UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

OntoForms: Generación automática de formularios para instanciar ontologías

Aplicación en sistema de prevención de cáncer de mama

Informe de Proyecto de Grado presentado por

Bruno Szilagyi

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Regina Motz
Edelweis Rohrer

Montevideo, 31 de diciembre de 2024

OntoForms: Generación automática de formularios para instanciar ontologías

Aplicación en sistema de prevención de cáncer de mama

Agradecimientos

Me gustaría comenzar dando las gracias a todas las personas que han contribuido para que este trabajo sea posible.

Primero doy las gracias a mis tutoras de tesis, Edelweis Rohrer y Regina Motz, por su apoyo, consejos y guía en todo el proceso de desarrollo de esta tesis. La guía ha sido esencial para poder sortear varios obstáculos encontrados, armar la definición del proyecto y luego llegar a realizar varias publicaciones.
También agradecer la colaboración de Yasmine Anchén de la Facultad de Medicina que aportó conocimiento importantísimo para la implementación del caso de estudio del sistema recomendador de estudios preventivos para el cáncer de mama.

Mi esposa Jimena merece un agradecimiento especial, por ser quien siempre ha estado a mi lado con su amor y su apoyo sincero. Su ánimo y apoyo hicieron que hoy esté terminando este proyecto. Luego me gustaría agradecer a todo el resto de la familia, mis padres, mi hermano, mis suegros, que han estado siempre pendientes y disponibles para ayudar en lo que se necesite para facilitar mis estudios.

Resumen

Las ontologías desempeñan un papel fundamental en la representación del conocimiento, ya que permiten estructurar y organizar la información de tal manera que los sistemas puedan comprenderla y procesarla. En este sentido, surge la necesidad de herramientas que faciliten la interacción de los usuarios con las ontologías. En particular, el uso de formularios que permitan la instanciación de ontologías, es decir, el proceso de completar las ontologías con datos concretos, resulta esencial para la recopilación y entrada de información de manera eficiente. Para abordar esta problemática, en este trabajo se propone una solución que consiste en la creación automática de formularios web a partir de estructuras ontológicas, destinados a facilitar el ingreso de datos en la ontología. Con este enfoque, se busca no solo mejorar la accesibilidad de los datos, sino también garantizar la coherencia y validez de la información ingresada.

Esta solución, OntoForms, es implementada con un conjunto de aplicaciones web que se interconectan para proveer el servicio de carga y manejo de ontologías así como el servicio de configuración y generación de formularios. Consiste en un servicio web backend realizado con tecnología Java y SpringBoot en donde utilizando el framework de Apache Jena para el manejo de ontologías en conjunto con el razonador Open Pellet, realizando razonamiento e inferencias sobre la ontología en tiempo real. Para almacenar las ontologías y configuraciones se utiliza un triplestore Fuseki. Se disponibiliza un administrador web para utilizar los servicios de OntoForms y poder realizar la visualización de las distintas ontologías que el usuario carga. El sistema OntoForms permite a las aplicaciones clientes crear interfaces gráficas para el ingreso de nuevas instancias en la ontología correspondiente, facilitando así una implementación reutilizable en diversos dominios. En este estudio, se aplica este servicio en el caso de OntoBreastScreen, un sistema diseñado para recomendar estudios preventivos del cáncer de mama. Durante la consulta médica, esta aplicación optimiza y agiliza la recomendación de estudios de prevención, proporcionando en línea información sobre el cálculo de riesgo y guías de recomendación de instituciones de salud reconocidas. A partir de los datos ingresados por el médico en el formulario generado por OntoForms, la aplicación elabora recomendaciones basadas en la ontología de prevención del cáncer de mama, BCSR-ONTO.

Palabras clave: Ontología, Generación automática UI, Formularios, Cáncer de mama

Índice general

Capítulo 1

Introducción

Actualmente, los sistemas que utilizan ontologías emergen como herramientas poderosas para la representación y organización del conocimiento. Las ontologías, al definir de manera formal los conceptos y las relaciones dentro de un dominio específico, permiten a los sistemas comprender y razonar sobre la información de una manera más efectiva. Uno de los principales beneficios de los sistemas basados en ontologías es su capacidad para facilitar la interoperabilidad entre diferentes aplicaciones y plataformas. Al proporcionar un marco común de referencia, las ontologías permiten que sistemas dispares compartan y comprendan datos de manera coherente, lo que es especialmente valioso en entornos colaborativos y en la integración de información de múltiples fuentes. Otro aspecto destacado es la capacidad de las ontologías para facilitar la automatización de procesos. Al estructurar el conocimiento de manera formal, los sistemas pueden aplicar razonamiento automático para inferir nueva información, detectar inconsistencias y validar datos, lo que reduce la carga de trabajo manual y mejora la eficiencia operativa. Esto es particularmente beneficioso en áreas como la medicina y la investigación, donde la calidad de la información es crucial. En este contexto, la necesidad de herramientas que faciliten la interacción de los usuarios con las ontologías se vuelve fundamental. En particular, el problema de instanciación de la ontología, es decir, el proceso mediante el cual se rellenan las ontologías con datos concretos, es una de las fases críticas en su utilización.

Las ontologías a menudo contienen una estructura compleja con múltiples clases, propiedades y relaciones. Sin una interfase adecuada los usuarios deben interactuar directamente con esta complejidad, lo que puede resultar en confusión y errores. La dificultad, para usuarios del sistema, no expertos en ontologías, en comprender cómo se relacionan los diferentes elementos puede llevar a una instanciación incorrecta. Para salvar este problema, los formularios sirven como guías que orientan a los usuarios sobre qué datos son necesarios y en qué formato deben ser ingresados. Sin esta orientación, los usuarios pueden no estar seguros de qué información proporcionar, lo que puede dar lugar a omisiones o entradas inexactas.

Los formularios pueden implementar además reglas de validación para garantizar que la información ingresada sea coherente y cumpla con los criterios establecidos por la ontología. Sin formularios, esta validación debe hacerse manualmente, lo que incrementa la posibilidad de errores. Adicionalmente,los formularios permiten una entrada de datos más organizada y sistemática, facilitando la recolección de información en un formato estandarizado. Instanciar una ontología sin formularios puede resultar en un proceso más largo y desorganizado, lo que dificulta la recolección de datos a gran escala. La creación de formularios a partir de ontologías se presenta como una solución efectiva para superar estos desafíos y garantizar que la información se ingrese de manera correcta y eficiente.

Este trabajo aborda esta problemática a traves del diseño y construcción de un software, OntoForms, que permite la creación automática de formularios web a partir de estructuras ontológicas, destinados a facilitar el ingreso de datos en la ontología.

Disponer de un servicio que permite ingresar datos a la ontología a través de un formulario web, y que además también genera automáticamente dicho formulario a partir de la ontología, presenta numerosas ventajas. Con el tiempo, surgen nuevas necesidades de los usuarios que requieren modificaciones o extensiones en la ontología, lo que implica que la interfaz de usuario también debe adaptarse. Para evitar la necesidad de modificar la aplicación cada vez que la ontología cambia, proponemos un enfoque en el que los formularios de ingreso de datos se generen de manera dinámica. Este enfoque no solo mejora la mantenibilidad de las aplicaciones basadas en ontologías, sino que también potencia su reusabilidad, permitiendo una evolución ágil y eficiente de la experiencia del usuario sin complicaciones adicionales.

Lo que distingue a OntoForms de otros enfoques similares es su estrategia de generación del formulario, que resuelve los siguientes aspectos:

- Los campos de la interfaz de usuario se ubican y agrupan en diferentes secciones, en forma automática.
- Se crean automáticamente instancias de clases de la ontología de forma transparente para el usuario en los casos donde no es necesaria la intervención del usuario directamente, por ejemplo, en las relaciones n-arias.
- Se hace uso de los servicios de razonamiento basados en la semántica formal de lógica descriptiva para asegurar la consistencia de los axiomas declarados. Se consideran además las inferencias en la generación de campos del formulario para asegurar la completitud en la recolección de los datos instancias de la ontología.

OntoForms está implementada con un conjunto de aplicaciones web que se interconectan para proveer el servicio de carga y manejo de ontologías así como el servicio de configuración y generación de formularios. Consiste en un servicio

web backend realizado con tecnología Java y SpringBoot en donde utilizando el framework de Apache Jena para el manejo de ontologías en conjunto con el razonador Open Pellet, realizando razonamiento e inferencias sobre la ontología en tiempo real. Para almacenar las ontologías y configuraciones se utiliza un triplestore Fuseki. Se disponibiliza un administrador web para utilizar los servicios de OntoForms y poder realizar la visualización de las distintas ontologías que el usuario carga. OntoForms es un desarrollo en código abierto disponible en GitHub en los siguientes enlaces:
https://github.com/brunoszilagyiibarra/ontoforms-backend y
https://github.com/brunoszilagyiibarra/ontoforms-admin

OntoForms fue validado con varias ontologías existentes y se llevó a cabo un caso de uso con usuarios reales en el ámbito del cáncer de mama. Utilizando la ontología BCSR-ONTO (Anchén, Rohrer, y Motz, 2023), se desarrolló la aplicación "OntoBreastScreen: Sistema recomendador de estudios preventivos para el cáncer de mama", con la colaboración de la Licenciada en Imagenología Yasmine Anchén, quien es experta en el tema. Esta aplicación facilita y agiliza la recomendación de estudios preventivos durante la consulta médico-paciente, proporcionando en línea información sobre el cálculo de riesgo y guías de recomendación de instituciones de salud reconocidas A partir de los datos ingresados por el médico en el formulario generado por Ontoforms, la aplicación elabora recomendaciones para la realización de estudios preventivos.

En el transcurso de este trabajo se realizó la publicación del siguiente poster: Szilagyi, B., Rohrer, E., Anchén, Y., and Motz, R. An Ontological Approach to Breast Cancer Screening: Risk Assessment and Personalized Testing Recommendations. En conferencia internacional ER2024: Companion Proceedings of the 43rd International Conference on Conceptual Modeling: ER Forum, Special Topics, Posters and Demos, October 28-31, 2024, Pittsburg, Pennsylvania, USA. https://ceur-ws.org/Vol-3849/poster-demo4.pdf

El resto del documento está organizado de la siguiente forma. El Capítulo 2 presenta trabajos relacionados. El Capítulo 3 describe OntoForms y se abordan distintos aspectos claves de los componentes implementados así como decisiones de diseño. En el Capítulo 4 se describe el caso de uso de OntoForms en la aplicación OntoBreastScreen Finalmente en el Capítulo 5 se presentan las conclusiones del proyecto y posibles trabajos futuros.

Capítulo 2

Trabajos relacionados

Este capítulo presenta trabajos que aportan a la instanciación de ontologías a través de formularios web. En particular son de interés aquellos trabajos donde el formulario web para el ingreso de instancias a la ontología es generado de forma automática. Del relevamiento realizado pudimos identificar como relevantes los siguientes trabajos que se presentan a continuación: (i)Liu, (ii) Peng, (iii) Rocchetti y Labandera, (iv) Vcelak, (vi) Gonçalves et al., y (vii) Rutesic et al. El capítulo termina con una sección de conclusiones donde se presenta el espacio de oportunidades para la propuesta de un servicio de generación automática de formularios para crear instancias de ontologías.

2.1. Trabajo de Fangfang Liu (2009)

En el trabajo de FangFang Liu "An ontology-based approach to Automatic Generation of GUI for Data Entry" (Liu, 2009) se presenta la problemática de diseñar una interfaz gráfica de usuario (GUI por sus siglas en inglés) que sea altamente personalizada para ontologías de los dominios de meteorología y oceanografía. Dada la cantidad de datos específicos que se requieren intercambiar en estos dominios, los expertos requieren interfaces diseñadas específicamente para cada uno y eso genera una dificultad a nivel de desarrollo ya que el diseño de cada interfaz es costoso. El proceso se inicia con la exploración de un archivo de ontología, presentado en una vista en forma de árbol. Esto se representa visualmente mediante una jerarquía que muestra las clases y sus subclases, ver Figura 2.1. Esta representación jerárquica permite a los expertos navegar a través de las categorías de la taxonomía y comprender las relaciones entre las distintas clases. A medida que los expertos del dominio analizan la ontología, seleccionan los atributos necesarios para sus requerimientos específicos mediante un sistema de casillas de verificación. Esta información se almacena en un documento de metadatos en formato XML. Una vez completadas las selecciones, el sistema genera automáticamente formularios HTML, ver Figura 2.2.

Este enfoque presenta varias ventajas significativas:

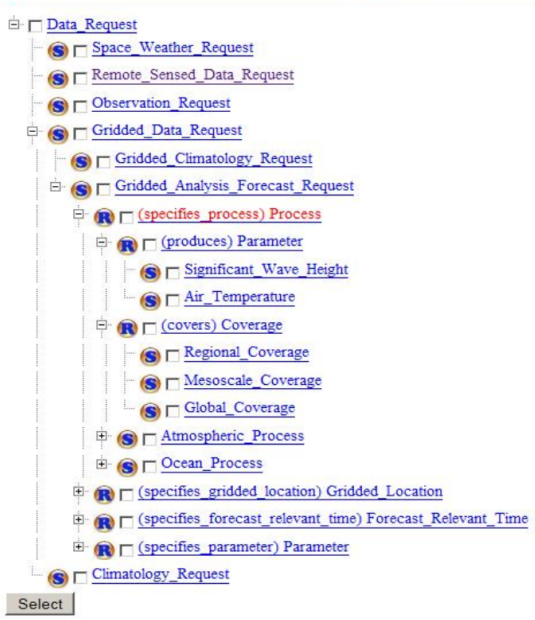

Figura 2.1: Vista de árbol.
(Liu, 2009)

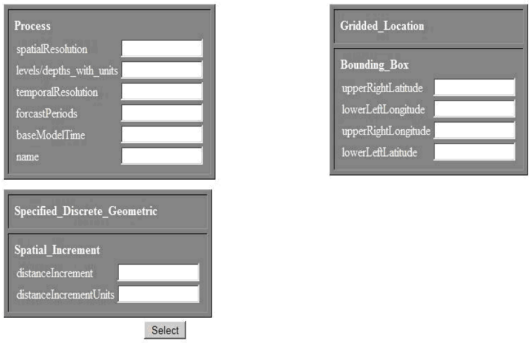

Figura 2.2: GUI generada por el sistema a partir de conceptos seleccionados.
(Liu, 2009)

- **Personalización sin programación:** Permite a expertos del dominio crear interfaces gráficas de usuario personalizadas sin tener que escribir una sola línea de código, lo que permite una personalización completa de los usuarios y sus requerimientos individuales.

- **Navegación intuitiva:** Se proporciona una vista de árbol que muestra

la jerarquía y las relaciones entre clases lo que ayuda a los expertos del dominio a navegar por el esquema de datos de una forma más comprensible.

- **Generación automática de formularios:** Dado que cada clase y atributo en la ontología está formalmente especificado, la generación de formularios se puede realizar de manera automática generando HTML correspondiente a los componentes, ahorrando tiempo y esfuerzo de desarrollo.

- **Prototipo:** Se utiliza el marco de trabajo provisto por Jena para el manejo de la ontología y se describen los algoritmos, esto ayuda a que los prototipos sean posibles de implementar en menor tiempo y con tecnologías actuales apartir de los algoritmos.

- **Adaptabilidad:** Este enfoque puede aplicarse a diferentes dominios como meteorología y oceanografío y otros.

A pesar de las ventajas, se identifican algunos desafíos en la implementación de este enfoque:

- **Dependencia del conocimiento del dominio:** Es fundamental que los expertos tengan un entendimiento sólido de la ontología para seleccionar correctamente los atributos necesarios. Un conocimiento limitado puede obstaculizar este proceso.

- **Complejidad en la integración:** La integración de diferentes sistemas y ontologías puede ser complicado, especialmente si hay discrepancias entre las definiciones y estructuras de datos en diferentes dominios. Se necesitan siempre al menos dos ontologías, una orientada al dominio y otra a la estructura del formulario, llamada .ºntología de nivel de tecnología"que asigna la ontología de dominio al esquema de datos subyacente donde se almacena la ontología.

- **Ambigüedad en la elección de clases:** Asociado al punto anterior, en la selección de clases y propiedades se puede llevar a una ambigüedad si no se establece claramente la relación entre diferentes elementos, lo que podría resultar en una GUI que no cumpla con las expectativas del usuario final.

- **Costo de desarrollo inicial:** Aunque el enfoque puede reducir los costos a largo plazo al facilitar la generación de GUIs personalizadas, el desarrollo inicial de la ontología para la estructura del formulario en el sistema de generación puede ser costoso y requerir mucho tiempo.

- **Sin ingreso de instancias:** Este trabajo se centra en dibujar el formulario pero no en instanciar la ontología con las instancias ingresadas por el usuario.

2.2. Trabajo de Peng et al. (2013)

Para ayudar a los expertos en el dominio de la fabricación (sin experiencia en ontologías) a proporcionar información de la calidad de la información sobre la capacidad del servicio de fabricación (MSC de sus siglas en inglés) utilizando el vocabulario de ontología, Peng et al. (Peng y Kang, 2013) proponen la arquitectura de interfaz de usuario de formulario dinámico extensible basado en ontologías (OXDF).

Hoy en día los MSC son publicados en las páginas de los proveedores o registrados en marketplaces. La información publicada en páginas web típicamente no es entendible por programas de computadora por lo que falta interoperabilidad. La información se puede extraer de estas páginas web mediante técnicas de extracción de datos pero a costo de perder precisión. Todos estos problemas pueden ser solucionados adoptando una ontología de dominio bien definida que permita a los proveedores individuales describir precisamente la información MSC sin ambigüedad y de manera que pueda ser entendible entre todos los interesados que compartan la ontología.

La estrategia principal del autor para la generación de formularios se basa en las siguientes tres innovaciones:

- **Navegación inteligente de onologías:** OXDF (ver figura 2.3 genera dinámicamente formularios y componentes de formulario a partir de las partes relevantes de la ontología según la información que el usuario desea ingresar. Esto permite a los usuarios navegar a través de una jerarquía de clases y seleccionar la que más se ajuste a sus necesidades.
- **Motor de búsqueda:** Se implementa un motor de búsqueda que ayuda a los usuarios a encontrar entidades relevantes dentro de la ontología, ya que las ontologías reales de este dominio son realmente extensas.
- **Mecanismo de actualización:** Permite a los usuarios definir nuevas terminologías y crear nuevas clases o propiedades en caso de que no se encuentren las adecuadas en la ontología existente. Esto asegura que el sistema sea flexible y evolutivo, adaptándose a las necesidades cambiantes de los usuarios.

Lo que se espera con el enfoque descrito, y que resulta atractivo para este caso, se resume en los siguientes aspectos.

- **Navegación intuitiva:** OXDF permite a los usuarios navegar interactivamente por la estructura de la ontología, creando formularios relevantes y elementos de formulario según las clases de la ontología, de modo que se pueda recopilar fácilmente la información específica. Si en la navegación el usuario no puede encontrar el concepto, puede buscarlo a través del asistente, que utiliza el motor de búsqueda inteligente o incluso crear el nuevo concepto.

