemas**, salvo que sea estrictamente necesaria [25].

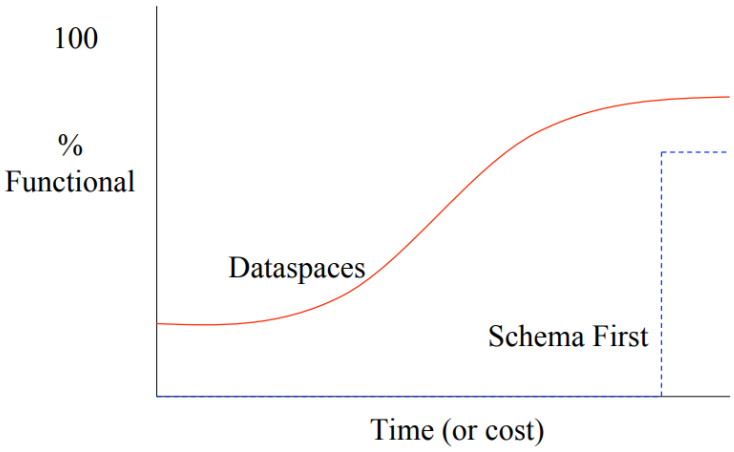


Figura 3.1: Utilidad vs coste: Comparación entre los espacios de datos y la integración tradicional [24]

Es importante entender que un espacios de datos no existe como una entidad propia. Un espacio de datos es el **conjunto de participantes** que lo componen y, en su nivel más básico, es simplemente el contexto entre dos participantes [28]. Es decir, los espacios de datos tienen un **diseño descentralizado**.

Los espacios de datos se tienden a organizar en torno a un **sector temático**. Algunos de los sectores de las iniciativas mas importantes a nivel internacional son: agricultura, energía, industria, salud o movilidad [29, 30]. Pero esta división se da solo en el plano de las **políticas y normas de gobernanza** de cada espacio de datos, ya que una organización puede utilizar una misma infraestructura para colaborar en varios espacios de datos.

[1] En integración de datos, un vocabulario es un conjunto de términos estandarizados que define y unifica el significado de los datos entre diferentes fuentes, facilitando su interoperabilidad y el intercambio de información [27].

3.1.1. Soberanía y confianza

La **soberanía y confianza** son considerados dos valores imprescindibles en un espacio de datos, de hecho, el Data Spaces Support Centre (DSSC), en la versión 1.5 de su documento *Plano para Espacios de Datos* [31], identifica 17 bloques básicos en el diseño de un espacio de datos. Estos bloques se pueden dividir en 6 pilares, y se pueden observar en la Figura 3.2. Uno de ellos es precisamente la **soberanía y confianza**. También son identificados como valores clave en [26, 32]. En esta subsección desarrollaremos estos dos conceptos.

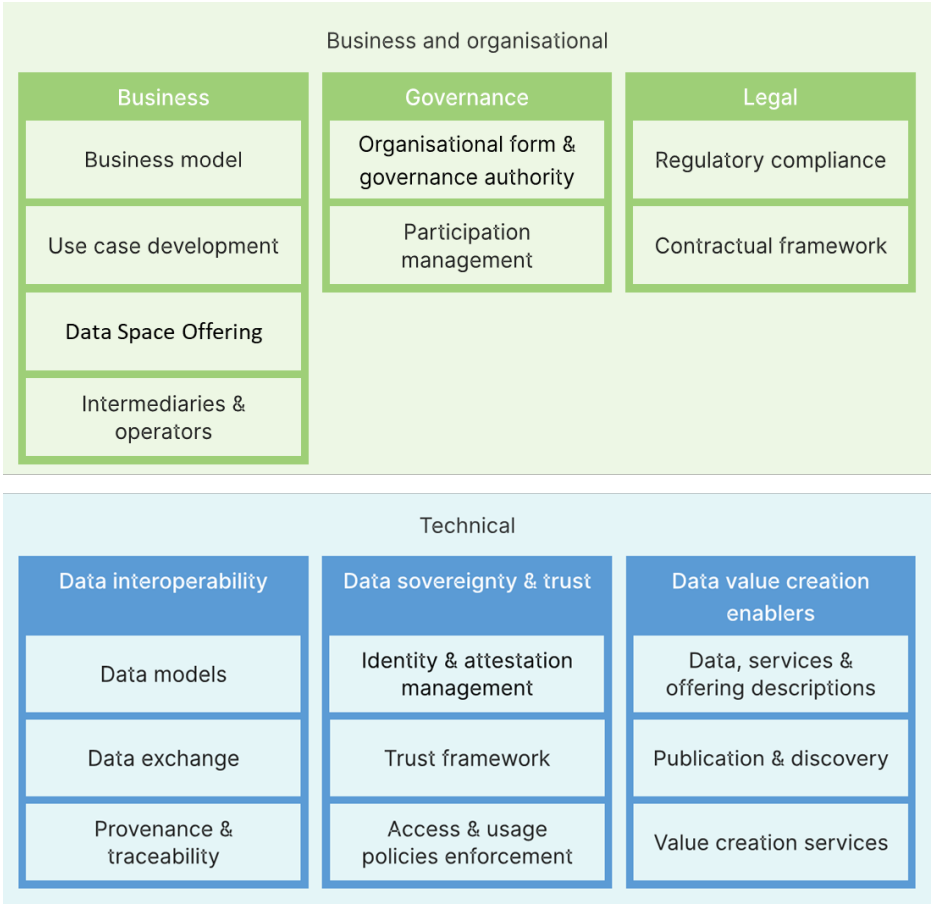


Figura 3.2: Bloques básicos de un Espacio de Datos [31]

- La **soberanía sobre los datos** se refiere a la capacidad de los proveedores de datos a decidir sobre el cómo y cuándo se pueden utilizar sus datos a lo largo de la cadena de valor [33]. El diseño descentralizado permite que los datos puedan permanecer en las organizaciones donde se producen. De esta manera, los proveedores pueden establecer las **normas de uso y acceso** que más convengan a su organización.

- La **confianza en el espacio de datos** se refiere a crear un entorno seguro donde los participantes puedan verificar sus identidades, y tengan la certeza de que las normas de uso y acceso que establecen para sus datos serán cumplidas.

Para proteger estos dos valores, el DSSC identifica tres bloques básicos de actuación, sus responsabilidades son las siguientes:

- **Certificación y Gestión de Identidad**: Abarca la necesidad de identificar a los participantes del espacio de datos. Para recibir un **certificado de pertenencia**, se seguirá un **proceso de verificación** en el que se comprobarán tanto la identidad del participante como el cumplimiento de los requisitos específicos que tenga cada espacio de datos.

- **Marco de Confianza**: Define los medios técnicos necesarios para verificar que los participantes cumplen con las reglas del espacio de datos. Es clave para proteger la soberanía de los datos, y para promover medidas de seguridad robustas.
- **Imposición de las Políticas de Uso y Acceso**: En este bloque se gestiona el control de acceso y uso de los datos, garantizando que el propietario mantenga el control sobre los mismos y que solo se utilicen para las acciones permitidas. Para ello, se establecen dos fases previas al acceso: la negociación y la aplicación. Los participantes deben firmar un contrato donde se acepten las condiciones, y posteriormente se comprueba si el contrato es válido.

3.1.2. Comparación con SGBD relacionales

Curry ofrece una comparación entre el paradigma del espacio de datos y los sistemas de gestión de bases de datos relacionales [25]. En un espacio de datos, las fuentes coexisten y evolucionan juntas a lo largo del tiempo, sin depender de un sistema rígido de gestión de datos, lo que las diferencia notablemente del enfoque tradicional basado en bases de datos relacionales. Esta comparación se muestra en la Tabla 3.1.

	SGBD	**Espacio de Datos**
Modelo	Relacional	Todos
Formatos	Homogéneo	Heterogéneo
Esquema	Esquema primero, datos después	Datos primero, esquema después o nunca
Control	Completo	Parcial
Liderazgo	De arriba a abajo	De arriba a abajo o de abajo a arriba
Consultas	Exactas	Aproximadas
Integración	Por adelantado	Incremental
Arquitectura	Centralizada	Descentralizada
Procesamiento en tiempo real	No	Aplicable

Tabla 3.1: Comparación entre SGBD relacionales y espacios de datos [25].

3.2. International Data Spaces Association (IDSA)

La asociación internacional de espacios de datos, es una asociación sin ánimo de lucro creada en 2017. Su objetivo es crear un **estándar global para los espacios de datos**, con un interés especial en garantizar la interoperabilidad entre diferentes plataformas y regiones, fomentar la colaboración entre sectores públicos y privados, y promover la confianza y la transparencia en el uso de los datos, todo mientras se asegura la soberanía y el control de los mismos por parte de los usuarios y las organizaciones participantes [34, 35].

Como parte de su misión, la IDSA desarrolla y mantiene: un **modelo arquitectónico de referencia** para espacios de datos, un **libro de normas**, y el **protocolo de comunicación** más extendido para espacios de datos. Además, la IDSA tiene un **programa de certificación** para garantizar que el software que consiga estos certificados cumple con ciertos estándares de seguridad, interoperabilidad y privacidad. Estas iniciativas se representan en la Figura 3.3.

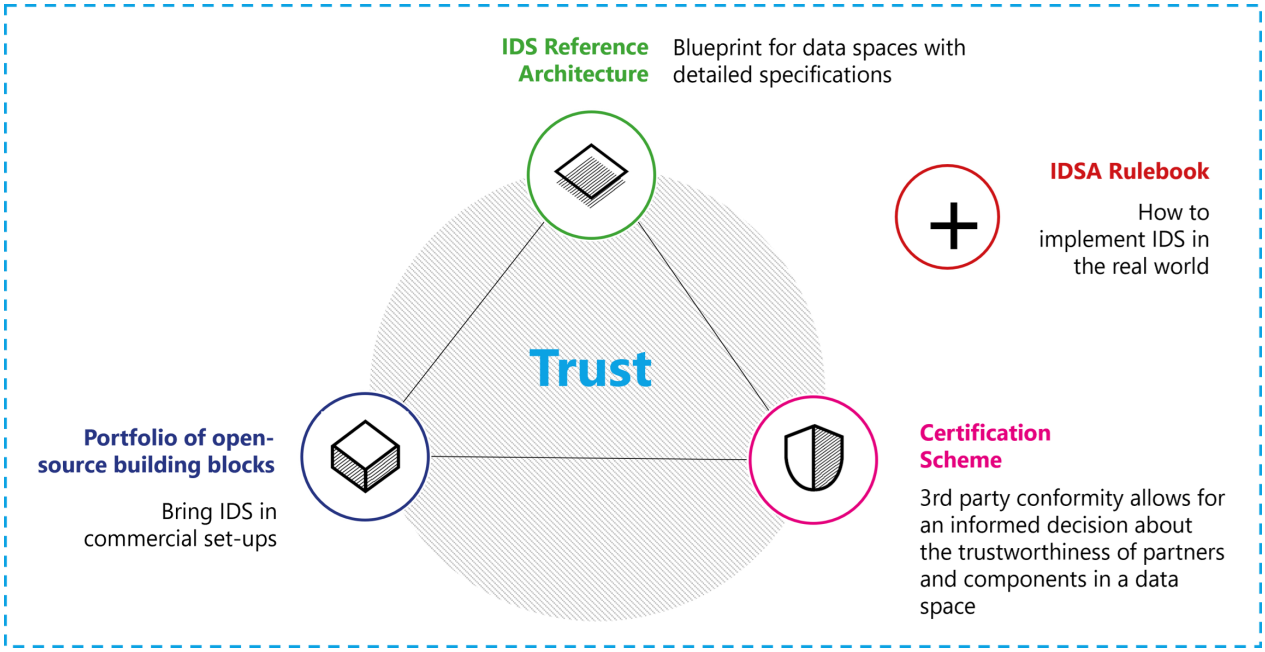


Figura 3.3: IDSA Magic Triangle [36]

3.2.1. IDS Reference Architecture Model

El Modelo Arquitectónico de Referencia para los Espacios de Datos Internacionales (RAM), es un documento de especificaciones arquitectónicas que ofrece una visión general de los **procesos, funcionalidades y componentes** necesarios para crear un espacio de datos funcional. En esta subsección, haremos una breve descripción de este documento, basándonos en su cuarta versión, la cual fue publicada en 2023 y es la versión más reciente disponible [36].

Comprender este modelo, aunque sea de forma superficial, nos servirá para entender el diseño de cualquier espacio de datos, incluso aunque no siga estas especificaciones. Muchos de los conceptos definidos en este documento son estándares que también se utilizan en otras implementaciones.

El RAM sigue los estándares comunes de arquitectura de sistemas, como ISO 42010, y utiliza una **estructura de cinco capas**, junto con **tres perspectivas transversales**, para abordar las preocupaciones de los diferentes interesados a diversos niveles de detalle, como se muestra en la Figura 3.4. A continuación, se detalla cada una de las capas que conforman este modelo, destacando cómo cada una contribuye a la estructura general del RAM.

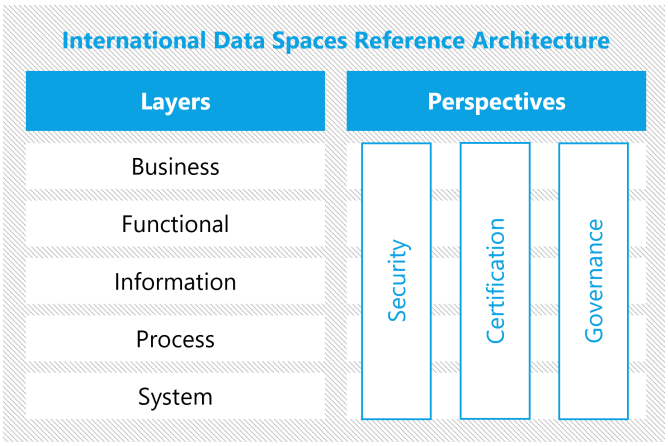


Figura 3.4: Estructura general del IDS Reference Architecture Model [36]

Capas

1. **Capa de negocio**: Define los roles de los participantes y sus actividades e interacciones, que se muestran en la Figura 3.5. Se distinguen cuatro categorías de ***roles de negocio***, que asumirán uno o más ***roles básicos*** dependiendo de los requisitos del negocio. Estos ***roles básicos*** son: Conector, Datos, Vocabulario, Identidad, Aplicación, Transacción y Servicio. Las categorías de ***roles de negocio*** son las siguientes:

   - **Participante principal**: proveedor de datos, consumidor de datos
   - **Intermediario**: intermediario de datos, intermediario de servicios, tienda de aplicaciones, intermediario de vocabularios, cámara de compensación, autoridad de identidad
   - **Desarrollador de software**: desarrollador de aplicaciones, desarrollador de conectores
   - **Cuerpo de gobernanza**: cuerpo de certificación, instalaciones de evaluación, organizaciones de estandarización, asociación internacional de espacios de datos

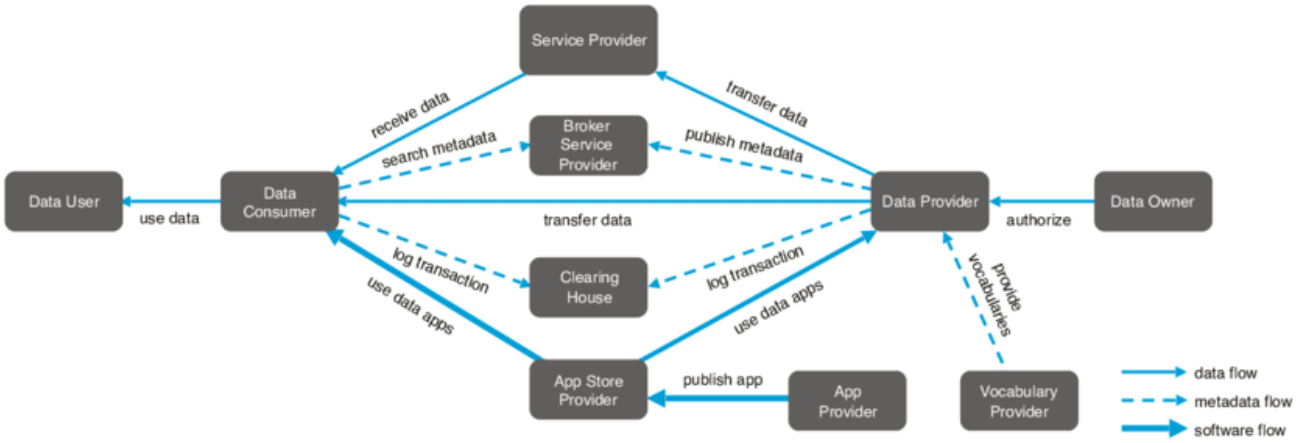


Figura 3.5: Roles y sus interacciones en un IDS [36]

2. **Capa funcional**: Establece los requisitos funcionales y las características derivadas que se deben implementar. Estos requisitos se agrupan en 6 categorías. Las áreas que cubre cada una se describen a continuación:

   - **Confianza**: roles, gestión de identidad, certificación de usuarios.
   - **Seguridad y soberanía**: autentificación y autorización, políticas de uso e imposición de uso, comunicación confiable y seguridad desde el diseño, certificación técnica.
   - **Ecosistema de datos**: descripción de las fuentes de datos, intermediación de metadatos y vocabularios.
   - **Interoperabilidad estandarizada**: operación, intercambio de datos.
   - **Aplicaciones que añaden valor**: procesamiento y transformación de datos, implementación de aplicaciones de datos, provisión de aplicaciones de datos, instalación y soporte de aplicaciones de datos.
   - **Mercados de datos**: liquidación y facturación, restricciones de uso y gobernanza, aspectos legales.

3. **Capa de información**: Define un modelo de información que describe a los recursos digitales en un espacio de datos, así como a sus elementos esenciales: participantes, componentes de infraestructura y procesos. El modelo se especifica en tres representaciones:

- **Conceptual**: Visión de alto nivel de los conceptos clave, sin estar vinculada a una tecnología o dominio específico. Está diseñada para una audiencia general, promoviendo un entendimiento compartido de los conceptos fundamentales.
- **Declarativa (Vocabulario IDS)**: Especificación formal del modelo, basada en estándares de la Web Semántica de W3C y vocabularios de modelado como DCAT, ODRL y SKOS. Ofrece una especificación a los conceptos de la representación conceptual, y define entidades del espacio de datos para facilitar el intercambio y búsqueda de metadatos estructurados.
- **Programática**: Dirigida a desarrolladores de software, esta representación adapta el Vocabulario IDS a las estructuras nativas de lenguajes de programación como Java, Python o C++. Permite por tanto a los desarrolladores el crear instancias del modelo de forma eficiente sin preocuparse por los detalles del procesamiento de ontologías.

4. **Capa de procesos**: Describe las interacciones entre los componentes de un espacio de datos, proporcionando una vista dinámica del RAM. Se definen los siguientes procesos:

   - La **incorporación** es el paso previo para acceder a los Espacios de Datos Internacionales, y requiere de dos pasos iniciales: el registro y la certificación de la organización, y la obtención de un conector IDS certificado.
   - El proceso de **ofrecimiento de datos** en los IDS consiste en poner a disposición de los consumidores los activos de datos a través de una auto-descripción que detalla la información relevante sobre los datos, como su acceso y políticas de uso. Esta descripción puede ser publicada en un intermediario de metadatos, lo que facilita la búsqueda y el acceso a los datos.
   - El proceso de **negociación de contratos** en los IDS comienza cuando el proveedor presenta una oferta de contrato, que el consumidor puede aceptar o modificar. Si ambas partes están de acuerdo, firman el contrato, que se almacena en sus conectores para su aplicación. En algunos casos, una cámara de compensación valida el contrato. Las negociaciones también pueden incluir contraofertas para ajustar las condiciones.
   - El **intercambio de datos** se divide en dos fases: la fase de control, donde se acuerdan las condiciones y se preparan los procesos necesarios, y la fase de transferencia, en la que se realiza el intercambio real de datos entre el proveedor y el consumidor.
   - El proceso de **publicación y uso de aplicaciones de datos** implica que el proveedor suba la app a una tienda IDS, a menudo con certificación. Los participantes pueden buscar y adquirir la aplicación mediante el conector IDS, para luego instalarla y usarla en tareas de procesamiento de datos. Todo esto se gestiona a través de la tienda IDS y sus interfaces.
   - El proceso de **aplicación de políticas** asegura que los datos se usen de acuerdo con las políticas acordadas, aplicando controles en tiempo real a través de una serie de pasos automatizados.

5. **Capa de sistema**: Describe una serie de componentes lógicos que se pueden considerar el núcleo técnico de los IDS. Utiliza los roles de la capa de negocios y los procesos de la capa de procesos, creando una arquitectura concreta de datos y servicios. En la Figura 3.6 se muestran los componentes de esta capa (a excepción del proveedor de identidad) y los procesos del RAM, como interacciones entre ellos. Los IDS tienen los siguientes componentes fundamentales:

   - **Proveedor de Identidad**: Gestiona la identificación, autentificación y autorización de los participantes para tomar decisiones de control de acceso confiables.
   - **Conector IDS**: Es un componente esencial en la red de IDS, facilitando el intercambio de datos mediante los puntos de acceso que expone. Debe ser accesible por conectores de otras organizaciones, lo que puede implicar la modificación de políticas de *firewall* o la creación de

zonas desmilitarizadas (DMZ). Un participante puede operar varios conectores para cumplir con requisitos de balanceo de carga, partición de datos, o criterios organizativos.

- **Tienda de Aplicaciones**: Es una plataforma segura para distribuir aplicaciones IDS (IDS Apps), que son activos de software independientes, funcionales, reutilizables y gestionables en un conector IDS.
- **Broker de Metadatos**: Es un conector IDS que gestiona la inscripción, publicación, mantenimiento y consulta de auto-descripciones de los conectores. Estas auto-descripciones contienen información sobre el conector, sus interfaces, capacidades y los metadatos de los datos ofrecidos.
- **Cámara de Compensación**: Es un conector IDS que se basa en un servicio de registro (*logging service*) para recopilar información relevante para la liquidación y facturación, así como para el control de uso.
- **Repositorio de Vocabularios**: Es una plataforma que gestiona y proporciona vocabularios estandarizados utilizados para describir datos, servicios, contratos y otros elementos dentro del espacio de datos. Su función principal es almacenar, mantener, publicar y documentar vocabularios que se usan en los casos de uso del IDS.

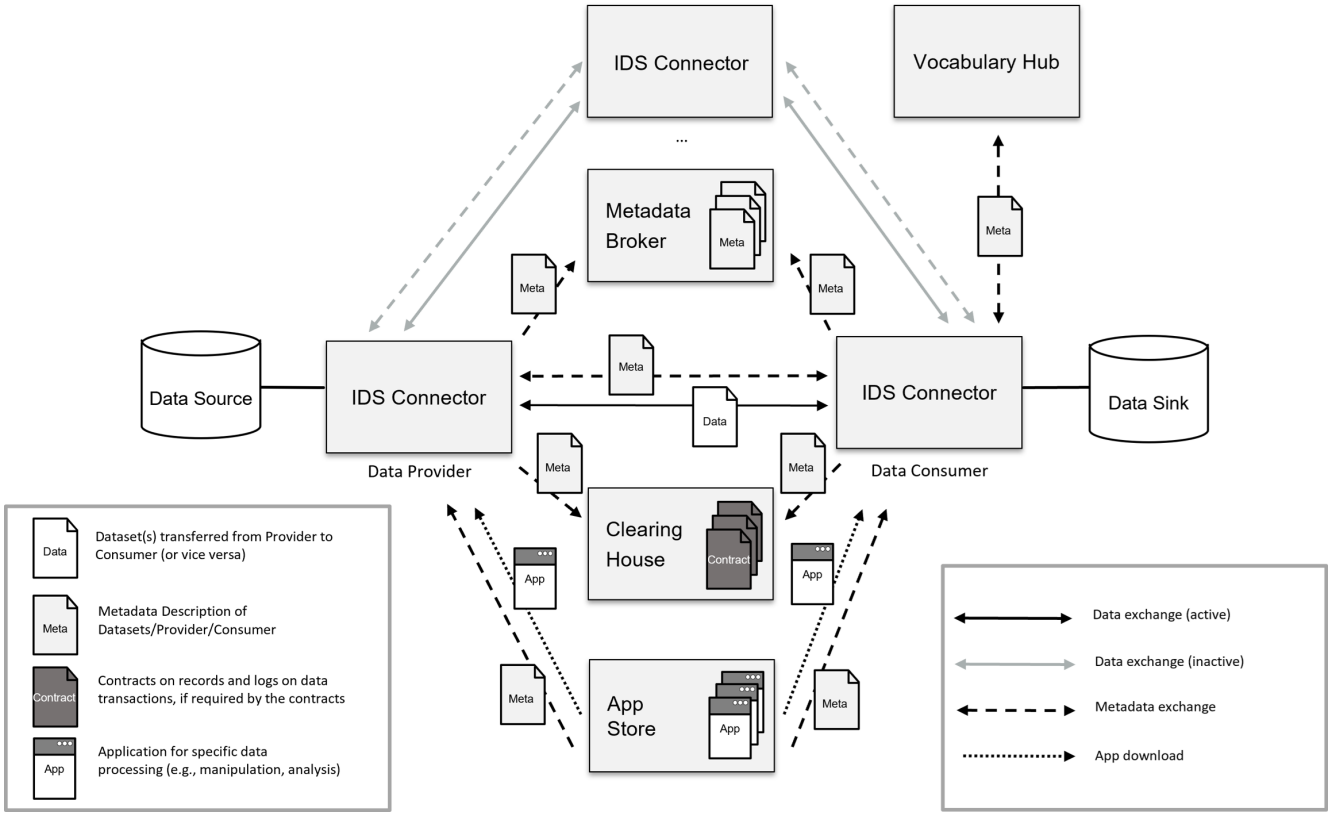


Figura 3.6: Procesos del RAM como interacción entre componentes de los IDS [36]

Perspectivas

1. **Perspectiva de seguridad**: Es imprescindible para asegurar la confianza en los espacios de datos y que la tecnología prospere. Provee los medios para **identificar dispositivos** en los IDS, **proteger la comunicación y las transacciones de datos**, y **controlar su uso** después de que hayan sido intercambiados. También define **requisitos de seguridad** para que el software pueda obtener un certificado para diferentes niveles de confianza. Estos son:

a) Integridad y autenticidad del *software stack*.
b) Verificación de integridad remota / certificación remota.
c) Protección de la integridad y confidencialidad de los datos en persistencia.
d) Aislamiento de procesos.
e) Registro de eventos.
f) Protección de claves utilizadas.
g) Protección contra administradores maliciosos.

2. **Perspectiva de certificación**: Deberán pasar por un **proceso de certificación** tanto las **organizaciones e individuos** que quieran participar en un IDS, como los **componentes** que se vayan a utilizar. La certificación de organizaciones e individuos se enfoca en la seguridad y la confianza, mientras que la certificación de componentes se enfoca además de a eso, al cumplimiento de los requisitos técnicos que aseguran la interoperabilidad. Esta perspectiva describe los procesos de certificación, enumerando los pasos necesarios, las partes implicadas y los diferentes niveles de certificados que se pueden obtener.

3. **Perspectiva de gobernanza de los datos**: Establece roles, funciones y procesos desde la perspectiva de gobernanza y *compliance* en los IDS. Si bien en los IDS se permite actuar conforme a reglas negociadas, no se imponen regulaciones fijas, ofreciendo un marco personalizable. La gobernanza es imprescindible para facilitar el intercambio seguro de datos, fomentar relaciones de confianza y la negociación de acuerdos y promover la transparencia.

   Frente a la creciente necesidad de gestionar datos que trascienden las fronteras de una sola organización, el IDS-RAM distribuye de manera federada los derechos de decisión, impulsando una gobernanza colaborativa esencial para el éxito en entornos digitales y de negocio innovadores.

3.2.2. Dataspace Protocol (DSP)

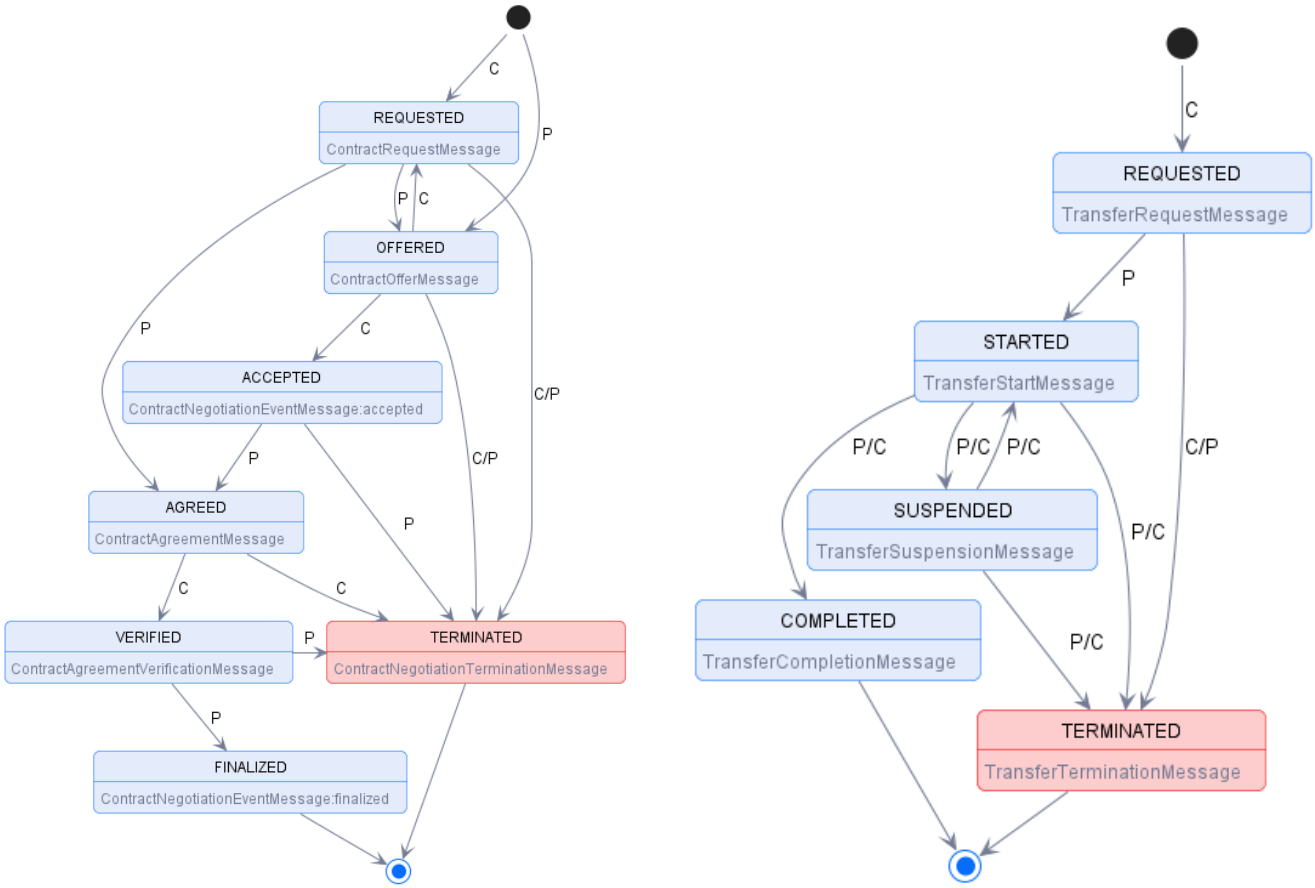
El protocolo de espacio de datos es un conjunto de especificaciones que regula la transferencia de **conjuntos de datos** entre entidades independientes. Estas especificaciones definen los esquemas y protocolos necesarios para que las entidades publiquen datos, negocien **acuerdos** y accedan a datos como parte de una federación de sistemas técnicos denominada **espacio de datos**. Utilizaremos como referencia la versión 2024-1, que es calificada como estable [37].

Este protocolo se puede dividir en tres más específicos: protocolo de **catálogos**, protocolo de **negociación de contratos** y protocolo de **proceso de transferencia**. Además, también incluye un **glosario de términos** donde se definen los conceptos clave de este protocolo y de los espacios de datos en general.

Para compartir conjuntos de datos es necesario administrar sus metadatos y sus políticas de acceso asociadas, los catálogos son la forma que utilizan los espacios de datos para permitir a los participantes encontrar los datos que necesitan. Un catálogo contiene metadatos sobre los conjuntos de datos, que proporcionan información imprescindible como su descripción, origen, formato, estructura, propiedades o políticas de acceso y uso. El **protocolo de catálogos** describe como se publica y accede a estos catálogos usando puntos de acceso HTTPS. Los catálogos se representan usando el Data Catalog Vocabulary (DCAT), y las políticas de acceso mediante el Open Digital Rights Language (ODLR).

El **protocolo de negociación de contratos** se describe como una máquina de estados (ver Figura 3.7a), donde el proceso de negociación va transicionando de un estado a otro, provocado por los mensajes que se mandan el proveedor y el consumidor (marcados con P y C en la figura). Cada proceso de negociación es identificado por un IRI. Los tipos de mensajes que se pueden enviar son: solicitud, oferta, acuerdo, verificación, evento de negociación y terminación.

El **protocolo de proceso de transferencia** controla el proceso de transferencia, ya que la transferencia de datos en sí es responsabilidad del protocolo de transporte. La transferencia de datos puede ser de dos tipos: *push* o *pull*, en la primera es el proveedor el que envía los datos a un punto de acceso del consumidor y en la segunda el consumidor solicita los datos a un punto de acceso del proveedor. Los estados que puede atravesar el proceso de transferencia se muestran en la Figura 3.7b.



(a) Proceso de negociación de un contrato

(b) Proceso de transferencia de datos

Figura 3.7: Máquinas de estados del Dataspace Protocol (DSP) [37]

3.2.3. IDSA Rulebook

El libro de reglas de IDSA está dirigido a los participantes de los espacios de datos, iniciativas de intercambio de datos, personas y empresas interesadas en los estándares de intercambio de datos y aquellos que quieren saber cómo configurar espacios de datos. En este documento se recogen todas las pautas necesarias para que cualquier organización pueda empezar a utilizar espacios de datos basados en IDS [38]. No todas estas pautas son obligatorias y abarcan los siguientes objetivos:

- Funcionalidad de los servicios comunes, así como su definición, procesos y servicios de roles específicos.
- Cómo implementar o usar un artefacto técnico de IDSA.
- Trabajo y la colaboración dentro de los servicios de datos.
- Base legal en cumplimiento con el entorno regulatorio para garantizar la confianza y la seguridad.

Es decir, el libro de reglas de IDSA es un marco de gobernanza que contiene todos los requisitos funcionales, técnicos, operacionales y legales para construir y operar espacios de datos basados en IDS.

3.3. Gaia-X Association

La asociación Gaia-X tiene sus orígenes en 2019, y entre sus miembros fundadores encontramos importantes empresas alemanas y francesas, centros de investigación, la IDSA o a los gobiernos de Alemania y Francia [39, 40]. Actualmente se organiza como una organización sin ánimo de lucro y tiene sede en Bélgica.

Gaia-X es otra de las más influyentes iniciativas en el ámbito de la **estandarización** de los espacios de datos [41], pero su objetivo va más allá que el de la IDSA. No se centra en los espacios de datos sino en otro concepto similar: **ecosistemas federados de datos**. En estos ecosistemas federados los participantes pueden ofrecen y consumir servicios y datos en competencia. Este concepto engloba a los espacios de datos, pero moviendo el foco de solo los datos para incluir también a los servicios, creando así una suerte de *cloud* descentralizada [41, 42].

Entre los proyectos que desarrolla Gaia-X para cumplir con su misión, destacan los siguientes [43]:

- **Estándares y directrices técnicas**: Documento de arquitectura, documento de cumplimiento, documento de gestión de identidad, credenciales y acceso y documento de intercambio de datos.
- **Implementación**: La comunidad de Gaia-X proporciona los componentes software que implementan sus especificaciones técnicas.
- **Cámara de compensación digital**: Gaia-X se encarga de supervisar y certificar organizaciones para que provean servicios de cámaras de compensación en el ecosistema de Gaia-X. Las cámaras de compensación forman el corazón del ecosistema, y se encargan de certificar que todos los participantes y servicios cumplen las reglas y principios de Gaia-X.
- **Academia**: Actúa como el centro de formación y difusión de conocimientos de la iniciativa. Se centra en capacitar a profesionales y actores del ecosistema digital en los principios, estándares y herramientas de Gaia-X, mediante cursos, talleres, seminarios y materiales educativos.
- **Proyectos avalados**: Iniciativas que han sido evaluadas y aprobadas por la asociación Gaia-X por cumplir con sus altos estándares de interoperabilidad, seguridad y gobernanza. Estos proyectos actúan como referentes y modelos de buenas prácticas dentro del ecosistema, demostrando soluciones efectivas y seguras que contribuyen a la soberanía digital europea y fomentan la colaboración entre diversos actores.

3.4. Implementaciones

En este trabajo vamos a utilizar componentes de Eclipse Dataspace Components (EDC) y de INES-Data para implementar nuestro proveedor. En esta sección describimos brevemente el contexto de estas iniciativas. Los detalles técnicos relevantes se detallan más adelante, en el Capítulo 5 (Sección 5.3: Contexto tecnológico).

Si estuviéramos interesados en encontrar implementaciones alternativas que sigan los estándares de los IDS, podríamos buscarlas por ejemplo en el informe de conectores de datos de IDSA (data connector report) [44], en la lista de conectores certificados por IDSA [45], o también en el folleto de casos de uso de IDSA [30]. Algunos de estos conectores se muestran en la Figura 3.8.

Name of connector	Maintainer	Open source	IDS certified	Identity management	Deployment options	Service level
Eclipse Dataspace Components (EDC)		✓			Not specified	Service level is best effort of the open-source-community
Data Space Integration	SAP			• X.509 • did:web • SSI	Cloud	Platform-aaS
FIWARE Data Space Connector	FIWARE FOUNDATION	✓		X.509	• On-premises • Cloud	• Connector-aaS • Self-service
GATE Dataspace Connector	GATE	✓	✓	X.509	On-premises	Platform-aaS
EONA-X EDC Connector	amadeus	✓		did:web	• On-premises • Cloud	• Connector-aaS • Self-service

Figura 3.8: Información sobre algunos conectores basados en el IDS-RAM, extraida de [44]

3.4.1. Eclipse Dataspace Components (EDC)

Eclipse Dataspace Components es un ***framework*** integral (concepto, arquitectura, código, ejemplos) que permite construir espacios de datos. El *framework* proporciona un conjunto básico de características, que los usuarios pueden reutilizar, ampliar y personalizar de acuerdo con sus necesidades. Se basa en las especificaciones del marco de gobernanza de Gaia-X y el protocolo de espacios de datos de IDSA [46].

El proyecto EDC comienza en junio de 2021, es administrado por la Eclipse Foundation y es por tanto de de código abierto. También es apoyado por algunas empresas y organizaciones de investigación: Amadeus, BMW Group, Fraunhofer, Huawei, Microsoft y TNO: Innovation for life.

EDC desarrolla los siguientes componentes: Conector, Catálogo federado, Centro de identidades, Servicio de registro y Panel de datos (GUI de administración). También provee muestras y un repositorio con un espacio de datos mínimo viable para apoyar la adopción por desarrolladores.

Dos ejemplos de espacios de datos que utilizan EDC son EONA-X (sector turismo, movilidad y logística) y Catena-X (sector industria automotriz). Ambos forman parte de los proyectos avalados por Gaia-X.

3.4.2. INESData

INESData (Infraestructura para la INvestigación de ESpacios de DAtos distribuidos) es una **incubadora de espacios de datos** en España liderada desde la Universidad Politécnica de Madrid (UPM). El proyecto tiene como objetivos fomentar la adopción de los espacios de datos, acelerar la creación de espacios de datos en España y contribuir al ecosistema global con cuatro espacios de datos nacionales: idioma, movilidad, medios y legal. El proyecto está financiado por el Ministerio de Transformación Digital de España y NextGenerationEU, en el marco del Programa UNICO I+D Cloud [47].

Los objetivos técnicos del proyecto son (transcrito de [4]):

- Proporcionar una **arquitectura lista para usar** para la creación de espacios de datos alineados con arquitecturas a nivel europeo (IDSA, Gaia-X), así como **modelos de gobernanza** para espacios de datos. Incluirá componentes horizontales que respalden estas arquitecturas y faciliten el despliegue de espacios de datos en contextos nacionales, regionales o locales, incluidos entornos de nube.
- **Implementar servicios de valor agregado basados en Inteligencia Artificial** para proporcionar una capa de conocimiento sobre el Espacio de Datos. Con especial atención en:

    - Servicios para el procesamiento del lenguaje natural que se pueden integrar en espacios de datos que dependen en gran medida de datos textuales.
    - Servicios de análisis de datos multimedia y multimodales a gran escala basados en procesamiento de señales y algoritmos de aprendizaje automático.
    - Servicios para generar y aprovechar gráficos de conocimiento a partir de fuentes estructuradas y semiestructuradas, incluida la explicabilidad de los algoritmos de aprendizaje automático.

- **Implementar múltiples espacios de datos** para demostrar los beneficios de los espacios de datos y la aplicabilidad de la tecnología relacionada. Algunas de ellas serán versiones nacionales, regionales o locales de espacios de datos comunes europeos actualmente en desarrollo.

El conector que desarrolla INESData para sus espacios de datos se basa en el *framework* de EDC. A fecha de 22 de febrero de 2025, los repositorios públicos asociados al proyecto de INESData son los siguientes [48]:

- inesdata-map
    - getstarted
    - gen-ai
    - graph-engine
    - editor-frontend
    - editor-backend
- inesdata-espacio-linguistico
    - elg-connector
    - elg-web-service
    - interface
    - demostradores
- inesdata-connector
- inesdata-connector-interface
- inesdata-registration-service
- inesdata-public-portal-frontend
- inesdata-public-portal-backend
- inesdata-local-env
- inesdata-ml-schema
- inesdata-mov-data-generation

3.5. Apoyo institucional

3.5.1. Unión Europea

Los espacios de datos son una propuesta clave en la **estrategia de digitalización** de la Unión Europea [49]. Los **datos son considerados esenciales** para el crecimiento económico, la competitividad, la innovación y la creación de empleo. Por eso, el objetivo de la UE es crear un **mercado único de datos** que fortalezca la competitividad europea a nivel global [50].

Los espacios de datos se alinean y son promovidos por dos de las regulaciones más recientes sobre los datos a nivel europeo como son la ley de gobernanza de datos (2022, Data Governance Act) [51] y la ley de datos (2024, Data Act) [52].

La UE promueve la creación de espacios de datos en 14 sectores estratégicos (p.e.: agricultura, salud, energía, movilidad o cultura) y financia iniciativas como el centro de apoyo para espacios de datos (DSSC, Data Spaces Support Centre) [29]. Otras iniciativas como GAIA-X, han contado desde sus inicios con el apoyo de los gobiernos de Alemania y Francia [40].

3.5.2. España

El Ministerio para la Transformación Digital presentó en noviembre de 2024 un ambicioso **Plan de Impulso de los Espacios de Datos Sectoriales**, apoyando así la agenda estratégica **España Digital 2026** [53, 54]. Este plan tiene como objetivo fomentar la innovación y mejorar la competitividad en todos los sectores productivos de España. Cuenta con un presupuesto total de 500 millones de euros y durará hasta el año 2026. En la Figura 3.9 se pueden ver las iniciativas del proyecto y sus presupuestos asignados.

EJE	ID	INICIATIVA	Presupuesto en M€	TOTAL en M€	%
1	#01	Casos de uso	110	160	32%
	#02	Casos de uso para el sector turismo	50		
2	#03	Kit Espacios de Datos	127	127	25%
3	#04	Productos y servicios tecnológicos para Espacios de Datos	44	44	9%
4	#05	Gestión de la demanda del dato público	20	20	4%
5	#01	Demostradores	40	139	28%
	#06	Plataforma Espacio de Datos Sector Turismo	35		
	#07	Espacio de Datos de la Nueva Economía de la Lengua	12		
	#08	Espacio de Datos de las Infraestructuras Urbanas Inteligentes	13		
	#09	Espacios de Datos de Desarrollo Regional	39		
6	#10	Promoción, Sensibilización y Capacitación	5	5	1%
-	#11	Centro de Referencia de Espacios de Datos Sectoriales	5	5	1%
TOTAL			500	500	100%

Figura 3.9: Tabla resumen con las iniciativas incluidas en el Plan de Impulso de los Espacios de Datos Sectoriales [54]

3.6. Ejemplo: movilidad

Los espacios de datos hacen factibles proyectos que no serían posibles sin ellos. Muy evidentemente, en aquellos casos donde ninguna entidad tenga por sí sola todos los datos necesarios. El beneficio de los espacios de datos es por tanto que permiten el desarrollo de **nuevos servicios innovadores**. Proporcionan el ecosistema necesario para acceder a los datos, y además permitiendo que los participantes mantengan su soberanía [26].

Los espacios de datos tienen aplicaciones en la inmensa mayoría de sectores económicos, pero esto es quizás mas evidente para el sector de la **movilidad**. El proyecto espacio de datos de movilidad (*Mobility Data Space*, MDS) es una iniciativa del gobierno federal alemán, y remonta sus orígenes a 2019. Está operativo desde 2022 y cuenta con más de 100 miembros interesados en el sector de la movilidad: empresas de transporte públicas y privadas, universidades e instituciones académicas y empresas de automoción [55].

El MDS es un proyecto avalado por Gaia-X y funciona como un mercado de datos de movilidad y logística. Algunos de los casos de uso que tiene el MDS son los cuatro que contiene el folleto de casos de uso de IDSA para el MDS [30], o los otros diez que podemos encontrar en su página web [56].

1. Mejores pronósticos de tráfico mediante aprendizaje automático (PTV Group)
2. Medición y pronóstico de la calidad del aire (ZF Group)

3. Uso sostenible de los motores eléctricos para vehículos eléctricos híbridos enchufables (CARUSO Dataplace)
4. Decisiones basadas en información de riesgos (Volkswagen Group)
5. Soluciones Avanzadas de Datos: Datos de vehículos y condiciones de las carreteras (Bridgestone Mobility Solutions)
6. Conexión de datos para una mayor seguridad vial (Esri)
7. Puntos peligrosos en el tráfico (Mobias)
8. Información local sobre riesgos (BMW)
9. 'Monitoreo de Estacionamiento' y 'Carretera Resbaladiza' (Mercedes-Benz)
10. Ecosistema urbano inteligente para una vida urbana centrada en el ser humano (Solita)
11. Asistente inteligente de ubicación DeepVolt (DeepVolt)
12. Paga según cómo conduces (Empresas aseguradoras)
13. Optimización basada en IA de las ofertas de movilidad actuales (highQ)
14. Información sobre la ocupación de plazas de estacionamiento, OptiPark ([ui!] Urban Mobility Innovations)

Los casos de uso del 4 al 9 están muy relacionados entre sí, siendo el tema común entre ellos la **seguridad vial**. Si estas empresas recogen datos mediante sensores instalados en sus vehículos – siempre con el consentimiento de los clientes – para, por ejemplo, detectar el mal estado en las carreteras, detectar condiciones climáticas adversas o detectar accidentes que ya hayan sucedido, podrían alertar en tiempo real a otros vehículos, incluso de diferentes compañías. Esto permitiría a los conductores tomar precauciones, recalcular rutas o adoptar las medidas necesarias. Además, la información podría compartirse con las autoridades, centros de control de tráfico y servicios de mantenimiento de carreteras para que actúen con la mayor celeridad posible.

Los casos de uso 9 y 14 hacen énfasis en el **estacionamiento**. Disponer de un mapa en tiempo real de plazas de aparcamiento libres, y de una estimación futura de su ocupación, serviría por ejemplo para **reducir la congestión de tráfico** reduciendo el tiempo necesario para encontrar una plaza de aparcamiento. Esta información también sería muy útil por ejemplo, para que las autoridades puedan mejorar la gestión de los aparcamientos.

Nótese que ambas aplicaciones se benefician de cuantos más participantes haya en el espacio de datos, mejorando el servicio por igual para todos. Los oferentes de datos se benefician por el precio que cobran a sus datos, y los consumidores de datos por el mejor servicio que ofrecen a sus clientes, pudiendo las empresas adoptar ambos roles. Con una mayor disponibilidad de datos aumenta su valor, se fomenta la innovación, la competencia y la creación de nuevos servicios y productos que redundan en un mayor beneficio para la sociedad en su conjunto.

Capítulo 4

Requisitos y análisis

En este capítulo describimos las fases iniciales de elicitación de requisitos y análisis del proveedor que desarrollaremos en este trabajo. Para ello, nos basaremos en las especificaciones de los espacios de datos internacionales (IDS), en concreto en el IDS RAM [36] y en el Dataspace Protocol (DSP) [37].

La **fase de elicitación de requisitos** tiene como objetivo identificar y formalizar las necesidades, expectativas y restricciones de los usuarios y demás partes interesadas para el desarrollo del sistema. De manera informal, describimos la funcionalidad del proveedor en la Sección 4.1: Descripción del sistema, en donde podemos identificar los actores principales, los casos de uso, algunas clases y algunos de los requisitos del sistema. Y de manera formal, detallamos los requisitos en la Sección 4.3: Requisitos. Además, también describimos los casos de uso en la Sección 4.4 y especificamos algunos de ellos.

La **fase de análisis** utiliza los requisitos y las descripciones abstractas de la fase anterior y las transforma en descripciones concretas que muestran de forma más evidente el comportamiento del sistema. Corresponden a esta fase la identificación de clases, la elaboración del Modelo del dominio (Sección 4.5) y la Realización en análisis de los casos de uso (Sección 4.6).

4.1. Descripción del sistema

El proveedor debe **integrarse en un espacio de datos**, el cual estará formado por **otros participantes** debidamente **identificados**. El proveedor tiene como propósito compartir con el resto de participantes unos datos que se almacenan en un ***warehouse***. Pero como paso previo para dar acceso a ellos, deberá proporcionar unos metadatos (**catálogo**) que contienen un **vocabulario** y las **políticas de acceso** para cada uno de los **activos** que ofrece. El proveedor no almacena los datos, a los cuales tiene acceso de forma externa. Si que almacena sin embargo los metadatos ya mencionados.

El **catálogo** es una lista que contiene todas las **ofertas** del proveedor. Cada **oferta** fija las condiciones (**política de acceso**) para acceder a uno o más **activos**. Los **activos** son metadatos que identifican, describen y representan a un determinado recurso (principalmente, datos). El **administrador del proveedor** es el encargado de gestionar el **catálogo**, **creando o modificando políticas de acceso**, **creando o modificando activos** y **creando o modificando ofertas**, es decir, asociando las políticas a los activos.

Los participantes que quieran **acceder a los datos** del proveedor deberán previamente firmar un **contrato** tras un **proceso de negociación**, aceptando una **oferta**. Después podrán solicitar la **transferencia** de los datos, indicando el **activo** al que quieren acceder y el **contrato** firmado.

La **identidad** de cada participante consiste en la serie de **certificados** que posee. Los tipos de **certificados** que existen son definidos y emitidos por la autoridad de gobernanza del espacio de datos y pueden ser requeridos como parte de una **política de acceso**. En última instancia es el **administrador** el encargado de **crear y modificar** la **identidad** del proveedor.

4.2. Roles de usuarios del sistema

Con nuestro proveedor interactuarán cinco tipos de usuarios, cada uno con diferentes intenciones, permisos y capacidades. Estas descripciones están basadas en el IDS RAM [36]. Son los siguientes:

- **Consumidor**: Participante del espacio de datos que quiere acceder a los datos.
- **Administrador**: Encargado de gestionar el proveedor. Administra la identidad, el catálogo, las políticas de acceso y los activos.
- **Repositorio de vocabularios**: Participante del espacio de datos que almacena vocabularios de todos los participantes, creando así un vocabulario armonizado para el espacio de datos.
- **Intermediario de metadatos**: Participante del espacio de datos que almacena ofertas de todos los participantes, con el objetivo de aumentar su disponibilidad.
- ***Warehouse***: Base de datos que almacena los datos que comparte el proveedor.

Omitimos al **proveedor de identidad**, a saber, participante del espacio de datos que proporciona servicios de autoridad de certificación (emisión de certificados) y un servicio de aprovisionamiento de atributos dinámicos (emisión de *tokens*), porque suponemos que el **administrador** adquiere los certificados necesarios directamente.

4.3. Requisitos

Los requisitos definen lo que un sistema informático debe hacer o cómo debe comportarse. Describen por tanto las funcionalidades, características y limitaciones que debe tener para cumplir con el objetivo del proyecto.

Queremos que el proveedor cumpla con los estándares de los espacios de datos internacionales y, por tanto, estos requisitos están orientados a implementar el Dataspace Protocol [37], y basados también en el resto de requisitos de seguridad, soberanía y confianza del IDS Reference Architecture Model [36].

Requisitos de información

RI-0 La identidad del sistema está formada por un identificador único, un certificado X.509 y los certificados específicos del espacio de datos (por ejemplo, de pertenencia o de entorno operacional[1]).

RI-1 Las políticas de acceso contendrán restricciones como, por ejemplo, prohibir el almacenamiento de los datos, o su transferencia a terceras partes. También pueden requerir al consumidor tener un certificado específico.

RI-2 Un activo puede describir cualquier tipo de recurso identificable como, por ejemplo, datos/información, aplicaciones o servicios. Los activos deben describir el tipo de contenido y su forma de acceso. Para ello tendrá atributos de identificador, nombre, descripción, versión y tipo de contenido. En caso de describir datos, se podrá detallar su semántica con un vocabulario. También se podrá establecer su precio, forma de cobro y métodos de pago aceptados.

[1] Un certificado de entorno operacional es emitido por el cuerpo de gobernanza del espacio de datos para garantizar que un participante cumple con ciertos requisitos de confianza y seguridad en relación con sus procesos organizativos y su entorno operativo [36].

Requisitos funcionales

RF-0 El sistema debe permitir al administrador definir su identidad (del sistema).

RF-1 El sistema debe permitir al administrador definir activos.

RF-2 El sistema debe permitir al administrador definir políticas de acceso.

RF-3 El sistema debe permitir al administrador definir ofertas, relacionando a una política de acceso con al menos un activo.

RF-4 El sistema debe permitir a los consumidores con identidad válida, consultar su identidad.

RF-5 El sistema debe permitir a los consumidores con identidad válida, acceder a su catálogo.

RF-6 El sistema debe permitir a los consumidores con identidad válida, negociar un contrato sobre una oferta concreta.

RF-7 El sistema debe permitir a los consumidores con identidad válida, iniciar un proceso de transferencia para un activo, si existe un contrato válido acordado por ambas partes.

RF-8 El sistema debe permitir a los consumidores con identidad válida, acceder a los datos, si existe un proceso de transferencia validado y activo.

Restricciones

RE-0 El sistema (el software) debe estar certificado por la IDSA. La certificación demuestra que el sistema provee la funcionalidad, interoperabilidad y nivel de seguridad requeridos.

RE-1 El sistema (la instancia) debe adquirir un certificado X.509 de un autoridad de certificación y un token de atributo dinámico de un servicio de aprovisionamiento de atributos dinámicos para identificarse[2].

RE-2 El sistema debe implementar el Dataspace Protocol de IDSA.

RE-3 El sistema debe tener acceso al *warehouse*.

Requisitos no funcionales

RNF-0 El sistema debe poder verificar la identidad de los consumidores, además de sus capacidades y características de seguridad.

RNF-1 El sistema debe permitir adaptar el catálogo para cada consumidor, ocultando o filtrando ofertas según criterios personalizados.

RNF-2 El sistema debe asegurar que las comunicaciones con el exterior estén encriptadas y su integridad protegida.

RNF-3 El sistema debe almacenar en un registro cada acción efectuada, accesos o transmisiones de datos e incidentes.

RNF-4 El sistema debe poder asegurarse de que las política de acceso que establece serán cumplidas por los consumidores.

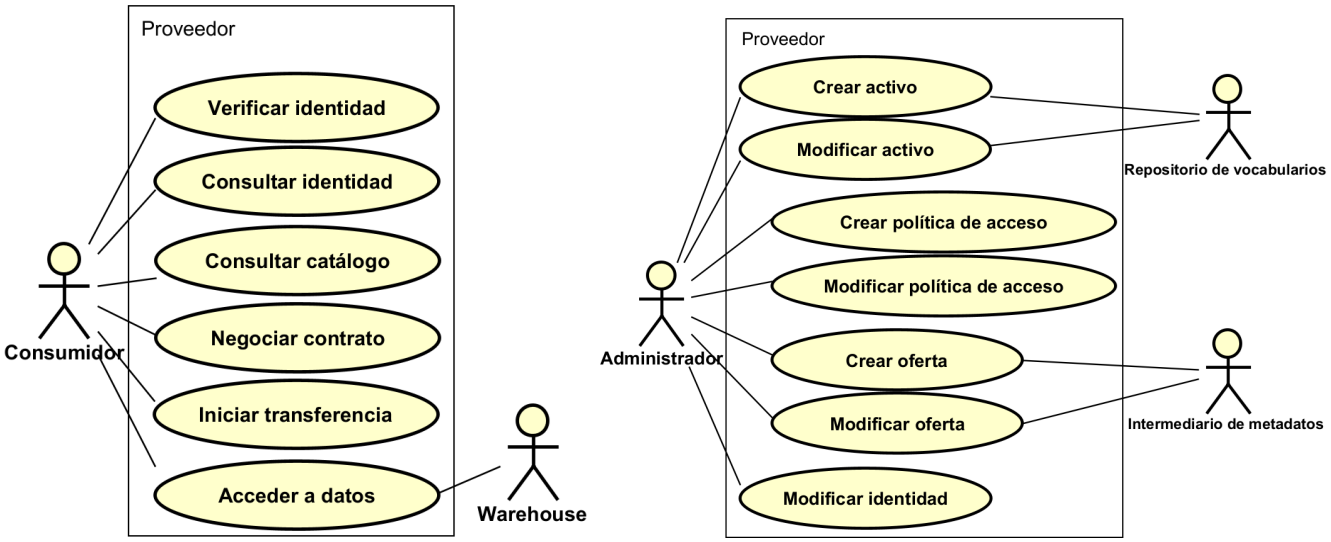
RNF-5 El sistema debe permitir el acceso a los datos de dos maneras: enviándolos directamente al consumidor o mediante un punto de acceso (*push*/*pull*).

[2]Un servicio de aprovisionamiento de atributos dinámicos (*Dynamic Attribute Provisioning Service*, DAPS) verifica la validez del software y la certificación del participante para emitir los token de atributo dinámico (*Dynamic Attibute Token*, DAT), que además de identificar al portador, contienen información dinámica como la ubicación del dispositivo, que podrían cambiar con el tiempo [36].

4.4. Casos de uso

Los casos de uso son una descripción detallada de cómo los usuarios interactúan con el sistema para alcanzar un objetivo. En esta sección hemos **identificado** todos los casos de uso del sistema y los describimos en la Figura 4.1: Diagrama de casos de uso. Además, **especificaremos** algunos de los casos de uso más relevantes: cuatro casos de uso del consumidor en la Subsección 4.4.1 y cuatro casos de uso del administrador en la Subsección 4.4.2 .

No es nuestro objetivo hacer una especificación exhaustiva de todos los casos de uso y por eso solo nos hemos centrado en ocho de ellos. Teniendo como referencia al resto, especificar los seis restantes no debería ser un ejercicio complicado.



(a) Casos de uso del consumidor

(b) Casos de uso del administrador

Figura 4.1: Diagrama de casos de uso

4.4.1. Del consumidor

Verificar identidad

Actores: Consumidor.

Precondiciones: Se ha producido un *TLS handshake* y se ha establecido un canal seguro.

Secuencia normal:

1. El Consumidor envía un *nonce*.
2. El sistema envíaa otro *nonce* y solicita la identidad del Consumidor.
3. El Consumidor envía su certificado de identidad y solicita la identidad del sistema.
4. El sistema verifica el certificado del Consumidor y envía su certificado de identidad.
5. El Consumidor verifica el certificado del sistema.
6. El sistema solicita el token de atributo dinámico (DAT) del Consumidor.
7. El Consumidor envía su DAT y solicita el del sistema.
8. El sistema verifica el DAT del Consumidor y envía el suyo.
9. El Consumidor verifica el DAT del sistema.

Alternativas y excepciones:

(4a, 8a) Si la verificación falla, el caso de uso queda sin efecto.

(5a, 9a) Si la verificación del consumidor falla, el caso de uso queda sin efecto.

Postcondiciones: El consumidor y el sistema han verificado sus identidades.

Consultar catálogo

Actores: Consumidor.

Precondiciones: El consumidor y el sistema han verificado sus identidades.

Secuencia normal:

1. El Consumidor solicita consultar el catálogo mandando una petición firmada.
2. El sistema comprueba que la firma de la petición es correcta.
3. El sistema filtra las ofertas del catálogo para devolver solo las que puede negociar el Consumidor.
4. El sistema devuelve el catálogo filtrado.

Alternativas y excepciones:

(2a) Si la firma no es correcta el sistema devuelve un error y el caso de uso queda sin efecto.

Postcondiciones: El Consumidor tiene acceso a todas las ofertas que puede negociar.

Negociar contrato

Actores: Consumidor.

Precondiciones: El consumidor y el sistema han verificado sus identidades.

Secuencia normal:

1. El Consumidor solicita negociar un contrato mandando una petición firmada con el id de una oferta.
2. El sistema comprueba que la sintaxis, el contenido y la firma de la petición son correctas.
3. El sistema crea una negociación con estado *solicitada*.
4. El sistema decide mandar una contraoferta.
5. El sistema manda la oferta al Consumidor y cambia el estado de la negociación a *ofertada*.
6. El Consumidor verifica que la sintaxis, el contenido y la firma de la oferta son correctas.
7. El Consumidor acepta la oferta y manda un acuerdo al sistema.
8. El sistema comprueba que la sintaxis, el contenido y la firma del acuerdo son correctas.
9. El sistema cambia el estado de la negociación a *aceptada*.
10. El sistema registra el acuerdo, se lo manda al Consumidor y cambia el estado de la negociación a *acordada*.
11. El Consumidor verifica que la sintaxis, el contenido y la firma del acuerdo son correctas.
12. El Consumidor registra el acuerdo e informa al sistema de su verificación.
13. El sistema cambia el estado de la negociación a *verificada*.
14. El sistema cambia el estado de la negociación a *finalizada*, crea un contrato, se lo manda al Consumidor y le informa de que la negociación ha terminado.

Alternativas y excepciones:

(2a, 6a, 8a) Si algo no es correcto el sistema devuelve un error y el caso de uso queda sin efecto.

(4a) Si el sistema acepta la petición el caso de uso continúa en el paso 10.

(4b) Si el sistema rechaza la petición el caso de uso queda sin efecto.

(4c, 10a, 14a) Si el sistema decide terminar la negociación el caso de uso queda sin efecto.

(7a) Si el Consumidor rechaza la oferta el caso de uso queda sin efecto.

(7b) Si el Consumidor decide mandar una contraoferta, el caso de uso continúa en el paso 1.

(7c, 12a) Si el Consumidor solicita terminar la negociación el caso de uso queda sin efecto.

Postcondiciones: El Consumidor y el sistema almacenan un contrato acordado por ambas partes.

Iniciar transferencia

Actores: Consumidor.

Precondiciones: El consumidor y el sistema han verificado sus identidades.

Secuencia normal:

1. El Consumidor solicita iniciar una transferencia mandando una petición firmada indicando el id del activo al que quiere acceder, el id de un contrato y la forma de acceso a los datos (*push*/*pull*).
2. El sistema comprueba que la sintaxis, el contenido y la firma de la petición son correctas.
3. El sistema comprueba que el activo existe.
4. El sistema comprueba que el contrato existe y que la oferta que se negocia en él contiene al activo al que se quiere acceder.
5. El sistema crea una transferencia con estado *solicitada*.
6. El sistema cambia el estado de la transferencia a *empezada* e informa al Consumidor.