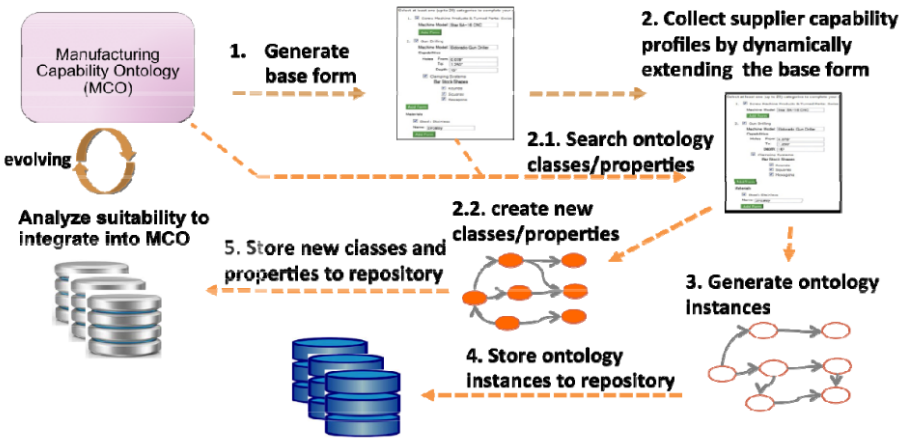

Figura 2.3: OXDF overview
(Peng y Kang, 2013)

- **Mantenimiento:** Permite a los usuarios crear conceptos completamente nuevos y agregar nuevas definiciones de clase y propiedad a la ontología, cambiando así la configuración del sistema para ciertos requisitos existentes y actualmente existentes.

- **Generación formularios:** Cada clase de la ontología de clases individuales se ve como una forma atómica, compuesta por el título de la clase más un conjunto de los atributos de la clase.

Por otro lado, se consideran las siguientes desventajas.

- **Curva de aprendizaje:** El conocer de ontologías, su sintaxis y semántica junto a sus usos son bastante sofisticados para los llamados usuarios sin experiencia y pueden presentar dificultades al momento de adoptar el enfoque.

- **Complejidad de la ontología:** Es natural que ontologías de dominio extensas y complejas planteen algunos desafíos para sus usuarios, ya que pueden parecer demasiado detalladas y abarcativas para que alguien pueda comprender el panorama completo y navegar a través de toda la estructura.

- **Sin configuración:** Este enfoque no permite configurar el formulario para seleccionar las clases o propiedades que se presentan u ocultan.

Por último, la mayor desventaja es que la implementación del enfoque no está disponible.

2.3. Trabajo de Rocchetti & Labandera (2016)

El trabajo de Néstor Rocchetti y Gonzalo Labandera titulado "Generación automática de formularios para el ingreso de datos en ontologías - Aplicación en la implementación de una ontología de percepciones de piezas de arte" (Rocchetti Martínez y Labandera, 2016) se centra en el desarrollo de una ontología que conceptualiza percepciones, medios de producción, soportes y familias de producción de piezas de arte en exposiciones. El objetivo principal del trabajo es facilitar el ingreso de datos a esta ontología mediante un generador automático de formularios.

Para la generación de formularios, se implementa una estrategia semiautomática a partir de una instancia de configuración previa, donde se definen las características del formulario a generar. Esta configuración incluye la elección de un concepto objetivo representando a la clase del individuo y los conceptos relacionados (llamados "anclas") representando a las propiedades que se utilizarán para construir los subformularios. Existen dos tipos de formularios: uno para ingresar nuevos individuos y otro para seleccionar individuos existentes a través de múltiple opción.

El generador se adapta a cambios en la ontología, lo que significa que si la estructura de la ontología se modifica, los formularios se ajustan automáticamente sin necesidad de recompilación de código.

Las ventajas que presenta este enfoque son:

- **Facilidad de uso:** Al utilizar formularios guiados, los usuarios que no están familiarizados con las ontologías pueden ingresar datos de manera controlada y sencilla, evitando la necesidad de interactuar directamente con la ontología.
- **Generación de preguntas contextualizadas:** Se implementa una estrategia que permite formular preguntas en lenguaje natural basadas en el contexto de los datos que se solicitan, facilitando así la comprensión de lo que se requiere al usuario.
- **Evolución:** El generador de formularios es capaz de adaptarse a diferentes ontologías, permitiendo su reutilización en otros dominios sin modificaciones significativas en el código.
- **Mantenimiento:** Los formularios se generan en tiempo de ejecución, lo que permite que cualquier cambio realizado en la ontología se refleje automáticamente ne los formularios, sin necesidad de recompilación de códi-

go. También el sistema permite la creación de formularios que pueden incluir tanto la entrada de nuevos individuos como la selección de opciones de un conjunto predefinido, lo que mejora la flexibilidad en la recolección de información.

Entre sus desventajas se encuentran:

- **Dependencia con la estructura de la ontología:** Cualquier cambio significativo en la ontología puede hacer que las configuraciones existentes de los formularios se vuelvan obsoletas, lo que requeriría una nueva configuración.
- **Necesidad de conocimiento previo:** Se podría decir que esta es una limitación importante, pues los usuarios administradores, quienes deben realizar la etapa de configuración, deben tener conocimientos de dominio tales como la ontología y sus conceptos.
- **Internacionalización:** El sistema está pensado principalmente para ontologías en español, lo que limita su aplicabilidad en contextos donde se necesiten otros lenguajes.
- **Reuso del prototipo:** El stack tecnológico del prototipo es antiguo y en la actualidad es difícil lograr ejecutarlo. Por otro lado, la interfaz gráfica se presenta desactualizada en cuanto a experiencia de usuario por lo que habría que actualizar componentes para mejorar el look & feel. Además, permite la inserción de instancias pero no la actualización de las existentes.

2.4. Trabajo de Vcelak et al. (2017)

El trabajo de Vcelak y sus colegas "Ontology-based Web Forms - Acquisition and Modification of Structured Data" (Vcelak, Kryl, Kratochvil, y Kleckova, 2017) presenta un método para la inserción y edición de datos de archivos RDF por medio de formularios web que se generan automáticamente de las ontologías.

La estrategia principal adoptada por los autores para la generación de formularios se fundamenta en la extracción automática de información de ontologías, utilizando específicamente el Web Ontology Language (OWL). En lugar de depender de un archivo de configuración externo, como en (Liu, 2009), los formularios se generan en tiempo de ejecución a partir de anotaciones integradas en las propias ontologías.Este enfoque garantiza que cualquier modificación en la ontología se refleje automáticamente en los formularios, permitiendo realizar cambios sin necesidad de una configuración manual. Los autores ponen en práctica un sistema que valida los datos que son introducidos por el cliente o por el servidor, de modo que todos los valores estén bien tipados y se ajusten a las restricciones de cardinalidad especificadas por la ontología.

Algunos de los beneficios que presenta este trabajo son:

- **Automatización de formularios:** Las actualizaciones automáticas de los formularios cuando cambian las ontologías son posibles desde el diseño ya que se crean según la necesidad utilizando las ontologías disponibles en tiempo de ejecución, sin necesidad de contar con una ontología específica para la estructura del formulario. Esto permite actualizaciones rápidas cuando las ontologías cambian. Por ejemplo: algunas configuraciones que se pueden realizar mediante anotaciones son el orden de las propiedades que se determina por una anotación de prioridad, o por la etiqueta o título del concepto, y por último el local-name o URI.

- **Validación de datos de entrada:** Hay una integración de mecanismos de validación de datos del lado del cliente y en el lado del servidor para asegurar que los datos ingresados cumplan con los tipos y las restricciones de cardinalidad especificadas en la ontología.

- **Configuración mínima:** Su intención es reducir la cantidad de trabajo de mantenimiento de formularios que se lleva a cabo manualmente al usar anotaciones en las ontologías para definir propiedades y restricciones. Anotaciones: rdfs: label, rdfs:domain y rdfs:range.

La mayor desventaja que presenta este trabajo es que aunque los autores dan una descripción clara del enfoque, la implementación de la solución no está disponible.

2.5. Trabajo de Gonçalves et al. (2017)

El trabajo de Gonçalves et al. “An ontology-driven tool for structured data acquisition using Web forms” (Gonçalves, Tu, Nyulas, Tierney, y Musen, 2017) fue concebido para generar formularios de UI para el dominio de la salud, aunque puede aplicarse a cualquier dominio [3]. El sistema propuesto tiene tres componentes principales: un archivo de configuración XML utilizado para la generación del formulario web, una ontología de dominio y un modelo de datos que describe el formulario. Este enfoque requiere un esfuerzo de configuración considerable por parte de un usuario administrador ya que además de la ontología de dominio es necesario proporcionar el archivo XML y también un modelo de datos para configurar el formulario. Esto también afecta la capacidad de mantenimiento de la solución ya que el archivo de configuración debe revisarse cuando la ontología de dominio evoluciona. Además, el formulario generado permite al usuario ingresar datos (instancias de ontología) pero no permite la actualización de los datos ya ingresados. Para utilizar la implementación provista es necesario actualizarla porque utiliza una tecnología antigua.

La arquitectura de la propuesta se muestra en la Figura 2.4 ilustrando las funcionalidades de representar, adquitir y consultar datos de un formulario. Este sistema cuenta con tres partes fundamentales, la primera un modelo de formulario que instrumenta la generación automática del formulario web, la segunda ontologías de dominio conteniendo la terminología para dar una descripción

formal a las entidades y la tercera un modelo de datos para la información ingresada que se encarga de vincular la información con la ontología del dominio y su terminología.

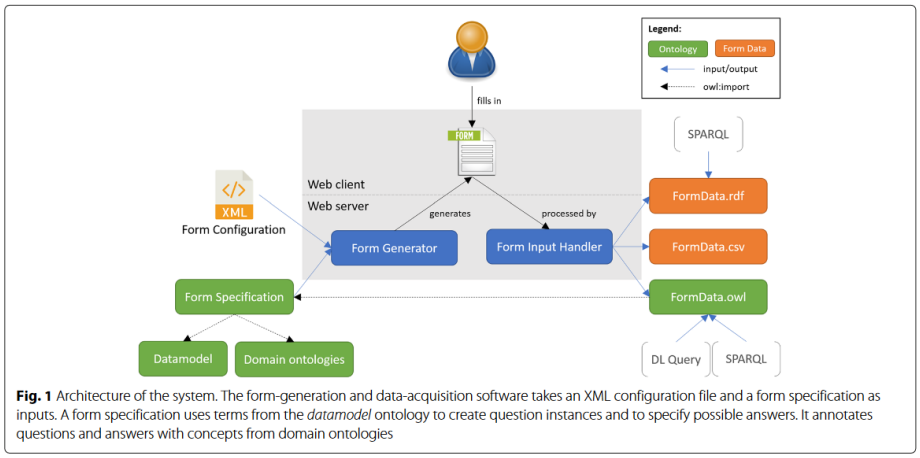

Fig. 1 Architecture of the system. The form-generation and data-acquisition software takes an XML configuration file and a form specification as inputs. A form specification uses terms from the *datamodel* ontology to create question instances and to specify possible answers. It annotates questions and answers with concepts from domain ontologies

Figura 2.4: Arquitectura del sistema - Goncalves (Gonçalves y cols., 2017)

A continuación se enumeran las principales ventajas de este enfoque.

- **Construcción de UI:** Flexibilidad para construir la interfaz gráfica ya que es de forma declarativa y se tienen muchas opciones para ”customizar.elementos.
- **Integración con ontologías**: Cada elemento de datos en los formularios está vinculado a descripciones detalladas en ontologías lo que proporciona un contexto y significado que no se obtienen con datos opacos.
- **Estructura de los datos recolectados:** Permite la recolección de los datos en un formato que está vinculado a una ontología por definición, facilitando así su análisis y consultas mediante SPARQL.
- **Evolución del sistema:** Se permite la Retro-compatibilidad ante cambios en ontología de formularios, las salidas se mantienen compatibles.
- **Flexibilidad y adaptabilidad:** Se tiene modelado un formulario genérico que puede extenderse para otros dominios más allá de la evaluación funcional clínica.

Las desventajas que presenta son:

- **Complejidad de configuración:** Si bien el enfoque es muy flexible, tiene un alto grado de configuración por parte de quien arma el formulario ya que debe proveer: DataModel, Domain Ontologies, Form Spec. Además el crear y configurar archivos XML para definir la estructura del formulario puede ser una barrera para usuarios sin conocimientos técnicos, dificultando la adopción del sistema.

- **Dependencia del modelado OWL:** La efectividad del sistema depende de la calidad y precisión de las ontologías utilizadas. Si las ontologías no están bien definidas pueden limitar la utilidad del sistema. Por ejemplo, la ontología que representa el formulario es algo extensa y se debe entender con exactitud como utilizarla y como proveer la ontología de dominio para lograr enriquecerla.

- **Prototipo:** La tecnología en la que está hecho el prototipo es antigua por lo que llevaría tiempo lograr una versión productiva, se debería reimplementar en un stack tecnológico reciente.

2.6. Trabajo de Rutesic et al. (2021)

Otro enfoque similar al de Gonçalves et al. es el propuesto por Rutesic et al., para el dominio de las pizzas (Rutesic, Radonjic-Simic, y Pfisterer, 2021). Además de la ontología de dominio FOODON, una ontología que describe la interfaz gráfica de usuario es un componente clave del enfoque. Por un lado, es una implementación simple que solo requiere cambiar la ontología de la interfaz cuando cambia la ontología de dominio, y que genera una interfaz amigable para el usuario. Además de ingresar nuevas instancias, esta interfaz permite agregar nuevas clases o propiedades a la ontología. Por otro lado, la configuración es demasiado manual para el usuario administrador, y la actualización de instancias no está resuelta de manera sólida. La implementación está disponible y utiliza una tecnología actualizada.

2.7. Conclusiones y oportunidades para nueva propuesta

En esta sección se realiza una comparación entre los trabajos relacionados presentados y se identifican oportunidades de desarrollo para un servicio de generación automática de formularios web para instanciar una ontología.

De los atributos mencionados de cada uno de los trabajos podemos observar que algunos son relevantes para usuarios, tanto administrativos (a cargo de gestionar las ontologías y la configuración de formularios) y los usuarios finales (usuarios que harán uso del formulario generado para el ingreso de datos), mientras otros atributos tratan acerca de la calidad de solución como componente

de software.

Para los usuarios administradores consideramos *esfuerzo de configuración* para evaluar cuan compleja es la tarea de configuración. Para el usuario final, el atributo de *usabilidad para edición de datos* es seleccionado para evaluar cuan amigable es el comportamiento de la interfaz de usuario en consideración si el formulario solo soporta insertar nuevas instancias o si además soporta la actualización de las mismas y si la solución implementa la generación automática de instancias de clases intermedias (clases intermedias en relaciones n-arias).
Otro atributo de calidad relevante para el usuario final, en particular para los usuarios expertos, es la capacidad de *agregar clases y propiedades*, el cuál permite a usuarios sin experiencia en Ontologías cambiar la estructura de la misma.
Respecto a la calidad de la solución, el atributo *mantenibilidad* es definido como "El grado de efectividad y eficiencia con el cuál un producto o sistema puede ser modificado para mejorarlo, corregirlo o adaptarlo a cambios en el ambiente y en requerimientos", en donde se incluye modularidad y reusabilidad del software. Además, es relevante el atributo de *actualización tecnológica* para evaluar cuán actualizado está la implementación del sistema con respecto al stack tecnológico.

Finalmente, se considera el atributo *uso de razonador* para medir si se utilizan los servicios de razonamiento para obtener los axiomas asociados a la ontología para obtener el formulario, además de afectar la calidad interna de la solución, esto puede influenciar en lo que se muestra para el usuario.

Teniendo en cuenta el análisis realizado, se desprende que la solución a plantear por el presente trabajo deberá considerar lo siguiente:

- Los enfoques en donde automatizan la recorrida de la Ontología para extraer los conceptos que forman parte del formulario son los más complejos. Siempre deberá haber una etapa mínima de configuración, aunque sea para definir la presentación en una interfaz de usuario del formulario. En este trabajo se plantea utilizar recorridas sobre la estructura de la ontología para poder recuperar los conceptos y sus relaciones para los cuales se deben generar los formularios (ver (Peng y Kang, 2013)) y luego excluir los conceptos que no se necesiten desde un menú de configuración. Se deberá definir convenciones que nos permitan reducir la cantidad de configuraciones a realizar aumentando el grado de automatización.

- Debemos permitir la visualización y edición de las instancias ya cargadas en el sistema.

- En todos los trabajos se presenta un componente encargado de automatizar la creación de componentes de interfaz gráfica a partir de un concepto. Por ejemplo: para relaciones de data-property debemos construir elemen-

tos HTML `<input>`, y para las object-properties construir elmentos de tipo combo-box con listados desplegables.

- Se debe tener una implementación de generación automática de formularios para una ontología que no dependa del un dominio sino que pueda extenderse sin costo a otros dominios.
- Los formularios deben ser generados en tiempo de ejecución y basados en la ontología. Esto asegura que el sistema se mantenga flexible y sea muy rápido introducir cambios.
- El sistema debe poder soportar internacionalización para manejar ontologías en múltiples idiomas.
- Debemos almacenar los datos recolectados y las propias ontologías en un formato RDF y en un triplestore para facilitar su acceso, análisis y consultas mediante SPARQL.
- El uso del sistema debe abstraer al usuario final del uso de ontologías, debe ser totalmente transparente y no necesitar modificar una línea de código para generar un nuevo formulario.
- Implementar un prototipo con tecnologías actuales para ejemplificar el uso de un sistema de estas características con un stack actualizado y en un caso de uso real de la actualidad.

En el siguiente capítulo se describe la contribución de este trabajo para resolver esta problemática planteada.

Capítulo 3

OntoForms

Este capítulo describe la solución realizada para generar de forma automática formularios de ingresos de datos a ontologías de dominios. El capítulo se organiza de la siguiente manera: en la primera sección se describe "OntoForms", la solución desarrollada.
La segunda sección explica en profundidad los componentes de "OntoForms Core API", haciendo foco en el algoritmo de generación de formularios, el uso de Jena como marco de trabajo y la integración con razonadores externos.
En la tercera sección se presenta un vistazo sobre el administrador web, los casos de uso que implementa y las funcionalidades que brinda al administrador, haciendo uso de una ontología de ejemplo.
En la cuarta sección se explica el caso de estudio "OntoBreastScreen: sistema recomendador de estudios preventivos para el cáncer de mama", se explica cómo se integró a OntoForms y el diseño de la solución de ese cliente.

3.1. Descripción general de la Arquitectura

Esta sección demuestra la arquitectura de la solución "OntoForms" y cómo se vincula ese componente de software con las aplicaciones clientes. "OntoForms" mejora la mantenibilidad de las aplicaciones basadas en ontologías, dado que cuando los requerimientos del dominio cambian, las ontologías evolucionan, entonces la interfaz gráfica de las aplicaciones pueden ser regeneradas, minimizando los cambios en el código fuente. Además del usuario final, quien hace uso de la aplicación para cumplir su rol en el dominio, esta solución también considera al usuario administrador, quien está encargado de actualizar ontologías, generar los formularios correspondientes y configurarlos si es necesario.