Alternativas y excepciones:

(2a, 3a, 4a) Si no se verifica, el sistema devuelve un error y el caso de uso queda sin efecto.

Postcondiciones: El sistema tiene un proceso de transferencia empezado.

4.4.2. Del administrador

Modificar identidad

Actores: Administrador.

Precondiciones: Ninguna.

Secuencia normal:

1. El Administrador introduce la identidad y su código de autorización.
2. El sistema comprueba que el Administrador está autorizado.
3. El sistema comprueba que la sintaxis y el contenido de la identidad son correctos.
4. El sistema crea y se asigna la nueva identidad.

Alternativas y excepciones:

(2a) Si el código de autorización no es válido, el sistema devuelve un error y el caso de uso queda sin efecto.

(3a) Si la identidad no es correcta, el sistema devuelve un error y el caso de uso queda sin efecto.

Postcondiciones: El sistema tiene una nueva identidad actualizada.

Crear activo

Actores: Administrador, Repositorio de vocabularios.

Precondiciones: Ninguna.

Secuencia normal:

1. El Administrador introduce su código de autorización y todos los datos de un nuevo activo: identificador, nombre, descripción, versión, tipo de contenido y forma de acceso al recurso que representa. Opcionalmente podrá introducir un vocabulario (y si se quiere publicar en un repositorio de vocabularios), precio, forma de cobro y métodos de pago.
2. El sistema comprueba que el Administrador está autorizado.
3. El sistema comprueba que la sintaxis y el contenido del activo son correctos.
4. El sistema crea el activo.
5. El sistema crea el vocabulario.
6. El sistema publica el vocabulario en un repositorio de vocabularios.
7. El Repositorio de vocabularios valida y almacena el vocabulario.

Alternativas y excepciones:

(2a) Si el código de autorización no es válido, el sistema devuelve un error y el caso de uso queda sin efecto.
(3a) Si el activo no es correcto, el sistema devuelve un error y el caso de uso queda sin efecto.
(5a) Si el Administrador no ha indicado vocabulario, el caso de uso termina con éxito.
(6a) Si el Administrador ha indicado que no quiere que se publique el vocabulario, el caso de uso termina con éxito.

Postcondiciones: El sistema tiene un nuevo activo.

Crear política de acceso

Actores: Administrador.

Precondiciones: Ninguna.

Secuencia normal:

1. El Administrador introduce su código de autorización y los datos de una nueva política de acceso: identificador, descripción y restricciones.
2. El sistema comprueba que el Administrador está autorizado.
3. El sistema comprueba que la sintaxis y el contenido de la política de acceso son correctos.
4. El sistema crea la política de acceso.

Alternativas y excepciones:

(2a) Si el código de autorización no es válido, el sistema devuelve un error y el caso de uso queda sin efecto.
(3a) Si la política de acceso no es correcta, el sistema devuelve un error y el caso de uso queda sin efecto.

Postcondiciones: El sistema tiene una nueva política de acceso.

Crear oferta

Actores: Administrador, Intermediario de metadatos.

Precondiciones: Ninguna.

Secuencia normal:

1. El Administrador introduce su código de autorización y los datos de una nueva oferta: nombre, activos a los que da acceso y la política de acceso asociada.
2. El sistema comprueba que el Administrador está autorizado.
3. El sistema comprueba que la sintaxis y el contenido de la oferta son correctos.
4. El sistema crea la oferta.
5. El sistema publica la oferta en un intermediario de metadatos.
6. El Intermediario de metadatos valida y almacena la oferta.

Alternativas y excepciones:

(2a) Si el código de autorización no es válido, el sistema devuelve un error y el caso de uso queda sin efecto.

(3a) Si la oferta no es correcta, el sistema devuelve un error y el caso de uso queda sin efecto.

Postcondiciones: El sistema tiene una nueva oferta, que pasará a formar parte del catálogo.

4.5. Modelo del dominio

Mostramos el modelo del dominio resumido en la Figura 4.2 y el modelo detallado en la Figura 4.3. Para representar las clases *Política de acceso* y *Oferta* hemos utilizado como referencia el modelo Open Digital Rights Language (ODRL) [57], que es usado por el Dataspace Protocol [37].

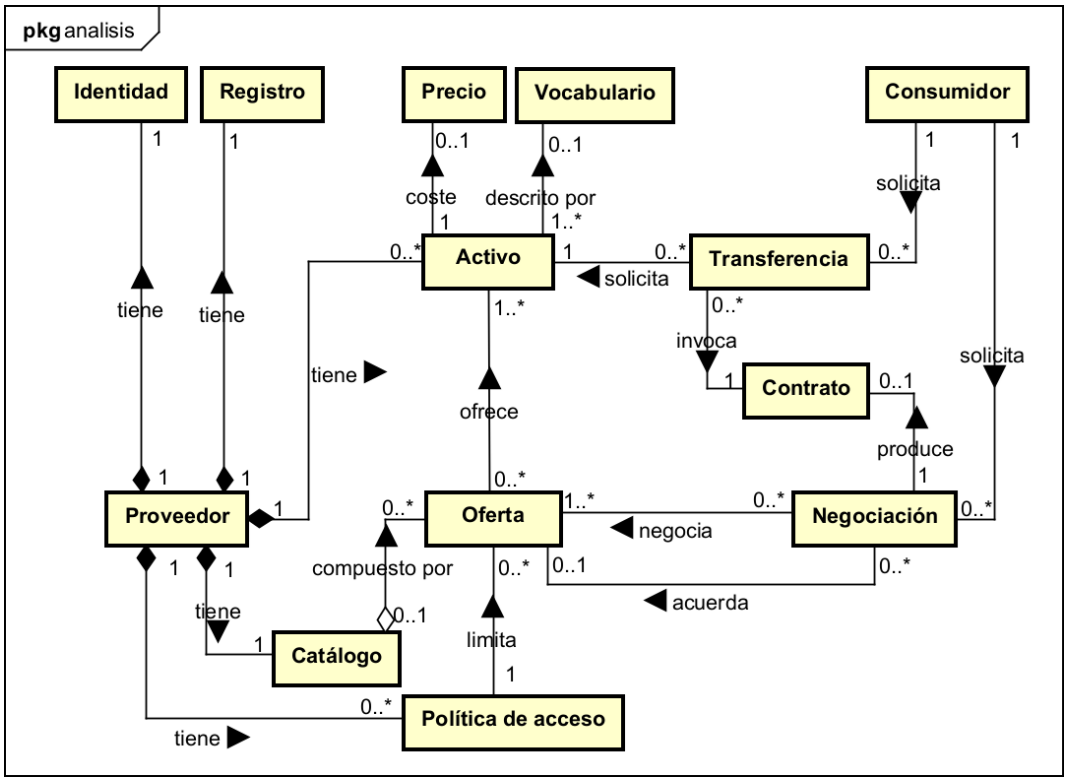


Figura 4.2: Modelo del dominio resumido

El proveedor tiene una identidad, un registro (*logger*), tiene activos que son opcionalmente descritos por una ontología y su coste fijado por precio, tiene políticas de acceso y tiene un catálogo. El catálogo se compone de ofertas, donde cada una ofrece al menos un activo y está limitada por una política de acceso. El consumidor empieza una negociación donde se negocian ofertas (y contraofertas) y si se llega a un acuerdo se produce un contrato, y solicita transferencias de un activo invocando un contrato.

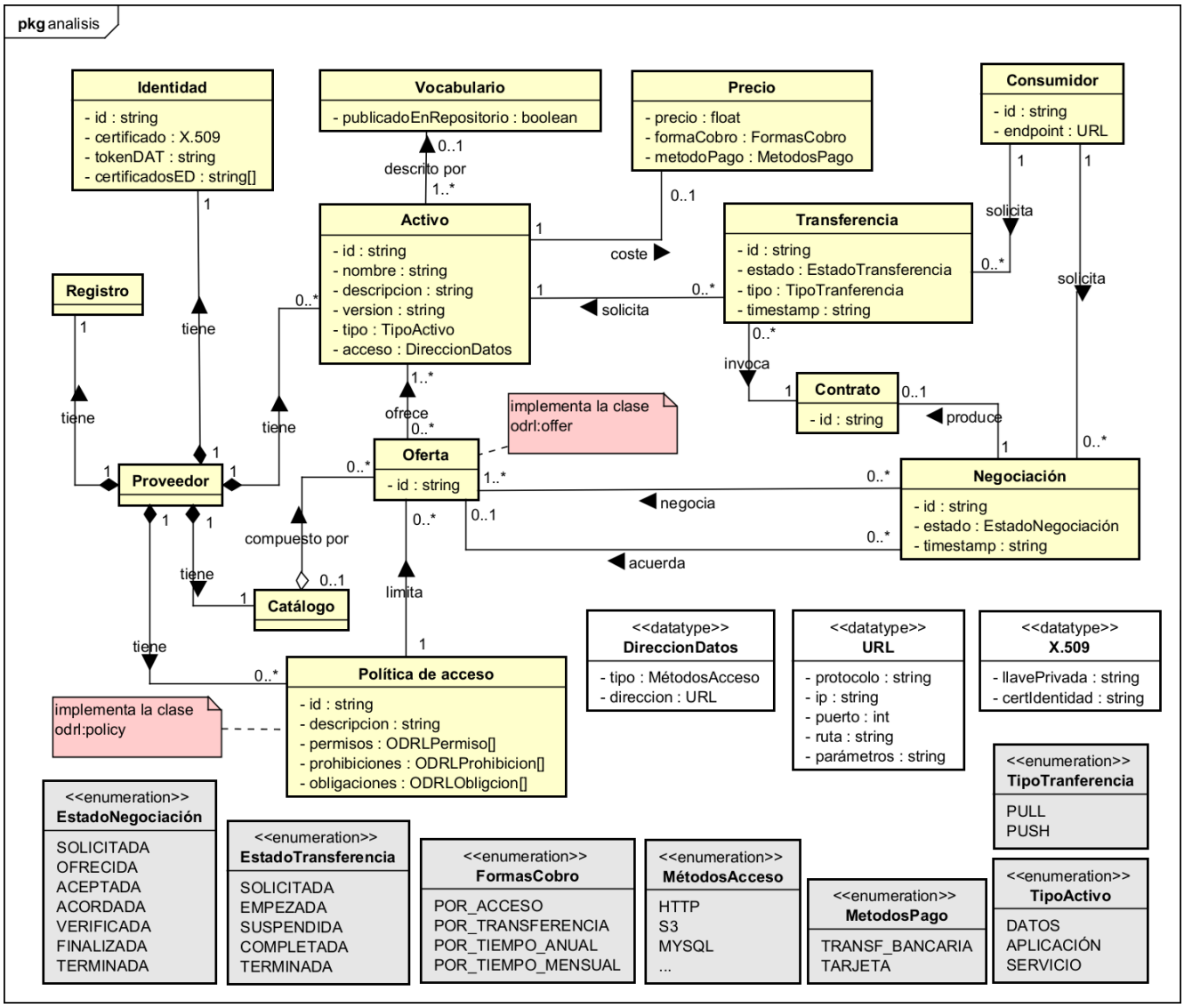


Figura 4.3: Modelo del dominio detallado

4.6. Realización en análisis de los casos de uso

En esta sección realizamos un diagrama de secuencia para las secuencias principales de tres de los casos de uso del consumidor que hemos especificado anteriormente. En la Figura 4.4 mostramos consultar catálogo, en la Figura 4.5 negociar contrato y en la Figura 4.6 iniciar transferencia. Hemos dividido el diagrama de secuencia de negociar contrato en 3 subfiguras para mejorar la legibilidad, correspondiendo a los pasos 1 al 5, 6 al 10 y 11 al 14 de la especificación del caso de uso.

Tampoco es nuestro objetivo que la realización en análisis de los casos de uso sea completa, pero sí pretendemos mostrar la interacción entre las clases del sistema. Por ejemplo, la clase proveedor sería la encargada de interactuar con los actores y crearía en su caso los activos y las políticas de acceso. La clase

catálogo sería la encargada de gestionar las ofertas del proveedor. Y por otra parte, la clase consumidor sería la encargada de crear las negociaciones y transferencias.

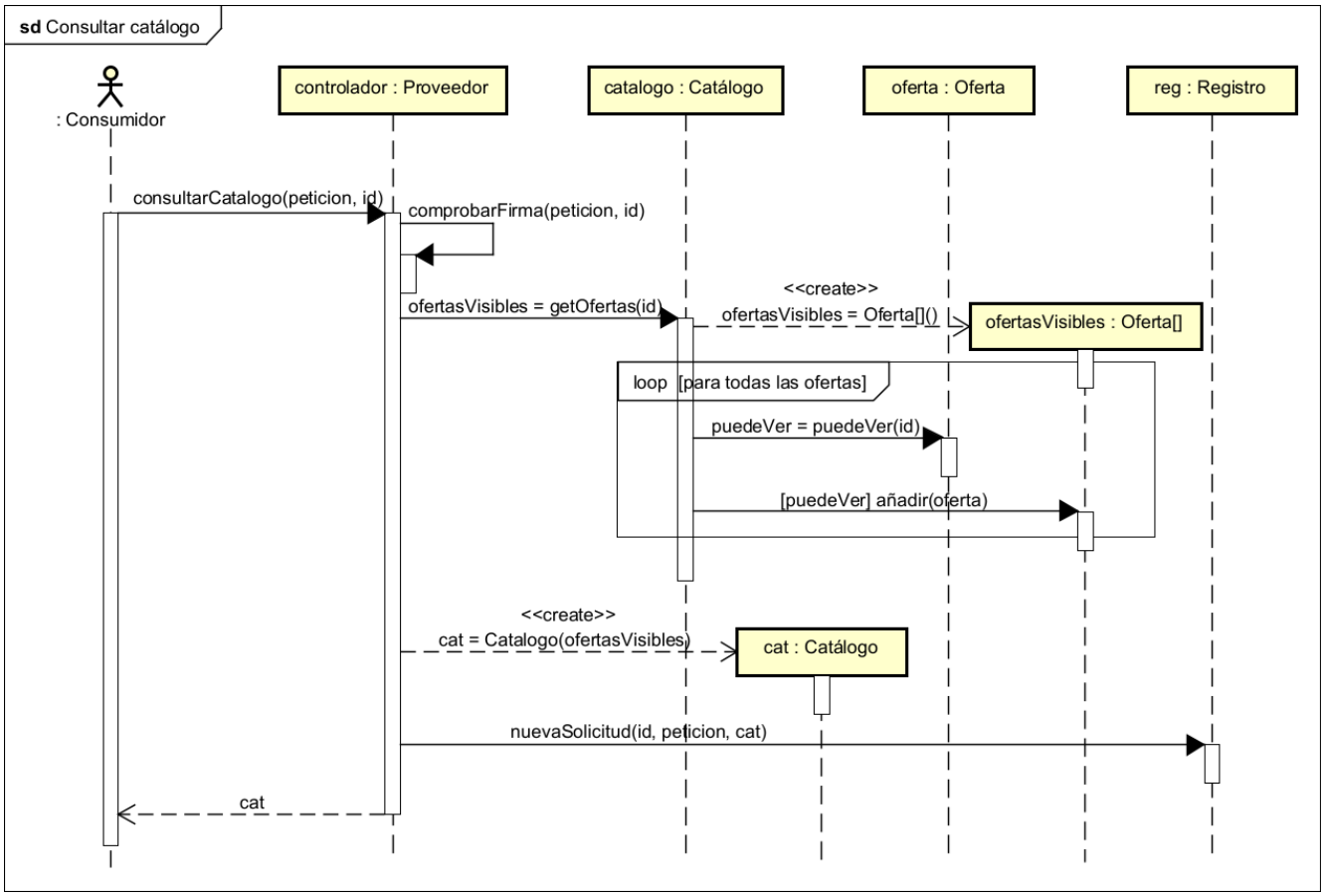
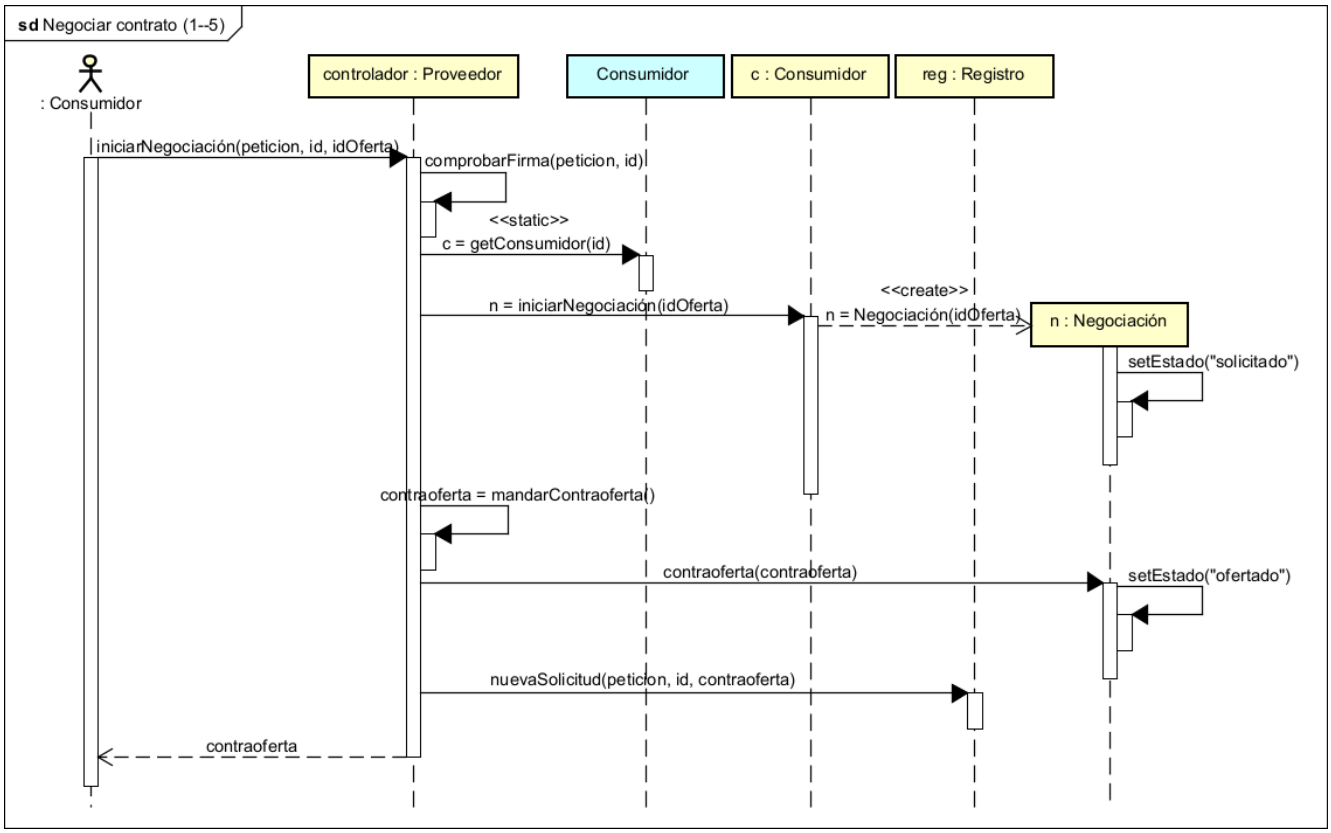
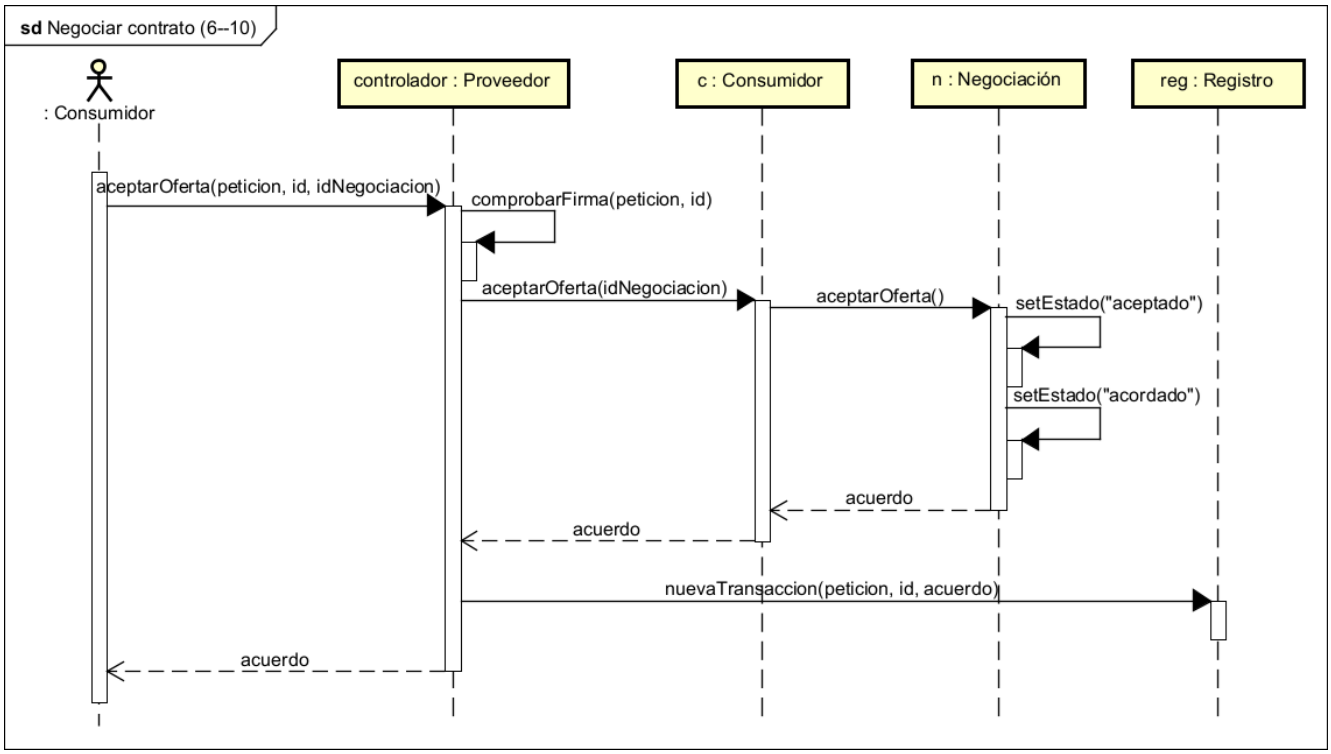


Figura 4.4: Diagrama de secuencia: Consultar catálogo

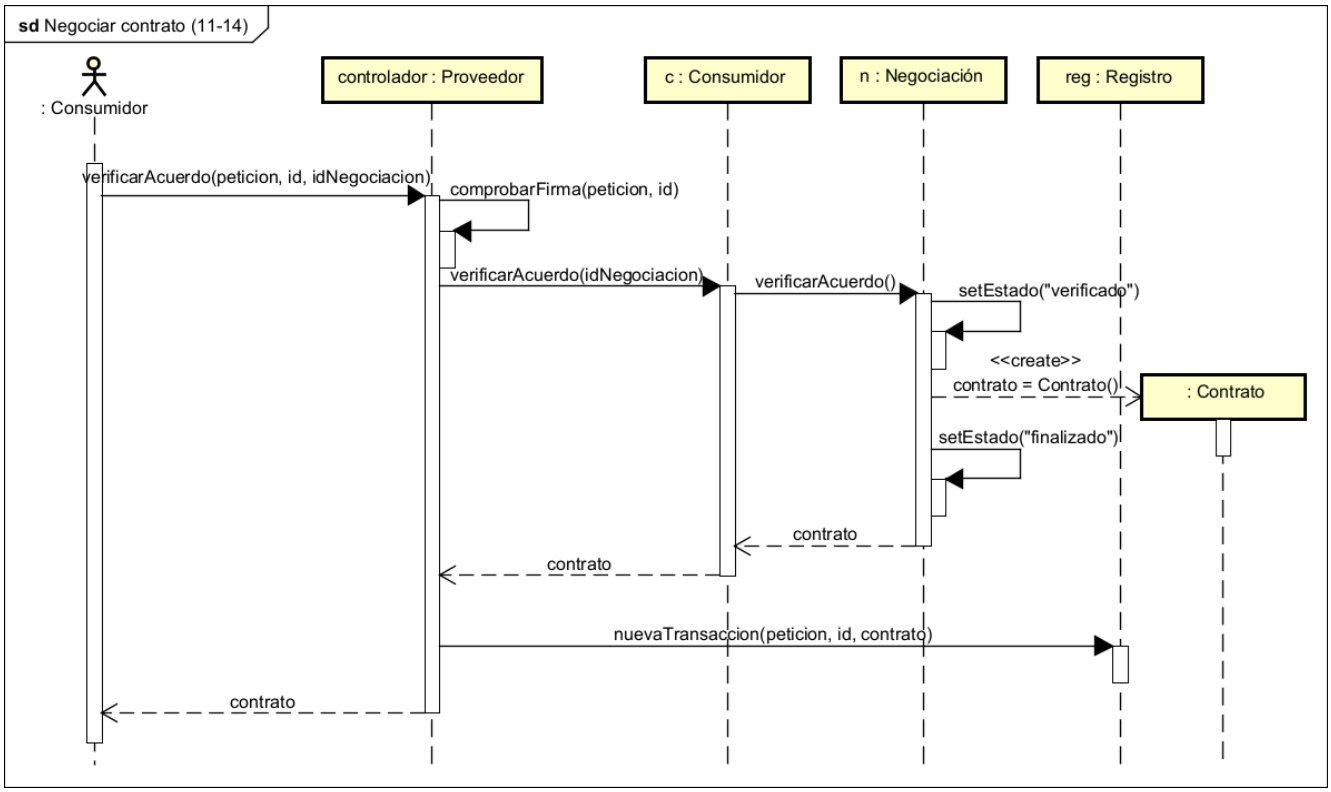


(a) Negociar contrato (pasos del 1 al 5)



(b) Negociar contrato (pasos del 6 al 10)

Figura 4.5: Diagrama de secuencia: Negociar contrato



(c) Negociar contrato (pasos del 11 al 14)

Figura 4.5: Diagrama de secuencia: Negociar contrato (cont.)

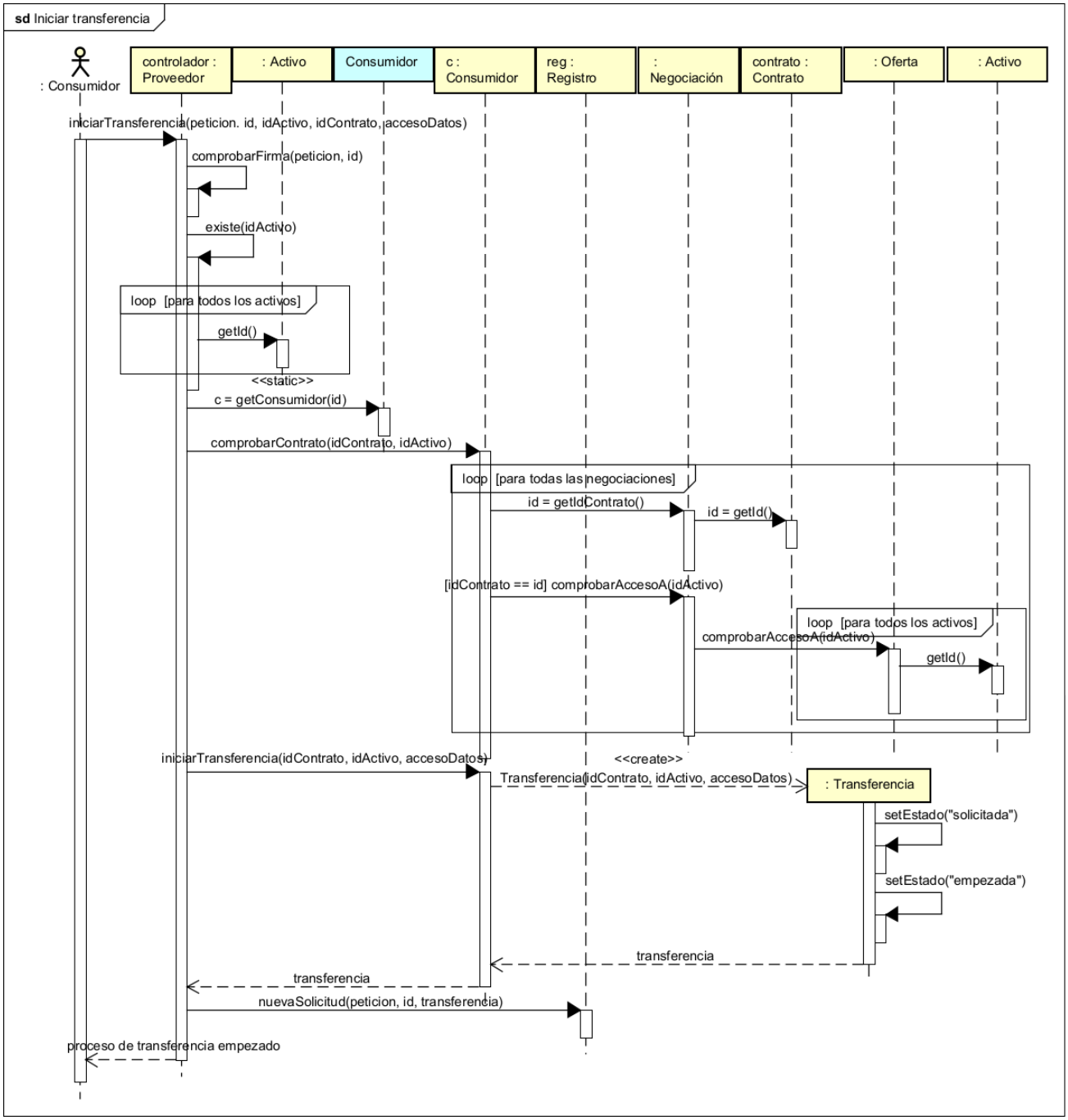


Figura 4.6: Diagrama de secuencia: Iniciar transferencia

Capítulo 5

Diseño

El diseño es el proceso donde se especifica la estructura, componentes, interfaces y comportamiento de un sistema **antes de implementarlo**. El diseño es por tanto el proceso intermedio entre el análisis de requisitos y la implementación. En este capítulo describimos el diseño, que hemos separado en el **diseño de los datos**, y el **diseño del proveedor**.

El **diseño de los datos** se refiere al diseño conceptual de las principales estructuras de datos del proveedor, en las que distinguimos dos áreas de ámbito. Por un lado tenemos que diseñar datos que incumben a todo un espacio de datos, estas serán las **credenciales** (Sección 5.1). Esto es necesario porque el proveedor no será diseñado para integrarse en ningún espacio de datos específico. Y por otro lado diseñaremos los datos que necesita el proveedor por sí mismo, estos serán las **políticas de acceso**, los **activos**, y la **ontología**. Todos estos se diseñan en la Sección 5.2: Metadatos.

El **diseño del proveedor** está condicionado por la elección de las dos tecnologías que hemos decidido utilizar en el proyecto, y de las que ya hemos hablado en el capítulo de espacios de datos (Sección 3.4). En la Sección 5.3: Contexto tecnológico volveremos a hablar sobre ellas, **explicando las tecnologías y estándares** que usaremos y todos los detalles técnicos relevantes para diseñar el proveedor. En este punto comentaremos también la **adaptación necesaria de los datos** para hacerlos compatibles con la elección de estas tecnologías, siguiendo los estándares correspondientes. Esta adaptación se comenta en la Sección 5.4: Adaptación de los datos a los estándares de EDC e INESData.

Elaboraremos el **diseño del proveedor** utilizando como piezas clave a los componentes de EDC e INESData. Nuestro diseño elegirá los componentes adecuados y describirá el uso de sus interfaces y como se conectan entre ellas. Este diseño se detalla en la Sección 5.5: Arquitectura lógica. La segunda parte del diseño se centra en la infraestructura de despliegue, que detallamos en la Sección 5.6: Arquitectura física. Para elaborar este diseño nos hemos basado en recomendaciones de EDC. Por último, describiremos en **detalle el diseño de EDC**. Esto es relevante para poder diseñar extensiones y personalizar los componentes, cosa que nosotros necesitamos hacer. Este se puede ver en la Sección 5.7.