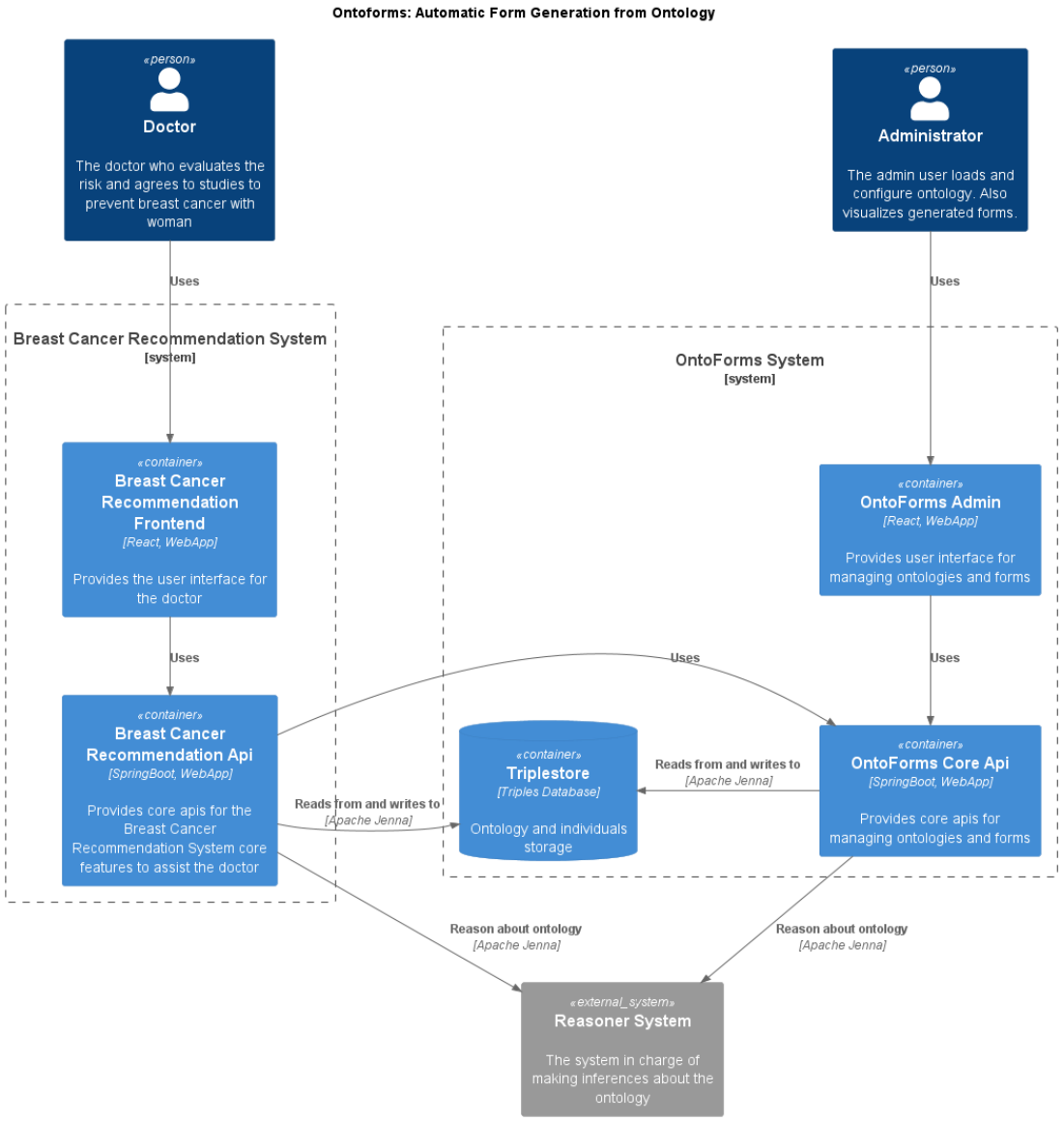

Figura 3.1: Arquitectura de OntoForms integrada al cliente OntoBreastScreen

En la figura 3.1 se ilustra la solución construida sobre una arquitectura cliente-servidor en tres capas, en donde se cuenta con una aplicación central "OntoForms Core Api" que tiene la responsabilidad de proveer los servicios para gestionar ontologías y configuraciones de formularios, además de obtener los formularios disponibles para una ontología. Estos servicios son utilizados por un usuario administrador vía "OntoForms Admin", un Administrador Web que hace uso de "OntoForms Core Api" para implementar las funciones de exploración de cada ontología cargada, previsualizar los formularios de ingreso de individuos para cada clase, así como también realizar personalizaciones sobre los formularios. Para almacenar las ontologías cargadas por los usuarios y las configuraciones realizadas para formularios se utiliza un Triplestore (*Apache Jena Fuseki servidor SPARQL*, s.f.).

La parte izquierda de la figura muestra un sistema aplicado a un dominio, el cual es operado por los usuarios finales. Aunque la implementación de la aplicación de dominio puede resolverse utilizando distintos estilos o patrones de arquitectura, para el caso de uso de la aplicación "OntoBreastScreen", se implementa una arquitectura en capas, con un frontend "Breast Cancer Recommendation Frontend" y un backend "Breast Cancer Recommendation Api", que utiliza los servicios provistos por "OntoForms Core Api" para presentar la interfaz para ingreso de individuos a la ontología. "OntoForms Core Api" provee una estructura de formulario para que el frontend la utilice y pueda dibujar su interfaz, a partir de la ontología de dominio. Una vez que el usuario ingresa los datos en el formulario, la aplicación de dominio es la encargada de implementar la estrategia de guardado de los mismos, sea en memoria o en el triplestore. Esto se decidió así, porque hay aplicaciones que manejan datos sensibles y necesitan estrategias especializadas para el manejo de datos personales.

La principal novedad de este trabajo es el uso del algoritmo de generación de formularios que es parte de "OntoForms Core Api", y el uso de razonadores externos integrados a la solución de generación de formularios y también a la aplicación de dominio para dar respuestas más completas, incluyendo el conocimiento que infiere el razonador.

3.2. Core Api & Web Administrator

Como se describe en la sección anterior, el sistema que implementa la solución de OntoForms está compuesto por una Api de backend, un administrador web y un almacenamiento para las ontologías.

Ontoforms Core Api está implementado con tecnología del stack de Java, utilizando el marco de trabajo de Spring Boot 3 y la versión 17 de Java. Esta aplicación web está organizada con un diseño en capas, API, servicios y persistencia. La Figura 3.2 muestra los principales componentes de la aplicación.

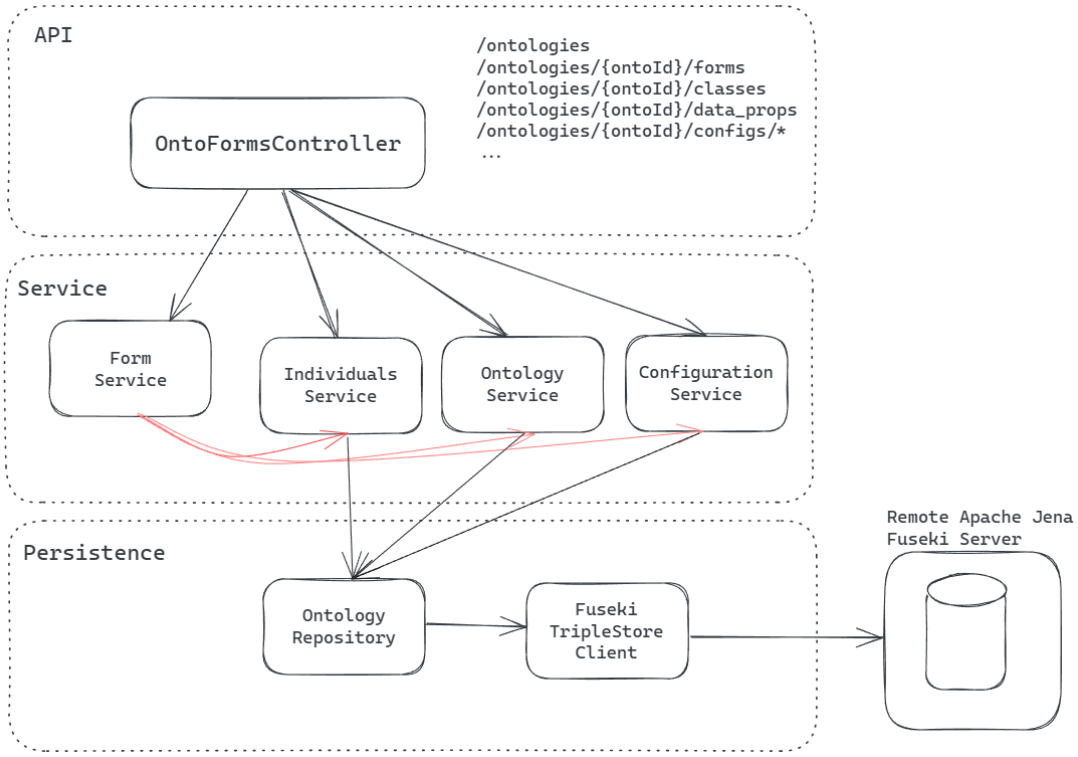

Figura 3.2: Vista de componentes de Ontoforms Core API

Este stack tecnológico fue elegido por la madurez del mismo y la versatilidad en las posibilidades de soluciones. En este proyecto utilizamos el spring-boot-starter-web que nos provee todos los módulos necesarios para poder levantar un servidor web Tomcat embebido en el jar generado por el proceso de compilación de la aplicación. Esto asegura un proceso de desarrollo y despliegue muy rápido. Además, se seleccionó la biblioteca de generación de código Lombok (*Project*

Lombok, s.f.) que apoya con el código repetitivo y nos permite centrarnos en la lógica de negocio propiamente dicha, acelerando el desarrollo y haciendo el código más mantenible y legible.
La aplicación se encuentra disponible en GitHub en los siguientes enlaces:
`https://github.com/brunoszilagyiibarra/ontoforms-backend` y
`https://github.com/brunoszilagyiibarra/ontoforms-admin`

Para el manejo de ontologías se utilizó **Apache Jena** ya que es un marco de trabajo que nos provee un soporte completo para trabajar con datos enlazados, no solo con ontologías. Entonces tenemos un soporte para RDF, SPARQL y ontologías en OWL; proporciona funcionalidades para almacenar, consultar y manipular ontologías y grafos de conocimiento. Uno de los principales requisitos de OntoForms es la integración con razonadores. En este sentido, Apache Jena incluye un motor de inferencia que soporta varios tipos de razonamiento sobre ontologías, y también soporta extensiones con razonadores externos como Open-Pellet. Otro requerimiento relevante es la capacidad de generar nuevas instancias y poder almacenarlas en la ontología. Jena provee un módulo para almacenar y gestionar grandes volúmenes de datos RDF en triplas mediante su triplestore TDB que es muy eficiente.Por último, Apache Jena tiene un SDK disponible para Java y, por tanto, se integra de manera natural al stack tecnológico utilizado en esta solución. Este SDK es modular, lo que permite seleccionar qué componentes utilizar según las necesidades y no importar módulos que no se necesitan. Por ejemplo, utilizar el módulo del motor de inferencias o solo el triplestore.

Los casos de uso que implementa la aplicación de OntoForms son los siguientes

- Cargar ontología en el sistema.
- Explorar ontología visualizando sus clases, propiedades e individuos.
- Explorar detalle de una clase de la ontología.
- Generar formulario de ingreso de individuo para una clase de la ontología.
- Configurar personalización de formulario
- Asociación de formulario con la aplicación dominio.

A continuación se describe el diseño e implementación de los casos de uso.

3.2.1. Cargar Ontología en el sistema

El objetivo de este caso de uso es que un usuario administrador pueda cargar y persistir ontologías en el sistema, permitiendo su uso y consulta posterior. El caso de uso se ejemplifica en la Figura 3.3.

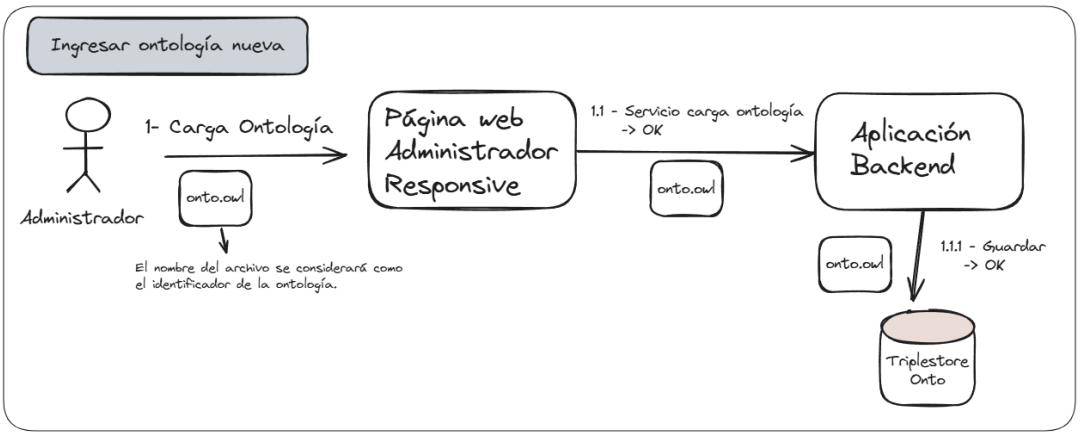

Figura 3.3: Caso de uso: Cargar Ontología

Dado que se requiere que se almacene la ontología para poder utilizarla en otros casos de uso, el triplestore se debe integrar al sistema.Investigando las distintas opciones de TripleStore, se llega a que Apache Jena Fuseki es una buena opción, ya que está integrado al SDK de Jena, tiene un rendimiento muy bueno para un gran volumen de tuplas y presenta un administrador web para realizar operaciones básicas.
Dado que en el triplestore se almacenan ontologías de dominio pobladas con individuos (Tbox y Abox) y configuraciones del formulario, se decide estructurar el repositorio con los datasets “/ontologies”, “/individuals” y “/configs”. En “/ontologies” e “/individuals” se almacenan Tbox y Abox respectivamente. Separar Tbox de Abox tiene ventajas en lo que respecta a la organización y el mantenimiento, puesto que la estructura de las ontologías está pensada para que vaya evolucionando, agregándose nuevas versiones, sin embargo, no se quiere perder los datos, que deben estar siempre disponibles. Además, tener los datos en un dataset separado facilita el reuso de los mismos con varias versiones de las ontologías.

La creación y configuración de los datasets generalmente se realiza una única vez. Sin embargo, en la etapa de desarrollo se utilizan varios ambientes volátiles, lo cual agrega una carga extra de configuración de los ambientes. Esto también puede suceder en producción ante nuevos ambientes, en su primer uso.
Para reducir el costo de esta tarea manual, se implementó en el proyecto un me-

canismo automático que se dispara al iniciar la aplicación y tiene como objetivo realizar la verificación de la existencia de estos datasets en el triplestore configurado; en caso de no existir, este mecanismo los genera. Para esto se utilizó la API de Fuseki server-protocol - DataSet and services (*Fuseki Server Protocol*, s.f.), que se puede ver en el Anexo A, en la Implementación A.1.

La carga de las ontologías en el dataset "/ontologies" presenta el desafío de que debemos almacenar múltiples ontologías y posiblemente algunas tengan colisiones en las uris. Esto se podría dar cuando los usuarios quieren subir varias versiones de la misma ontología. La solución debe soportar estas colisiones, es decir, cada definición de ontología no debe interferir con las demás.Para resolver este requisito se define que cada ontología sea almacenada en un grafo con nombre dentro del dataset. Dado que esta solución prioriza, entre otros, el atributo de calidad de usabilidad de la aplicación para el usuario administrador, se opta por no solicitarle al usuario datos que pueden calcularse, por lo que se decide que el identificador de la ontología sea el nombre del archivo con el cual se sube al sistema. Este identificador va a ser el nombre del grafo a utilizar para almacenar el TBOX (ver ejemplo en la figura 3.4)

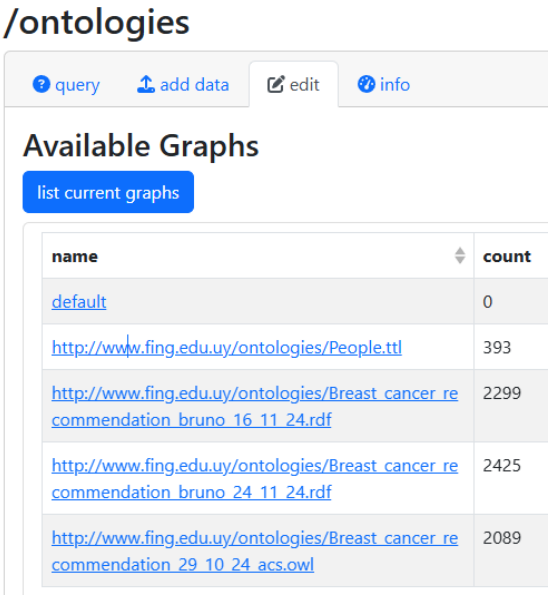

Figura 3.4: Triplestore Ontology DataSet named graphs

La solución se implementa solamente para los formatos RDFXML, TURTLE, NTRIPLES y JSONLD; cualquier otro formato será rechazado con un mensaje de formato no válido al usuario.
A su vez, se configura el servidor para aceptar ontologías de hasta 100MB, y

haciendo uso del formato HTTP MultiPart en la API de OntoForms hacemos un uso eficiente en la transmisión permitiendo guardar sin problemas este tipo de ontologías.

Se implementa una sección de manejo de ontologías que le presenta al usuario administrador un listado de ontologías cargadas en el sistema y un componente file-chooser para poder seleccionar un archivo local y realizar la carga (ver figura 3.5). Se brinda feedback al usuario tanto por éxito como por error (ver figura 3.6).

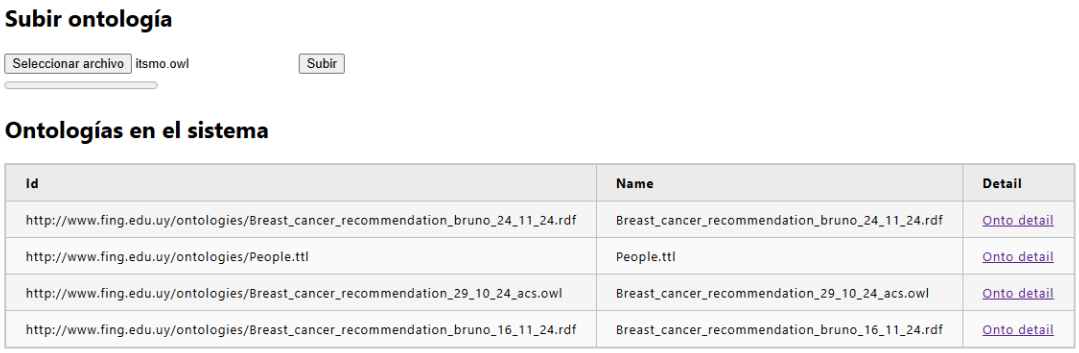

Figura 3.5: Carga ontología en administrador

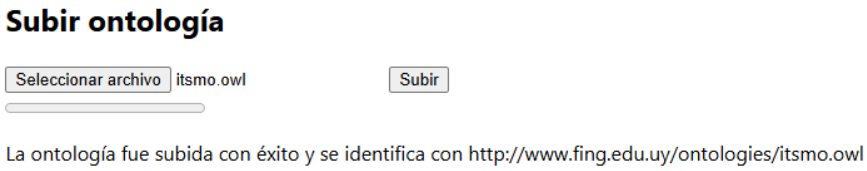

Figura 3.6: Carga ontología con éxito

En la implementación de la carga de la ontología, se construyó un cliente que denominamos “FusekiTripleStoreClient”, el cual implementa las operaciones de manejo del repositorio mediante la SDK de Apache Jena y el módulo de triplestore. En este caso, para cada operación generando una conexión RDFConnectionFuseki al host y dataset, y luego usando la interface de RDFConnection para ejecutar las operaciones.