5.1. Credenciales

El primer paso en el diseño de los datos está relacionado con el **diseño de un espacio de datos** en sí mismo. Debería ser la labor de la organización que promueva el espacio de datos la de **definir las credenciales** que hay en el espacio de datos y las **condiciones para adquirirlas**. Nuestro proveedor no será diseñado para integrarse en ningún espacio de datos específico y por tanto será nuestra labor diseñarlas. Es decir, que este diseño depende de un diseño previo (el del espacio de datos) que está ausente.

Las credenciales forman la identidad de un participante en un espacio de datos y son necesarias entre otras cosas para poder diseñar las políticas de control de acceso. Hemos decidido diseñar tres credenciales aunque solo utilizaremos la primera para las políticas de acceso. El motivo es para no perjudicar la interoperabilidad con otros espacios de datos que usen credenciales distintas, que al ser la más básica no será problemático adaptarla. Las otras dos se pueden tomar como ejemplos de credenciales que se podrían llegar a utilizar.

Para representar el diseño conceptual de las credenciales hemos decidido utilizar un diagrama de clases, en el que destacamos las entidades, sus atributos y las relaciones entre ellas. Este diagrama se muestra en la Figura 5.1, y la descripción detallada del significado de cada atributo en la Tabla 5.1. Además, damos también una breve descripción informal a continuación:

1. **Credencial de Miembro**: Contiene información del participante y de su membresía en el espacio de datos.

2. **Credencial de Tipo de Uso**: Define el uso que el participante tendrá con los datos. Puede tomar los siguientes valores: personal, comercial y académico.

3. **Credencial de Participante Confiable**: Certifica que el participante ha pasado por una auditoría por el cuerpo de gobernanza del espacio de datos (emisor de las credenciales). Supone haber validado la información legal y oficial sobre la organización o individuo.

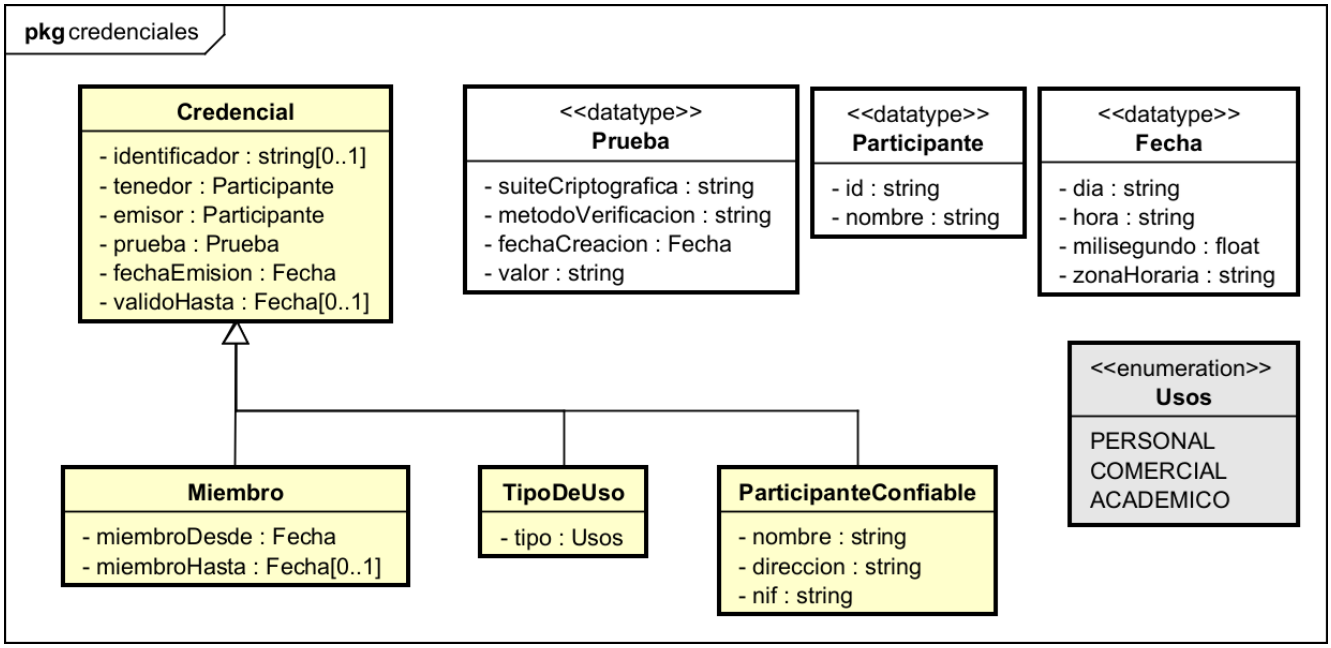


Figura 5.1: Modelo conceptual de las credenciales: diagrama de clases

Para modelar la *Prueba* de las credenciales nos hemos inspirado en las VCs (ver Subsección 5.3.2: Identidad digital) [58].

La utilidad adicional de las credenciales es aportar seguridad y confianza a los participantes del espacio de datos. Por ejemplo, una política de uso podría prohibir el uso comercial de unos datos, pero requiriendo la credencial de *TipoDeUso* se gana seguridad, evitando depender solo de la buena fe de los participantes.

ENTIDAD	ATRIBUTO	DESCRIPCIÓN
Credencial	identificador	Identificador de la credencial
	tenedor	Tenedor o portador de la credencial
	emisor	Emisor de la credencial
	prueba	Emitida por el emisor para demostrar la validez de la información de la credencial
	fechaEmision	Fecha en el que se ha emitido la credencial
	validoHasta	Fecha hasta la que es válida la credencial
Miembro	miembroDesde	Fecha desde la que es miembro el tenedor
	miembroHasta	Fecha hasta la que es miembro el tenedor
TipoDeUso	tipo	Identifica el uso que tenedor dará a los datos
ParticipanteConfiable	nombre	Nombre legal completo del tenedor
	direccion	Dirección completa del tenedor
	nif	Número de identificación fiscal del tenedor
Prueba	suiteCriptografica	Conjunto de algoritmos de cifrado utilizados para poder verificar la credencial
	metodoVerificacion	Método que se debe utilizar para verificar la prueba
	fechaCreacion	Fecha en la que se ha emitido la prueba
	valor	Firma del emisor, un binario codificado en hexadecimal como cadena de caracteres
Participante	id	Identificador del participante
	nombre	Nombre, o alias del participante
Usos	PERSONAL	Fines personales e individuales
	COMERCIAL	Fines comerciales
	ACADEMICO	Fines académicos y de investigación

Tabla 5.1: Modelo conceptual de las credenciales: descripción de los atributos

5.2. Metadatos

La segunda tarea del diseño de los datos involucra ahora sí a nuestro proveedor en particular. Los metadatos son lo que nos permite compartir nuestro repositorio, y son la parte fundamental de cualquier proveedor de un espacio de datos. Los metadatos deben **describir con claridad qué datos se comparten, y con qué condiciones**. Serán accesibles para cualquier miembro del espacio de datos, para que pueda decidir la utilidad que le aportan. En esta sección describimos el diseño de las políticas de acceso, los activos, y la ontología.

Para diseñar la ontología hemos utilizado como referencia los libros *A Semantic Web Primer* de Antoniou y Van Harmelen (2004) [59] y *Semantic Web for the Working Ontologist* de Allemang y Hendler (2011) [60].

5.2.1. Políticas de acceso y uso

Las políticas de acceso y uso establecen las condiciones que un consumidor debe cumplir para acceder o utilizar los datos. Algunas posibilidades podrían ser la ya mencionada de prohibir el uso comercial, o también: prohibir almacenar los datos, prohibir sacar los datos de la Unión Europea, requerir citar al proyecto App-PI, acceso especial para investigadores, etc.

Para este trabajo no establecemos ninguna restricción adicional al acceso de los datos[1], y **estableceremos una única política que solicite ser miembro del espacio de datos**. Definir políticas adicionales es una cuestión relevante, pero que dejaremos como trabajo futuro. Esto es algo que depende en gran medida de un trabajo posterior, como de los datos que se quieran compartir y de su sensibilidad, o de las necesidades que se tengan en cada momento. Lo volveremos a mencionar en el Capítulo 7.

Representaremos las políticas usando una simplificación de ODRL, como ya se hizo en el análisis, ya que este es el modelo que utiliza el Dataspace Protocol para expresarlas. El modelo ODRL completo se muestra en la Figura 5.2, y nuestra simplificación en la Figura 5.3.

Para representar nuestro modelo conceptual hemos vuelto a utilizar un diagrama de clases. Del modelo ODRL hemos eliminado *Asset* y *Party* (y sus colecciones) porque no son necesarias. Recordamos que las políticas se elaboran como algo general, y que son las ofertas las que especifican después las políticas que se aplican para cada activo. El significado de cada atributo y relación se describe en más detalle en la Tabla 5.2.

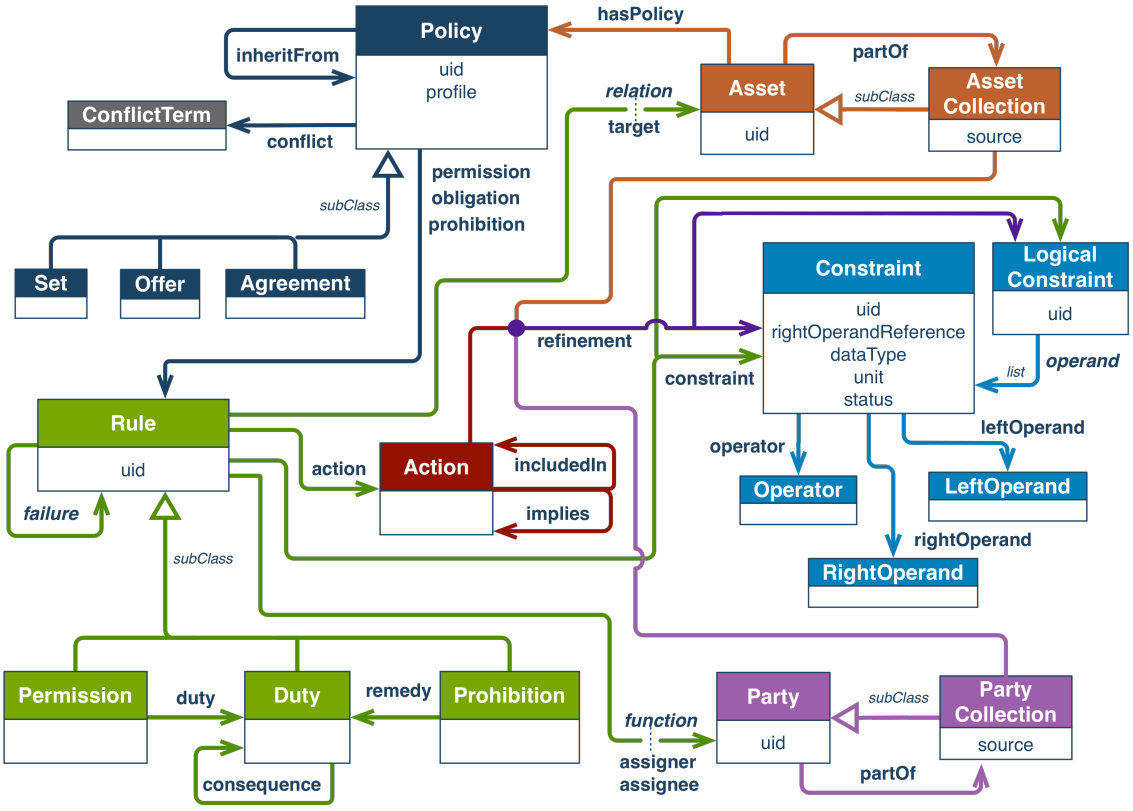


Figura 5.2: ODRL Information Model [57]

De acuerdo con el modelo de la Figura 5.3, nuestra única política de acceso podría expresarse como: una instancia de política que otorgue el permiso de usar (un activo determinado), con la restricción de *Credencial de Miembro == válida*.

[1]Con restricción adicional hacemos referencia a la API del repositorio App-PIMD. Para este trabajo solo utilizaremos métodos de acceso público de esta API, por lo que los datos que ofrezca el proveedor también serán públicos.

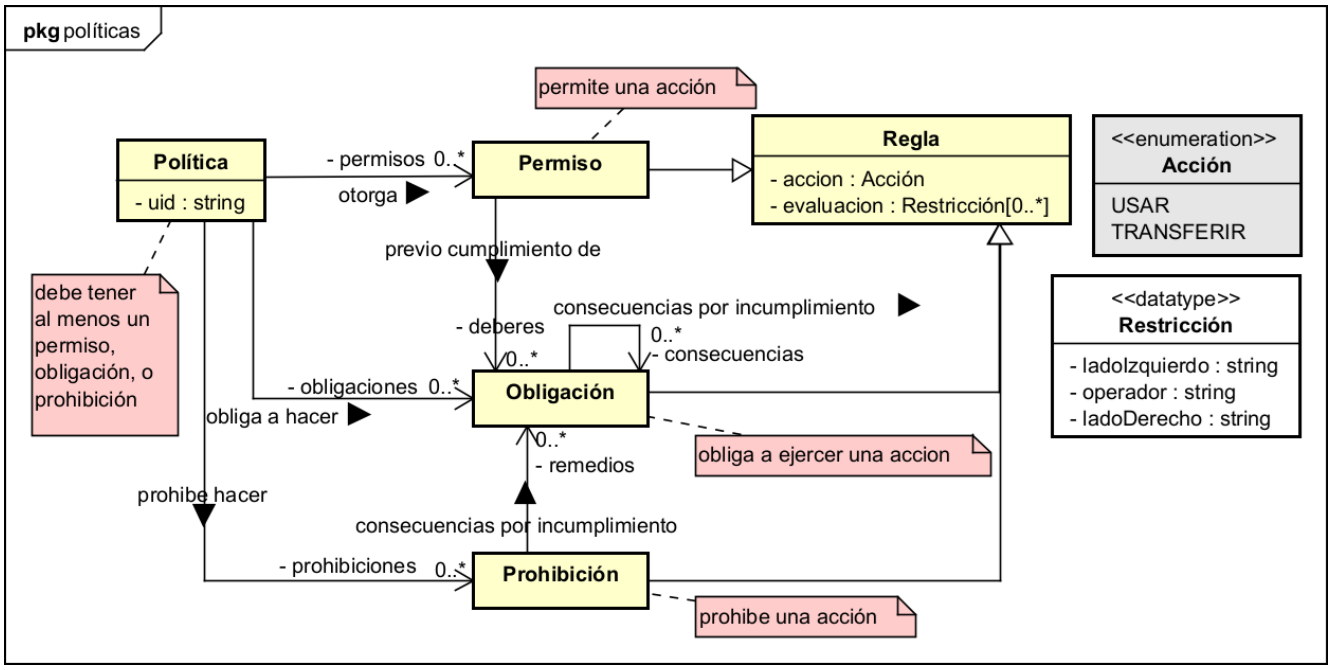


Figura 5.3: Modelo conceptual de las políticas: diagrama de clases

ENTIDAD	ATRIBUTO O RELACIÓN	DESCRIPCIÓN
Política	uid	Identifica de forma única a la política
	permisos	Conjunto de acciones que permite hacer la política
	obligaciones	Conjunto de acciones de obliga a hacer la política
	prohibiciones	Conjunto de acciones que prohíbe hacer la política
Regla	accion	Tipo de acción permitida / obligada / prohibida
	evaluacion	Expresión lógica que evalúa si se cumple la regla
Permiso	deberes	Conjunto de acciones de obliga a hacer para otorgar el permiso
Obligación	consecuencias	Conjunto de acciones de obliga a hacer en caso de incumplir la obligación
Prohibición	remedios	Conjunto de acciones de obliga a hacer en caso de incumplir la prohibición
Acción	USAR	Se refiere a un uso genérico
	TRANSFERIR	Se refiere a la transferencia a un tercero
Restricción	ladoIzquierdo	Lado izquierdo de la expresión lógica
	operador	Operador de la expresión lógica
	ladoDerecho	Lado derecho de la expresión lógica

Tabla 5.2: Modelo conceptual de las políticas: descripción de los atributos

5.2.2. Activos

Los activos son una parte imprescindible del proveedor, ya que son los encargados de describir cualquier tipo de dato que se puede compartir. La naturaleza de estos datos no está limitada a únicamente datos estáticos, pudiendo representar también flujos de datos (*data streams*, eventos), o incluso una serie de cálculos computacionales costosos.

Sin embargo nosotros solo compartiremos datos estáticos, que se alojan en última instancia en el *warehouse*. Para diseñar los activos, podemos pensar entonces en todos los tipos de consultas que nos gustaría ejecutar para compartir sus resultados. El *warehouse* aloja datos de aplicaciones, indicando los permisos que utiliza, y valorando el riesgo que supone cada uno para la privacidad con diferentes metodologías. Es decir, que la respuesta a nuestra pregunta son las aplicaciones, que son el objeto de interés central. También hay otras consultas que podrían aportar valor, pero las hemos decidido dejar como trabajo futuro. Un ejemplo sería las valoraciones de los riesgos para la privacidad de cada permiso, que se calculan con diferentes metodologías, y algunas son aportación propia desde la UVa.

Una vez decidido que vamos a compartir aplicaciones, también podemos decidir si compartirlas por separado, o agrupadas. Pero como no es excluyente hemos preferido hacerlo de ambas maneras. Es decir, definiremos dos categorías de activos. Por un lado los activos que representan **grupos de aplicaciones que comparten temática** (por ejemplo: activo de aplicaciones de comunicación, de banca, de educación, etc). Y por otro lado los activos que representan a una **aplicación individual** (por ejemplo: whatsapp, discord, duolingo, etc).

En cuanto al modelo conceptual de los activos, nos hemos inspirado en el Data Catalog Vocabulary (DCAT), de nuevo siguiendo el modelo que utiliza el Dataspace Protocol. Para representarlo hemos utilizado un diagrama de clases, que se muestra en la Figura 5.4, y también detallamos el significado de los atributos no triviales en la Tabla 5.3.

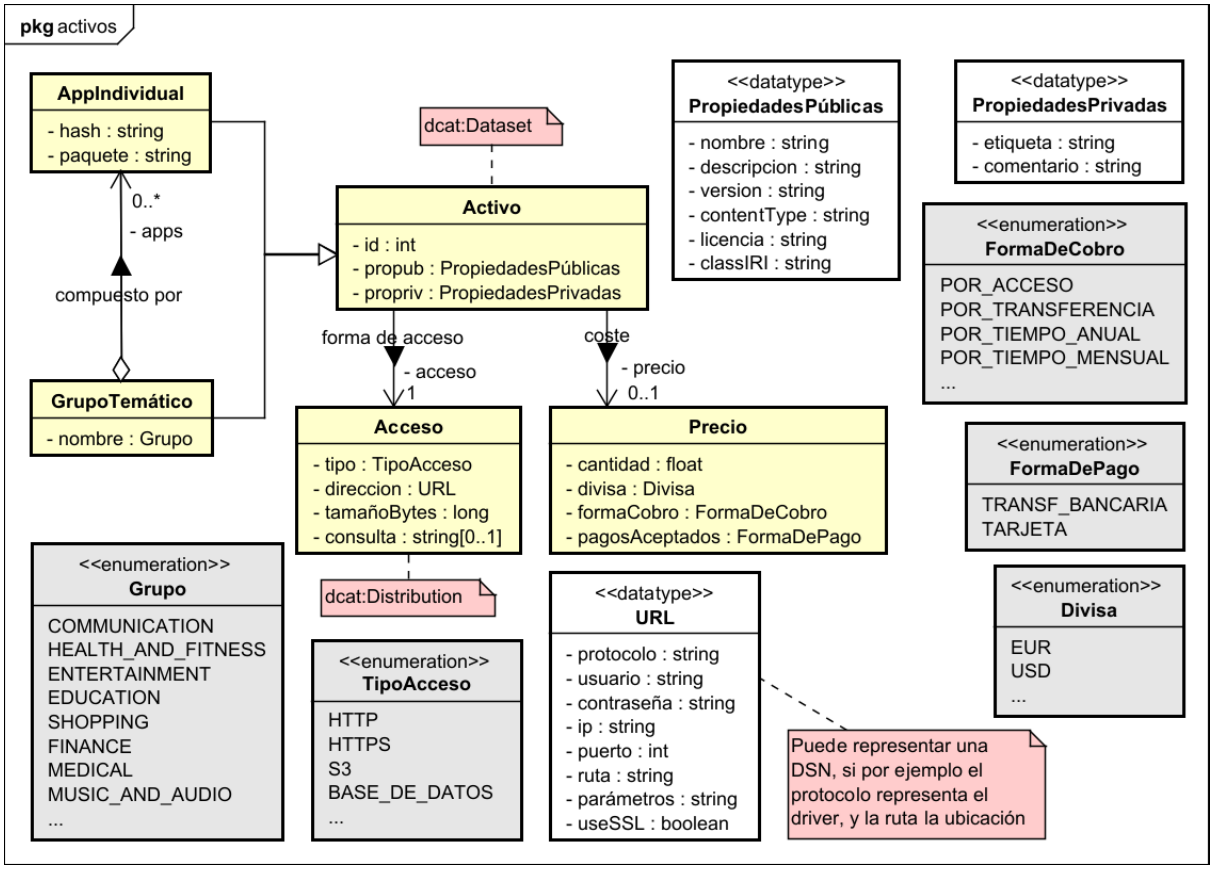


Figura 5.4: Modelo conceptual de los activos: diagrama de clases

La clase *Activo* se inspira en la clase *dcat:Dataset*, que representa a un conjunto de datos que se publica y hace disponible en una o varias distribuciones (*dcat:Distribution*). Las distribuciones en principio no contienen la información de acceso a los datos físicos, ya que esto se omite del catálogo. En su lugar, se ofrecerían las formas que soporta el proveedor para acceder a los datos (las mencionadas en análisis *push/pull* sobre HTTP, por ejemplo). El propósito de la clase *Acceso* es por tanto de uso exclusivo interno del proveedor, para localizar la ubicación física de los datos.

El consumidor tendrá acceso a estos activos a través del catálogo, donde estarán asociados a una política. Y como ya hemos mencionado, en ningún caso se le mostrará la ubicación real de los datos. También tenemos la opción de definir propiedades privadas para añadir información a los activos que tampoco se mostrará al consumidor.

ENTIDAD	ATRIBUTO O RELACIÓN	DESCRIPCIÓN
Activo	id	Identificador del activo
	proppub	Propiedades que se publicarán en el catálogo
	propriv	Propiedades internas para etiquetar los activos
	acceso	Detalla la ubicación física de los datos y su forma de acceso
	precio	Detalle el coste de acceder a los datos
AppIndividual	hash	Identifica de forma única la aplicación (hash SHA256 del apk)
	paquete	Nombre del paquete de la aplicación
GrupoTemático	nombre	Nombre de la categoría que agrupa a las aplicaciones
	apps	Aplicaciones que pertenecen al grupo
Acceso	tipo	Indica el tipo de acceso
	direccion	Ubicación física de los datos o de la máquina que los contiene
	tamañoBytes	Cantidad de bytes que ocupan los datos
	consulta	Recupera los datos, si se ubican en un SGBD
Precio	cantidad	Valor monetario
	divisa	Divisa en la que se expresa el precio
	formaCobro	Indica la forma en la que se cobra por el activo, ya sea por tiempo, o por cada acceso, u otro método
	pagosAceptados	Medios de pago aceptados para el cobro
Propiedades Públicas	nombre	Nombre de los datos
	descripcion	Descripción de los datos
	version	Versión actual de los datos
	contentType	Formatos en los que se puede entregar los datos
	licencia	Licencia con la que se ofrece a los datos
	classIRI	IRI de la clase RDF que describe a los datos

Tabla 5.3: Modelo conceptual de los activos: descripción de los atributos

5.2.3. Ontología

De acuerdo con Gruber [61, 62], una ontología es una especificación explícita de una conceptualización. Es decir, que una ontología define (especifica) los conceptos, relaciones y otras distinciones relevantes para modelar un dominio.

Las ontologías son uno de los estándares de la Web Semántica, donde tienen muchas aplicaciones. Algunas de ellas son compartir el conocimiento de una comunidad, permitir la interoperabilidad entre bases de datos, la búsqueda cruzada en múltiples bases de datos y la integración de servicios web [62]. En nuestro caso la ontología servirá para **compartir** metadatos de aplicaciones. Esto es un factor clave para el diseño, ya que podremos eliminar las entidades y relaciones que no sirvan a este propósito.

La ontología se basa en el modelo conceptual del *warehouse* (Figura 5.5), elaborado por Alejandro Pérez de la Fuente [63]. En su diagrama podemos observar 9 entidades, 3 de las cuales son débiles, y 2 especializaciones. También hay 10 relaciones, 5 de las cuales son identificativas, y una ternaria.

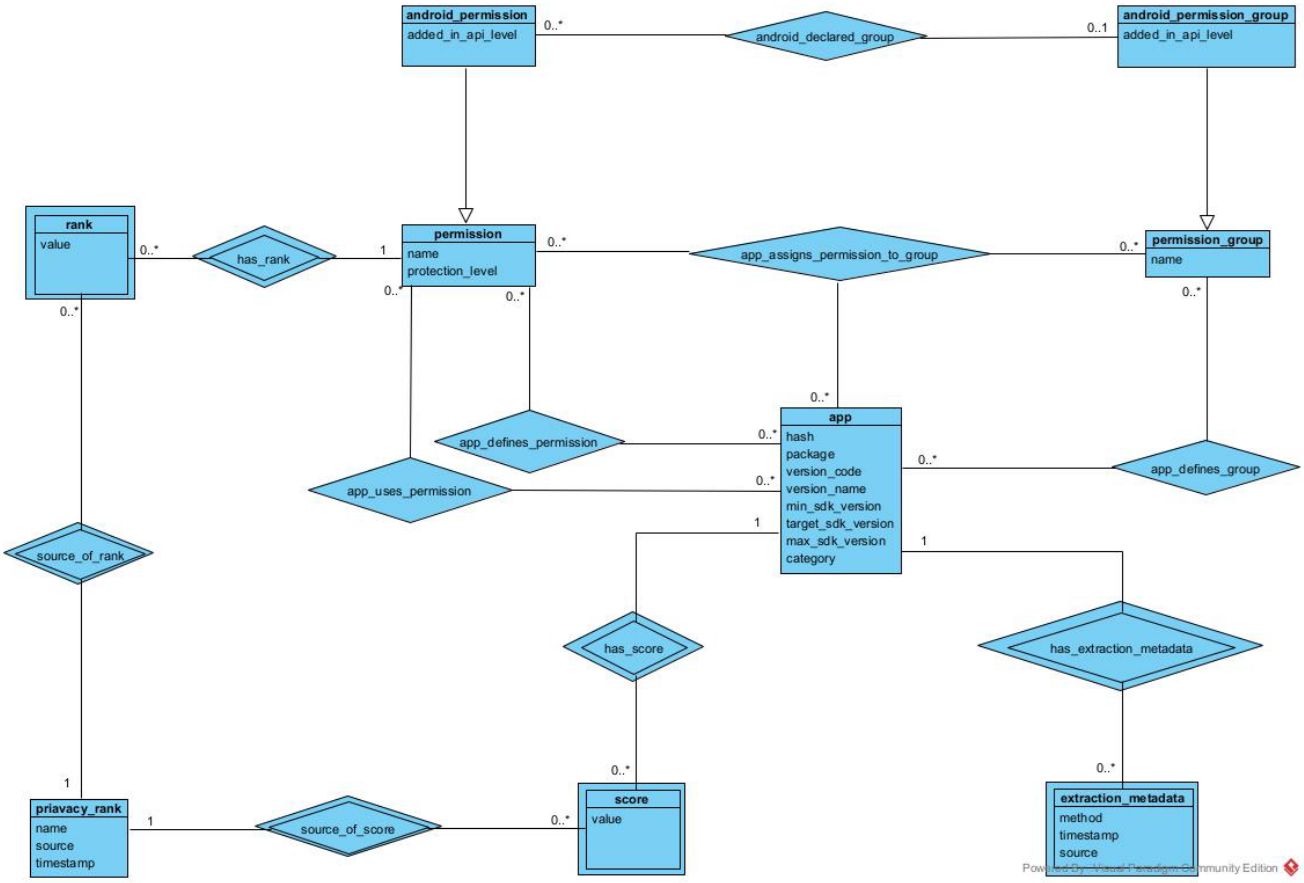


Figura 5.5: Diagrama conceptual de los datos del *warehouse* [63]

Hemos decidido usar OWL para la ontología, porque necesitaremos expresar restricciones de cardinalidad. Esta es la única característica de OWL que utilizaremos que está ausente en RDFS. Empezaremos describiendo las clases, con una taxonomía que mostramos en la Figura 5.6. Hemos omitido algunas flechas de *rdf:type* por que se pueden inferir indirectamente. Haremos lo mismo en el resto de diagramas, aunque usando el color para diferenciar los tipos. Todos los hemos elaborado con *draw.io* [6].

Las 4 entidades principales de nuestra ontología se corresponden con las 4 entidades fuertes y genéricas del modelo conceptual del *warehouse*, a saber, *app*, *permission*, *permission_group*, y *privacy_rank*[2], que renombraremos respectivamente como *App*, *Permission*, *PermissionGroup* y *ScoringSystem*.

[2]privacy_rank es una métrica que mide el nivel de intrusión de cada permiso [63]. Esta entidad es por tanto usada de referencia para puntuar tanto permisos como aplicaciones.

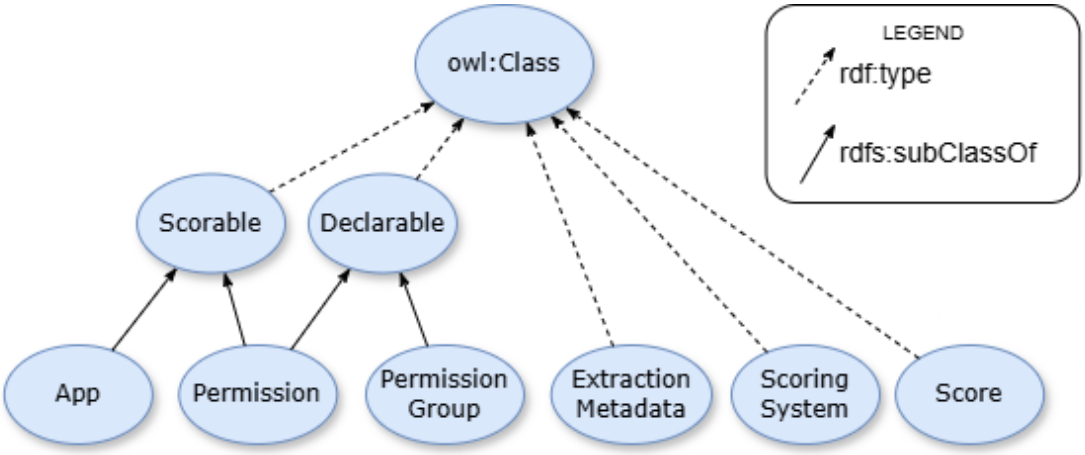


Figura 5.6: Taxonomía de clases: grafo RDF

Incluiremos también dos clases adicionales: *Score*, que reúne el significado semántico de las entidades débiles *score* y *rank*. Esta clase representa la puntuación de aplicaciones y permisos, usando un sistema de puntuación particular. Y la clase *ExtractionMetadata*, que se corresponde con la entidad débil *extraction_metadata*.

Las dos superclases que incluimos en nuestra ontología son *Scorable* y *Declarable*, que representan cosas puntuables, y cosas declarables en el *Android mainifest*. Las dos clases puntuables son *App* y *Permission*, cuya puntuación sería una instancia de *Score*. Y las dos clases declarables son *Permission* y *PermissionGroup*, que son declarados por una *App*.

Y en cuanto a las relaciones, nos interesarán solo las que sean relevantes desde la perspectiva de una *App*, por lo que excluiremos las dos relaciones entre permisos y grupos de permisos. Como hemos decidido utilizar OWL, podemos usar *owl:ObjectProperty* para describir las propiedades que relacionan objetos en vez de estar limitados a *rdfs:Property*. En la Figura 5.7 mostramos también con una taxonomía todas las propiedades (de objeto) de la ontología.

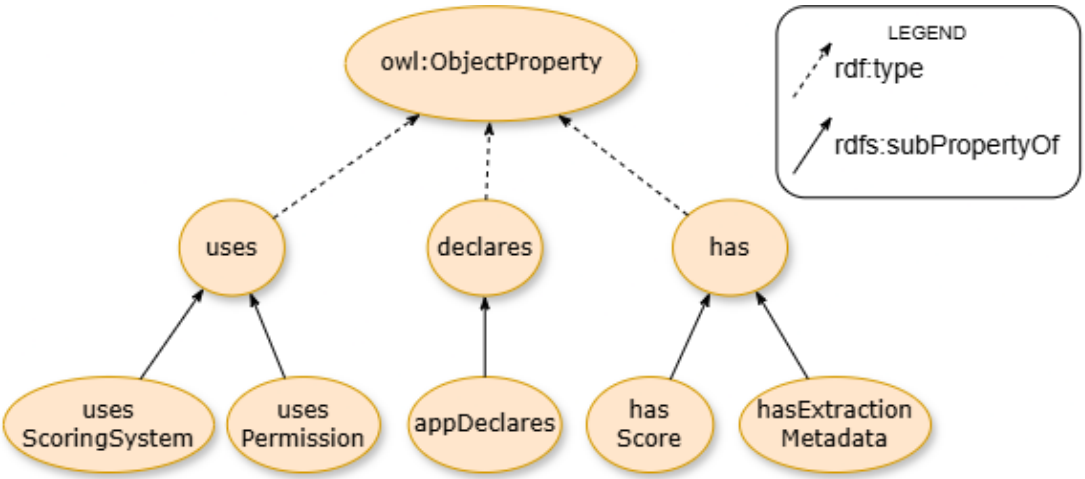


Figura 5.7: Taxonomía de propiedades de objeto: grafo RDF

uses es la propiedad general que representa uso, que se especializa en *usesScoringSystem* y *usesPermission*. Estas propiedades representan las relaciones *app_uses_permission*, *source_of_rank* y *source_of_score*.

has es la propiedad general que representa tenencia, que se especializa en *hasScore* y *hasExtractionMetadata*. La primera representa a las relaciones *has_rank* y *has_score*, mientras que la segunda a la relación *has_extraction_metadata*.

declares es la propiedad que representa definición, que solo tiene una especialización: *appDeclares*. Esta propiedad representa a las relaciones *app_defines_permission* y *app_defines_group*.

Habiendo ya descrito todas las clases, propiedades de objetos, y sus significados, podemos elaborar entonces un grafo RDF completo de la ontología. De este diagrama solo se excluyen los atributos (*owl:DatatypeProperty*), y la restricción, que detallaremos a continuación. Hemos utilizado el color azul para las clases y el color naranja para las propiedades. El modelo se muestra en la Figura 5.8.

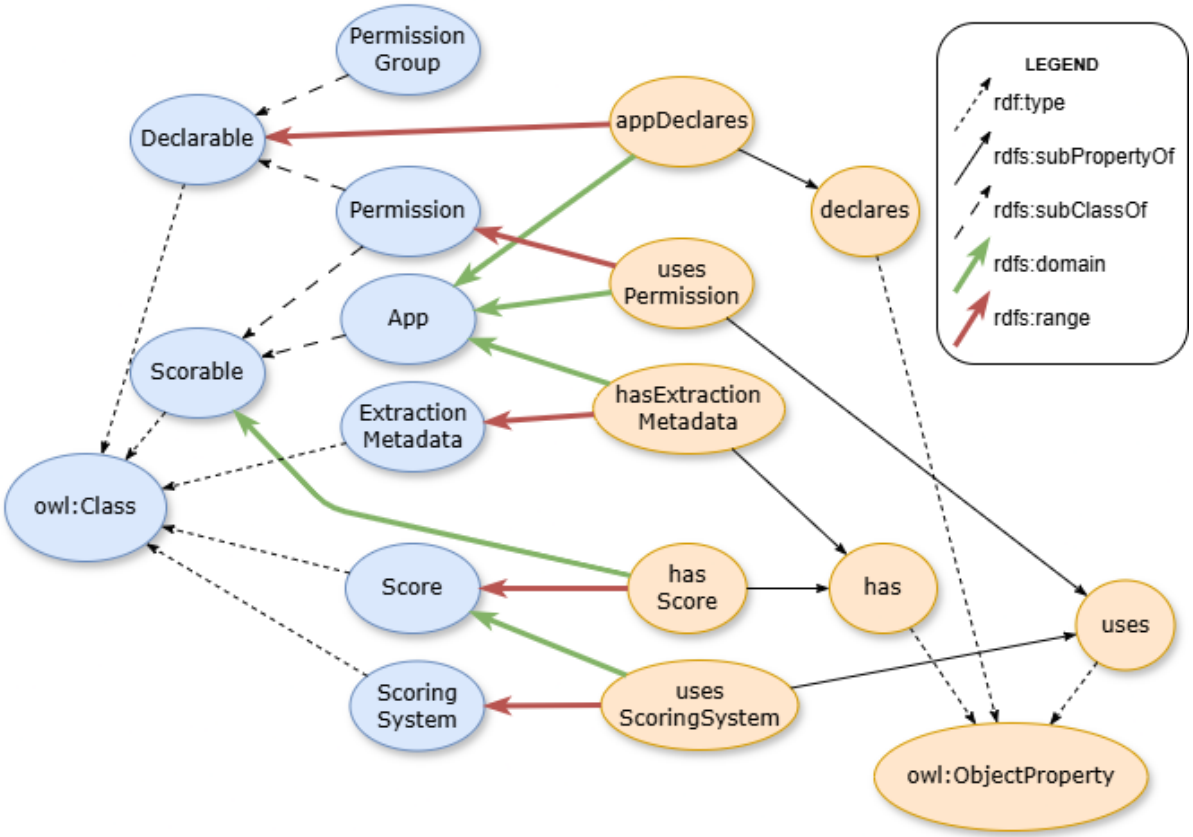


Figura 5.8: Ontología completa: grafo RDF

La única restricción que hay en la ontología es sobre *Score*, porque cada instancia de esta clase debe estar siempre relacionada con exactamente un sistema de puntuación. El diseño de esta restricción se muestra en la Figura 5.9.

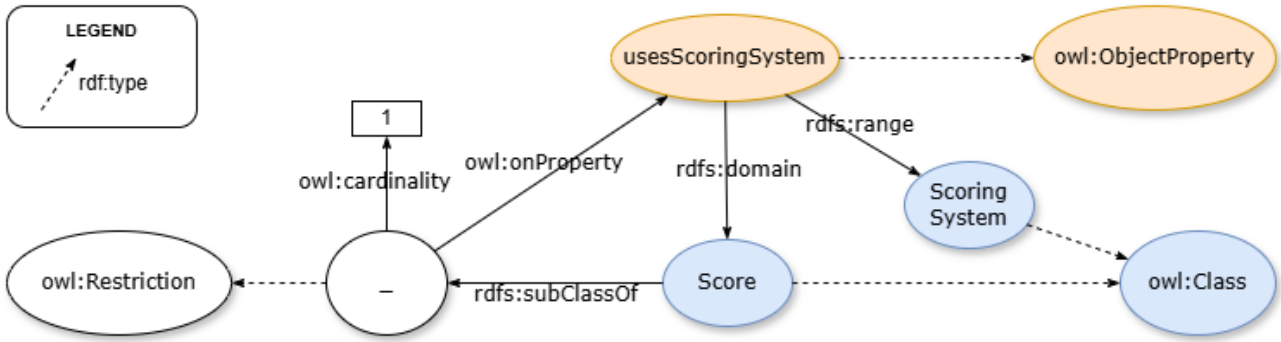


Figura 5.9: Restricción sobre *Score*: grafo RDF

Sobre los atributos, mostramos sus definiciones en la Tabla 5.4. La tabla representa tripletas RDF, donde cada fila representa a una misma *owl:DatatypeProperty* (sujeto), el nombre de la columna indica el predicado, y el contenido de cada celda el objeto.

rdfs:label	rdfs:domain	rdfs:range
hash	App	xsd:string
package		xsd:string
category		xsd:string
versionCode		xsd:integer
versionName		xsd:string
minSDKVersion		xsd:integer
targetSDKVersion		xsd:integer
maxSDKVersion		xsd:integer
extractionSource		xsd:string
extractionMethod		xsd:string
extractionTimestamp		xsd:dateTime

(a) Propiedades cuyo dominio es App

rdfs:label	rdfs:domain	rdfs:range
name	Scoring System	xsd:string
source		xsd:string
timestamp		xsd:dateTime
value	Score	xsd:float
name	Permission Group	xsd:string
addedAPILevel		xsd:string
name	Permission	xsd:string
protectionLevel		xsd:string
addedAPILevel		xsd:string

(b) Resto de dominios

Tabla 5.4: Definición de las propiedades de datos de la ontología

5.3. Contexto tecnológico

Para elaborar el diseño del proveedor hemos elegido utilizar los componentes de EDC e INESData. Esta decisión nos condiciona a usar otras tecnologías y estándares que describiremos brevemente en esta sección. Para ver más en detalle el contexto de EDC e INESData, nos referimos a la Sección 3.4: Implementaciones donde contamos más información sobre ellas.

Hemos organizado estas tecnologías por categorías para facilitar su compresión. Empezaremos describiendo los estándares, para después ubicarlos en el contexto en el que se utilizan (protocolos y componentes). Como resumen, las tecnologías que se usarán se describen a continuación:

- **Web Semántica y Representación de Datos**: JSON-LD.
- **Identidad Digital**: Verifiable Credentials (VCs), Decentralized Identifiers (DIDs), JSON Web Token (JWT).
- **Protocolos**: Dataspace Protocol (DSP), Decentralized Claims Protocol (DCP), Open Authorization 2.0 (OAuth2).
- **Componentes**:
  - EDC: Control Plane, Data Plane, Identity Hub.
  - INESData: Connector, Connector Interface.
  - Otros: *Warehouse*, PostgreSQL Database, Hashicorp Vault, Servidor de autorización OAuth2.

5.3.1. Web semántica y representación de datos

La web semántica es una extensión de la web tradicional donde los datos tienen un modelado semántico. De esta manera que las máquinas puede entender, compartir e interpretar la información de forma más inteligente. Estos modelos semánticos se basan en estándares como RDF, RDFS y OWL para definir significados y relaciones entre los datos, permitiendo una mejor interoperabilidad y automatización [60].

JSON-LD Estándar del W3C basado en JSON para la serialización de modelos semánticos [64]. El Dataspace Protocol (DSP), y los componentes de EDC utilizan JSON-LD como estándar de serialización.

5.3.2. Identidad digital

Verifiable Credentials (VCs) Las Verifiable Credentials (VCs) son un estándar desarrollado por el W3C para la emisión, presentación y verificación de credenciales digitales de manera segura, verificable y descentralizada. Estas credenciales permiten representar información verificable sobre una entidad en un formato interoperable, sin depender de una autoridad centralizada para su validación [58].

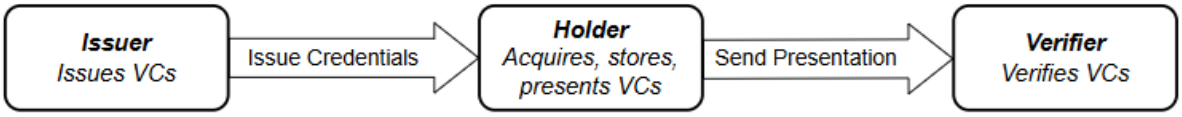


Figura 5.10: Roles y flujo de información en el VC Data Model [58]

Una VC es un conjunto de declaraciones (*claims*) y otros metadatos que criptográficamente pueden demostrar quién es el *emisor*, y que están libres de manipulaciones. Un *tenedor* de VCs puede crear Verifiable Presentations (VPs) reuniendo declaraciones para presentarlas ante un *verificador*. Normalmente las VPs tendrán un periodo de vida corto.

La forma preferida de representación para las VCs y VPs es JSON-LD.

Decentralized Identifiers (DIDs) Los DIDs son un nuevo tipo de identificador que posibilita una identidad digital verificable y descentralizada. Los DIDs están asociados a una entidad (por ejemplo: organizaciones, personas o documentos) y la identifican. A diferencia de los identificadores típicos federados, los DIDs han sido diseñados para poder estar desacoplados de registros centralizados, proveedores de identidad y autoridades de certificación [65].

Los DIDs son URIs que asocian a un sujeto de un DID con un documento DID, permitiendo interacciones confiables relacionadas con dicho sujeto. En el contexto de las VCs, se utilizan para identificar a los sujetos de las declaraciones en las credenciales [58, 65].

El Identity Hub de EDC utiliza específicamente *did:web*, que es un método que se basa en servidores web y DNS existentes para la resolución de los documentos DID [66].

JSON Web Token (JWT) JSON Web Token (JWT) es un formato compacto de representación de declaraciones (*claims*) que se transmiten entre dos partes. Permite ser firmado o encriptado para garantizar su integridad [67]. Se pueden utilizar para transmitir las credenciales de un usuario, junto con sus permisos/rol y autorizar el uso o acceso a un recurso.

5.3.3. Protocolos

Dataspace Protocol (DSP) El protocolo de espacios de datos es un estándar desarrollado por la IDSA, del cual ya se ha hablado en profundidad en la Subsección 3.2.2: Dataspace Protocol (DSP).

La interacción entre los participantes en un espacio de datos se lleva a cabo a través de agentes participantes (conectores). Aunque la mayoría de las interacciones ocurren entre conectores, también se requieren interacciones con otros sistemas como un proveedor de identidad [68].

El proveedor de identidad es el encargado de proporcionar la información requerida para garantizar la confianza en el espacio de datos. La validación de la identidad de un agente participante y la verificación de declaraciones (*claims*) adicionales son mecanismos fundamentales. Este protocolo no establece ni la estructura ni el contenido de las identidades y declaraciones, que pueden ser distintas en cada espacio de datos [68].

Decentralised Claims Protocol (DCP) El DCP es un protocolo desarrollado por eclipse y supone una extensión del DSP. El DCP admite el uso múltiples anclas de confianza (*trust anchors*) y permite a cada participante gestionar y verificar las presentaciones sin necesidad de recurrir a sistemas externos fuera de su control [69]. Su alcance es el siguiente:

- Especificar un formato para tokens de identidad autoemitidos.
- Definir un protocolo para almacenar y presentar VCs y otros recursos de identidad.
- Definir un protocolo para que las partes soliciten credenciales a un emisor de credenciales.

Open Authorization 2.0 (OAuth2) OAuth2 es un protocolo de autorización diseñado para permitir a aplicaciones de terceros acceder a recursos de un usuario en otra aplicación. Es un estandar propuesto por el IETF en el RFC 6749 [70].

Los componentes de EDC permiten usar OAuth2 para acceder a sus APIs, característica que utiliza el conector de INESData [71]. En concreto se utilizan tokens de acceso en el formato JWT para autorizar a los administradores o a otros participantes del espacio de datos.

5.3.4. Componentes

Los componentes de EDC (y el conector de INESData) están implementados en **Java**, en concreto con el JDK 17. En esta sección damos una **explicación genérica de los componentes que utilizaremos**, explicando las tareas que realizan, y las interfaces que debemos utilizar. Separaremos estos componentes en tres categorías: EDC, INESData y Otros. En la Sección 5.7: Diseño detallado entramos en detalles más técnicos sobre EDC.

EDC

EDC es un *framework* que proporciona implementaciones de componentes con una serie de funcionalidades básicas. Es decir, que no se adaptan a ningún caso de uso particular. Por tanto es responsabilidad de las organizaciones que utilizan sus componentes, la de adaptarlos a sus necesidades, programando en su caso extensiones utilizando su modelo de extensión[3] [28].

EDC recomienda que el conector se separe en dos (plano de control y plano de datos) para que puedan ser gestionados y escalados por separado. En la Figura 5.11 se muestra esta división junto con la comunicación entre los planos del proveedor y consumidor.

Cada componente de EDC tiene una identidad, siendo posible desplegarlos en múltiples entornos de ejecución según las necesidades de escalabilidad requeridas. Los componentes que utilizaremos son:

- **Control Plane**: División del conector que se encarga de: recopilar catálogos, crear contratos, administrar transferencias de datos y monitorizar el cumplimiento de las políticas de uso. Para administrar el plano de control se debe utilizar la Management API[4]. Los planos de control del consumidor y proveedor se comunicarán usando el Dataspace Protocol (DSP).

[3] En la Sección 5.7: Diseño detallado se entra en más detalle sobre como se crean extensiones en EDC.

[4] La Management API es una interfaz RESTful que permite gestionar el plano de control. A través de esta API se crean los activos y las políticas, o se administran los contratos y procesos de transferencia activos. La documentación completa de esta API se puede encontrar en este enlace.

- **Data Plane**: División del conector que se encarga de las transferencias de datos. Es administrado por el plano de control, usando la Data Plane Signaling API (DPS)[5]. También expone una API pública que permite el acceso a datos del formato *HttpData-PULL*, para lo cual emite tokens de acceso no renovables.

- **Identity Hub**: Almacena y administra de forma segura VCs, incluida su presentación y su proceso de emisión y reemisión. Además administra las llaves públicas y privadas, y los documentos DIDs. Se basa en el DCP, permitiendo el uso de un modelo de identidad descentralizado. Un único Identity Hub puede almacenar las VCs para diferentes espacios de datos y diferentes conectores dentro de una misma organización. Para gestionar el componente se debe usar la Identity API, que permite crear y modificar las identidades (contextos), y crear y modificar pares de claves.

Para autorizar el acceso a la Management API y a la Identity API EDC actualmente permite utilizar una clave de acceso o un proveedor externo de OAuth2. Es muy importante que niguna de estas APIs se exponga a al red pública.

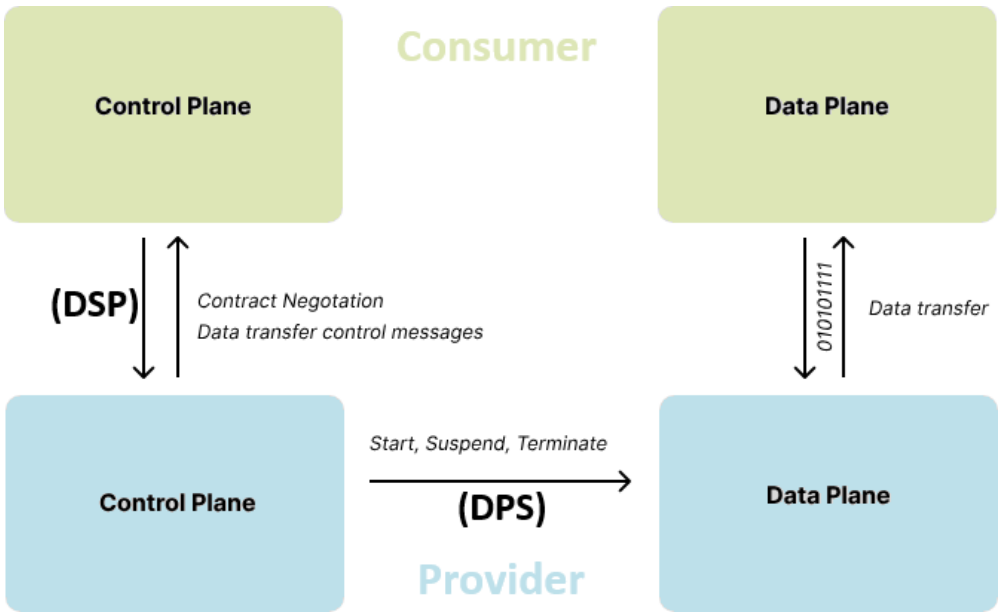


Figura 5.11: EDC Connector: Plano de control y plano de datos, adaptado de [28]

Algunos de los principios arquitectónicos fundamentales de los componentes de EDC son los siguientes [28]:

- **Asincronía**: Todas las modificaciones externas a las estructuras de datos internas son asíncronas.
- **Procesamiento en un solo hilo**: El plano de control está diseñado en torno a un conjunto de máquinas de estado secuenciales que emplean bloqueo pesimista para evitar condiciones de carrera y otros problemas.
- **Idempotencia**: Las solicitudes que no provocan una modificación son idempotentes. Lo mismo aplica cuando se aprovisionan recursos externos.
- **Tolerancia a errores**: El diseño del plano de control prioriza la corrección y la fiabilidad sobre la baja latencia. Esto significa que, incluso si un socio de comunicación no está disponible debido a un error transitorio, el sistema está diseñado para manejar ese error e intentar superarlo.

[5]Se puede consultar mas información sobre la DPS API en el siguiente enlace.

INESData

En su misión de promover e implementar espacios de datos propios, INESData desarrolla una gran variedad de componentes. Sus funciones y diseño se detallan en su entregable: *E5. Componentes horizontales para espacios de datos* [71].

Para diseñar nuestro proveedor utilizaremos su conector y su interfaz gráfica. Un cambio importante del conector de INESData con respecto al de EDC es que no utiliza el Identity Hub para verificar la identidad de otros participantes, sino el protocolo OAuth2. Para autorizar el acceso a la Management API también utiliza OAuth2.

- **Connector**: Se basa en el conector de EDC y agrupa en un único componente el plano de control y el plano de datos. Implementa funcionalidades adicionales como el almacenamiento de activos en los servidores de INESData, un nuevo servicio de registro (*logs*), la gestión de vocabularios o la búsqueda por texto entre las propiedades de los activos.

- **Connector Interface**: Según [71]:

  > Es la interfaz gráfica de usuario (GUI) desarrollada en Angular para la gestión del conector INESData. Utiliza las API del EDC para mostrar las funcionalidades del conector, como la creación y gestión de assets, catálogo y políticas de acceso a los datos. Facilita una experiencia de usuario integrada que permite interactuar con el conector de manera visual e intuitiva.

Otros

Utilizamos tecnologías específicas en vez de un SGBD relacional genérico o un gestor de secretos genérico porque EDC proporciona ls módulos necesarios para utilizar estas en particular. Sería posible usar otros proveedores de estas tecnologías, pero el diseño debería ampliarse para incluir nuevas extensiones que admitan su uso. En el caso del servidor de autorización, no es importante la aplicación específica, sino el estándar JWT y el protocolo OAuth2.

- ***Warehouse***: Elaborado por Alejandro Pérez de la Fuente en su TFG [63]. Para poder realizar consultas tenemos disponible la API de App-PIMD, por ejemplo consultando las aplicaciones individualmente por su hash, o nombre de paquete. Una versión actualizada de la documentación de la API se puede consultar en este enlace.

- **PostgreSQL Database**: SGBD relacional de código abierto. Utilizado por el EDC Control Plane y por el conector de INESData para la persistencia de activos, políticas, ofertas, contratos, etc.

- **Hashicorp Vault**: Herramienta de gestión de secretos y protección de datos diseñada para almacenar, controlar y acceder de manera segura a credenciales, pares de claves, claves API, certificados y otros secretos en entornos distribuidos. Usado por el EDC Identity Hub y por el conector de EDC e INESData.

- **Servidor de Autorización OAuth2**: Componente que forma parte del *framework* de autorización OAuth2. Emite los tokens de acceso para una aplicación tras verificar la identidad del usuario.

5.4. Adaptación de los datos a los estándares de EDC e INESData

Credenciales en EDC

El componente de EDC responsable de gestionar los recursos de identidad es el Identity Hub, que en el caso de las credenciales utiliza el estándar **W3C Verifiable Credentials** (VCs), y para los identificadores **W3C Decentralized Identifiers** (DIDs), en particular *did:web*.

Nuestro modelo para las credenciales **incluye toda la información mínima** que requieren las VCs, y por tanto solo necesitaremos cambiar el nombre de las propiedades, y reorganizarlas de forma apropiada. Nuestro modelo de credenciales solo admite hacer una declaración sobre un solo sujeto, que además coincide con el tenedor. Las VCs en cambio no tienen esta restricción, ya que cada declaración (*claims*, contenidas en la propiedad *credentialSubject*) debe indicar al sujeto al que hace referencia. Es decir, que los modelos son compatibles, al ser las VCs un modelo más general. Las credenciales se deben serializar usando JSON-LD.

Otro estándar que es relevante para las credenciales es el **did:web**. Esto quiere decir que deberemos crear un identificador para nuestro proveedor, que al resolverse como si fuera una URL de la web, nos proporcione un **documento DID**. Estos documentos se alojan en el Identity Hub y deben contener los puntos de acceso para el conector, el servicio de credenciales y las llaves públicas. El Identity Hub se encarga de crear estos documentos, por lo que nosotros solo debemos crear los contextos de los participantes con la Identity API.

Credenciales en INESData

Las credenciales utilizando los componentes de INESData requieren una aclaración adicional, ya que no se utiliza directamente el concepto de credencial. En su lugar se utilizan tokens de acceso JWT siguiendo el protocolo OAuth2. Los JWT alojan declaraciones (*claims*) por lo que también nos sirve nuestro modelo de credenciales.

Políticas de acceso y uso

Nuestro modelo es compatible con el de EDC e INESData porque también utilizan ODRL (al implementar el DSP). Se deben serializar usando JSON-LD.

Activos

EDC implementa los activos con una única clase, y por tanto **no permite utilizar especializaciones** como las que tenemos nosotros. Si podemos en cambio definir todas las propiedades adicionales que queramos, mientras estén bien documentadas. Para esto usaremos una **extensión a nuestra ontología** que comentaremos en la siguiente subsección.

Estas nuevas propiedades se deben incluir en la propiedad *edc:properties*[6]. La clase Acceso se corresponde a la propiedad *edc:dataAddress*, que ya define las propiedades necesarias para que podamos utilizarla sin problema. Los activos también se serializan con JSON-LD.

[6]El prefijo *edc* hace referencia al espacio de nombres con la IRI: https://w3id.org/edc/v0.0.1/ns/.

Ontología

La ontología **no necesita ser adaptada**, ya que no depende de ninguna tecnología. Sin embargo si que **necesitamos ampliarla** para poder documentar los activos adecuadamente.

Definiremos dos nuevas propiedades de objeto, cuyo rango son dos clases enumeradas. La propiedad *assetCategory* indica la categoría de un activo, y la propiedad *thematicGroupName* indica el grupo temático de un activo cuya categoría es *THEMATIC_GROUP*, aunque esta última restricción no está incluida en el modelo. La definición de estas dos propiedades se muestra en la Figura 5.12.

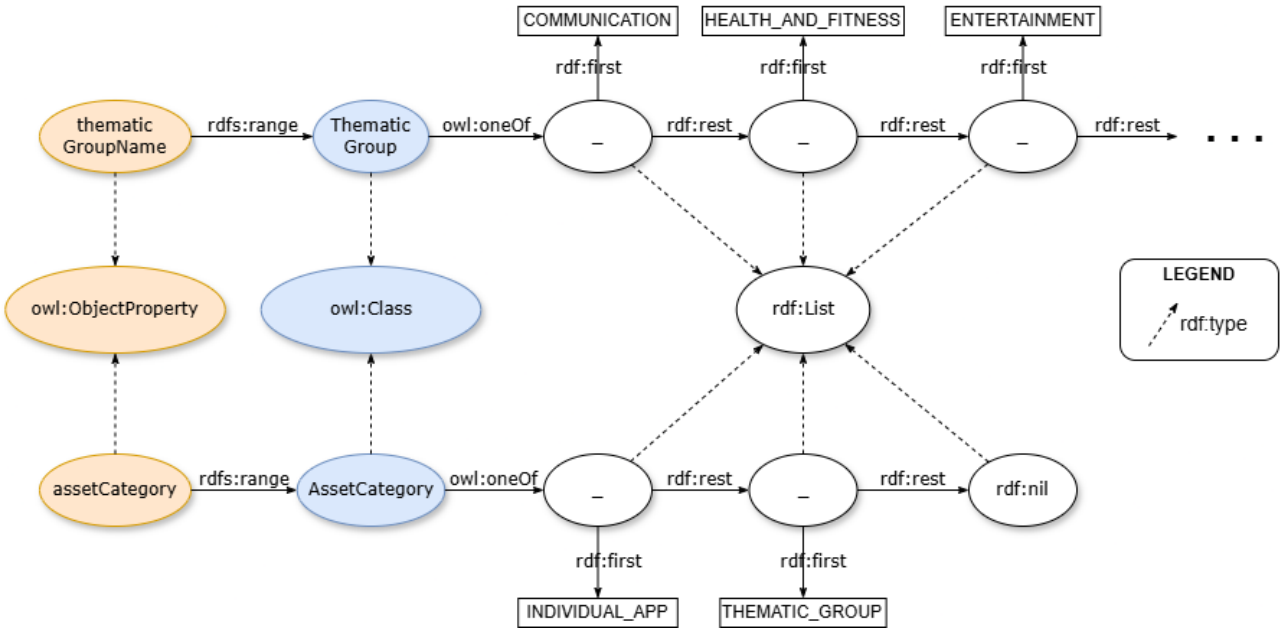


Figura 5.12: Extensión a la ontología: grafo RDF

No es necesario definir propiedades para el id, nombre, descripción, tipo de contenido o versión, porque podemos utilizar las propiedades de EDC, definidas en su espacio de nombres. Tampoco crearemos nuevas propiedades para expresar el precio porque en la práctica no lo podremos utilizar.

5.5. Arquitectura lógica

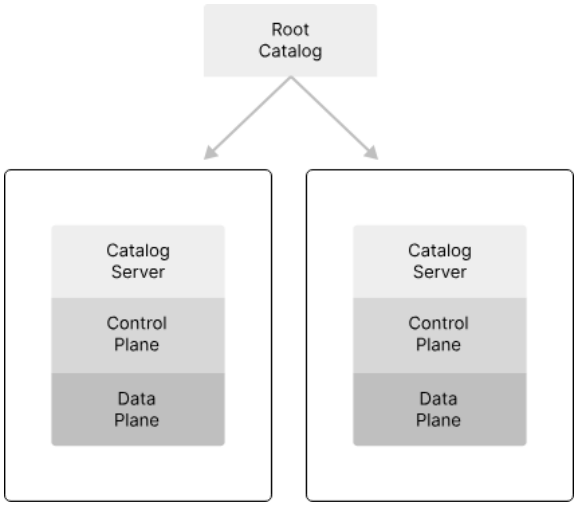
En esta sección elaboramos el diseño de la **arquitectura lógica de dos proveedores**, uno para cada tecnología. Aunque la arquitectura de ambos proveedores es muy parecida, no son exactamente compatibles y por este motivo creemos que es preferible exponer los dos.

La arquitectura de los proveedores sigue el **patrón de microservicios**. Cada servicio se corresponde a uno de los componentes que hemos descrito en la Subsección 5.3.4: Componentes, donde serán independientes unos de otros.

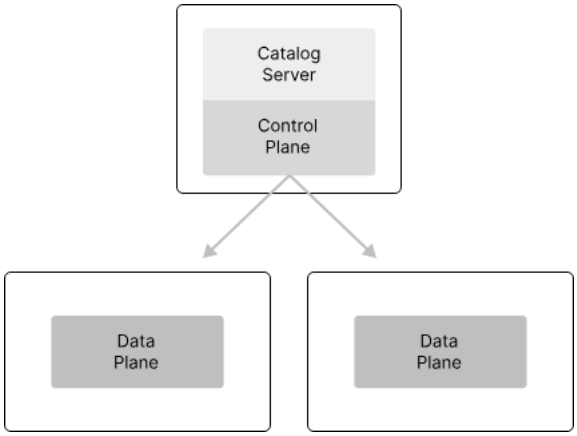
Una consideración previa antes de diseñar los proveedores, es la de tener en cuenta a la organización donde se integrarán. Hay que recordar que una misma infraestructura permite colaborar en varios espacios de datos, o que podría haber varios departamentos interesados en colaborar en espacios de datos. Las posibles soluciones a esta cuestión, y la que hemos elegido, se detallan en la Subsección 5.5.1. Después, en la Subsección 5.5.2 y en la Subsección 5.5.3 describimos el diseño de cada proveedor y elaboramos para cada uno un diagrama de componentes.