3.2.2. Explorar las ontologías cargadas

El objetivo de este caso de uso es que un usuario administrador pueda visualizar para cada ontología cargada un detalle de la misma al interactuar con ella. Esta exploración es interactiva, el usuario puede navegar entre las clases, propiedades e individuos de la ontología. El caso de uso se ejemplifica en la figura 3.7.

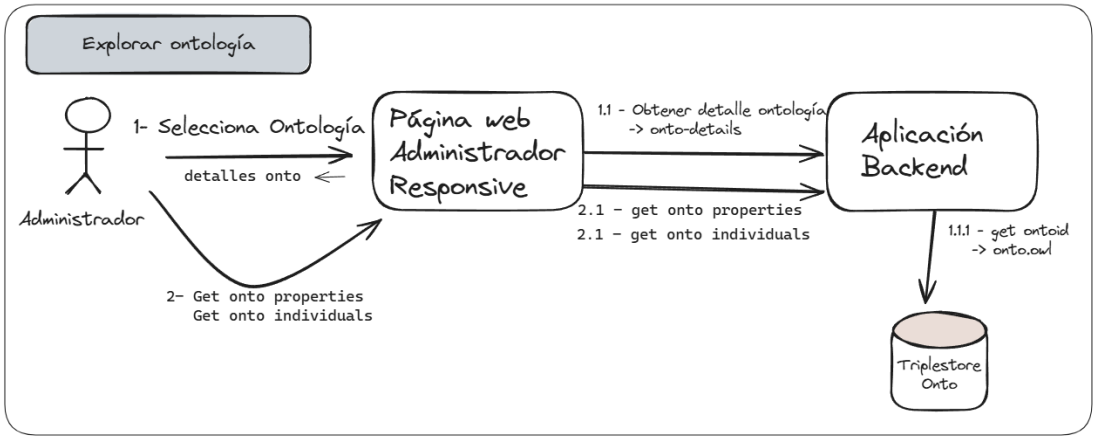

Figura 3.7: Caso de uso: Explorar Ontología

Se busca que el usuario administrador pueda visualizar todo los datos de la ontología, y para ello distinguimos cuatro secciones de interés, las clases de la ontología, las data properties, las object properties y los individuos.
El primer paso para esto es que el usuario tenga la posibilidad de visualizar todas las ontologías cargadas en el sistema y pueda seleccionar una para obtener el detalle de la misma. En "OntoForms Core Api" se implementó una funcionalidad para acceder al tripleStore y al dataset de "/ontologies" para listar todos los grafos con nombre, eso representa a todas las ontologías cargadas en el sistema. El código de ejemplo se puede ver en A.2. Esta funcionalidad se disponibiliza al administrador mediante la implementación de un panel en forma de tabla (debajo del componente para cargar ontología) donde se muestra ese listado y se agrega por cada fila un CTA (call to action) para visualizar el detalle de la ontología seleccionada (ver figura 3.8).

Ontologías en el sistema

Id	Name	Detail
http://www.fing.edu.uy/ontologies/Breast_cancer_recommendation_bruno_24_11_24.rdf	Breast_cancer_recommendation_bruno_24_11_24.rdf	Onto detail
http://www.fing.edu.uy/ontologies/People.ttl	People.ttl	Onto detail
http://www.fing.edu.uy/ontologies/Breast_cancer_recommendation_29_10_24_acs.owl	Breast_cancer_recommendation_29_10_24_acs.owl	Onto detail
http://www.fing.edu.uy/ontologies/itsmo.owl	itsmo.owl	Onto detail
http://www.fing.edu.uy/ontologies/Breast_cancer_recommendation_bruno_16_11_24.rdf	Breast_cancer_recommendation_bruno_16_11_24.rdf	Onto detail

Figura 3.8: Panel de visualización ontologías

Una vez que el usuario selecciona una ontología cargada para explorar, se utilizan varios servicios de "OntoForms Core Api" para poblar la interfaz gráfica. Se utiliza un servicio para obtener las clases de la ontología, otro para las data-properties, las object-properties y por último uno para los individuos. La obtención de individuos de la ontología se implementa obteniendo la ontología guardada en el triplestore que corresponde al identificador de ontología seleccionado y luego esa ontología se carga en un objeto `OntModel` utilizando la SDK de Jena. Sobre este objeto se ejecuta la operación de listar individuos, y posteriormente a una proyección de estos objetos a un modelo para la vista es que obtenemos los resultados. Esta funcionalidad se puede ver en la Figura 3.9

Detalle de la Ontología

Classes | Data properties | Object properties | Individuos

uri	Label
http://www.semanticweb.org/mdebe/ontologies/example#Mary_Doe	Mary_Doe
http://www.semanticweb.org/mdebe/ontologies/example#Nick_Carraway	Nick_Carraway
http://www.semanticweb.org/mdebe/ontologies/example#Jay_Gatsby	Jay_Gatsby
http://www.semanticweb.org/mdebe/ontologies/example#Sarah_Doe	Sarah_Doe
http://www.semanticweb.org/mdebe/ontologies/example#Daisy_Buchanan	Daisy_Buchanan

Figura 3.9: Explorar ontología - ver individuos

Un punto importante de esta solución es que el modelo ontológico se construye con un razonador asociado, Open-Pellet, que más adelante se detalla.

Para la sección de propiedades de una ontología, se utiliza un enfoque similar; se tiene un servicio dedicado para las data-properties y otro para las object-properties. En ambos servicios, se obtiene la ontología del triplestore y se carga como un modelo ontológico de Jena que usa el razonador Open-Pellet. Para las data-properties se utiliza la funcionalidad de `OntModel.listDatatypeProperties()` del modelo y para las object-properties la función `OntModel.listObjectProperties()`. El valor agregado está en el post-procesamiento, una vez tenemos estos listados de propiedades, para agregar los atributos de nombre propiedad, dominio y rango. Para definir el valor de estas etiquetas se implementó el patrón Visitor para el procesamiento de los distintos tipos de nodo RDF, como se muestra en la Figura 3.10.

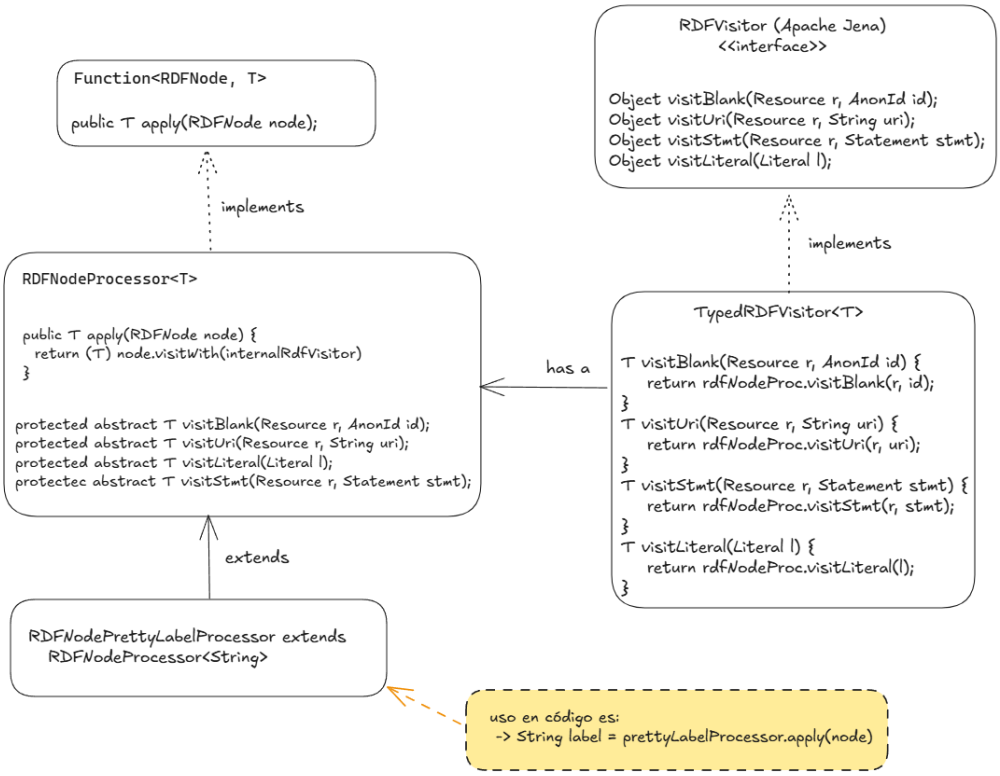

Figura 3.10: RDF Processor - Visitor pattern

Los lineamientos de la implementación para resolver la etiqueta mostrable dado un nodo, son los siguientes:

- Para los literales se devuelve el texto del literal.

- Para las Uris se intenta primero obtener el valor de rdfs:label, en caso de que no exista esta anotación se utiliza el localName del nodo.
- Para los statements se utiliza el nombre local del predicado.
- Para los Blank, se verifica si es una expresión de tipo Union, Intersection, Restriction u OntProperty, y se realiza una implementación particular para cada una, recorriendo sobre la estructura. Si no es ninguna de las anteriores se devuelve el localName.

Además de estas reglas, para el caso en que el dominio o rango no estén definidos (o sea, rango/dominio global) se define la etiqueta "«Undefined»".
En la figura 3.11 se muestra la interfaz gráfica que maneja el administrador para visualizar las data-properties y en la figura 3.12, la sección donde visualiza las object-properties.

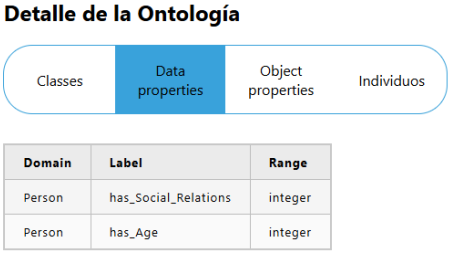

Figura 3.11: Visualizar data-props ontología

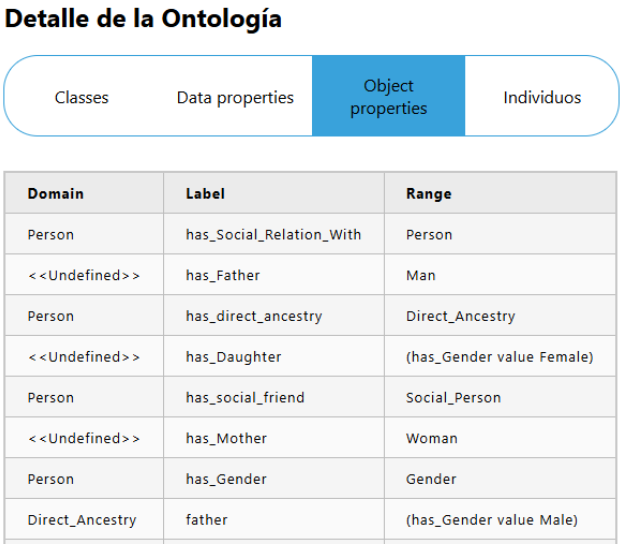

Figura 3.12: Visualizar obj-props ontología

Por último, la sección más compleja es la visualización de clases de la ontología. El enfoque directo sería listar todas las clases utilizando una función similar a la anterior `OntModel.listClasses()`, pero este enfoque terminaría en una lista no jerarquizada de las clases, resultando confusa la visualización en ontologías con gran número de clases. En esta implementación se sigue el mismo enfoque de (Liu, 2009), que presenta la navegación de los conceptos en forma de árbol. Se implementa el modelo de árbol que se muestra en la Figura 3.13.

El desafío se encuentra entonces en cómo armar el árbol, jerarquizando las clases de la ontología y definiendo la estructura del árbol. Para realizar la re-

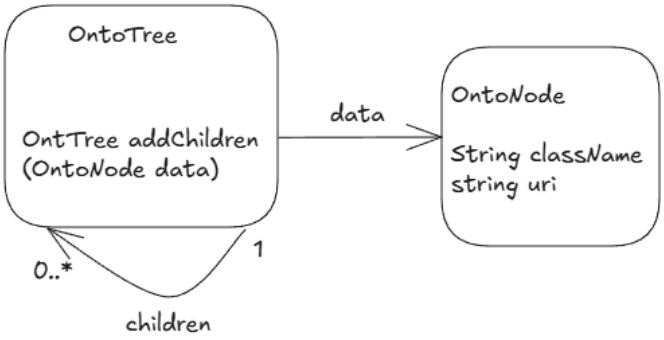

Figura 3.13: Modelo de Árbol clases ontología

construcción de este árbol en base a las clases de la ontología se implementa un algoritmo recursivo con memoria, que comienza con un árbol que tiene únicamente la clase OWL.Thing. Se obtienen las clases equivalentes y las clases de primer nivel, y por cada clase de primer nivel se agrega un nodo y se procesan sus hijos. El Algoritmo 1 describe los pasos que se siguen.

Algorithm 1 Obtener las clases de nivel superior de la ontología

Input: *ontoId* (identificador de la ontología)
Output: *resTree* (árbol de clases)

```
1: ontoModel ← obtenerModelo(ontoId)
2: resTree ← nuevoArbol(OWL.Thing)
3: equivClasesMap ← obtenerMapaClasesEquivalentes(ontoModel)
4: clasesPrimerNivel ← obtenerClasesPrimerNivel(ontoId)
5: orderedPrimerNivel ← ordenarClasesSegunEtiqueta(clasesPrimerNivel)

6: for all clazz in orderedPrimerNivel do
7:     subTree ← agregarNodo(resTree, clazz)
8:     agregarSubClasses(clazz, subTree, equivClasesMap)
9: end for
10: Return: resTree
```

Durante la implementación de este algoritmo se buscó que las ontologías se visualicen de forma similar a la interfaz de la herramienta Protégé. Las principales dificultades fueron el manejo de clases equivalentes y clases definidas con expresiones. También presentó algo de dificultad la implementación de la función para obtener las clases de primer nivel, ya que hubo que excluir casos especiales como clases de intersecciones o clases con equivalencias. Por otra parte,

todas las etiquetas agregadas como dato a cada nodo del árbol se calcularon a partir del componente `RDFNodePrettyLabelProcessor` mencionado anteriormente.

Del lado del administrador web, también hubo que implementar una recorrida recursiva de la estructura para dibujar la interfaz ; el resultado se puede ver en la figura 3.14. Cada componente del árbol está pensado para que al hacer click dispare un CTA, el mismo se describe en la próxima sección.

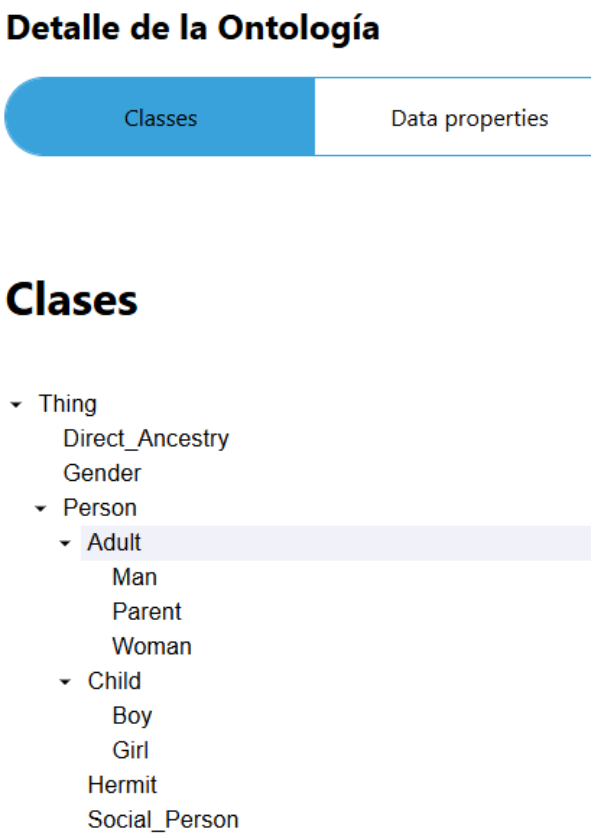

Figura 3.14: Modelo de Árbol clases ontología

3.2.3. Explorar clase de ontología

El objetivo de este caso de uso es que un usuario administrador pueda visualizar el detalle de una clase de la ontología al interactuar con ella en el árbol de clases de la ontología. Esta exploración es interactiva para que el usuario pueda

navegar entre las clases de la ontología, y pueda previsualizar la información de interes. El caso de uso se ejemplifica en la figura 3.15.

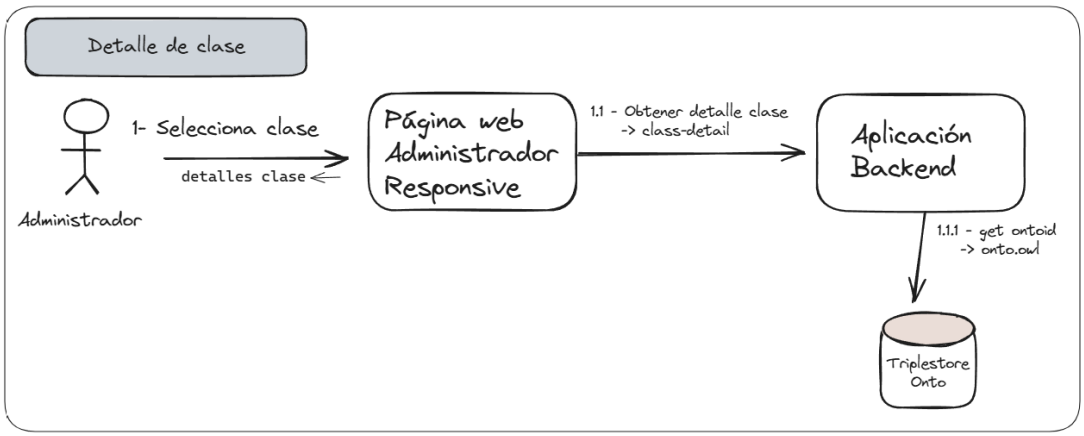

Figura 3.15: Caso de uso: Detalle de clase en ontología

Este caso de uso aunque parece simple es uno de los bloques fundamentales del trabajo para poder resolver la creación del formulario. La dificultad radica en dada una clase entender que significa que una propiedad pertenezca a una clase y luego cuales serían los métodos posibles para encontrar todas ellas.
Intuitivamente todas las propiedades de una clase las pensamos como aquellas cuyo dominio es la propia clase o una expresión que la involucre.

Recuperar estas relaciones usando SPARQL implica desarrollar consultas con lógica muy compleja.
Como es descripto en (*RDF as Frames*, s.f.), una de las consecuencias del paradigma de mundo abierto es que no es posible responder a la pregunta ¿Que propiedades pueden ser aplicadas a recursos de la clase X? Estrictamente, la respuesta a nivel de RDF es çualquier propiedad". Pero para responder esta pregunta en la cuál las propiedades tienen una clase particular en su dominio, se debe tener en cuenta también las propiedades que no tienen su dominio definido, y propiedades que tienen jerarquia de propiedades.