5.5.1. Topología

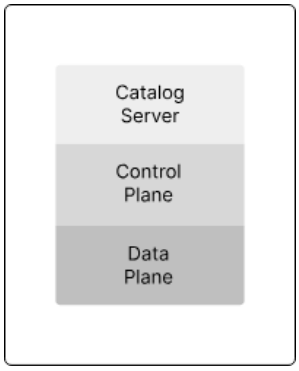
El diseño del proveedor está condicionado por las necesidades de la organización que lo opera. En este caso podríamos pensar en la Universidad de Valladolid, en su Escuela de Ingeniería Informática, o en su Departamento de Informática. EDC por ejemplo, sugiere varias opciones sobre las topologías de diseño posibles, que llama dominios de gestión (*management domains*) [28]. Tres de estas opciones se muestran en la Figura 5.13, donde cada recuadro muestra un dominio de gestión independiente.



(a) Dominio distribuido



(b) Dominio distribuido con catálogo y Control Plane centralizado



(c) Dominio único

Figura 5.13: Topologías de dominios de gestión [28]

El diseño del proveedor podría contemplar por ejemplo un servidor de catálogo raíz y un proveedor de identidad para toda la organización, permitiendo así que cada facultad o departamento operen sus propios servidores de catálogo y conectores independientes (Figura 5.13a). Usando otra topología, la organización podría tener un único servidor de catálogo y Control Plane, permitiendo a cada departamento tener solo su propio Data Plane (Figura 5.13b). De esta forma la negociación de contratos estaría centralizada dentro de la organización, mientras que de la primera forma cada departamento podría definir sus propias políticas, y negociar sus propios contratos.

En nuestro caso diseñaremos un proveedor con solo un conector, usando un dominio de gestión único, sin tener en cuenta otros departamentos (Figura 5.13c).

5.5.2. Proveedor con EDC

El diseño con EDC se corresponde al primer incremento de la planificación, donde todavía no incluimos una interfaz gráfica. El administrador interactuará con el proveedor mandado peticiones HTTP a sus dos APIs asignadas[7]. Es importante notar que los componentes de EDC no soportan HTTPS, y por el riesgo a la seguridad que supone recomiendan no exponer estas APIs a redes no seguras. Para simplificar el diseño de este proveedor para autorizarnos utilizaremos una *API key*, en vez de un proveedor externo de OAuth2.

En la Subsección 5.3.4: Componentes ya hemos mencionado las APIs que expone cada componente y el papel que desempeñan en el sistema. En la Figura 5.14 detallamos a mayores las relaciones entre los componentes y las comunicaciones con elementos externos. Hemos usado el color azul para mostrar las interfaces a las que accede el administrador del proveedor, y el morado para mostrar las interfaces a las que accederían los consumidores.

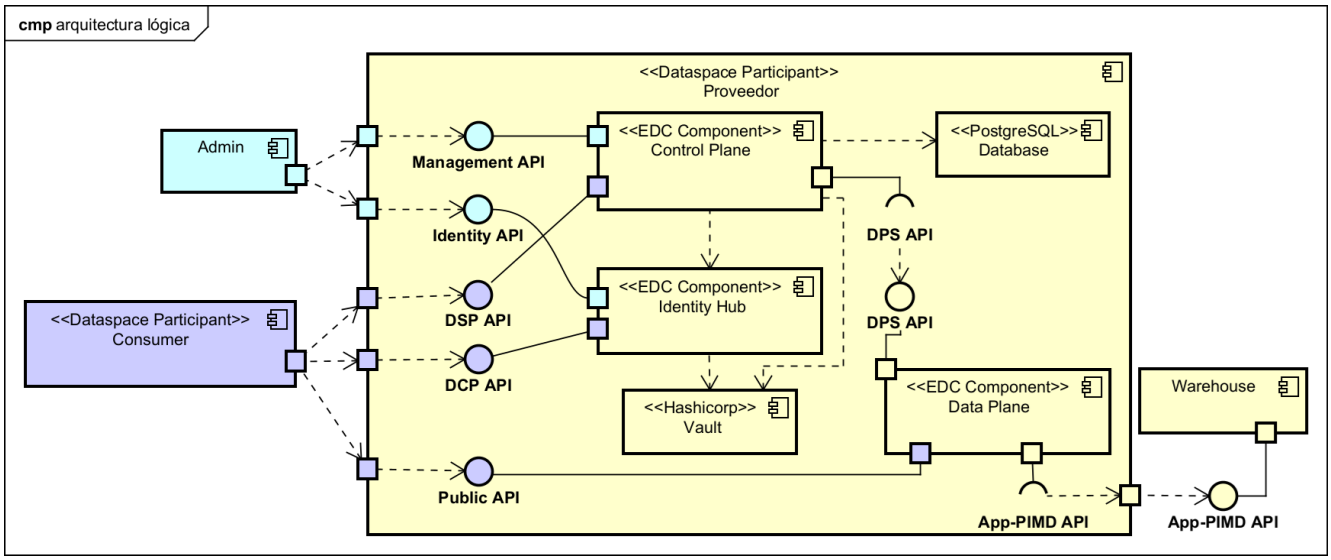


Figura 5.14: Diagrama de componentes: proveedor con EDC

Nuestro proveedor estará formado por los siguientes cinco componentes: EDC Identity Hub, EDC Control Plane, EDC Data Plane, PostgreSQL Database y Hashicorp Vault. Mientras que los componentes externos serán: Administrador, Consumidor y *Warehouse*.

Hay varias implementaciones posibles para el EDC Control Plane, en concreto para personalizar la persistencia de los activos, políticas y ofertas (llamados contratos en el lenguaje de EDC). La implementación por defecto persiste a estas entidades en memoria, pero es posible utilizar una base de datos PostgreSQL incluyendo las extensiones apropiadas. Nuestro diseño usa la segunda opción.

De la misma manera, el EDC Identity Hub también está preparado para utilizar una Hashicorp Vault para almacenar todos los secretos, aunque es necesario incluir extensiones adicionales.

También tenemos que tener en cuenta que para poder usar la API de App-PIMD con el EDC Data Plane **necesitamos programar una extensión**, como ya nos advierte EDC de esta posibilidad. Los detalles sobre la extensión se describen con más detalle en la Subsección 5.7.4. También tenemos la posibilidad de añadir soporte a más protocolos de transferencia de datos a los consumidores. EDC implementa por defecto HTTP, transferencias basadas en S3, y Kafka. Nosotros solo utilizaremos HTTP y por tanto no nos hará falta programar más extensiones.

[7]Para mandar peticiones HTTP se podrían usar por ejemplo la línea de comandos (curl), postman o un programa de python (módulo requests).

5.5.3. Proveedor con INESData

El proveedor con INESData incluye una interfaz gráfica para que el administrador pueda gestionar el conector, pero a parte tiene las siguientes diferencias con respecto al diseño con EDC:

1. El conector no se separa en plano de control y plano de datos.
2. El servidor de autorización adopta el rol de ancla de confianza en el espacio de datos.
3. Uso de OAuth2 para autorizar el acceso a la Management API.
4. Desaparece el Identity Hub y en su lugar se usarán JWT en cada petición del DSP.

La Figura 5.15 muestra estos cambios en forma de un diagrama de componentes. Las interfaces del conector (Management, DSP y API pública) son las mismas, cambiando solo su método de autorización, que requerirán un token emitido por el servidor de autorización. Para verificar que los tokens son válidos, el conector y su interfaz podrán validarlos mediante una petición al servidor de autorización, o de forma local si ya conocen su llave pública.

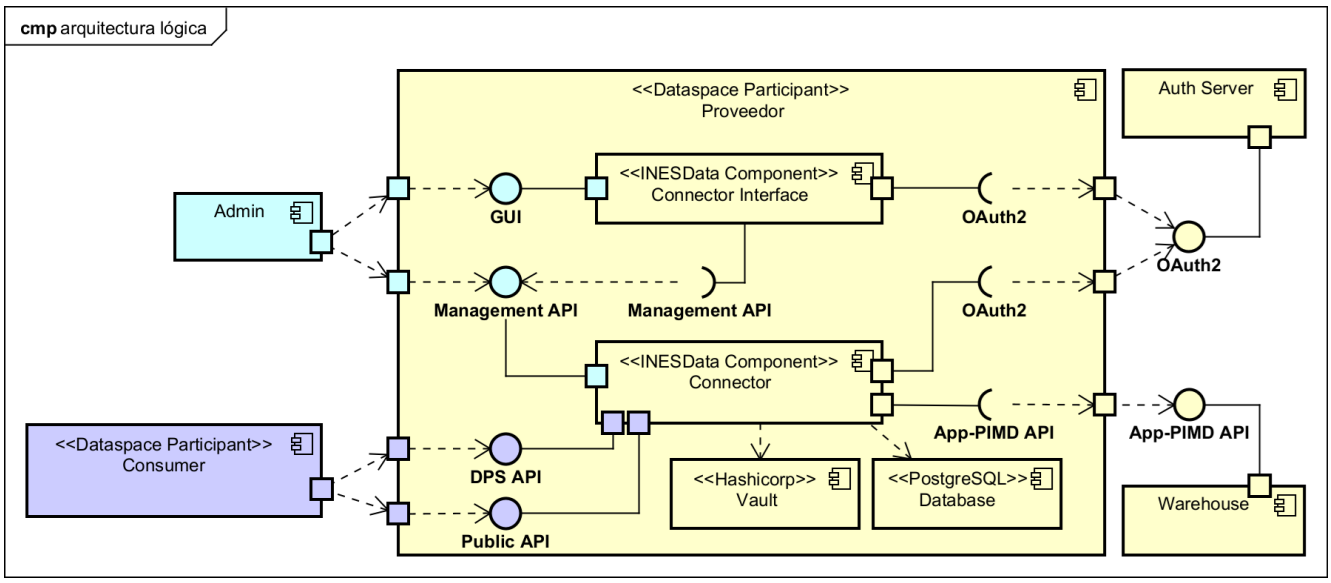


Figura 5.15: Diagrama de componentes: proveedor con INESData

5.6. Arquitectura física

La arquitectura física describe la organización de los componentes de hardware y de software sobre una infraestructura específica. Es decir, representa la estructura real sobre la que se ejecuta el software. Para ello elaboraremos un diagramas de despliegue de UML para cada diseño lógico.

Para elaborar este diseño y favorecer su interoperabilidad, escalabilidad y gestión, utilizaremos Docker y Kubernetes. En la Subsección 5.6.1 comentaremos las particularidades de estas dos tecnologías y cómo las utilizaremos. Los diseños físicos los detallamos en la Subsección 5.6.2 para el de EDC y en la Subsección 5.6.3 para el de INESData.

5.6.1. Tecnologías

Utilizar los componentes de EDC o de INESData no nos obliga a utilizar ninguna arquitectura de despliegue en específico. La decisión de emplear Docker y Kubernetes se motiva por las muchas ventajas que nos proporcionan a la hora de gestionar la infraestructura, siendo mucho más sencillo adaptarla a cambios de disponibilidad o uso. Por ejemplo, Kubernetes nos permite también utilizar el mismo diseño físico tanto en local como en *cloud*.

Nos hemos inspirado a usar estas tecnologías por el Minimum Viable Dataspace (MVD) de EDC [16], y por el diseño de los espacios de datos de INESData [71]. Una breve definición de estas tecnologías es la siguiente:

Docker Tecnología de contenedores que permite empaquetar aplicaciones junto con sus dependencias creando así entornos aislados.

Kubernetes Plataforma de orquestación de contenedores que permite gestionar el despliegue, la escalabilidad y la disponibilidad de aplicaciones en contenedores.

Antes de elaborar el diseño físico es necesario familiarizarse con la arquitectura de un clúster de kubernetes, la cual mostramos en la Figura 5.16. Como se puede ver, un clúster se compone de un plano de control (o potencialmente varios), también llamados *master nodes*, y una serie de *worker nodes*. Un nodo es una máquina física o virtual. Dentro de cada nodo hay una serie de procesos ajenos a la aplicación que se aseguran del buen funcionamiento del clúster.

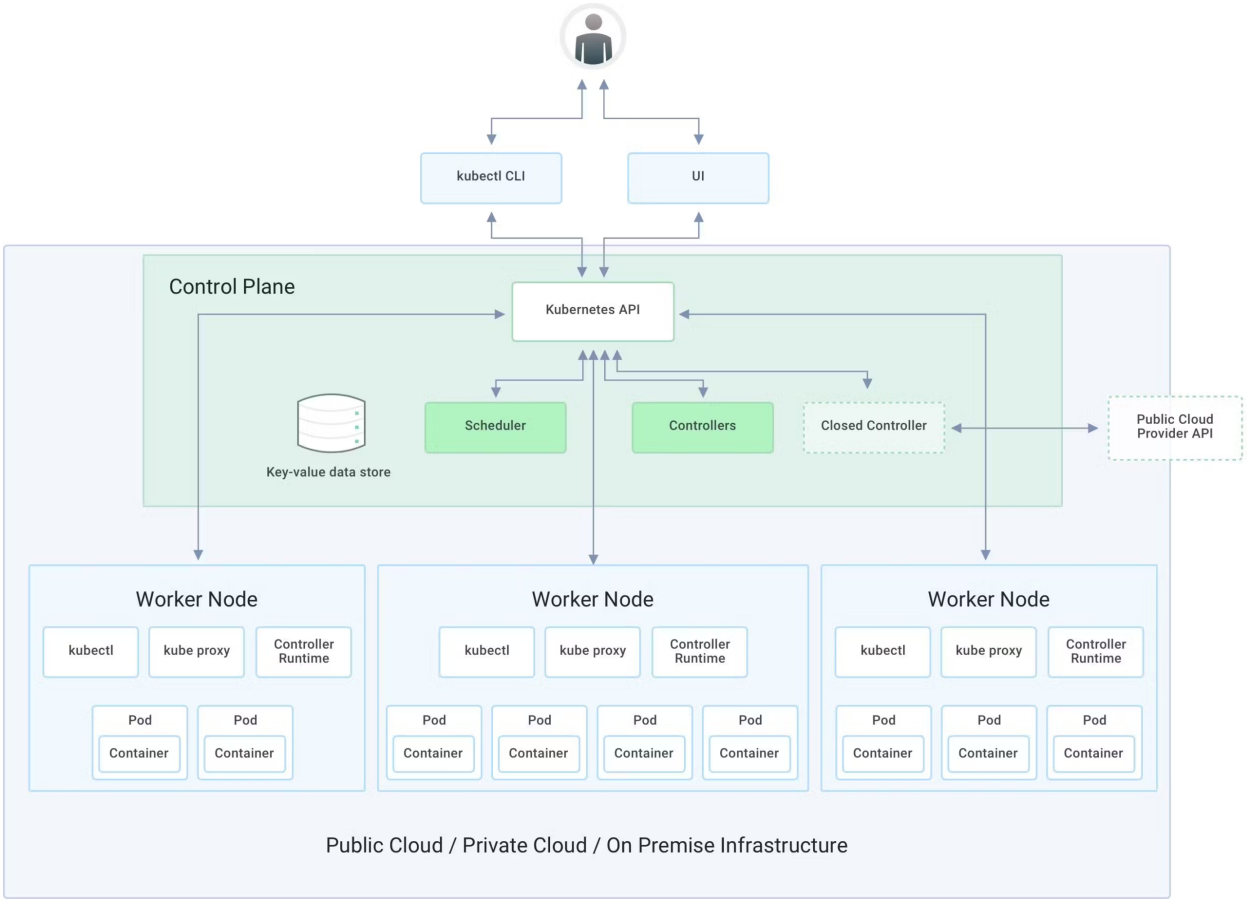


Figura 5.16: Arquitectura de un clúster de Kubernetes [72]

Adaptando esta arquitectura a nuestro caso, los nodos serán máquinas virtuales, y tendremos solo un *worker node*, en el cual habrá una *pod* por cada componente lógico. Dentro de cada *pod* habrá un contenedor, que ejecutará la imagen de docker del componente correspondiente. Para ambos diseños físicos, cada componente lógico se corresponderá con un contenedor de Docker.

5.6.2. Proveedor con EDC

El diseño del proveedor de EDC es *on premise*. En concreto sobre el *Portatil 2* (ver Sección 2.2: Gestión de los recursos). Este proveedor está formado por 5 componentes, por lo que tenemos 5 *pods*, cada una con un contenedor.

Mostramos en la Figura 5.17 el diagrama de despliegue del proveedor. Para intentar facilitar su lectura hemos utilizado colores para agrupar a los nodos que tienen las mismas características. De morado están las *pods*, de naranja los contenedores de Docker, de azul los procesos de Kubernetes, y de verde pardoso los artefactos.

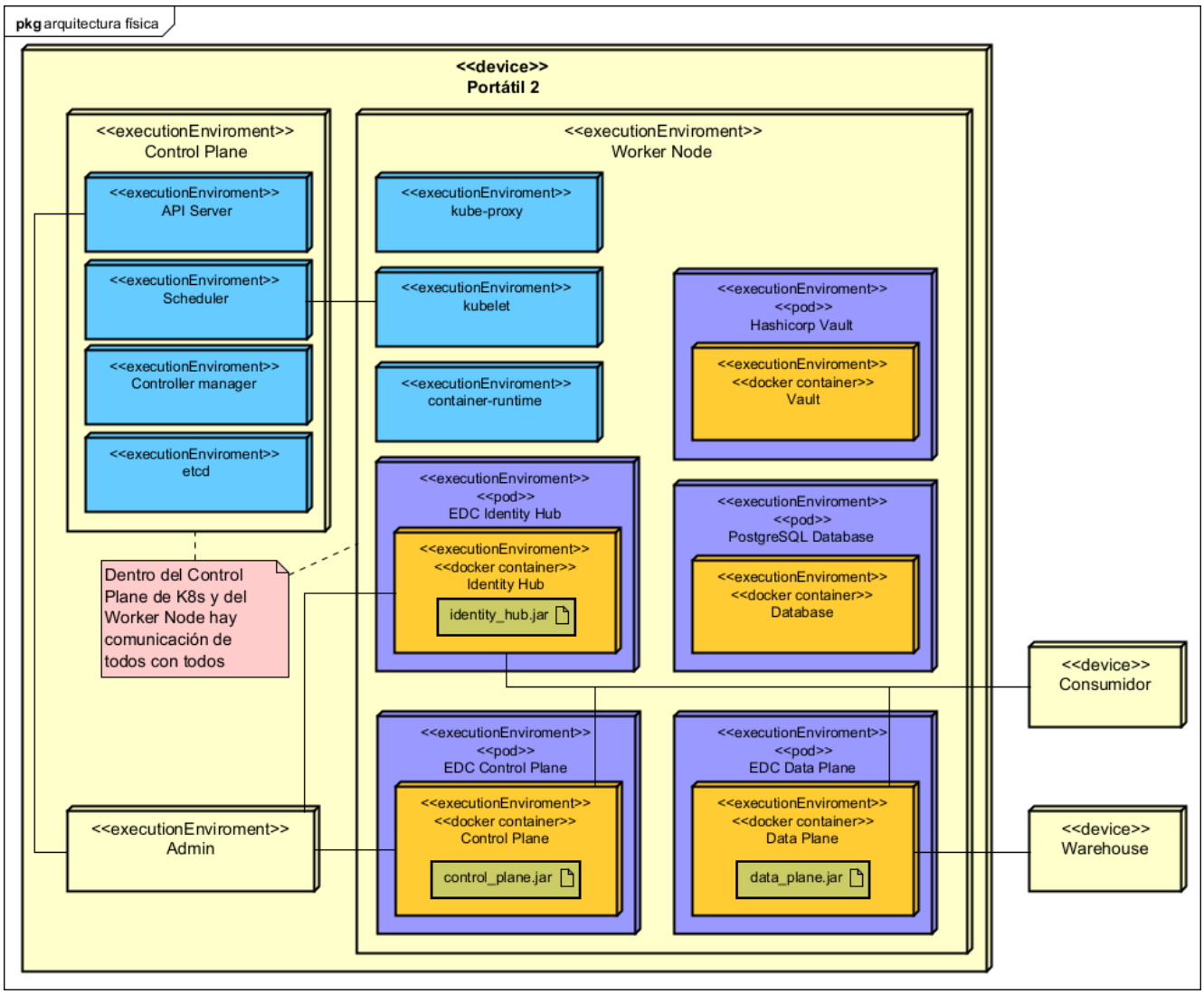


Figura 5.17: Diagrama de despliegue: proveedor con EDC

5.6.3. Proveedor con INESData

El diseño del proveedor con INESData es en entorno *cloud*, en concreto sobre Google Cloud. Al estar basado en Kubernetes, no hay grandes diferencias con respecto al diseño anterior. Solo cabe destacar que para acceder al proveedor, el administrador deberá comunicarse desde otro dispositivo, en nuestro caso desde el *Portatil 1*.

El diagrama de despliegue de la Figura 5.18 muestra este diseño físico, en el que tenemos una pod menos y un componente externo adicional (*Auth Server*).

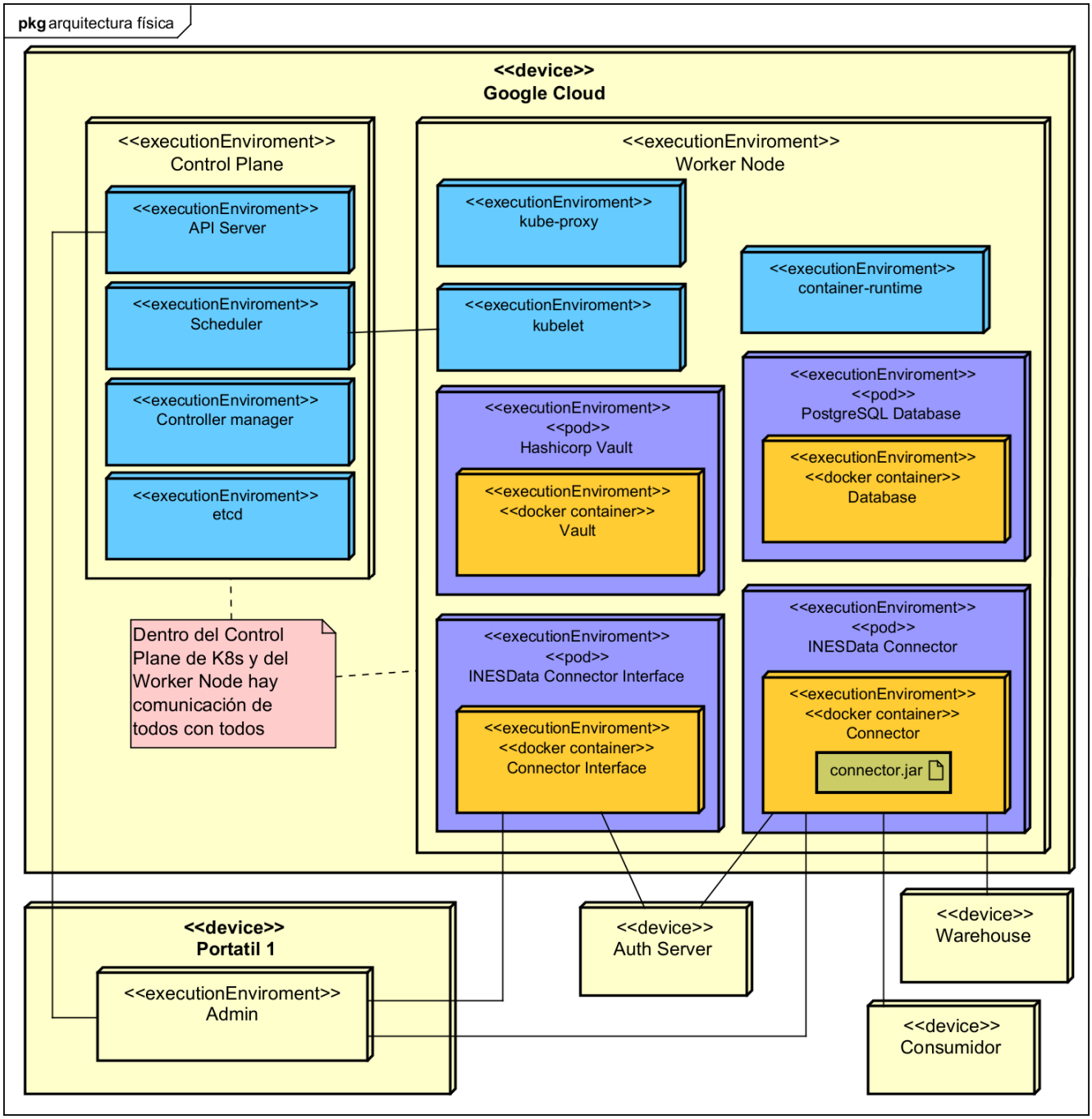


Figura 5.18: Diagrama de despliegue: proveedor con INESData

5.7. Diseño detallado

En esta sección vemos partes más específicas del diseño del proveedor. Nuestro objetivo es diseñar una extensión para permitir al proveedor acceder al *warehouse* usando la API de App-PIMD. La implementación del cliente HTTP de EDC soporta el acceso a fuentes de datos con HTTPS, pero no admite certificados autofirmados, que es el que usa la API de App-PIMD. Además también tendremos que transformar los datos de JSON a JSON-LD, y proporcionar grupos temáticos de aplicaciones.

Aunque antes de diseñar esta extensión explicaremos brevemente el contexto necesario: la arquitectura de EDC (Subsección 5.7.1), cómo crear extensiones (Subsección 5.7.2), y específicamente extensiones para el Data Plane (Subsección 5.7.3). Finalmente daremos todos los detalles del diseño de la extensión en la Subsección 5.7.4.

5.7.1. Arquitectura de EDC

La arquitectura del *framework* de EDC se puede resumir en la Figura 5.19. EDC se basa en un sistema de módulos, donde cada componente tiene unos módulos centrales básicos, a los que se puede añadir funcionalidades adicionales eligiendo una serie de extensiones. Las extensiones se pueden crear implementando las interfaces en los módulos SPI (*Service Provider Interface*) [73].

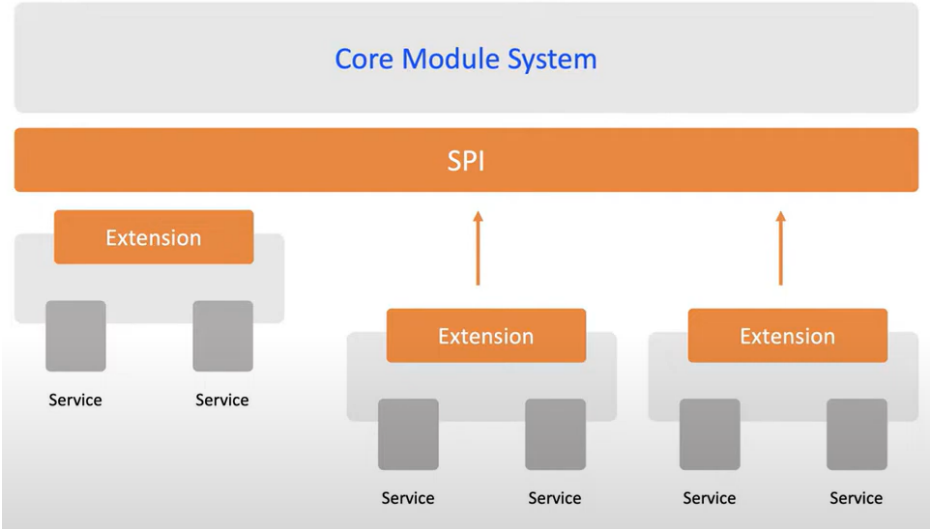


Figura 5.19: Arquitectura de EDC [73]

Según Jim Marino (EDC *Lead Architect*) [73], los principios arquitectónicos de EDC son:

- **Modularidad**: Toda la funcionalidad se contribuye como módulos, lo cual permite construir componentes ligeros usando las mínimas dependencias posibles.
- **Extensibilidad**: Todas las características tienen un punto de extensión, por lo que se pueden intercambiar la implementación de cualquier módulo, o crear nuevas capacidades y características.
- **Adaptabilidad**: Permite adaptar los componentes para desplegarlos en cualquier entorno: *cloud*, *on-premise* o *edge*. Los componentes no tienen por que tener siempre las mismas capacidades, que se pueden aumentar o disminuir dependiendo del caso de uso.
- **Resiliencia**: EDC es capaz de aprovechar infraestructura de alta disponiblidad ya existente. Por ejemplo: servicios cloud, gestión con clústeres, o infraestructuras de transferencia de datos.

5.7.2. Extensiones

Como ya hemos mencionado, las extensiones son la unidad básica para personalizar los componentes de EDC. Para crear una extensión básica necesitamos al menos dos cosas: una clase que implemente la interfaz *ServiceExtension* y un fichero *plugin* en el directorio *src/main/resources/META-INF/services/* con el nombre *org.eclipse.edc.spi.system.ServiceExtension* en el que introduciremos el nombre completamente cualificado de la clase implementadora [28].

A continuación mostramos un ejemplo de una extensión vacía *SampleExtension*, basado en la documentación de EDC [28].

```
// fichero
// src/main/java/com/example/extensions/SampleExtension.java
public class SampleExtension implements ServiceExtension {

    @Override
    public void initialize(ServiceExtensionContext context) {
        // do something
    }
}
```

Listing 5.1: Ejemplo de extension [28]

```
// fichero
// src/main/resources/META-INF/services/org.eclipse.edc.spi.system.ServiceExtension
...
com.example.extensions.SampleExtension
...
```

Listing 5.2: Ejemplo de fichero plugin

Normalmente las extensiones proporcionan un servicio al componente, pudiendo requerir también utilizar los servicios de otras extensiones. Para manejar estas dependencias, EDC utiliza el **patrón SPI**. Cada extensión debe indicar los servicios de los que depende y que proporciona. Si una extensión proporciona un servicio, esta debe incluir su interfaz en el módulo SPI. De esta manera las extensiones nunca dependen de otras extensiones, sino exclusivamente del módulo SPI [28]. Una representación de este patrón se puede ver en la Figura 5.20.

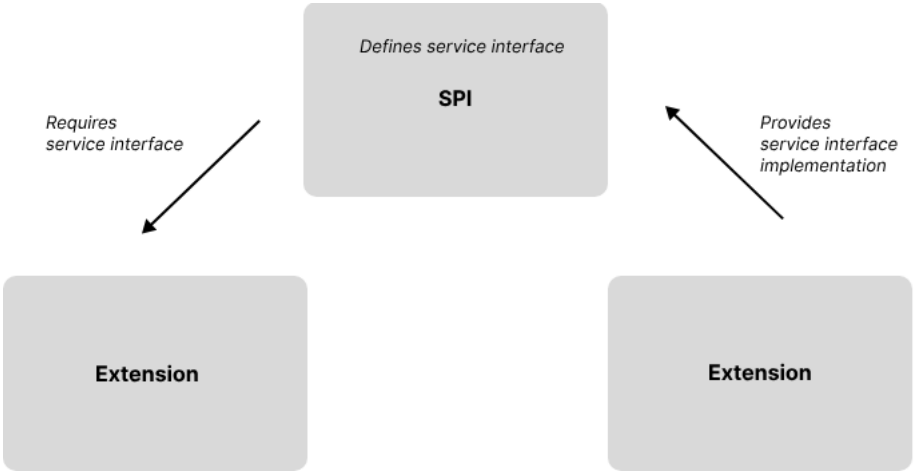


Figura 5.20: Patrón SPI [28]

5.7.3. Data Plane Framework

Para poder diseñar nuestra extensión, debemos utilizar el *Data Plane Framework*, que consiste en una serie de SPIs y de implementaciones por defecto. Nosotros solo necesitaremos utilizar las interfaces que tratan el origen de los datos, porque para enviar los datos al consumidor utilizaremos HTTP, como ya hemos indicado en la Subsección 5.5.2: Proveedor con EDC.

Las tres interfaces del Data Plane SPI que vamos a necesitar son: *PipelineService*, *DataSourceFactory* y *DataSource*. El servicio *PipelineService* nos permite registrar *DataSourceFactory*, que al recibir una petición de transferencia de datos, identificará cual de las factorías que tiene registradas pueden satisfacer la petición y se la enviará. La *DataSourceFactory* creará entonces el *DataSource* correspondiente, que se encargará de recuperar los datos.