RDF as Frames

Apache Jena presenta esta noción en el apartado de RDF as Frames (*RDF as Frames*, s.f.), en donde se muestra la complejidad de trabajar con datos RDF por la flexibilidad que este lenguaje permite al ser válido por ejemplo una propiedad *edad* ser aplicada a cualquier recurso, sin limitarse solamente a recursos de tipo Persona.

La idea planteada es basarse en la propiedad rdfs:domain para poder restringir las aplicaciones de las propiedades a solamente los recursos de esa clase de dominio. Se definen entonces las propiedades de una clase como solo aquellas propiedades que no agregan ninguna información de tipos una vez que se aplican a los recursos que ya se sabe que son de dicha clase, como se ilustra con el siguiente fragmento:

```
<rdfs:Class rdf:ID="LivingThing" />

<rdfs:Class rdf:ID="Animal">
  <rdfs:subClassOf rdf:resource="#LivingThing">
</rdfs:Class>

<rdfs:Class rdf:ID="Mammal">
  <rdfs:subClassOf rdf:resource="#Animal">
</rdfs:Class>

<rdf:Property rdf:ID="hasSkeleton">
  <rdfs:domain rdf:resource="Animal" />
</rdf:Property>
```

Listing 3.1: Código XML de Ontología Ejemplo

para la pregunta, ¿hasSkeleton es una de las propiedades de Animal?
La respuesta: Sí, porque cualquier recurso de Animal debería tener la propiedad hasSkeleton. Entonces, similarmente, cualquier recurso que es un Mammal también es un animal, entonces hasSkeleton también es propiedad de Mammal. Sin embargo, hasSkeleton no es propiedad de LivingThing porque no se sabe automáticamente que LivingThing es un animal, podría ser otra cosa. Si LivingThing tuviera la propiedad hasSkeleton, entonces agregaríamos información de tipo agregando la inferencia de que LivingThing es también un Animal. Para expresiones más complejas en el dominio, se debe revisar las inferencias que surgen de la expresión de dominio. Por ejemplo, para una expresión de intersección en el dominio, A intersección B, se hace una demostración en donde considerar que una propiedad con intersección como dominio pertenece a las propiedades de A o B resulta falso. En cambio, si la expresión es de unión, entonces sí la propiedad puede ser considerada tanto de A como de B.
Como las sub-propiedades heredan las restricciones de dominio de sus padres, entonces las propiedades de una clase también tienen en consideración las sub-propiedades de aquellas propiedades que ya sean parte del conjunto.
En el caso de propiedades globales, o sea, aquellas que no definen el dominio, se considera que pertenecen al conjunto de propiedades de la clase.

Obtención de propiedades

Para que las funciones asociadas con la vista de frames de una clase devuelvan los resultados de forma completa dado un modelo ontológico es necesario que se le pueda anexar un razonador. En el caso del modelo para TBOX, ana-

lizando los disponibles en la documentación, el razonador que más se ajusta a nuestras necesidades ha sido el perfil OWL_MEM_TRANS_INF dado que es para modelos almacenados en memoria lo cual hace muy rápido manejar ontologías pequeñas a medianas y las consultas de inferencias pueden realizarse sin necesidad de acceso a disco. Además, es un razonador eficiente en la inferencia sobre propiedades transitivas (como rdfs:subClassOf o rdf:type). Utiliza la transitividad para propagar información de manera efectiva a través de las relaciones de la ontología. Este razonador soporta todas las características de OWL Full, lo cual lo hace bastante versátil.

Para entonces obtener las propiedades de una clase se carga el OntoModel desde el repositorio triplestore con un el razonador vinculado a los TBOX (perfil OWL_MEM_TRANS_INF) y luego se utiliza la funcionalidad de Jena para obtener la vista de Frames `class.listDeclaredProperties()`. El ejemplo se puede visualizar en el siguiente fragmento 3.2

```
1 public List<OntProperty> getPropertiesByOntoIdAndClassUri(String ontoId, String
      classUri) {
2     OntModel model = getOntologyModelForTBoxById(ontoId);
3     return model.getOntClass(classUri).listDeclaredProperties()
4             .filterKeep(p -> p.isObjectProperty() || p.isDatatypeProperty())
5             .toList();
6 }
```

Listing 3.2: Obtener propiedades Frame-Like de una clase

Notar que todas las funciones de listado de Apache Jena retornan un ExtendedIterator, lo cuál implementa el patrón Iterator y brinda facilidades para recorrer el listado de resultados.
Acá hacemos uso de esto, y filtramos los resultados para solo quedarnos con las propiedades object-properties y data-properties. Luego ejecutamos la función `iter.toList()` para volver al mundo de las listas built-in de java.
Todas estas propiedades son transformadas a un modelo descriptor (DTO) en el cuál utilizamos el componente `RDFNodePrettyLabelProcessor` mencionado anteriormente para obtener las etiquetas de nombre, rango y dominio.

Obtención de individuos

Para obtener los individuos de una clase se necesita un modelo ontológico enfocado en realizar inferencias en ABOX, mediante pruebas hemos visto que el perfil utilizado para TBOX (perfil OWL_MEM_TRANS_INF) no fue muy efectivo ya que hubo muchas inferencias que no lograba realizar a nivel de las instancias. Fue por esto que buscamos integrar un razonador externo open-pellet al sistema. La integración fue bastante directa, solo necesitamos agregar la dependencia `implementation 'com.github.galigator.openllet:openllet-jena:2.6.5` y luego utilizar el perfil `PelletReasonerFactory.THE$_$SPEC` al cargar el modelo Jena de la ontología.

Para obtener los individuos de una clase, se utiliza el selector `model.listIndividuals(class).toList()` y luego realizamos la correspondencia contra nuestro descriptor con el `RDFNodePrettyLabelProcessor`.

GUI Administrador

En el administrador web, se presenta el árbol con todas las clases de la ontología en donde se puede expandir y/o contraer aquellas clases que presenten sub-clases, y al presionar sobre cada nodo del árbol se ejecuta un CTA que procesa y muestra los detalles que corresponden a la clase almacenada en el nodo.
Se puede ver el resultado a nivel de la interfaz del administrador en la siguiente figura 3.16

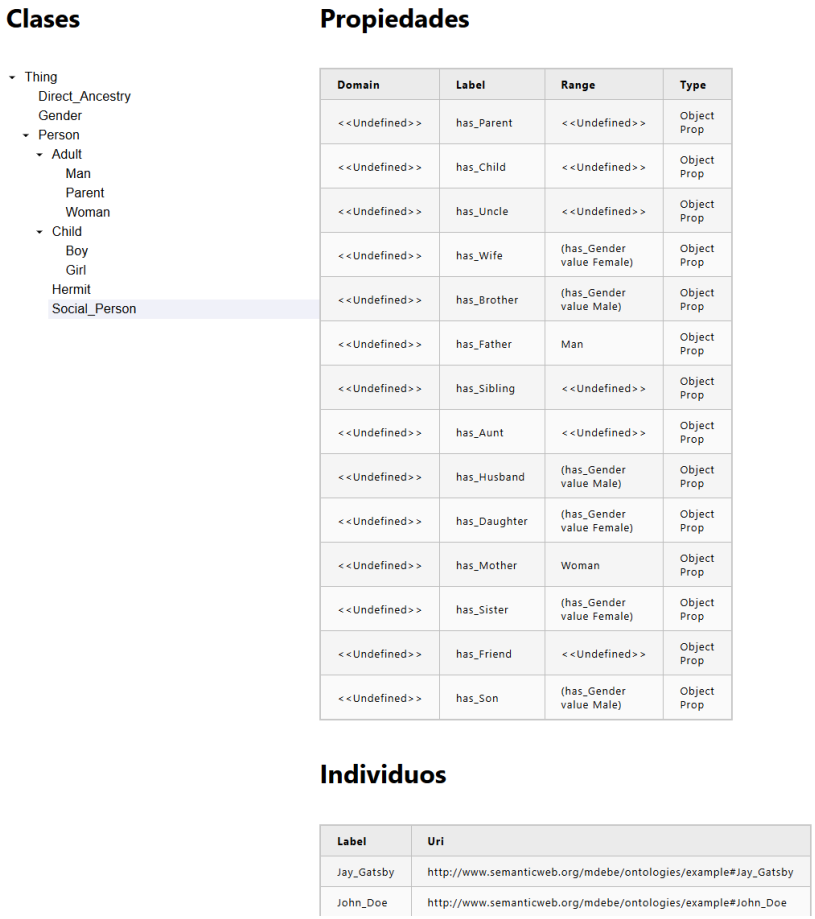

Figura 3.16: Propiedades e Individuos de una clase

3.2.4. Previsualizar formulario de clase

El objetivo de este caso de uso es que el usuario administrador pueda explorar para alguna ontología cargada todas las clases de la misma y los formularios de ingreso de individuos asociados a cada clase. Esta definición se puede ver en la siguiente figura 3.17

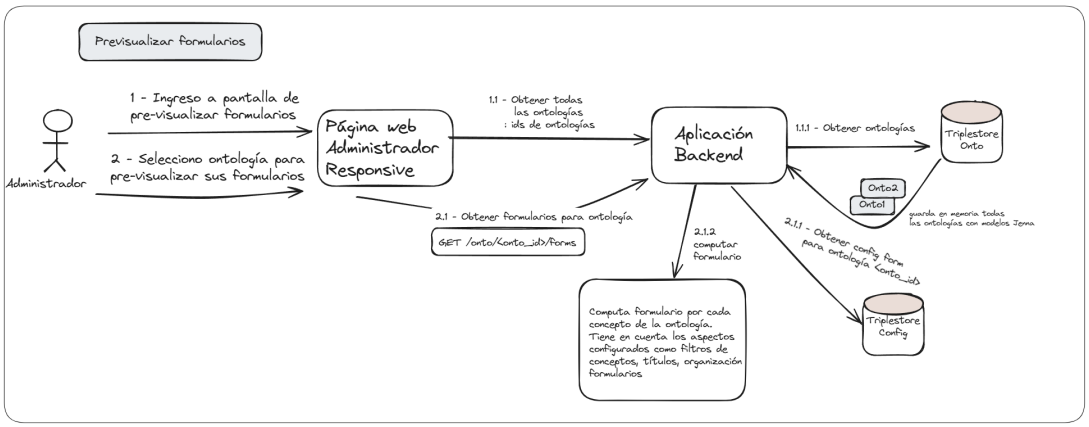

Figura 3.17: Caso de uso: Previsualizar formulario de clase

Este caso de uso fue construido a partir de todas las funciones desarrolladas anteriormente. Como se muestra en la figura, el primer paso del caso de uso es presentarle al usuario administrador un listado de ontologías cargadas en el sistema en forma de listado para que él pueda seleccionar una. Esta funcionalidad re-utiliza la función de explorar ontologías cargadas. La vista del administrador se puede ver en la siguiente figura 3.18

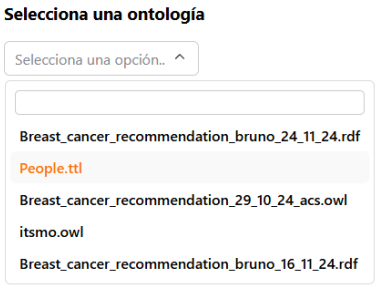

Figura 3.18: Listado de ontologías

La selección de ontología tiene un CTA para disparar el cálculo del árbol de clases asociado a la ontología seleccionada, esto habilita al usuario administrador a realizar una navegación interactiva para poder seleccionar cada clase de interés y poder previsualizar el formulario. Se muestra en la siguiente figura 3.19

Figura 3.19: Arbol de clases de la ontología

Una vez seleccionada la clase, se realiza una llamada al servicio de formularios de OntoForms Core Api pasando el identificador de la ontología y la uri de la clase seleccionada (llamada "main class"). El resultado corresponderá a un subgrafo de la ontología, accesible desde la "main class.ª través de las propiedades declaradas explícitamente así como también de las propiedades inferidas por el razonador.
La instanciación del formulario de esta clase es implementada utilizando la vista de RDF as Frames de Jena y complementando con configuraciones para personalizar el formulario. Básicamente involucra el llenado de todas las data-properties y object-properties en conjunto a los posibles valores de individuos para cada una. El sub-grafo es computado con un algoritmo recursivo usando una recorrida depth-first en la estructura de clases, similar a la recorrida de un spanning tree en el grafo de la ontología.

El algoritmo 2 ilustra el procesamiento de la ontología realizado por OntoForms Core API. El algoritmo inicia con la "Main Classz continúa construyendo una o más secciones que organizan una colección de campos. Los componentes definidos en el algoritmo 2 son los siguientes:

- $\mathcal{O}$: Representa a la ontología de dominio.
- $\mathcal{O}^{\models}$: Denota el cierre deductivo de $\mathcal{O}$.
- C: Refiere a la main class.
- **SubC** $= \{SC_1, \ldots SC_n\}$: Representa el conjunto de subclases derivadas de C.

- **Props** = $\{p_1, \ldots p_m\}$: Constituye al conjunto de object o data properties asociadas con el dominio de C o cualquier superclase de C.
- $Range(p)$: Indica el rango de la propiedad p
- $Form$: Describe la estructura del formulario generado por $\mathcal{O}^{\models}$.
- $Section$: Define la estructura que organiza campos y subsecciones en el formulario.

Algorithm 2 Generación de formulario a partir de clase

Input: $\mathcal{O}^{\models}$, C
Output: $Form$
  SubC ← obtain subclasses of C
  if SubC is empty **then**
    Props ← obtain properties with domain C
    for all p_i in **Props** $(1 \leqslant i \leqslant m)$ **do**
      if p_i is not calculated **then**
        if p_i is a data property **then**
          $Section$ ← add a new field
        end if
        if p_i is an object property **then**
          if $Range(p_i)$ is not empty and p_i is transparent **then**
            $Section$ ← add OntoForms($\mathcal{O}^{\models}$, $Range(p_i)$)
          else
            $Section$ ← add a new field
          end if
        end if
      end if
    end for
    $Form$ ← add $Section$
  else
    for all SC_i in **SubC** $(1 \leqslant i \leqslant n)$ **do**
      $Form$ ← add OntoForms($\mathcal{O}^{\models}$, SC_i)
    end for
  end if
  return $Form$

El algoritmo 2 es ejecutado de forma recursiva al definir cada subclase como clase padre hasta que la clase más específica es alcanzada (recorrida en profundidad). Estas subclases se corresponden con una nueva sección en el formulario, la cuál contiene un conjunto de campos para ingresar los valores de las data y object properties asociadas con esta clase.

Una descripción de las características en alto nivel del algoritmo es la siguiente:

- Por cada data-property se crea un nuevo campo en el formulario (o sección) para ingresar el dato.
- Por cada object-property se crea un campo en el formulario (o sección) en donde se selecciona uno o varios individuos (depende de si la relación es funcional o no). Los individuos son cargados por el algoritmo utilizando la búsqueda de individuos para la expresión del rango.
- Si la clase del rango ha sido configurada como "transparente", por ejemplo, no relevante para al usuario, en vez se generan campos nuevos y una sección en donde se agregan todas las propiedades de la clase "transparente". Se puede ver como si se estuviera generando un nuevo individuo de la clase transparente y se estuviera incrustando su formulario en una sección del formulario original.

Como nota adicional, recordar que por la definición de RDF as Frames, si la expresión del dominio de alguna de las propiedades contiene a la clase principal pero es parte de una expresión más compleja, puede ser que esa propiedad no aparezca en el formulario como parte de las propiedades de la clase principal, depende de si agrega información de tipos o no.

La salida del algoritmo se representa en JSON, en donde las aplicaciones cliente son encargadas de interpretarlo y generar la UI más adecuada para mostrarlo. Un ejemplo (recortado) de esta salida se puede ver en A.3 y la interfaz gráfica generada en el administrador se muestra en la figura 3.20.

3.2.5. Personalización de formularios

En el armado de los formularios se sigue la estrategia de convención antes que configuración. Esto ayuda a que el administrador pueda previsualizar formularios, no necesitando realizar configuraciones manuales y solo lo haga cuando la aplicación de ese formulario para determinada aplicación así lo requiera.

Las configuraciones que se brindan a través del administrador web son las siguientes:

- Cambio en textos sobre campos formularios
- Clases para ingreso de nuevos individuos (clases transparentes)
- Ocultar propiedades
- Aplicaciones vinculadas.

Las configuraciones se necesitan persistir para que las mismas perduren en el tiempo y puedan ser fácilmente recuperables. En este trabajo se decidió persistir estas configuraciones en el triplestore utilizado, en el dataset /config"siguiendo

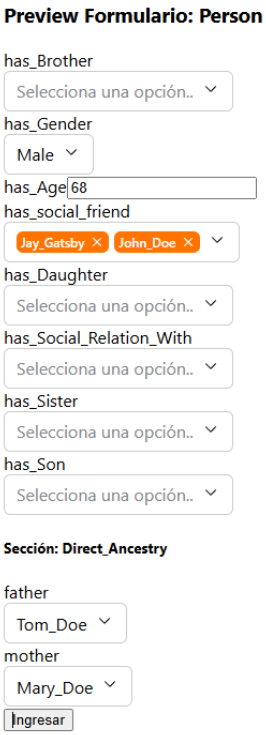

Figura 3.20: UI Administrador para previsualización formulario generado

la estrategia de tener un grafo con nombre por ontología.
Para estructurar la configuración en cada grafo con nombre se armó una ontología que la describa; en la siguiente figura 3.21 se puede ver un esquema de la estructura.

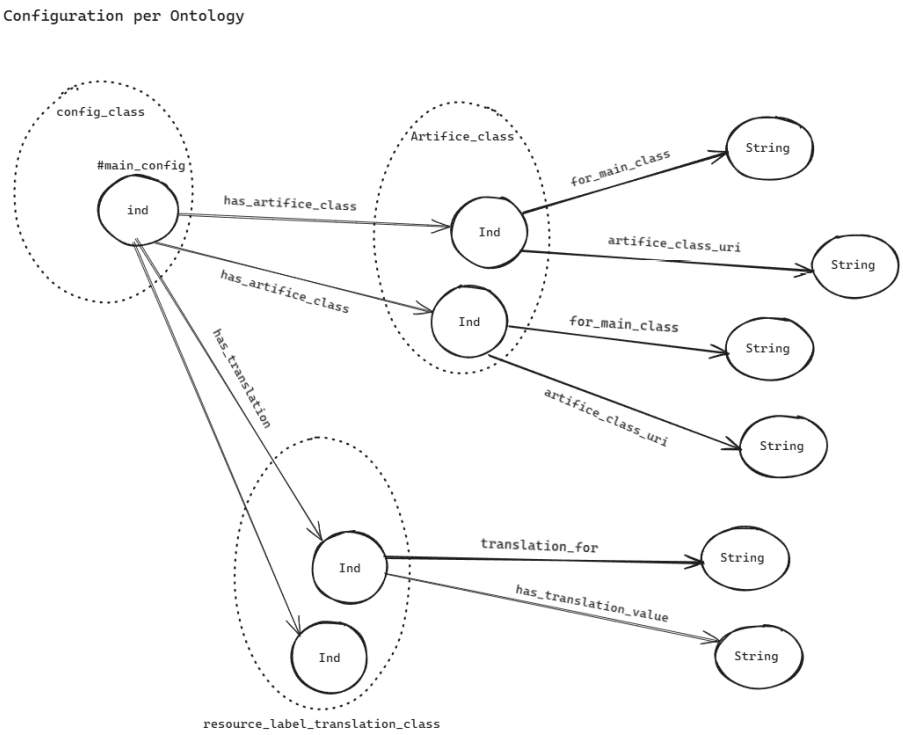

Figura 3.21: Ontología de configuración

Texto en campos formulario

Sucede que los campos del formulario están etiquetados con las labels definidas en la ontología y esto puede no ser del todo claro para el usuario que utilizará el formulario. Es por eso que el administrador debe tener la posibilidad de modificar el texto que se muestra sobre los campos del formulario. Para hacer esto, el usuario administrador debe brindar una URI que corresponde a la URI de la propiedad y luego una traducción. Esto se guarda en la ontología de configuración asociada a la ontología seleccionada y se utilizará al momento de construir el formulario. En la siguiente figura 3.22 se puede visualizar un ingreso de esta configuración y el posterior resultado sobre el formulario 3.23.

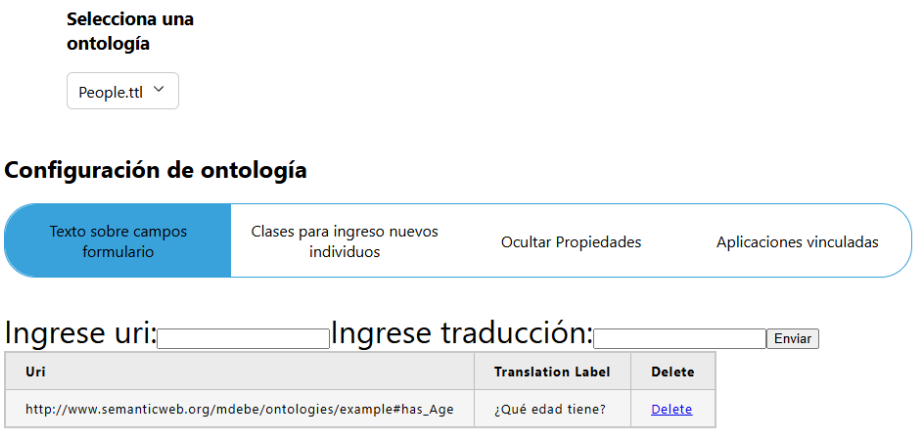

Figura 3.22: Configuración de traducciones

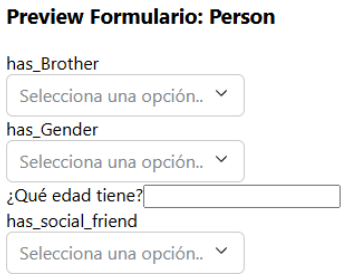

Figura 3.23: Formulario con textos configurados

Por otro lado, la configuración persistida para este caso se puede visualizar con el siguiente fragmento turtle 3.3

```
<http://www.ontoforms.com/config#main_config>
rdf:type <http://www.ontoforms.com/config> ;
<http://www.ontoforms.com/config/has_translation>
    [ rdf:type
        <http://www.ontoforms.com/config/resource_label_translation_class>
        ;
      <http://www.ontoforms.com/config/has_translation_value>
             " Que  edad tiene?" ;
      <http://www.ontoforms.com/config/translation_for>
             "http://www.semanticweb.org/mdebe/ontologies/example#has_Age"
    ] .
```

Listing 3.3: Código de configuración textos

Clases transparentes

Esta configuración surgió por la necesidad de no depender que existan instancias pre-definidas en la ontología para ciertas clases que agrupan configuraciones. Por ejemplo, si se tiene una clase Persona que tiene una object-property "tiene_edad"donde el rango de la misma es una clase Edad que a su vez tiene una data-property representando la edad y otras relaciones, entonces no sería deseado tener que vincularnos en el formulario con instancias de la clase Edad, sino que transparentar esa clase e incluir todas sus propiedades en una sección para completar una nueva instancia dinámicamente.

En la siguiente figura 3.24 se puede visualizar el ingreso de esta configuración, en donde agregamos la URI de la clase que queremos transparentar y una main-class uri para que la clase sea transparente pero solo al construir el formulario de la main-class. En la siguiente figura se muestra el resultado del formulario generado con una clase transparente representada como sección 3.25 El código 3.4 demuestra como queda la configuración para clases transparente persistida en el repositorio.

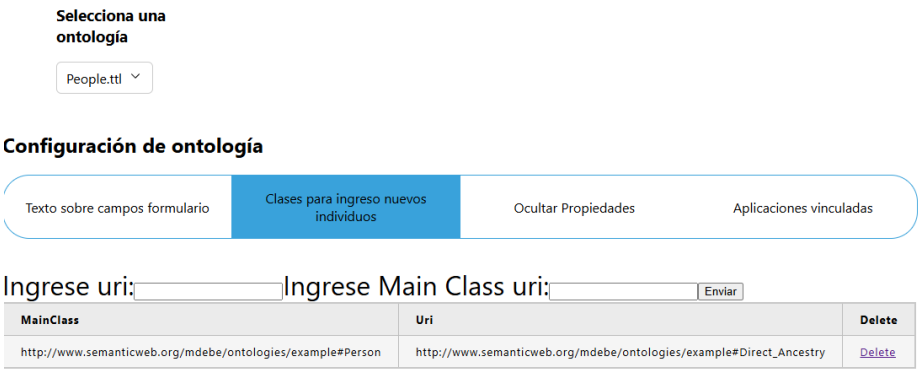

Figura 3.24: Configuración de clases transparentes

```
<http://www.ontoforms.com/config#main_config>
  rdf:type <http://www.ontoforms.com/config> ;
  <http://www.ontoforms.com/config/has_artifice_class>
    [ rdf:type <http://www.ontoforms.com/config/artifice_class> ;
      <http://www.ontoforms.com/config/artifice_class_uri>
             "http://www.semanticweb.org/mdebe/ontologies/example#Direct_Ancestry" ;
      <http://www.ontoforms.com/config/for_main_class>
             "http://www.semanticweb.org/mdebe/ontologies/example#Person"
    ] ;
```

Listing 3.4: Código de configuración clases transparentes

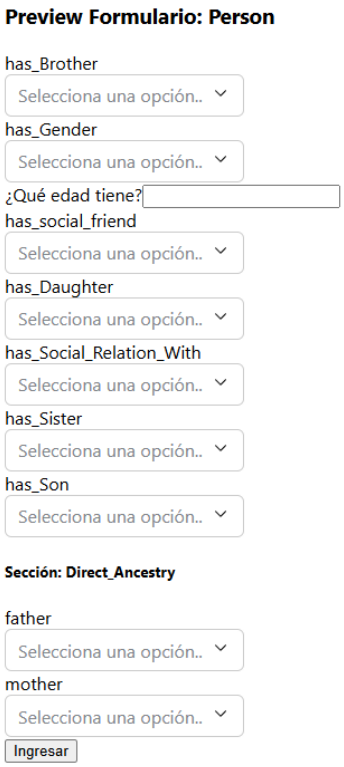

Figura 3.25: Formulario con clase transparente

Ocultar propiedades

Dada la estrategia de generación de formulario por defecto que es generar el mismo a partir de una Main-class y recopilar todas las propiedades asociadas a ésta, puede suceder que algunas propiedades no sean de interes para que se ingresen por un usuario final, por ejemplo propiedades calculadas por el razonador u propiedades que no apliquen al caso de uso.
Es por esto que se require tener la posibilidad de poder configurar propiedades a ocultar según la main-class seleccionada. Para esto se tiene el panel de la figura 3.26 en donde se puede ingresar la uri de una clase y la main-class para configurar el ocultamiento en el formulario generado.

El formulario resultante luego de aplicar el ocultamiento de las propiedades es el que se muestra en la figura 3.25 y un fragmento del código asociado se puede ver en 3.5

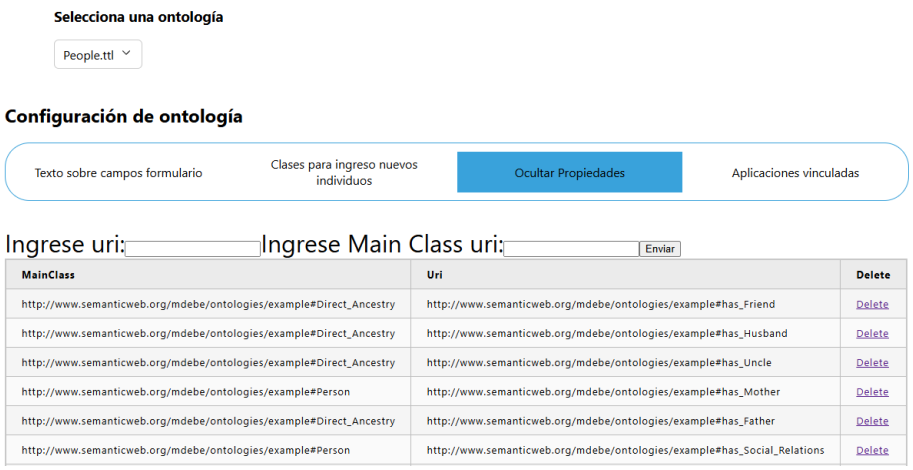

Figura 3.26: Configuración de props a ocultar

```
<http://www.ontoforms.com/config#main_config>
  rdf:type <http://www.ontoforms.com/config> ;
  <http://www.ontoforms.com/config/has_calculated_property>
    [ rdf:type <http://www.ontoforms.com/config/calculated_property_class> ;
      <http://www.ontoforms.com/config/calc_prop_for_main_class>
              "http://www.semanticweb.org/mdebe/ontologies/example#Direct_Ancestry" ;
      <http://www.ontoforms.com/config/calc_prop_uri>
              "http://www.semanticweb.org/mdebe/ontologies/example#has_Uncle"
    ] ;
```

Listing 3.5: Código de configuración ocultar propiedades

Asociar ontología a aplicación

Para que las aplicaciones que usan OntoForms utilicen formularios de forma dinámica, cada aplicación se identifica por un nombre de aplicación. En nuestro caso el nombre de la aplicación de ejemplo será .ºntoBreastScreen". El objetivo es poder vincular un identificador de ontología para el uso en la aplicación y entonces que la aplicación cliente lo pueda obtener y utilizar en los servicios subsecuentes como en la generación de formulario. Con esto se logra cambiar o actualizar la ontología que las aplicaciones utilizan de forma centralizada y sin líneas de código en las aplicaciones cliente aumentando así la flexibilidad y mantenimiento del sistema.

La configuración que se le presenta al usuario administrador es directamente el nombre de la aplicación, ver figura 3.27. Al ingresar el nombre de la aplicación, esta app se vinculará con la ontología seleccionada. Notar que podemos tener varias aplicaciones utilizando la misma ontología, pero cada aplicación solo puede usar una ontología.

A nivel de la configuración persistida, se puede ver como queda en el siguiente código 3.6.

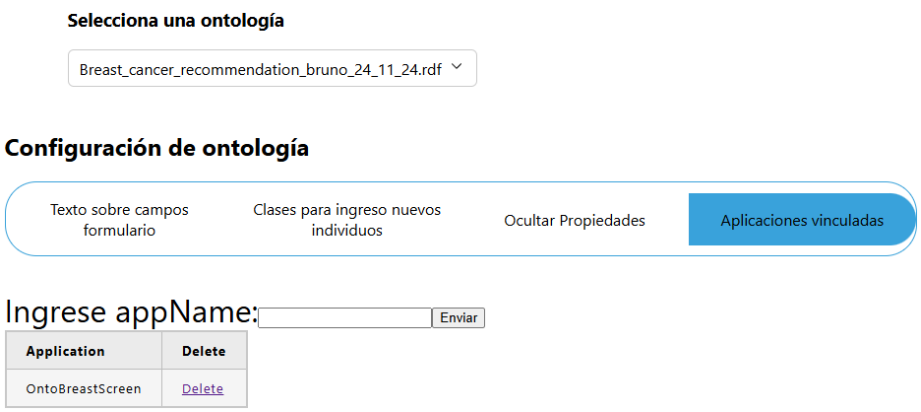

Figura 3.27: Configuración para vincular ontología a aplicación

```
<http://www.ontoforms.com/config#main_config>
  rdf:type <http://www.ontoforms.com/config> ;
  <http://www.ontoforms.com/config/has_app_onto_mapping>
    [ rdf:type <http://www.ontoforms.com/config/app_onto_mapping_class> ;
      <http://www.ontoforms.com/config/has_related_app>
              "OntoBreastScreen"
    ] .
```

Listing 3.6: Código de configuración vincular ontología a App

Capítulo 4

OntoBreastScreen: Recomendador de estudios médicos preventivos para cancer de mama

La importancia de detectar y prevenir enfermedades en una etapa temprana no puede ser negada y resulta aún más importante para enfermedades oncológicas. En los últimos años se ha producido un cambio significativo en las estrategias de prevención. Desde enfoques estandarizados hasta otros más personalizados, que se centran en los perfiles de riesgo de los pacientes y preferencias. Para el cáncer de mama, las estrategias se centran en estimar el riesgo de que una mujer desarrolle la enfermedad aplicando diferentes modelos de riesgo (Kim y Bahl, 2021) y realizando pruebas de detección según directrices de reconocidas organizaciones de salud (Cardoso y cols., 2019).

El caso de estudio consiste en desarrollar una aplicación OntoBreastScreen basada en la ontología BRCS-Onto (Anchén, Rohrer, y Motz, 2024). Este diseño de la aplicación provee una forma estructurada y estandarizada para representar y organizar el conocimiento acerca del riesgo de cáncer de mama y los estudios. Usando la ontología como modelo conceptual para la aplicación, habilita tener flexibilidad para futuras extensiones. Este enfoque también tiene la ventaja de que la consistencia del modelo puede ser verificada por un razonador ante cualquier cambio en la ontología.
Esta ontología interactúa con las calculadoras de riesgo basadas en diferentes modelos o guías y cada una puede estar potencialmente basada en distintos datos de entrada y metodologías. La aplicación permite a los médicos utilizar diferentes calculadoras de riesgo en un marco de trabajo unificado para obtener recomendaciones de estudios específicas y acordes a las guías de recomendación

reconocidas por las instituciones de salud internacionales.
Adicionalmente, la aplicación de frontend será en parte generada utilizando el sistema de generación automática de formularios de OntoForms.

4.1. Descripción general de la arquitectura y componentes

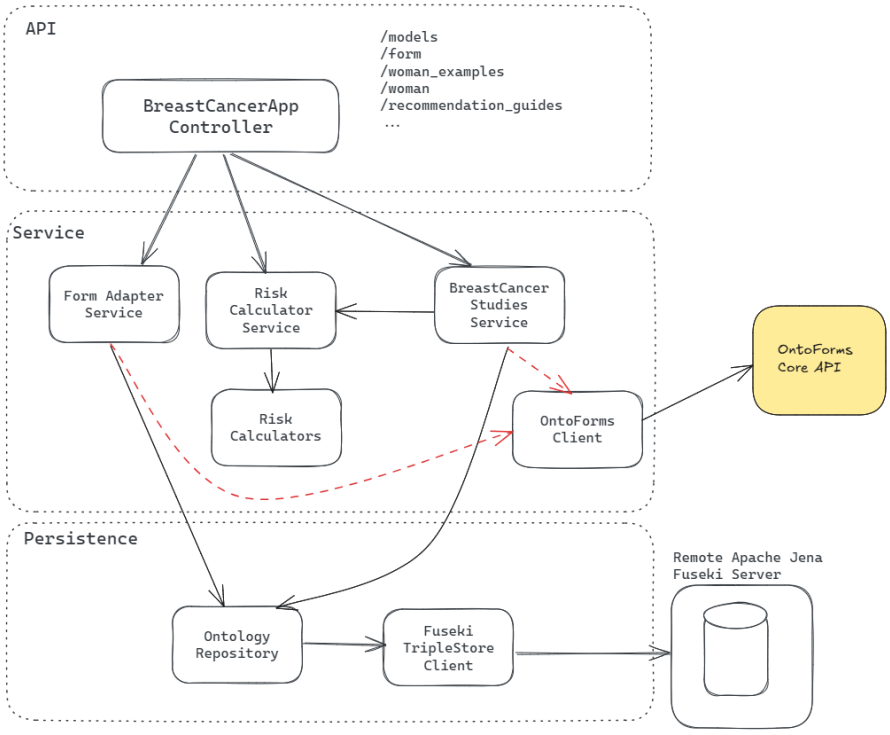

Figura 4.1: Diseño de componentes OntoBreastScreen API

Basado en el diseño propuesto en la Figura 3.1 para aplicaciones cliente de Ontoforms, se utilizará una arquitectura cliente-servidor con un frontend web que será la interfaz con el usuario final con el objetivo de visualizar e interactuar con el formulario presentado para ejecutar el caso de uso planteado.
Para el backend se utiliza el framework de Spring Boot y Java 17 como stack principal, por las mismas razones que se argumentaron en la construcción de OntoForms. Para el frontend se utiliza el stack de Node.js con React para tener los componentes más actuales.

El diseño interno de este componente de OntoBreastScreen API es el que podemos ver en la Figura 4.1.

En este diseño se distingue el módulo de calculadoras de riesgo, en donde se implementa un patrón strategy para la implementación concreta de cada calculadora: IBIS, ACS y MSP UY. Estas calculadoras y su implementación serán detalladas más adelante.
También para resolver gran parte de los casos de uso se tiene comunicación mediante un cliente HTTP con OntoFormsCore API. El resto de las operaciones se resuelven con lógica aplicativa orientada al negocio de la aplicación, como por ejemplo el cómputo de las recomendaciones dado el perfil de riesgo del paciente.

Dado el uso pretendido para la aplicación, se decide hacer una Single Page Application en donde cada pantalla pertenece a un flujo de pantallas y da la sensación de que "van pasando hacia adelante o retrocediendo". Esto asegura un uso y entendimiento sencillo. (ver Figura 4.2)
Por otro lado, se planteó que el sistema fuera internacionalizable para soportar tanto inglés como español. Esto se implementó por medio de los mecanismos usuales de internacionalización i18n y, a nivel de la ontología, se hizo uso de los strings con lenguaje.
Como se verá más adelante, el diseño estético de la página está orientado a una paleta de colores que hace referencia a los colores de la lucha contra el cáncer de mama.

En las siguientes secciones se ampliará la funcionalidad y la vista de cada una de las páginas descriptas en el flujo de la figura 4.2.