5.7.4. Extensión para el warehouse

La extensión para habilitar al proveedor a acceder al *warehouse* se llamará *WarehouseExtension* y tiene la siguiente funcionalidad:

1. Acceso a la API de App-PIMD con HTTPS.
2. Transformación de Apps de JSON a JSON-LD.
3. Acceso a grupos temáticos de aplicaciones.

Dividiremos las responsabilidades creando tres nuevas fuentes de datos que implementan la interfaz *DataSource*. *HttpsDataSource* se encargará de acceder a un punto de acceso HTTPS, sin hacer ninguna transformación a la información recibida. *AppDataSource* transformará además los datos (de aplicaciones) de JSON a JSON-LD. Y *AppGroupDataSource* se encargará de solicitar todas las aplicaciones del mismo grupo temático, para lo cual habrá que enviar más de una petición.

Cada factoría crea su *DataSource* correspondiente y depende de la factoría anterior para reutilizar sus servicios. Además, incluimos en un módulo SPI propio, el esquema de las *DataAddress* para cada factoría. Este esquema se utiliza para validar la sintaxis de las direcciones.

La clase *WarehouseExtension* creará una instancia de cada factoría y las registrará en el *PipelineService*.

Hemos hecho dos diagramas para representar el diseño de la extensión. En la Figura 5.21 mostramos las dependencias de la clase *WarehouseExtension*, y en la Figura 5.22 mostramos el diseño de las factorías y del resto de la extensión.

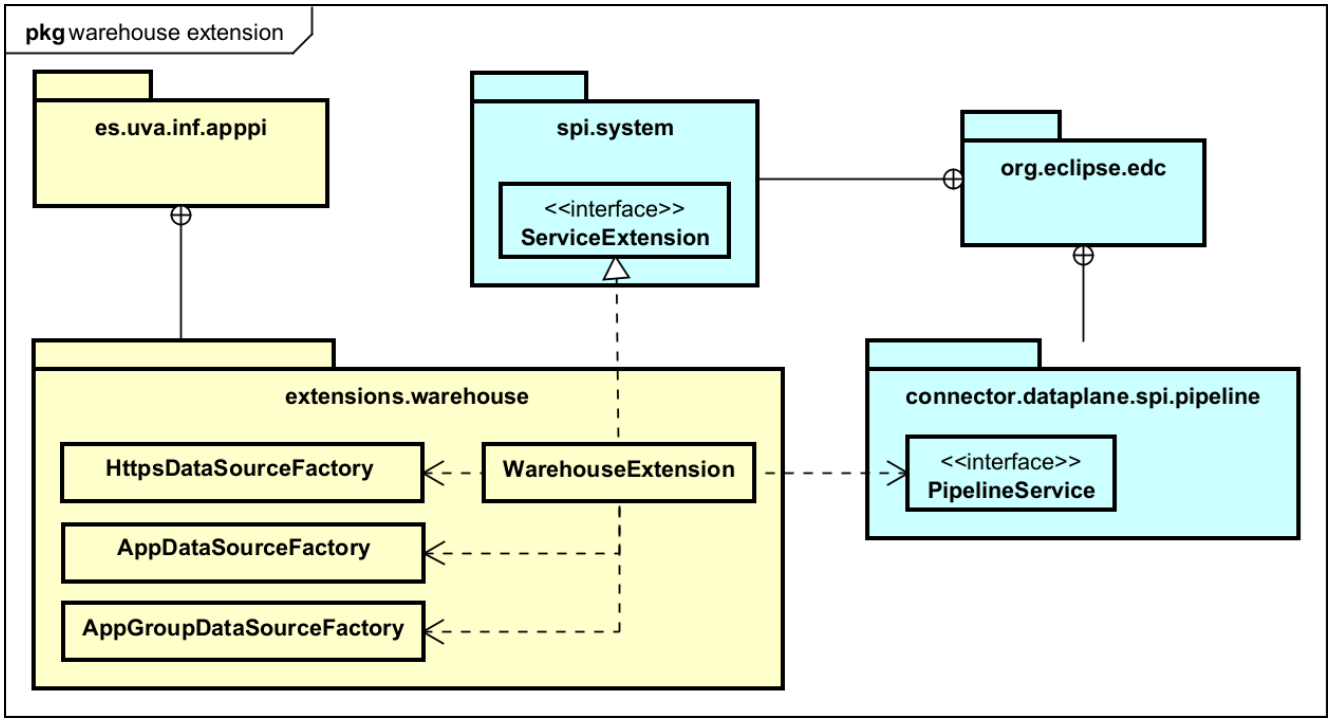


Figura 5.21: Diseño de *WarehouseExtension*

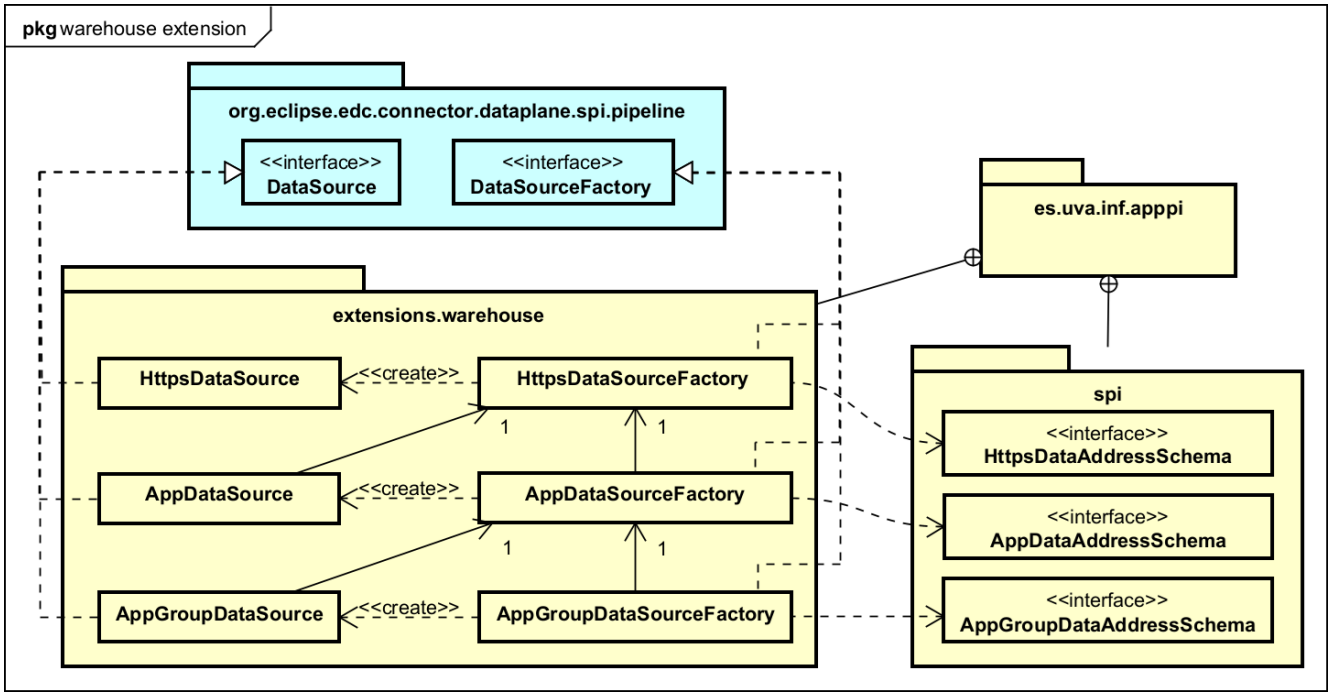


Figura 5.22: Diseño de las factorías

Capítulo 6

Implementación, despliegue y pruebas

La implementación es el proceso en el que se **lleva a la práctica el sistema** siguiendo la documentación de diseño. En esta fase es imprescindible también comprobar que el sistema funciona adecuadamente y conforme a lo esperado, para lo cual diseñaremos un **conjunto de pruebas**. En este capítulo proporcionamos toda la información, decisiones y fragmentos de código necesarios para entender y replicar este proceso.

Para poder probar adecuadamente el sistema no solo será necesario implementar un proveedor, sino que necesitaremos al menos dos participantes. Implementaremos así un **espacio de datos en miniatura** al cual llamaremos **AppSpace**. Ambos participantes tendrán las mismas especificaciones, pero uno de ellos adoptará el rol de proveedor y el otro de consumidor.

Empezamos el capítulo con la Sección 6.1, donde hacemos un listado exhaustivo de las tecnologías que hemos utilizado. A continuación, en la Sección 6.2, describimos los **pasos para implementar AppSpace** que hemos seguido, entrando en detalle sobre los dos repositorios que hemos utilizado como referencia y aprovechado para acelerar el proceso de implementación.

El último paso de la implementación consiste en poblar al proveedor de metadatos, los cuales tienen una especial relevancia al estar trabajando con espacios de datos. Por eso dedicamos la Sección 6.3 a detallar la **implementación de los metadatos**, dando ejemplos representativos de las credenciales, políticas, activos y ofertas. También proporcionamos la definición de la ontología.

Para terminar la implementación, en la Sección 6.4 explicamos la **organización del código** y el contenido de los directorios y archivos importantes. El código está disponible en el **repositorio público de GitLab**: codigo-tfg-alfonso-cabrero.

Sobre el despliegue, en la Sección 6.5 describimos las **instrucciones para desplegar AppSpace**, que podemos entender como la traducción a comandos de los pasos de la Sección 6.2, una vez ya desarrollado todo el código.

Por último, en la Sección 6.6 describimos las **pruebas** que hemos realizado para validar el correcto funcionamiento del sistema, que consistirán en pruebas unitarias y pruebas de sistema.

6.1. Entorno de desarrollo

Nuestro entorno tecnológico de desarrollo está formado por tres grupos de herramientas, según el motivo que hemos tenido para elegirlas. El primer grupo son las que hemos decidido utilizar en la fase de diseño, el segundo grupo es el que necesitamos para aprovechar el repositorio de EDC Minimum Viable Dataspace (MVD) , y el tercer grupo son las que hemos elegido por comodidad y familiaridad en su uso.

- Herramientas elegidas en diseño.

  - **JDK 17** Kit de desarrollo de Java que incluye todo lo necesario para ejecutar aplicaciones Java: los componentes de EDC e INESData.

  - **Docker** Tecnología de contenedores que permite empaquetar aplicaciones junto con sus dependencias creando así entornos aislados.

  - **Kubernetes** Plataforma de orquestación de contenedores que permite gestionar el despliegue, la escalabilidad y la disponibilidad de aplicaciones en contenedores.

  - **JSON-LD** Estándar del W3C basado en JSON para la serialización de modelos semánticos.

- Herramientas que utiliza el EDC Minimum Viable Dataspace (MVD).

  - **Gradle** Herramienta de automatización de compilación que se utiliza para gestionar dependencias, compilar código y empaquetar aplicaciones.

  - **Terraform** Herramienta de infraestructura como código (IaC) que permite definir, provisionar y administrar recursos de manera declarativa. Se utiliza para automatizar la creación y configuración de infraestructura.

  - **Kind** Herramienta para ejecutar clústeres locales de Kubernetes utilizando contenedores Docker como nodos.

  - **Postman** Herramienta para probar y desarrollar APIs mediante peticiones HTTP. Usaremos colecciones de postman para probar la Management API.

  - **Openssl** Usaremos la herramienta CLI para la generación y gestión de claves criptográficas y certificados.

- Herramientas elegidas por comodidad o familiaridad en su uso.

  - **Python 3.13** Lenguaje de programación interpretado de alto nivel. Lo utilizaremos para poblar el proveedor creando las políticas, activos y ofertas.

  - **JUnit 5** Framework de pruebas unitarias para aplicaciones Java, diseñado para facilitar la escritura y ejecución de tests de forma estructurada y eficiente.

  - **Turtle** Formato de serialización para expresar datos en RDF de manera compacta y legible. Permite definir prefijos para simplificar las referencias a URIs. Lo usaremos para crear la ontología.

  - **VS Code** Editor de código ligero y extensible. Ofrece múltiples herramientas avanzadas como para depuración de código o integración con Git.

  - **Git** Sistema de control de versiones que facilita la colaboración y la gestión de cambios en el código.

  - **GitLab** Plataforma de desarrollo colaborativo basada en Git que usaremos para alojar el repositorio de desarrollo del trabajo y para compartir el código.

    Enlace al código: https://gitlab.inf.uva.es/alfcabr/codigo-tfg-alfonso-cabrero.

6.2. Proceso de implementación

El proceso de implementación del proveedor, con nuestra elección de tecnologías, consiste en los siguientes pasos:

1. Descargar e instalar las **tecnologías necesarias** (detalladas en Sección 6.1: Entorno de desarrollo).
2. Implementar la **extensión** para el *EDC Data Plane*.
3. Crear las **imágenes para cada componente** (de docker) implementando y ejecutando los *scripts* de **gradle**[1].
4. Crear el **clúster de kubernetes** ya sea con **kind** o con **Google Kubernetes Engine** (GKE).
5. Definir la **infraestructura necesaria** implementando y ejecutando los *scripts* de **terraform**.
6. **Poblar el proveedor** con identidad, políticas, activos y ofertas desde un *script* de **python**.

Aunque como ya hemos mencionado en la introducción del capítulo, no implementaremos solo un proveedor, sino un espacio de datos en miniatura (AppSpace). Esto en la práctica solo supone definir infraestructura adicional usando las mismas imágenes pero con diferente configuración.

Para acelerar el esfuerzo de implementación aprovecharemos el trabajo de dos repositorios: EDC Minimum Viable Dataspace [16] e INESData Dataspace Local Enviroment [74], de los que hablamos más en detalle en la Subsección 6.2.1 y en la Subsección 6.2.2 respectivamente. También hemos utilizado el repositorio del conector de INESData [75] porque para añadirle nuestra extensión necesitamos el código fuente.

6.2.1. EDC Minimum Viable Dataspace

El EDC MVD es un repositorio de demostración en el que se muestra el funcionamiento de los componentes de EDC y que pretende facilitar la adopción de su tecnología por los desarrolladores [46]. El enlace a este repositorio es: github.com/eclipse-edc/MinimumViableDataspace. Aunque el MVD no está diseñado para entornos de producción [16], sí que se adapta a nuestros objetivos al permitirnos probar su funcionamiento. No obstante, si se quisiera utilizar en un entorno real, en un trabajo posterior se deberían abordar los atajos que se han tomado, para desarrollar así un proveedor seguro.

El espacio de datos MVD lo forman dos participantes: un proveedor y un consumidor. Pero el proveedor tiene además dos departamentos: *Q&A*, y *Manufacturing*, cada uno con su dominio de gestión independiente. En la Figura 6.1 mostramos todos los componentes que forman el MVD, que hemos elaborado con *draw.io* [6], basándonos en las descripciones de [16].

El servidor de catálogo es un componente simple que solo atiende peticiones de solicitud de catálogo, y en este caso unifica los catálogos de los dos departamentos del proveedor. El consumidor por ejemplo sigue el mismo diseño que el proveedor de la Subsección 5.5.2.

El participante emisor del espacio de datos no existe como tal. El MVD ha optado por simular a este participante mediante un servidor web (nginx) que aloja un documento DID que los participantes pueden consultar para verificar la autenticidad de las credenciales.

Modificaremos el MVD eliminando el conector del departamento de *Manufacturing* y el servidor de catálogo, pudiendo aprovechar así gran parte del código ya desarrollado. De esta manera tendremos un espacio de datos simétrico con dos participantes.

[1] Los componentes para la base de datos (postgres), el gestor de secretos (vault), el servidor de autorización (keycloak) y la interfaz del conector (inesdata) se descargan de un repositorio de imágenes.

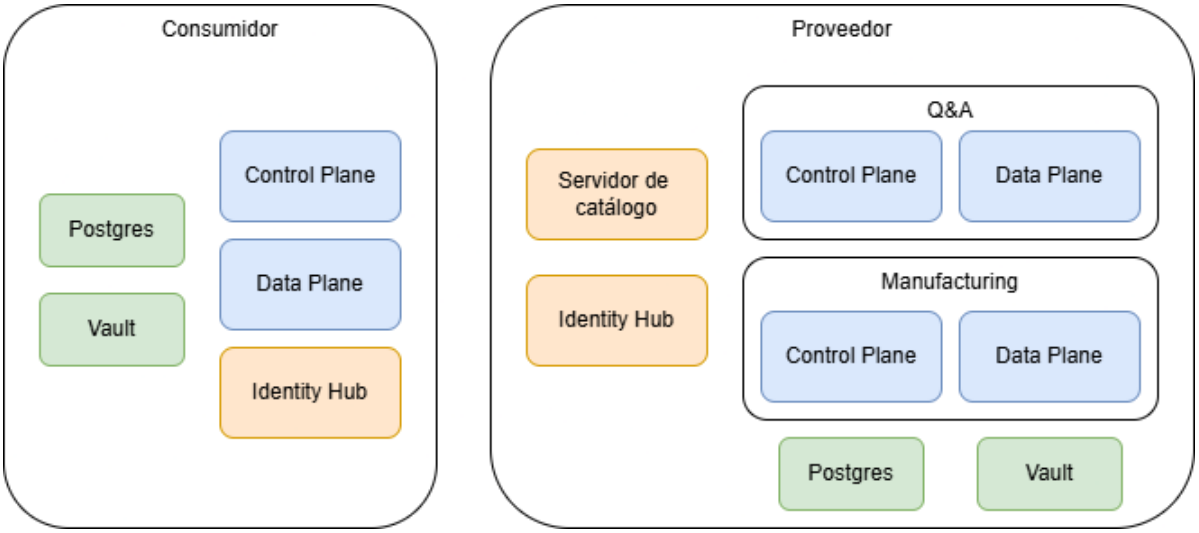


Figura 6.1: Escenario del Minimum Viable Dataspace (MVD)

6.2.2. INESData Dataspace Local Enviroment

El Dataspace Local Enviroment (DLE) de INESData tiene el mismo objetivo que el EDC MVD. Facilita el uso de sus componentes por otros desarrolladores y les permite probarlos en un espacio de datos real. Se puede acceder al repositorio a través del siguiente enlace: inesdata-local-env [74].

A diferencia con el MVD, en el DLE hay un emisor y por tanto tenemos 3 participantes. El emisor es la entidad que gobierna el espacio de datos y emite los *tokens* que permiten operar los conectores y la comunicación segura entre los participantes. A parte del servidor de autorización (keycloak), el emisor cuenta con otros componentes de INESData que enriquecen el espacio de datos, aunque nosotros no los utilizaremos. El *Registration Service* ayuda a federar el catálogo del espacio de datos manteniendo una lista de los participantes registrados. Y el portal público ofrece información del espacio de datos de manera abierta, para que los interesados ajenos al espacio de datos puedan consultar la estructura de gobernanza o algunos de los datos disponibles [71].

En la Figura 6.2 mostramos el escenario del DLE, con todos los componentes que lo forman. La hemos elaborado con la herramienta *draw.io* [6] basándonos en un diagrama del repositorio DLE [74].

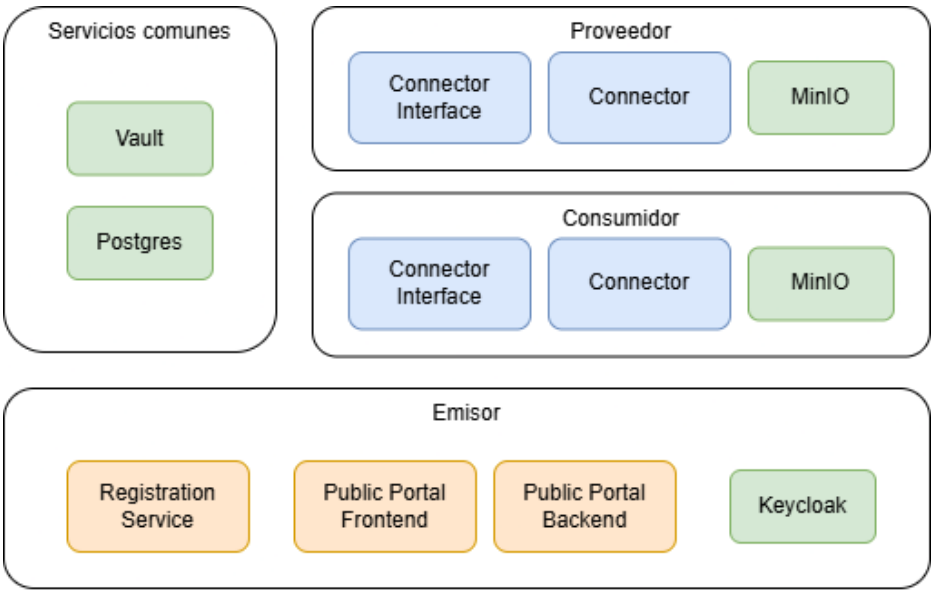


Figura 6.2: Escenario del Dataspace Local Enviroment (DLE)

Los participantes principales (proveedor y consumidor) comparten entre ellos la base de datos y el gestor de secretos (la base de datos también con el emisor) para evitar tener réplicas de estos componentes. También tienen su propio servidor de almacenamiento de objetos (MinIO), aunque nosotros prescindiremos también de este último porque usamos App-PIMD.

Para desplegar el entorno, el DLE utiliza docker compose en lugar de un clúster de kubernetes. Para respetar el diseño original hemos intentado adaptar la configuración del repositorio para usar terraform y kind. Aunque tras comprobar que no era una tarea sencilla e intentarlo durante más de una semana con solo éxito parcial hemos optado por mantener la implementación con docker compose.

6.3. Implementación de los datos

La implementación de los datos se basa en los modelos de diseño de las secciones 5.1, 5.2 y 5.4. En el caso de las políticas, activos y ofertas utilizaremos el modelo de EDC, como ya mencionamos en diseño y cuya documentación podemos encontrar en [28].

Credenciales

Como ya mencionamos en el diseño de las credenciales (Sección 5.1), solo utilizaremos la credencial de miembro, y que es obligatoria para cualquier comunicación con cualquier participante de AppSpace. A continuación mostramos la credencial en el formato VC, que usará el proveedor en la implementación de EDC:

```
{
    "@context": [ "https://www.w3.org/ns/credentials/v2" ],
    "@type": [ "VerifiableCredential", "MembershipCredential" ],
    "issuer": {
            "@id": "did:web:appspace-issuer",
            "name": "AppSpace Issuer Entity"
    },
    "name": "Membership Credential",
    "description": "Membership Credential in the AppSpace Data Space",
    "validFrom": "2025-03-01T00:00:00Z",
    "validUntil": "2025-04-01T00:00:00Z",
    "credentialSubject": {
            "@id": "did:web:appspace-provider",
            "claims": {
                    "memberSince": "2025-03-01T00:00:00Z",
                    "memberUntil": "2025-12-31T23:59:59Z"
            }
    }
}
```

Listing 6.1: Credencial de miembro del proveedor

En la implementación con EDC de AppSpace no existe como tal un participante emisor de las credenciales. Por eso crearemos las credenciales manualmente y las firmaremos usando unas llaves autogeneradas con openssl. Después serán almacenadas en la Vault correspondiente donde se utilizarán por el Identity Hub para crear *Verifiable Presentations* (VPs).

En la implementación con INESData no usamos VCs, pero hacemos algo que en la práctica resulta equivalente: crearemos un rol en Keycloak de miembro del espacio de datos. Por tanto los participantes que consigan este rol podrán solicitar un token de acceso y con él demostrar que son miembros de AppSpace, lo mismo que harían con una VC o una VP.

Políticas

En la Subsección 5.2.1 definimos una única política para nuestro proveedor que solo solicite ser miembro del espacio de datos. Hemos tomado la implementación de esta política directamente del EDC MVD porque se adapta perfectamente a nuestras necesidades, y la mostramos a continuación:

```
{
  "@context": [ "https://w3id.org/edc/connector/management/v0.0.1" ],
  "@type": "PolicyDefinition",
  "@id": "require-membership",
  "policy": {
      "@type": "Set",
      "permission": [
          {
              "action": "use",
              "constraint": {
                  "leftOperand": "MembershipCredential",
                  "operator": "eq",
                  "rightOperand": "active"
              }
          }
      ]
  }
}
```

Listing 6.2: Política única del proveedor [16]

El servicio encargado de evaluar las políticas y sus expresiones es el *Policy Engine*, el cual tiene un modelo de extensión que nos permite añadir *policy functions* para poder evaluar políticas propias. En nuestro caso, al utilizar una política de EDC no serán necesarias modificaciones. Para más información del *Policy Engine* y cómo funciona se puede consultar la documentación de EDC [28].

Activos

Como ya detallamos en el diseño de los activos (Subsección 5.2.2), tenemos dos categorías: aplicaciones individuales y grupos. Los activos son una instancia de la clase *edc:Asset*, que tienen dos propiedades: *edc:properties*, en la que describiremos con precisión el activo del que se trata, y *edc:dataAddress*, en la que describimos cómo se accede al activo en cuestión. A continuación mostramos un activo representativo para cada categoría:

```
{
  "@context": {
      "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
      "edc": "https://w3id.org/edc/v0.0.1/ns/",
      "dct": "https://purl.org/dc/terms/",
      "apps": "https://w3id.org/apppi/v0.3/"
  },
  "@id": "app.com.discord",
  "@type": "edc:Asset",
  "properties": {
      "dct:type": { "@id": "https://w3id.org/apppi/v0.3/App" },
      "name": "Discord Privacy Metadata",
      "description": "Privacy metadata about the discord android app",
      "description@es": "Metadatos de privacidad de la app de android discord",
      "apps:assetCategory": "INDIVIDUAL_APP",
      "apps:hash": "00006852e35635388...265f923e2477e2907fd",
```

```
        "apps:package": "com.discord",
        "dct:format": "application/ld+json"
    },
    "dataAddress": {
        "@type": "edc:DataAddress",
        "type": "Warehouse-App",
        "apps:hash": "00006852e35635388...265f923e2477e2907fd"
    }
}
```

Listing 6.3: Activo de una aplicación individual: Discord

```
{
    "@context": {
        "@vocab": "https://w3id.org/edc/v0.0.1/ns/",
        "edc": "https://w3id.org/edc/v0.0.1/ns/",
        "dct": "https://purl.org/dc/terms/",
        "apps": "https://w3id.org/apppi/v0.3/"
    },
    "@id": "app.group.communication",
    "@type": "edc:Asset",
    "properties": {
        "dct:type": {
          "@id": "https://w3id.org/apppi/v0.3/App", "@container": "@list"
        },
        "name": "Communication Apps Privacy Metadata",
        "description": "Privacy metadata about communication android apps",
        "description@es": "Metadatos de privacidad de aplicaciones de comunicacion",
        "apps:assetCategory": "THEMATIC_GROUP",
        "apps:thematicGroupName": "COMMUNICATION",
        "dct:format": "application/ld+json"
    },
    "dataAddress": {
        "@type": "edc:DataAddress",
        "type": "Warehouse-AppGroup",
        "apps:thematicGroupName": "COMMUNICATION"
    }
}
```

Listing 6.4: Activo de un grupo temático de aplicaciones: Comunicación

Ofertas

Las ofertas relacionan los activos con las políticas, y que EDC llama contratos. En un contrato de EDC podemos relacionar a dos políticas distintas: *accessPolicy* y *contractPolicy*, e indicar los activos a los que se aplica con *assetsSelector*, de una forma parecida a las restricciones en las políticas. *accessPolicy* determina la visibilidad de la oferta, es decir, si un consumidor concreto puede acceder a la oferta y solicitar negociarla a través del catálogo. *contractPolicy* en cambio, determina las condiciones que debe cumplir el consumidor para acceder a cualquiera de los activos.

Para nuestro proveedor, tendremos una única oferta. De esta manera, cualquier miembro de AppSpace solo necesitará negociar un contrato una vez para acceder a cualquier activo. Ambas políticas que hemos mencionado serán *require-membership* (nuestra única política). A continuación mostramos la implementación de esta oferta:

```
{
  "@context": [ "https://w3id.org/edc/connector/management/v0.0.1" ],
  "@id": "general-offer",
  "@type": "ContractDefinition",
  "accessPolicyId": "require-membership",
  "contractPolicyId": "require-membership",
  "assetsSelector": {
      "@type": "Criterion",
      "operandLeft": "https://w3id.org/edc/v0.0.1/ns/id",
      "operator": "in",
      "operandRight": [
          "app.group.communication",
          "app.group.banking",
          "app.group.education",
          "...",
          "app.com.whatsapp",
          "app.com.discord",
          "app.com.duolingo",
          "..."
      ]
  }
}
```

Listing 6.5: Oferta única del proveedor

Ontología

Para implementar la ontología hemos decidido usar Turtle, que es mucho más compacto y legible que JSON-LD para grandes documentos. Hemos seguido al detalle el diseño de la Subsección 5.2.3, y el diseño de la extensión de la Sección 5.4. A continuación mostramos la definición de la ontología:

```
@prefix apps: <https://w3id.org/apppi/v0.3/> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix dct:  <http://purl.org/dc/terms/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vann: <http://purl.org/vocab/vann/> .
@prefix voaf: <http://purl.org/vocommons/voaf#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix cc:   <http://creativecommons.org/ns#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.

apps:
    a voaf:Vocabulary, owl:Ontology ;
    rdfs:label "App-PI Ontology"@en ;
    dct:title "The App Privacy Impact Ontology"@en ;
    rdfs:comment """
    This ontology defines classes and properties to
    describe metadata about the privacy of mobile apps.
    """@en ;
    cc:license <http://www.apache.org/licenses/LICENSE-2.0> ;
    dct:creator _:AlfonsoCabrero ;
    dct:created "2025-02-27"^^xsd:date ;
    dct:modified "2025-04-14"^^xsd:date ;
    owl:versionInfo "0.3.0" ;
    owl:versionIRI <https://w3id.org/apppi/v0.3/> ;
    vann:preferredNamespaceUri "https://w3id.org/apppi/v0.3" ;
    vann:preferredNamespacePrefix "apps" ;
.
```

Listing 6.6: Definición de la ontología

El resto de la implementación de la ontología está disponible en el repositorio de GitLab, en el directorio *ontology*. La definición se encuentra en el archivo principal *Ontology.ttl*, que hace referencia al resto de archivos usando *owl:imports*. Las definiciones de las clases se encuentran en las carpetas *classes* y *superclasses*, además de las propiedades de datos y restricciones correspondientes. Las propiedades de objeto se definen en el archivo *ObjectProperties.ttl*, y la extensión en *Extension.ttl*.

6.4. Organización del código

Todo el código desarrollado y utilizado en este trabajo está disponible en el repositorio público de GitLab accesible desde este enlace: https://gitlab.inf.uva.es/alfcabr/codigo-tfg-alfonso-cabrero. En esta sección describimos el contenido de este repositorio: sus directorios y sus ficheros más importantes.

- **components-edc**: Contiene un proyecto de gradle que permite compilar los tres componentes de EDC que usamos en el trabajo: Control Plane, Data Plane y Identity Hub. Usa la versión 0.12.0 de los módulos de EDC (marzo 2025).
  - **extensions**: Contiene módulos independientes que se pueden añadir a los componentes. Entre ellas está la extensión *app-pi-warehouse*, que sigue el diseño de la Subsección 5.7.4. El resto de extensiones son necesarias para el funcionamiento del MVD.
  - **launchers**: Contiene los módulos de gradle que crean las imágenes de cada componente. Cada subdirectorio compila un componente diferente: *controlplane*, *dataplane* y *identity-hub*.
  - **spi**: Contiene los módulos SPI. Solo tiene *app-pi-data-plane*, donde se definen el esquema de las *DataAddress* que define la extensión *app-pi-warehouse*, tal y como se indica en la Subsección 5.7.4.
  - **build.gradle.kts**: Fichero principal de configuración de gradle en el que se establecen las tareas, dependencias y plugins.
  - **settings.gradle.kts**: Fichero de configuración adicional de gradle en el que se definen los módulos y submódulos del proyecto (componentes y extensiones).
- **components-inesdata**: Contiene un proyecto de gradle que permite compilar el conector de INESData, y su estructura es la misma que la del directorio *components-edc*. Usa la versión 0.10.0 de los módulos de EDC (octubre 2024). Es un proyecto de gradle a parte por que la versión de EDC es incompatible con la que utilizan los otros componentes.
- **deployment**: Contiene un proyecto de terraform y otros ficheros de configuración para la fase de despliegue.
  - **assets**: Contiene credenciales, secretos y archivos de configuración.
  - **edc**: Contiene los archivos del entorno principal para desplegar AppSpace con EDC.
  - **inesdata**: Contiene los (insuficientes) archivos del entorno principal para desplegar AppSpace con INESData. Contiene también el archivo *docker-compose.yml* que reemplaza esta funcionalidad.