Los accesos a la aplicación y los repositorios son los siguientes:

- Aplicación: http://risk-prevention-studies.web.elasticloud.uy/
- Repo frontend: https://github.com/brunoszilagyiibarra/ontobreastscreen-app
- Repo backend: https://github.com/brunoszilagyiibarra/ontobreastscreen-backend

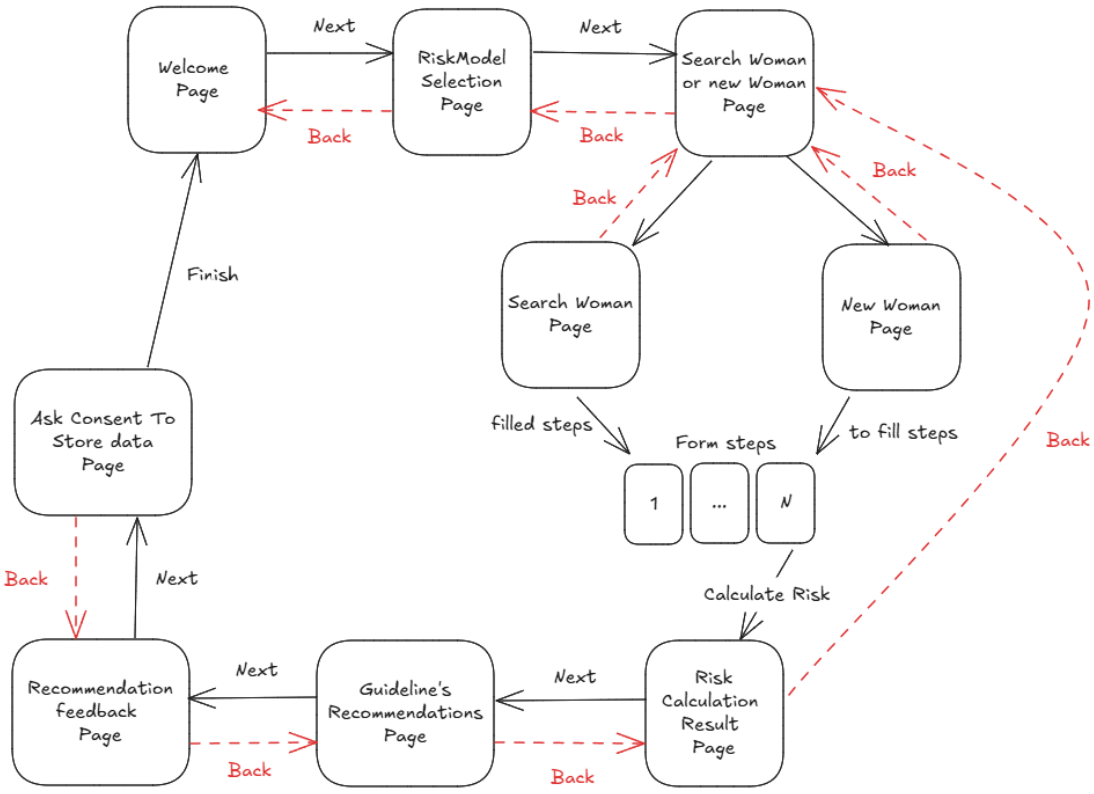

Figura 4.2: Diseño de navegación OntoBreastScreening

4.2. Página de bienvenida y estructura SPA

La pantalla (ver Figura 4.3) de bienvenida es el punto de inicio de toda la aplicación. En ella se puede distinguir el panel con el mensaje de bienvenida junto a un botón de ayuda que indica el propósito de la aplicación. Como parte de la estructura de la single-page application se distingue la sección de header, en donde tenemos una imagen relacionada al dominio y un panel con el nombre de la aplicación junto al símbolo de lucha contra el cáncer. Por otro lado, el footer está dividido en dos, del lado izquierdo se presenta la información de contacto con la posibilidad de redactar un mail (CTA ya abre el correo con un asunto y destinatario), la información sobre la versión de la aplicación y el tipo de licencia que se utiliza, en este caso estamos bajo licencia MIT.
Del lado derecho tenemos el panel de selección de idiomas, en donde en cualquier momento el usuario puede presionar uno de los dos botones y cambiar de Español a Inglés o viceversa. Como se mencionó anteriormente, esto se logró implementando i18n a nivel de frontend, en donde todos los textos están inter-

nacionalizados y se generó un componente que escucha estas actualizaciones de selección de país y ajusta el lenguaje seleccionado para reflejar automáticamente el cambio de idioma. Por el lado de OntoForms, se implementa el uso del header Accept-Language al invocar sus servicios y, del lado del servidor, esto es tenido en cuenta para intentar obtener etiquetas con lenguaje en la ontología.

OntoBreastScreen APP

Bienvenidos a OntoBreastScreen

Una herramienta para evaluación de riesgo de contraer cancer de mama y recomendación de estudios preventivos.

Iniciar evaluación

Contacto

Correo: bruno.szilagyi.ibarra@gmail.com
Versión: 2.9.0

Licencia: Licencia MIT

English Español

Figura 4.3: Pantalla de bienvenida

4.3. Selección de modelo riesgo

Esta pantalla presenta al usuario final una lista de modelos de riesgo disponibles, el cálculo puede ser basado en calculadoras de riesgo o en guías que dan pautas para definirlo. En la figura 4.4 se pueden ver los modelos disponibles para su selección.

Figura 4.4: Pantalla de selección modelo - Listado disponibles

Mientras no se seleccione modelo, el botón de continuar se encuentra deshabilitado ya que es condición necesaria para poder avanzar. El listado de modelos es cargado desde el triplestore, consultando el dataset `/ontologies` y por el identificador de ontología asociado a la aplicación `OntoBreastScreen` obtenido mediante consulta a OntoForms. Una vez obtenido el modelo Jena que representa a la ontología, simplemente se realiza la búsqueda de individuos de la clase RiskModel, presentando en pantalla metadata de nombre modelo y URI del mismo (para posterior identificación de la selección realizada).
Una vez que se realiza la selección, se carga un texto asociado a cada modelo que agrega información para el usuario final, por ejemplo, un enlace al sitio de la calculadora o guía y un texto explicativo, ver Figura 4.5

Figura 4.5: Pantalla de selección modelo - CTA modelo

Como observación, el texto explicativo hoy está fijado definido en el frontend

para los modelos existentes; en caso de agregarse un modelo desconocido del lado de la ontología, el mismo no tendrá texto asociado hasta que no se haga cambio en el código.

4.4. Selección de ingreso

Esta pagina tiene como objetivo brindar la posibilidad al usuario entre buscar un paciente ya ingresado al sistema o ingresar un paciente nuevo. En la Figura 4.6 se muestra la interfaz gráfica simple en donde se presentan dos botones para la funcionalidad dicha.

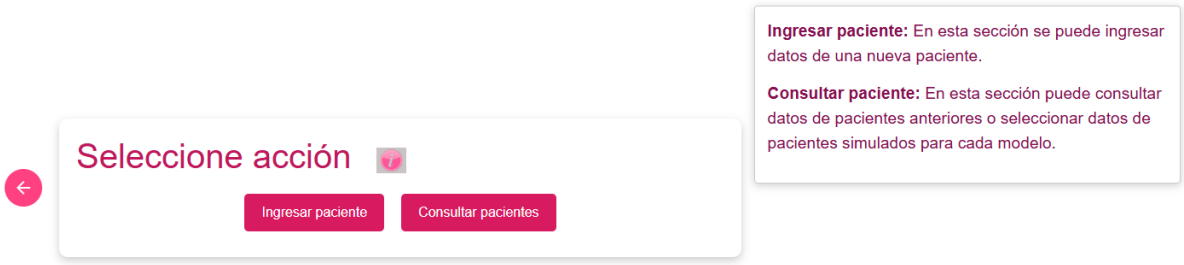

Figura 4.6: Pantalla de selección busqueda o ingreso

4.5. Búsqueda de mujer

En esta página la idea es que el médico realice una búsqueda de los pacientes ya ingresados al sistema y pueda luego previsualizar los datos ingresados y el resultado de riesgo con su recomendación.
Dado que este es un prototipo, las instancias que se disponibilizan al usuario son ejemplos pre-cargados en la api-backend según el modelo de riesgo que el usuario seleccionó al inicio del flujo, ya que los ejemplos deben tener los datos solicitados por ese modelo. En la siguiente Figura 4.7 se puede visualizar la pantalla y los individuos disponibles para su selección bajo el modelo de IBIS.

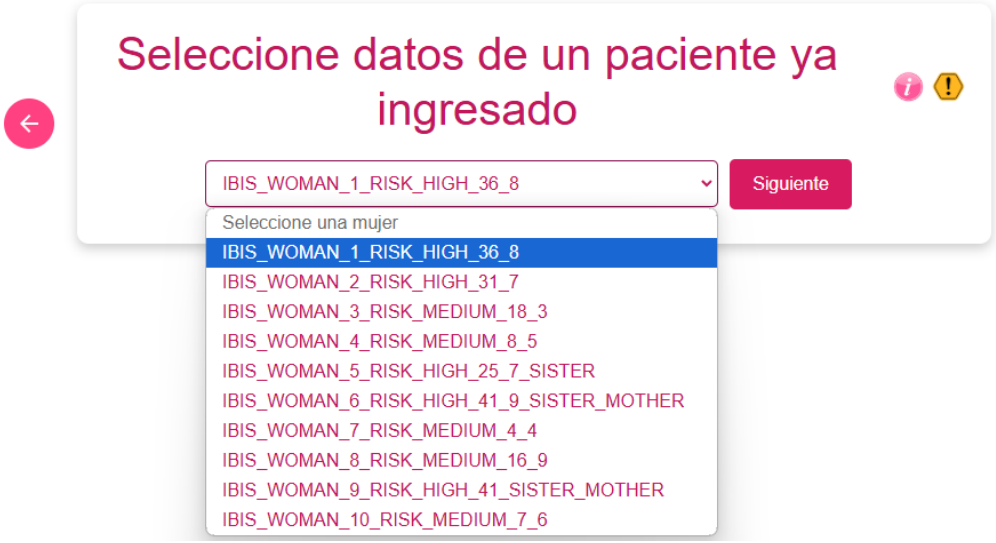

Figura 4.7: Pantalla de búsqueda de pacientes ingresados

Un aspecto a destacar de la implementación de este caso, es que los ejemplos pre-fijados son cargados en el backend mediante la implementación de un patrón strategy en las calculadoras de riesgo disponibles en donde cada una tiene un conjunto de ejemplos disponibles y se encarga de levantar archivos JSON con la representación de los datos de un individuo que implementa el ejemplo. En la siguiente Figura 4.8 se muestra esta estrategia.

El uso de este patrón nos brinda flexibilidad y mejora de mantenimiento al poder agregar nuevos ejemplos rápidamente sin tener que modificar mucho código, solamente un JSON nuevo y agregar a la lista correspondiente dentro de la calculadora. También, si se llegara a agregar una nueva calculadora, con este método ya se tiene automáticamente la funcionalidad implementada y no hay que modificar código base para soportarlo, solamente programar la calculadora específica.

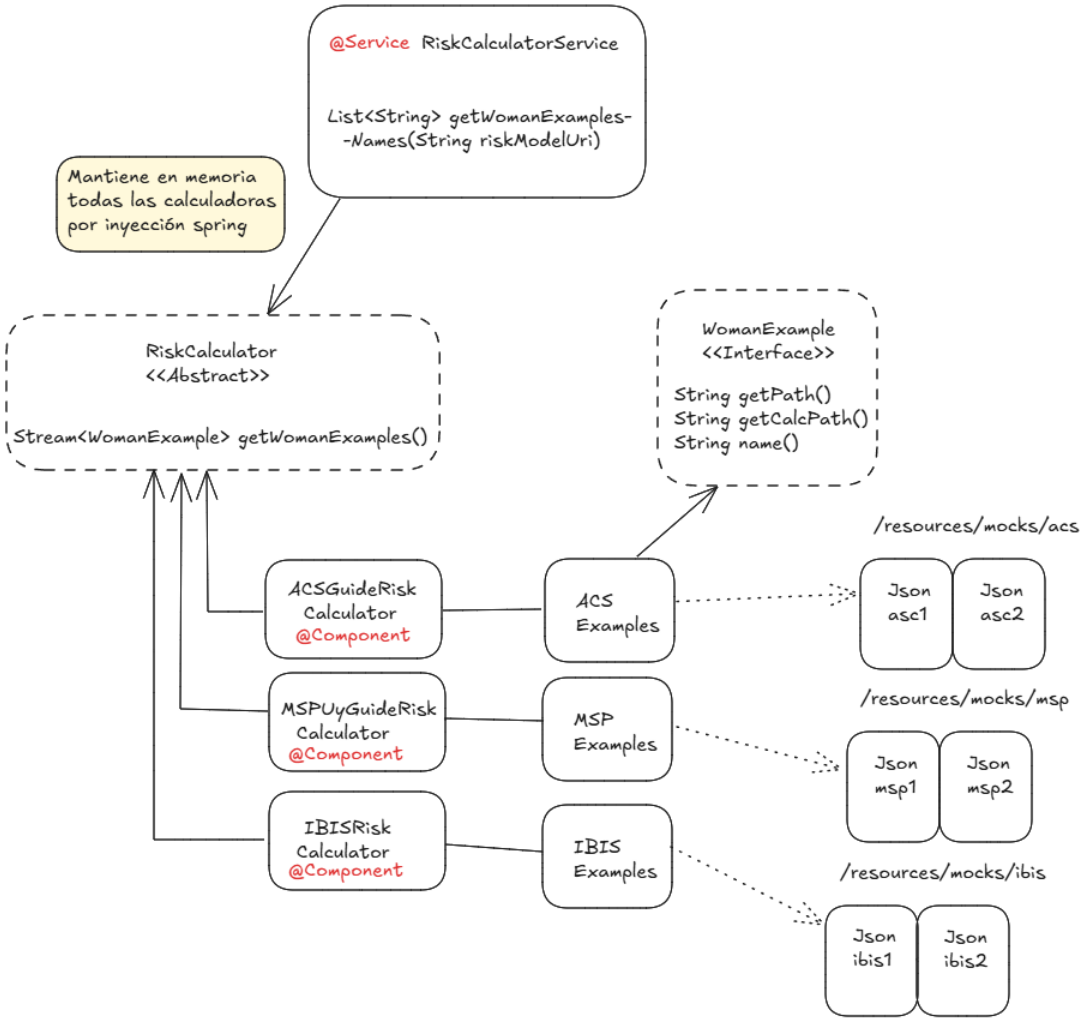

Figura 4.8: Implementación de estrategy en calculadoras de riesgo - Mocks

4.6. Formulario ingreso o visualización mujer

Esta es la pantalla principal de esta aplicación, en donde estamos usando plenamente la ontología brindada para construir una interfaz en donde los usuarios finales puedan ingresar datos para generar instancias de los individuos. Para la implementación de este caso de uso se tiene en cuenta el diagrama de la ontología BCSR-Onto como se muestra en la Figura 4.9. Dado que desde la clase mujer, que es a la que le queremos ingresar datos, no se cuenta con todas las preguntas asociadas al modelo de riesgo, se debe hacer una adaptación para obtener las preguntas de la clase Question y unificar con las respuestas en Answer hasta que no se adapte el diseño de la Ontología para relacionar los datos necesarios directamente vía propiedades alcanzables navegando desde la mujer.
El esquema implementado se puede ver en la siguiente Figura 4.10 en dónde primero se consulta a OntoForms por el formulario que corresponde a la mujer, y luego se ejecuta un Adapter que genera nuevas secciones incluyendo una sección por Factor de riesgo en las preguntas y obteniendo los individuos a través de la relación con Answer.

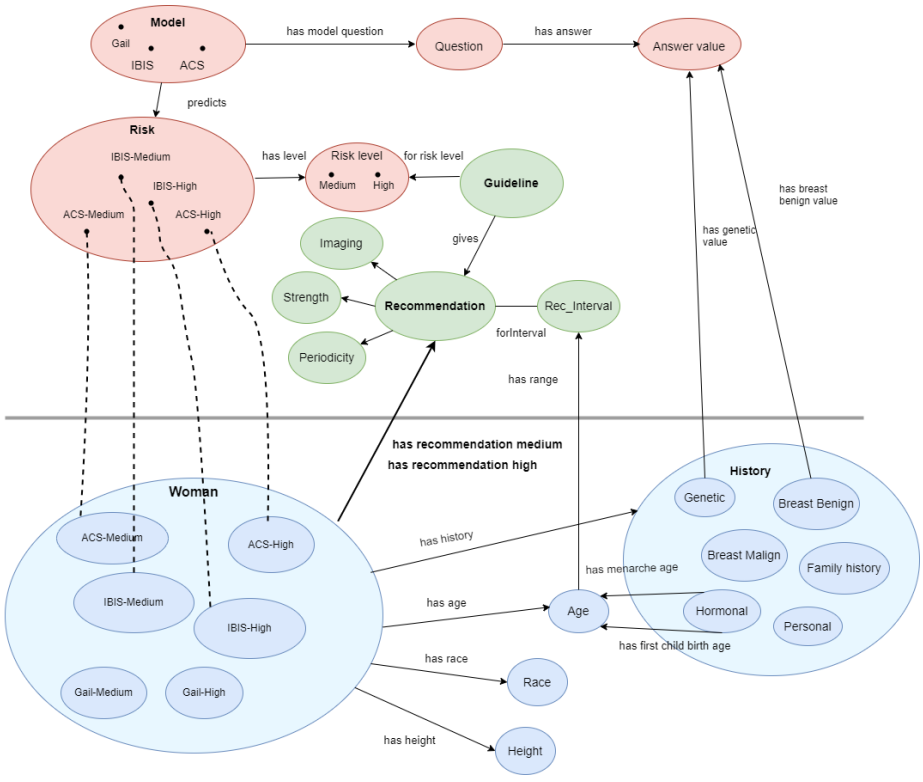

Figura 4.9: BCSR-onto

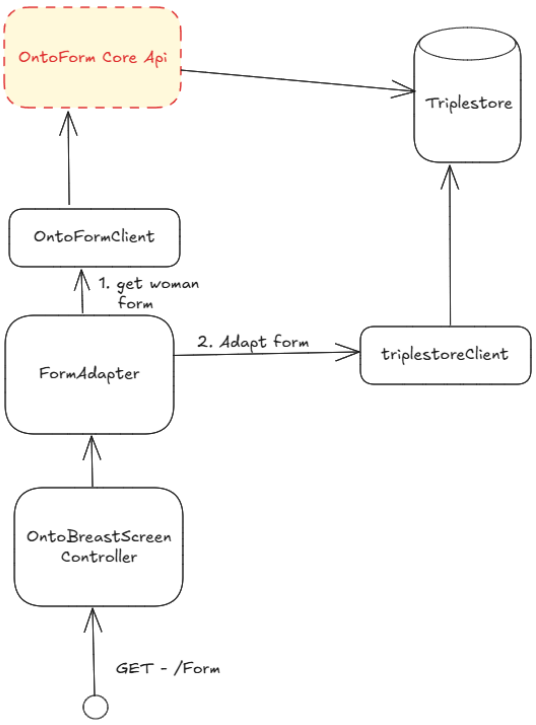

Figura 4.10: Form adaptor patrón

Una vez obtenido el formulario, la estrategia a nivel de frontend fue realizar un formulario por pasos, en donde cada sección se representa por un paso del formulario. Esto es una personalización posible a nivel de componentes de frontend soportada por la estructura de formularios de OntoForms. Además, cada campo del formulario renderiza un combo-box o un campo de dato simple según el tipo de campo. Por otro lado, cada campo tiene trazabilidad con el identificador mediante la uri de la propiedad y para los individuos la uri de cada uno de ellos. Esto permite que al momento de ingresar los datos, en el backend se pueda hacer la correspondencia correctamente, sabiendo a qué propiedad pertenece cada dato ingresado.