    - **modules**: Módulos de terraform con la configuración de cada componente: *connector*, *inesdata-connector*, *identity-hub*, *vault*, etc. Se reutilizan para el consumidor y proveedor.
    - **seed**: Contiene los scripts de python que pueblan al espacio de datos con identidad, activos, políticas y ofertas.
- **ontology**: Implementación de la ontología en Turtle.
- **test**: Contiene cuadernos de Jupyter en Python para realizar pruebas a los espacios de datos.

6.5. Instrucciones de despliegue

Continuando con la Sección 6.2, en esta sección especificamos en detalle como podemos desplegar AppSpace usando el repositorio de código del trabajo. Tenemos la opción de desplegar el espacio de datos usando tanto EDC como INESData, para lo cual deberemos seguir estas instrucciones con algunos cambios. Se dividen en los siguientes tres pasos:

- Paso 1: Compilar el código y construir las imágenes.
- Paso 2: Configurar y ejecutar las imágenes (en un clúster / con docker compose).
- Paso 3: Poblar el espacio de datos.

Para las instrucciones de los pasos 1 y 2 hemos utilizado ficheros e instrucciones del repositorio EDC MVD en su versión 0.12.0 (enlace) [16], y del repositorio INESData DLE en su versión 0.9.0 [74]. Para ejecutar estas instrucciones se necesitará una consola compatible con POSIX, y tener disponible el siguiente software:

- JDK 17+
- Python 3
- Docker
- KinD (solo EDC)
- Kubernetes (solo EDC)
- Terraform (solo EDC)

Los comandos se deben ejecutar en el directorio que se indica con el comando `cd`, y en caso de que no se indique niguno, desde el directorio raíz del repositorio de código.

Paso 1: Compilar el código y construir las imágenes

Usando EDC

```
cd components-edc
./gradlew build -x checkstyleMain -x checkstyleTest -x test -x javadoc
./gradlew dockerize -Ppersistence=true
```

Usando INESData

```
cd components-inesdata
./gradlew build
./gradlew dockerize
```

Paso 2: Configurar y ejecutar las imágenes

Usando EDC

1. Crear el clúster.

```
kind create cluster -n mvd --config deployment/kind.config.yaml
```

2. Cargar las imágenes en KinD.

```
kind load docker-image -n mvd controlplane:latest dataplane:latest \
  identity-hub:latest
```

3. Desplegar un Ingress con Nginx. Actúa como proxy y permite acceder desde la máquina local al clúster.

```
kubectl apply -f \
https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
```

4. Esperar a que el controlador Ingress esté disponible.

```
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s
```

5. Desplegar el espacio de datos, escribe 'yes' después de `terraform apply`.

```
cd deployment/edc
terraform init
terraform apply
```

Usando INESData

```
cd deployment/inesdata
docker compose up
```

Paso 3: Poblar el espacio de datos

Usando EDC

```
cd deployment/seed
python3 ./identity.py
python3 ./policies_assets_contracts.py
```

Usando INESData

```
cd deployment/seed
python3 ./inesdata_policies_assets_contracts.py
```

6.6. Pruebas

Las pruebas son una parte indispensable en el ciclo de vida del desarrollo de software, siendo una etapa crítica para garantizar la calidad del producto final. A través de ellas, es posible identificar defectos, validar las funcionalidades y asegurar que el sistema se comporta conforme a lo esperado.

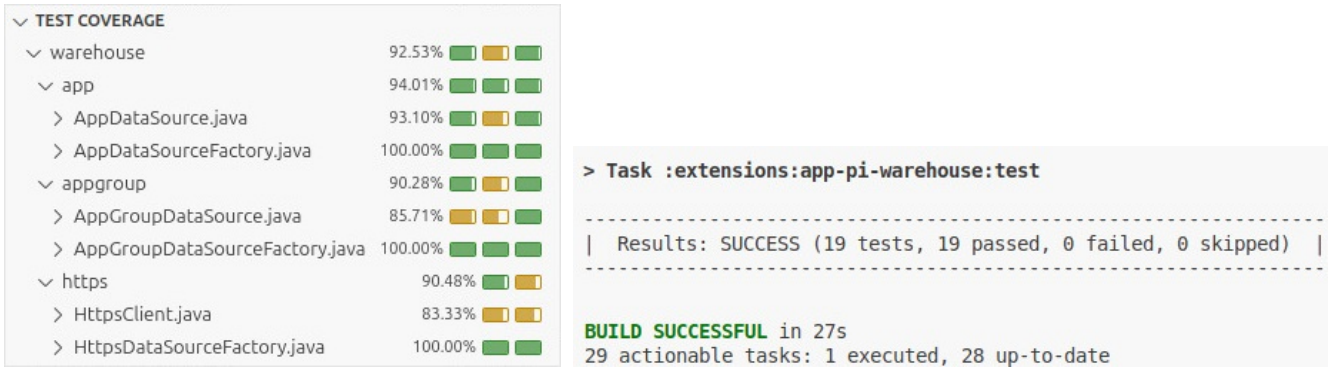
En este trabajo hemos realizado dos tipos de pruebas a nuestro sistema, **unitarias** para la extensión del Data Plane, y **de sistema** para AppSpace en su conjunto. Las pruebas unitarias tienen el objetivo de validar el único código que modifica el funcionamiento del proveedor. En cambio, para probar el resto del código utilizaremos pruebas de sistema. Así podremos comprobar que todos los componentes interactúan adecuadamente, y el funcionamiento global del sistema.

6.6.1. Pruebas unitarias

Hemos realizado las pruebas a la extensión del Data Plane utilizando el *framework* de pruebas JUnit en su versión 5. Hemos realizado un total de 19 pruebas, que suponen una cobertura de 196 de las 210 líneas de código de la extensión, un 93,33 %.

Estas pruebas simulan peticiones para iniciar la transferencia de datos provenientes del Control Plane, pero sin enviar los datos a su destinatario. De esta forma, creamos mensajes tanto válidos como inválidos que prueban el comportamiento de la extensión en escenarios normales y de error. Estos escenarios de error incluyen atributos faltantes (al definir la forma de acceso en un activo no se define la categoría, o el hash de la aplicación), o errores en ellos (categoría que no existe, hash que no existe). Dos escenarios que no hemos podido simular son la falta de conexión con la API (que no responda) o recibir una respuesta de una aplicación mal formada (que el JSON no válido, falten atributos o algún nombre esté mal escrito).

Estas pruebas certifican que el conector tiene acceso al repositorio App-PIMD, y que la transformación a JSON-LD de las aplicaciones se hace adecuadamente.



(a) Vista de la cobertura con VSCode

(b) Resultado de las pruebas con Gradle

Figura 6.3: Resultado de las pruebas unitarias

6.6.2. Pruebas de sistema

Las pruebas de sistema se realizan con AppSpace completamente operativo, y consisten en probar el acceso a los datos. Es condición necesaria para acceder a los datos, que todo el espacio de datos funcione adecuadamente, incluyendo a todos los componentes, su interacción y sus metadatos. De esta manera podremos probar el resto del sistema, verificando así la validez de los scripts de terraform, de python, de las credenciales y del resto de metadatos, etc.

Hemos hecho estas pruebas al final de cada incremento, en las que el consumidor solicita el acceso a los siguientes datos:

- Petición de 6 activos de aplicaciones individuales.
- Petición de 2 activos de grupos temáticos de aplicaciones.

Hemos elegido las aplicaciones para que fueran lo más heterogéneas posibles entre sí, es decir, de distintas categorías, que al menos una defina un grupo de permisos, que al menos una no tenga utilice ni declare permisos, etc. Las respuestas a cada petición se han procesado también por JSON-LD Playground [76] para verificar que los modelos semánticos generados son correctos. Cada petición de datos tiene los siguientes pasos:

1. Consulta de catálogo.
2. Negociación de contrato.
3. Inicio de proceso de transferencia.
4. Acceso a datos.

Pruebas primer incremento

Las pruebas se realizan desde un cuaderno de Jupyter en Python, que se puede encontrar en el directorio *test* del repositorio de código con el nombre *edc_request_asset.ipynb*. Todas las peticiones se realizan sin incidencias y con éxito.

En el cuaderno también se puede ver el resultado de la última prueba que se ha realizado, en la que se solicitó el activo *app.com.discord* (enlace). También mostramos la visualización del modelo semántico de este activo en la Figura 6.4, obtenido con una captura de pantalla de JSON-LD Playground [76].

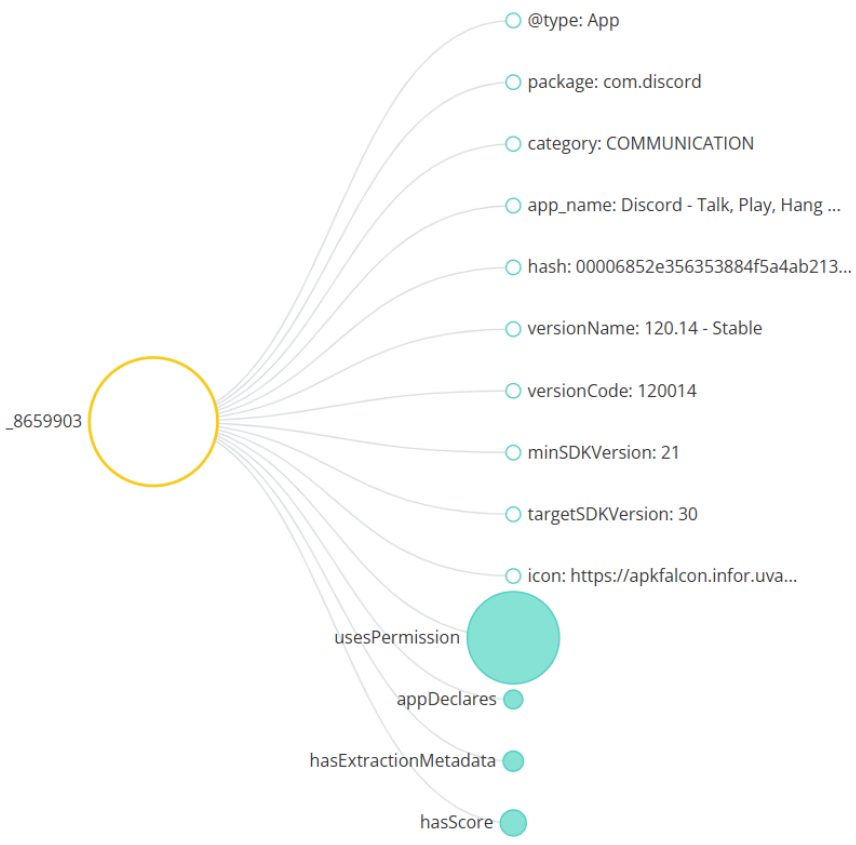


Figura 6.4: Visualización con JSON-LD Playground del activo *app.com.discord*

Pruebas segundo incremento

En el segundo incremento tampoco se incluye la interfaz gráfica, a si que las pruebas también se realizan desde un cuaderno de Jupyter. Se puede encontrar junto al otro cuaderno con el nombre *inesdata_request_asset.ipynb*. Todas las peticiones se realizan sin incidencias y con éxito.

En el cuaderno se puede ver el resultado de solicitar el activo *app.group.finance* (enlace).

Pruebas tercer incremento

En el tercer incremento utilizamos la interfaz gráfica del conector de INESData. Hemos podido utilizarlo para consultar todos los datos del conector (activos, ofertas, contratos, historial de transferencias, etc), y también para crearlos aunque no para modificarlos.

Lo que no hemos conseguido es visualizar el catálogo para poder iniciar el proceso de petición de un activo. En ausencia de un manual del componente es difícil saber si esto se debe a algún error en la configuración, a un error en el código, o si esta funcionalidad no está implementada todavía. Para asegurarnos de que el componente accedía correctamente al catálogo hemos revisando su código fuente. Hemos podido modificar los ficheros de configuración para cambiar la ruta de la API de catálogo, aunque así tampoco hemos conseguido que se visualizara correctamente.

En conclusión, calificamos estas pruebas como fallidas. Deberemos esperar a la publicación de un manual que nos permita diagnosticar y arreglar el fallo de la interfaz.

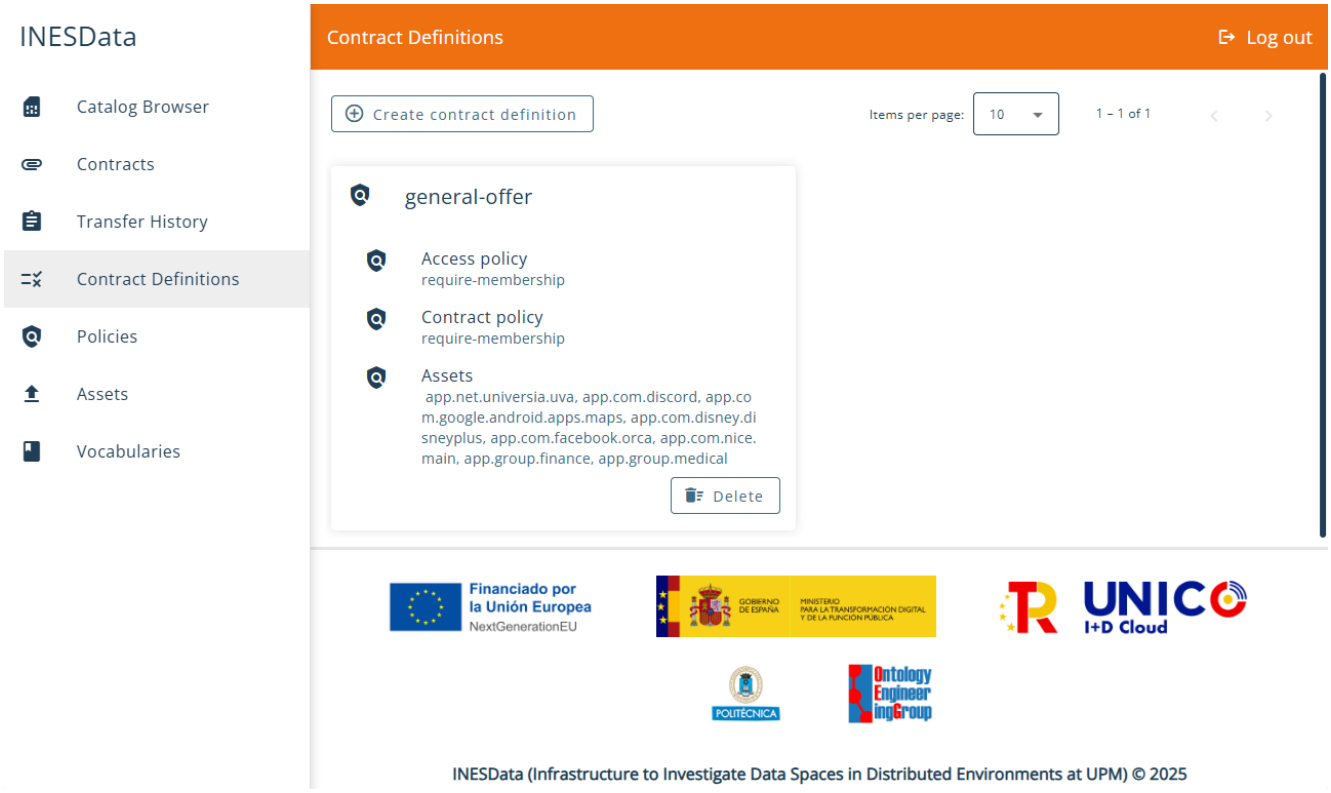


Figura 6.5: Vista de las ofertas desde INESData Interface Connector

Capítulo 7

Conclusiones y trabajo futuro

Este capítulo final está dedicado a reflexionar sobre los resultados del trabajo, haciendo un resumen de los aprendizajes y logros conseguidos. Además, y con especial importancia en el contexto de este trabajo, propondremos líneas de mejora y trabajo futuro, fundamentales para garantizar su continuidad y resolver las limitaciones pendientes.

7.1. Conclusiones

A lo largo de este trabajo hemos analizado el estado actual de los espacios de datos, y hemos identificado sus principales ventajas y fortalezas. Hemos analizado en profundidad los estándares más reconocidos en la actualidad, y los hemos aplicado diseñando un proveedor. Pese a ser aún una tecnología emergente, prometen crear un entorno seguro de intercambio de datos centrado en la confianza, la soberanía y la interoperabilidad, favoreciendo así la innovación, la colaboración y la prosperidad.

También hemos estudiado la importancia que tienen las políticas de acceso y uso y normas de gobernanza para proteger la soberanía de los datos y promover la confianza, y visto el estado del arte actual con el modelo ODRL, los modelos de identidad descentralizados como las credenciales verificables (VCs) y los identificadores descentralizados (DIDs), o los basados en OAuth2.

También hemos aprendido la relevancia de las ontologías y los modelos semánticos, especialmente para la integración de información, la web semántica y los servicios basados en datos distribuidos. Además, nos hemos iniciado en el diseño de ontologías creando una propia modelando metadatos sobre aplicaciones móviles.

Y utilizando todos estos aprendizajes, los hemos conseguido aplicar con éxito desarrollando un proveedor capaz de compartir el repositorio de App-PIMD, así como creando un pequeño espacio de datos (AppSpace) en el que hemos podido comprobar algunas capacidades de esta tecnología.

Compartir, pero con garantías

En este trabajo nos hemos centrado en compartir datos que ya son accesibles de manera pública, pero el potencial de los espacios de datos hace que también sea posible compartir datos que *a priori* no dejaríamos libremente al público. A través de las políticas de acceso y uso, el proveedor que hemos desarrollado habilita a compartir una mayor variedad de datos, incluso aunque sean especialmente sensibles.

Ya hemos mencionado un ejemplo de nuevos datos que desde el proyecto App-PI podrían decidir compartir, y esas son las valoraciones del riesgo para la privacidad de cada permiso de Android, usando metodologías diferentes.

Para ofrecer y proteger estos activos, el proveedor permite definir nuevas políticas con un formato muy flexible, capaz de adaptarse a las condiciones de cada proyecto. Algunos ejemplos de estas políticas son los siguientes: uso no comercial, uso para investigación, uso permitido solo en Europa (o cualquier otra geografía), prohibición de almacenar los datos, prohibición de transferir a terceros u obligación de citar la fuente de los datos.

Objetivos del trabajo y limitaciones del proveedor

Hemos cumplido con el principal objetivo del trabajo, que era compartir el repositorio App-PIMD a través de un proveedor. Lo hemos implementado con dos tecnologías, EDC e INESData, que soportan además dos modelos de identidad diferentes. Aunque actualmente se trata de una solución aislada, la hemos diseñado pensando en facilitar su integración futura a un espacio de datos adecuado. Esto permitirá que el repositorio sea accesible para un público mucho más amplio cuando surja un proyecto apropiado.

Por otra parte, al haber estado trabajado con un *framework* tan reciente, nos hemos encontrado con una limitación. Las APIs de EDC para la gestión del conector, o para enviar mensajes entre participantes solo permiten usar HTTP, lo cual supone un grave riesgo de seguridad y no permite exponer al proveedor a una red pública o a una red no segura. Sin duda, la tecnología seguirá madurando y este problema será resuelto en versiones posteriores.

Por último, si bien al principio del trabajo nos planteamos desplegar AppSpace en Google Cloud como objetivo complementario, finalmente no lo hemos hecho. En su lugar, hemos concentrado nuestros esfuerzos en el proveedor, que era lo más importante. Con el tiempo adicional, hemos podido mejorar su diseño y estudiar más detenidamente algunos conceptos de los espacios de datos y de las tecnologías que hemos utilizado.

7.2. Trabajo futuro

Relativo al proveedor

Las líneas de trabajo futuro más claras son las orientadas a superar los retos tecnológicos actuales, como los que hemos comentado. Sin duda, estos se solucionarán en un futuro cercano, a medida que la tecnología siga avanzando.

La continuidad de este proveedor pasará por adaptarlo e integrarlo en el espacio de datos de seguridad que surja en el futuro (o en varios). Esto implicará posiblemente el tener que revisar su arquitectura para asegurar la compatibilidad con los estándares que adopte este espacio de datos, además de incorporar las innovaciones tecnológicas que se vayan desarrollando.

Relativo al proyecto App-PI

Una línea de mejora que permitirá simplificar el proveedor, pero que requiere modificar la API de App-PIMD, será la de integrarlo con la ontología. Este cambio la acercaría en mayor medida a donde se producen los datos, y donde es más probable que sea modificada.

Con este cambio, además, la API podrá transformar sus respuestas a JSON-LD, que junto con utilizar un certificado emitido por una autoridad de certificación reconocida, eliminaría la necesidad de incluir la extensión al Data Plane en el conector.

Además, para que el proveedor pueda acceder a estos nuevos métodos, sería recomendable convertirlo en un usuario autorizado con mayores permisos en la API.

Bibliografía

[1] European Commission. The European Data Market study 2024-2026. Enlace, 2024.

[2] Data Spaces Support Centre (DSSC). Why data spaces? A business and user's perspective. Enlace, 2023.

[3] Universidad de Valladolid Grupo de Investigación en Ingeniería de la Privacidad. Proyecto App-PI (App Privacy Impact). Enlace. Visitado el 9 de mayo de 2025.

[4] INESData. Sobre el proyecto. Enlace. Visitado el 22 de febrero de 2025.

[5] tryQA.com. What is Incremental model- advantages, disadvantages and when to use it? Enlace. Visitado el 1 de febrero de 2025.

[6] JGraph Ltd. draw.io. Enlace. Visitado el 9 de mayo de 2025.

[7] TeamGantt. Project Management Software Tool. Enlace.

[8] Gregory M. Becker. *A practical risk management approach*. Paper presented at PMI® Global Congress 2004—North America, Anaheim, CA. Newtown Square, PA: Project Management Institute., 2004. Enlace.

[9] GLASSDOOR. Sueldos de Junior Software Engineer. Enlace. Visitado el 9 de mayo de 2025.

[10] GLASSDOOR. Sueldos de Tech Lead. Enlace. Visitado el 9 de mayo de 2025.

[11] asesorias.com. Horas de trabajo anuales. ¿Cómo calcularlas? Enlace. Visitado el 9 de mayo de 2025.

[12] Seguridad Social. Bases y tipos de cotización 2024. Régimen general de la Seguridad Social. Enlace. Visitado el 9 de mayo de 2025.

[13] Leburó. Precio de Coworking en Valladolid. Enlace. Visitado el 9 de mayo de 2025.

[14] PcComponentes.com. Portátil Acer Gaming Aspire 5 A515-58GM Intel Core i5-1335U/16GB/512GB SSD/RTX 2050. Enlace. Visitado el 9 de mayo de 2025.

[15] PcComponentes.com. Lenovo Ideapad 330 Intel Core i7-8550U/8GB/256GB SSD. Enlace. Visitado el 9 de mayo de 2025.

[16] Eclipse Dataspace Components. Minimum Viable Dataspace v0.12.0. Enlace, 2025.

[17] Google Cloud. Precios de Google Kubernetes Engine. Enlace. Visitado el 9 de mayo de 2025.

[18] Google Cloud. Precios de Cloud SQL. Enlace. Visitado el 9 de mayo de 2025.

[19] European Central Bank. Euro foreign exchange reference rates: US dollar (USD). Enlace.

[20] Red Eléctrica de España ESIOS (Sistema de Información del Operador del Sistema). Término de facturación de energía activa del PVPC 2.0TD. Enlace con vista previa. Descarga de datos.

[21] Selectra. Factura de la luz: desglose y explicación de conceptos. Enlace. Visitado el 9 de mayo de 2025.

[22] Dirección General de Gobernanza Pública administracion.gob.es. Impuesto especial sobre la electricidad. Enlace. Visitado el 9 de mayo de 2025.

[23] Selectra. Impuestos e IVA de la luz en 2025: cambios y evolución. Enlace. Visitado el 9 de mayo de 2025.

[24] Michael Franklin, Alon Halevy, and J Widom. Data-spaces: a new abstraction for data management. *SIGMOD Record, December*, 2005. Enlace.

[25] Edward Curry. *Dataspaces: Fundamentals, Principles, and Techniques*, pages 45–62. Springer International Publishing, Cham, 2020. Enlace.

[26] Fraunhofer ISST. Wie baut man Data Spaces? | Data Researchers #8. Enlace, 2020.

[27] World Wide Web Consortium. *Linked Data Glossary*. Enlace, 2013.

[28] Eclipse Dataspace Components. Documentation. Enlace. Visitado el 9 de mayo de 2025.

[29] European Commission. Common European Data Spaces. Enlace, 2024.

[30] International Data Spaces Association (IDSA). Use case brochure. Enlace, 2022.

[31] Data Spaces Support Centre (DSSC). *Data Spaces Blueprint v1.5: Building Block Overview*. Enlace, 2024.

[32] datos.gob.es. ¿Por qué espacios de datos? Enlace, 2024.

[33] International Data Spaces Association (IDSA). Data sovereignty. Enlace. Visitado el 9 de mayo de 2025.

[34] International Data Spaces Association (IDSA). Our mission and vision. Enlace. Visitado el 9 de mayo de 2025.

[35] Internationa Data Spaces Association (IDSA). GitHub. Enlace. Visitado el 9 de mayo de 2025.

[36] International Data Spaces Association (IDSA). *Reference Architecture Model (RAM) version 4*. Enlace, 2023.

[37] International Data Spaces Association (IDSA). *Dataspace Protocol version 2024-1*. Enlace, 2024.

[38] International Data Spaces Association (IDSA). Rulebook. Enlace. Visitado el 9 de mayo de 2025.

[39] GAIA-X. Members Directory. Enlace. Visitado el 9 de mayo de 2025.

[40] Ministerio Federal de Economía y Protección del Clima (BMWK; Bundesministerium für Wirtschaft und Klimaschutz). Franco-German Position on GAIA-X. Enlace, 2020.

[41] Hendrik Meyer zum Felde, Maarten Kollenstart, Thomas Bellebaum, Simon Dalmolen, and Gerd Stefan Brost. Extending actor models in data spaces. *Companion Proceedings of the ACM Web Conference 2023*, 2023.

[42] GAIA-X. About Gaia-X. Enlace. Visitado el 9 de mayo de 2025.

[43] GAIA-X. Services & Deliverables. Enlace. Visitado el 9 de mayo de 2025.

[44] International Data Spaces Association (IDSA). Data Connector Report | No. 16 - September 2024. Enlace, 2024.

[45] International Data Spaces Association (IDSA). Certified data connectors. Enlace. Visitado el 9 de mayo de 2025.

[46] Eclipse Foundation. Eclipse Dataspace Components. Enlace. Visitado el 9 de mayo de 2025.

[47] Elena Montiel and Víctor Rodríguez. Primera aproximación al proyecto INESData. Enlace, 2023.

[48] INESData. GitHub. Enlace (etiqueta inesdata-project) Enlace (usuario INESData). Visitado el 22 de febrero de 2025.

[49] datos.gob.es. El kit de iniciación a los espacios de datos. Enlace, 2023.

[50] European Commission. European data strategy. Enlace, 2020.

[51] European Commission. European Data Governance Act. Enlace, 2022.

[52] European Commission. Data Act. Enlace, 2024.

[53] Ministerio para la Transformación Digital y de la Función Pública. Plan de Impulso de los Espacios de Datos. Enlace, 2024.

[54] datos.gob.es. Plan de Impulso de los Espacios de Datos Sectoriales. Enlace, 2024.

[55] Mobility Data Space. Página web. Enlace. Visitado el 12 de febrero de 2025.

[56] Mobility Data Space. MDS Use Cases of our members. Enlace. Visitado el 12 de febrero de 2025.

[57] World Wide Web Consortium. *ODRL Information Model 2.2*. Enlace, 2018.

[58] World Wide Web Consortium. *Verifiable Credentials Data Model v2.0*. Enlace, 2025.

[59] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[60] Dean Allemang and James Hendler. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

[61] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993. Enlace.

[62] Thomas R. Gruber. *Encyclopedia of Database Systems* | *Ontology*, pages 1–3. Springer New York, 2016. Enlace.

[63] Alejandro Pérez de La Fuente. *Data Warehouse* para el estudio de la privacidad de aplicaciones móviles. Enlace, 2023.

[64] World Wide Web Consortium. *JSON-LD 1.1 A JSON-based Serialization for Linked Data*. Enlace, 2020.

[65] World Wide Web Consortium. *Decentralized Identifiers (DIDs) v1.0*. Enlace, 2022.

[66] World Wide Web Consortium Credentials Community Group. *did:web Method Specification*. Enlace, 2024.

[67] Internet Engineering Task Force (IETF). *JSON Web Token (JWT)*. Enlace, 2015.

[68] Eclipse Foundation. Eclipse Dataspace Protocol. Enlace. Visitado el 9 de mayo de 2025.

[69] Eclipse Foundation. Eclipse Dataspace Decentralized Claims Protocol. Enlace. Visitado el 9 de mayo de 2025.

[70] Internet Engineering Task Force (IETF). *The OAuth 2.0 Authorization Framework*. Enlace, 2012.

[71] INESData. E5. Componentes horizontales para espacios de datos. Enlace, 2025. Visitado el 9 de mayo de 2025.

[72] Kong Learning Center. Understanding the Basics of Kubernetes Architecture. Enlace. Visitado el 9 de mayo de 2025.

[73] Eclipse Dataspace Components. EDC Conceptual Overview and Architecture. Enlace, 2022.

[74] INESData. Dataspace Local Enviroment v0.9.0. Enlace, 2025.

[75] INESData. INESData Connector v0.9.1. Enlace, 2025.

[76] Digital Bazaar Inc. JSON-LD Playground. Enlace. Visitado el 9 de mayo de 2025.

Apéndice A

Herramientas utilizadas

Investigación y documentación

1. **LaTeX**: Lenguaje de marcado que facilita la preparación de documentos con estructura compleja. Utilizado para escribir la presente memoria.
   a) **MiKTeX**: Compilador de LaTeX de código abierto válido para Windows.
   b) **TeXstudio**: Editor de código LaTeX de código abierto (IDE).
   c) **tablesgenerator.com**: Herramienta online que facilita la creación de tablas con LaTeX (enlace).
2. **Google Scholar**: Motor de búsqueda que facilita buscar literatura académica.
3. **ChatGPT**: Chatbot de inteligencia artificial desarrollado por OpenAI. Usado para tareas de revisión ortográfica, síntesis y traducción.

Planificación y diseño

1. **TeamGantt**: Herramienta online para gestionar proyectos mediante diagramas de Gantt. Utilizada para elaborar el cronograma del proyecto (Figura 2.3).
2. **draw.io**: Herramienta online para crear diagramas y figuras de cualquier tipo. Utilizada para representar: la Estructura de División del Trabajo (Figura 2.2), el diseño de la ontología (figuras 5.6, 5.7, 5.8, 5.9 y 5.12) y los escenarios de los dos repositorios de código utilizados en el trabajo (Figura 6.1 y Figura 6.2)
3. **Canva**: Plataforma online de diseño gráfico. Utilizada para la figura Figura 2.5, en la que se muestra la matriz de índices de prioridad de riesgos.
4. **Astah Professional**: Herramienta de modelado UML. Utilizada para los diagramas de los capítulos de análisis y diseño.
5. **Python (matplotlib) y git**: Lenguaje de programación (librería para elaborar gráficos) y sistema de control de versiones. Utilizados en conjunto para elaborar la figura de evolución del número de palabras por capítulo (Figura 2.6).

Desarrollo, gestión y pruebas del código

A parte de las ya mencionadas en la Sección 6.1: Entorno de desarrollo, hemos utilizado también **ChatGPT**, como asistente de programación.

Otros

Para la comunicación con los tutores hemos utilizado la función de correo electrónico de **Microsoft Outlook**.