En lo que respecta al caso de uso de búsqueda y el caso de uso de ingreso, en lo que se diferencia es que en el caso de uso de búsqueda el formulario viene con los valores ya cargados desde la API Backend y los campos no son editables, mientras que en el caso de uso de ingreso, todos los campos son editables y se despliegan para el usuario.
A continuación se pueden ver ejemplos de estos formularios para la búsqueda, en las figuras 4.11 y 4.12 y los formularios para el ingreso en las figuras 4.13 y 4.14.

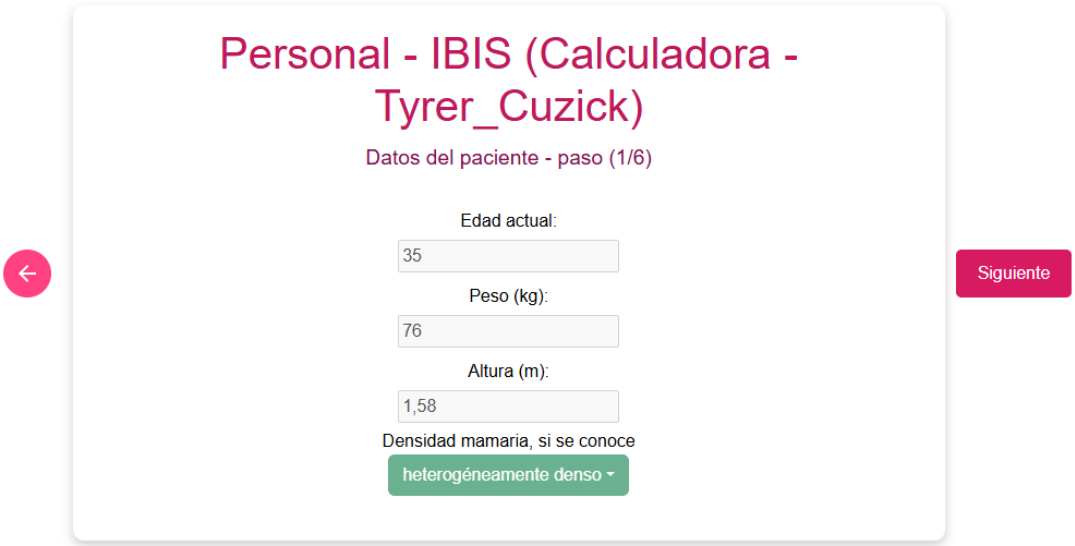

Figura 4.11: Form búsqueda - step 1

Se puede observar que en la parte superior del formulario, se incluye el nombre de la sección y la calculadora seleccionada. En el subtítulo se muestra el avance en el formulario, contando los pasos totales y el paso actual. Notar que el formulario es dinámico según el modelo de riesgo seleccionado, por ejemplo,

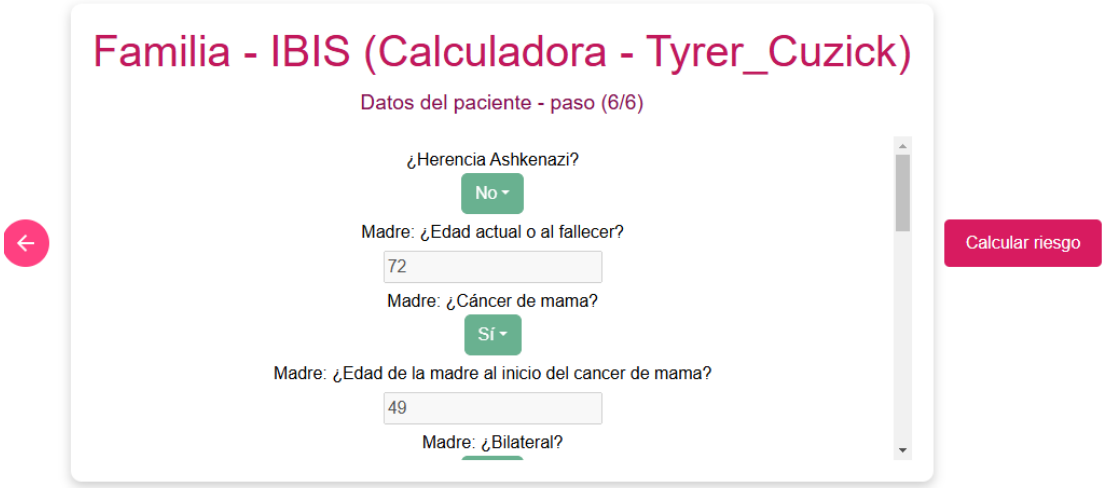

Figura 4.12: Form búsqueda - step 6

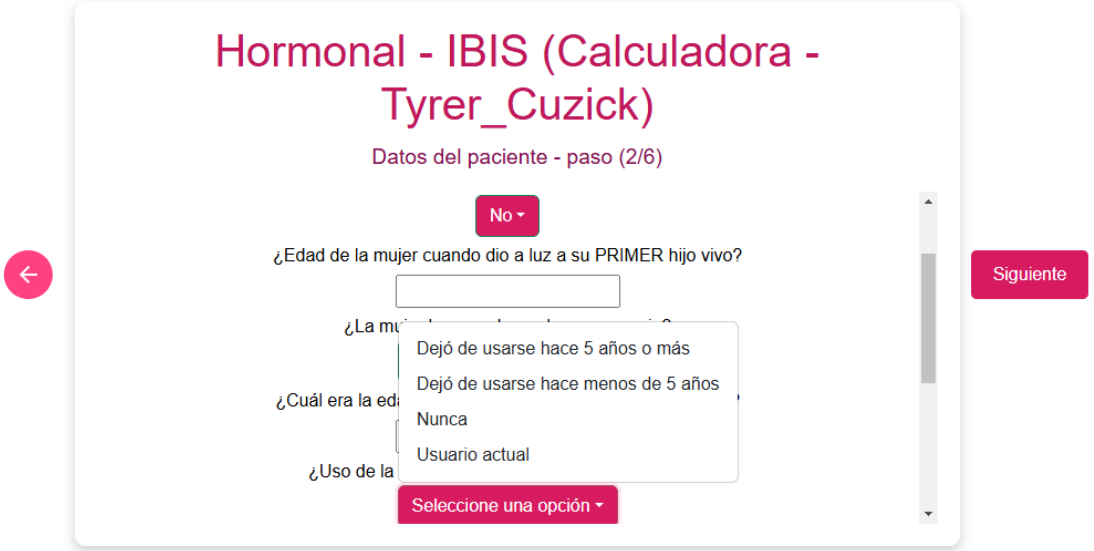

Figura 4.13: Form ingreso - step 2

Figura 4.14: Form ingreso - step 3

para MSP UY se muestra el formulario de búsqueda en las siguientes Figuras 4.15 4.16.

Figura 4.15: Form busqueda MSP - step 1

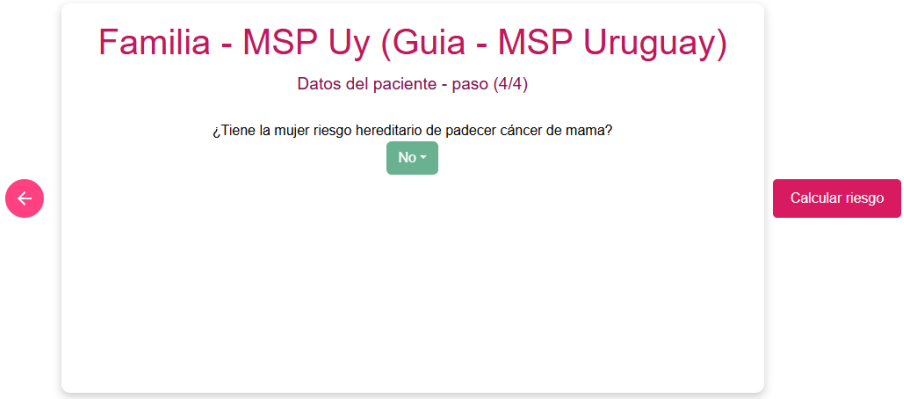

Figura 4.16: Form busqueda MSP - step 4

4.7. Resultado cálculo riesgo

Al presionar el botón de `Calcular riesgo` de la pantalla de formulario, los datos del formulario son enviados al backend, en donde se pasan a la calculadora asociada al modelo de riesgo para su procesamiento. Si estamos ante un caso de una mujer ya ingresada en el sistema, entonces el cálculo del riesgo ya se encuentra realizado y se devuelve el dato de un cache. Sino se propagan todos los datos obtenidos en el formulario a la instancia concreta de calculadora.
A continuación mostraremos la implementación del cálculo de riesgo según cada calculadora o guía disponible. En esta implementación se tiene MSP UY, ASC e IBIS.

Para el cálculo de riesgo basado en la guía MSP UY solo es necesario obtener el valor de alguna de las tres preguntas realizadas y si alguno de ellos es verdadero entonces el riesgo será HIGH, sino MEDIUM en caso contrario. El fragmento de código que implementa esto es el siguiente 4.1
En cada cálculo de riesgo estamos retornando un objeto con los metadatos: riskLevel, riskModelCalculatorType y riskModel. Esto nos da la posibilidad de mostrar estos datos en pantalla al usuario para brindarle mayor contexto del cálculo, tal como se ve en la Figura 4.17

```
public RiskCalculation doRiskCalculation(Map<String, String>
    womanFormData) {
    RiskLevel resultRisk = RiskLevel.MEDIUM;

  if(UY_HEREDITARY_RISK_YES.getUri().equals(
        womanFormData.get(UY_HEREDITARY_RISK_QUESTION.getUri())) ||
    UY_CHEST_RADIOTERAPHY_YES.getUri().equals(
        womanFormData.get(UY_CHEST_RADIOTERAPHY_QUESTION.getUri())) ||
    UY_HIPERPLASIA_ATIPIA_YES.getUri().equals(
        womanFormData.get(UY_HIPERPLASIA_ATIPIA_QUESTION.getUri()))) {
        resultRisk = RiskLevel.HIGH;
    }

   return RiskCalculation.builder()
        .riskLevel(resultRisk)
        .riskModelCalculatorType(getRiskModelCalculator())
        .riskModel(getRiskModel())
        .build();
}
```

Listing 4.1: Calculadora MSP UY - Calcuar riesgo

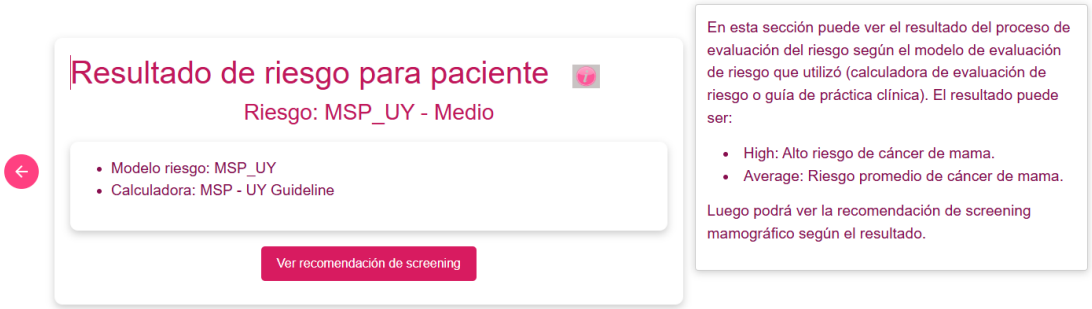

Figura 4.17: Cálculo riesgo MSP

Para el cálculo de riesgo basado en la guía ASC se sigue un enfoque similar al anterior siendo solo necesario obtener el valor de alguna de las tres preguntas realizadas y si alguno de ellos es verdadero entonces el riesgo será HIGH, sino MEDIUM en caso contrario. El fragmento de código que implementa esto es el siguiente 4.3 y la figura en donde se ve el resultado es la siguiente 4.18. En este caso se utilizó la búsqueda de un ejemplo ACS que diera riesgo HIGH, notar como la calculadora se muestra como `Mocked` dado que el resultado ya estaba en cache y era un ejemplo pre-cargado.

```
public RiskCalculation doRiskCalculation(Map<String, String> womanFormData) {
    RiskLevel resultRisk = RiskLevel.MEDIUM;

  if(ACS_CHEST_RADIOTERAPY_YOUNG_AGE_YES.getUri().equals(
        womanFormData.get(ACS_CHEST_RADIOTERAPY_YOUNG_AGE_QUESTION.getUri())) ||
    ACS_HISTORY_BREAST_CANCER_YES.getUri().equals(
        womanFormData.get(ACS_HISTORY_BREAST_CANCER_QUESTION.getUri())) ||
    ACS_GENETIC_MUTATION_YES.getUri().equals(
        womanFormData.get(ACS_GENETIC_MUTATION_QUESTION.getUri()))) {
            resultRisk = RiskLevel.HIGH;
        }

   return RiskCalculation.builder()
        .riskLevel(resultRisk)
        .riskModelCalculatorType(getRiskModelCalculator())
        .riskModel(getRiskModel())
        .build();
}
```

Listing 4.2: Calculadora ACS - Calcuar riesgo

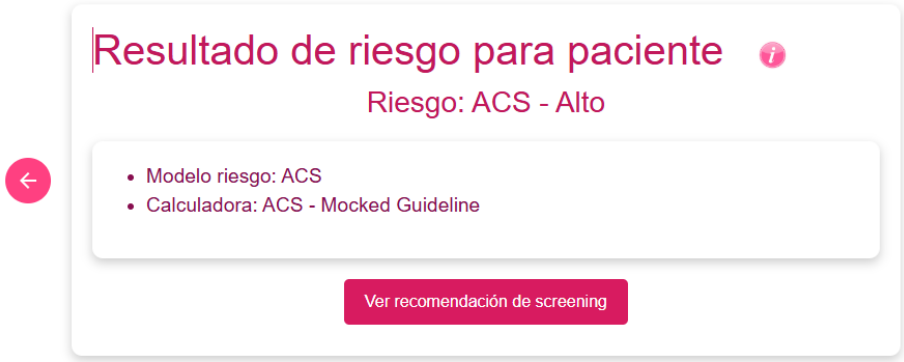

Figura 4.18: Cálculo riesgo ACS

Para el caso de IBIS es más complejo ya que el cálculo de riesgo se realiza mediante una calculadora (*IBIS v8 calculator*, s.f.). Evaluamos las opciones y dado que no hay API disponible para el uso de la calculadora, se decidió hacer uso de alguna de las calculadoras publicadas online.
Se seleccionó la calculadora publicada en `https://ibis.ikonopedia.com`, la idea es completar los datos del formulario usando lo que ingresó el usuario, calcular y obtener el resultado llevándolo a nuestra aplicación. Para realizar este trabajo se implementó la solución de la Figura 4.19. La limitante de esta solución es que solamente se pueden realizar 5 cálculos por día, ya que hay un control de intentos en la página de ikonopedia. Para la versión productiva del prototipo se necesitará comprar una licencia de la calculadora.

La implementación de esta estrategia es básicamente delegar al componente de scraper y obtener el resultado de allí; en caso de error de cálculo, se asume

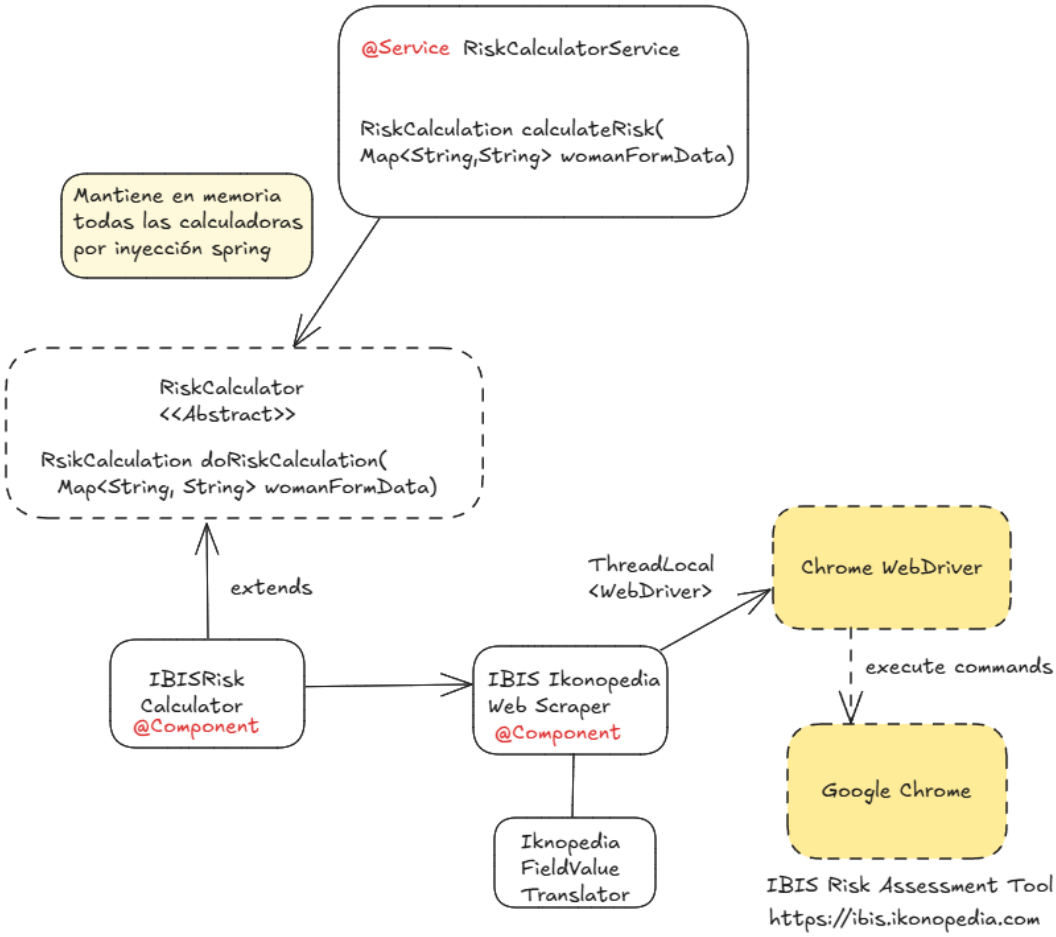

Figura 4.19: Solución web scraper para calculadora IBIS

riesgo Medio.

```
public RiskCalculation doRiskCalculation(Map<String, String>
    womanFormData) {
  try{
        return ibisIkonopediaWebScrapper.calculateRisk(womanFormData);
    } catch (RiskCalculatorException e) {
        return RiskCalculation.builder()
                .riskLevel(RiskLevel.MEDIUM)
                .riskModelCalculatorType(getRiskModelCalculator())
                .riskModel(getRiskModel())
                .build();
    }
}
```

Listing 4.3: Calculadora ACS - Calcuar riesgo

4.8. Recomendación según guía

En esta página de recomendación, el objetivo es que el médico seleccione una guía de recomendaciones para el riesgo obtenido. Estas recomendaciones y las guías obtenidas se obtienen mediante la ejecución del razonador sobre los datos ingresados del a mujer y el riesgo calculado. La siguiente Figura 4.20 muestra la interfaz gráfica generada.

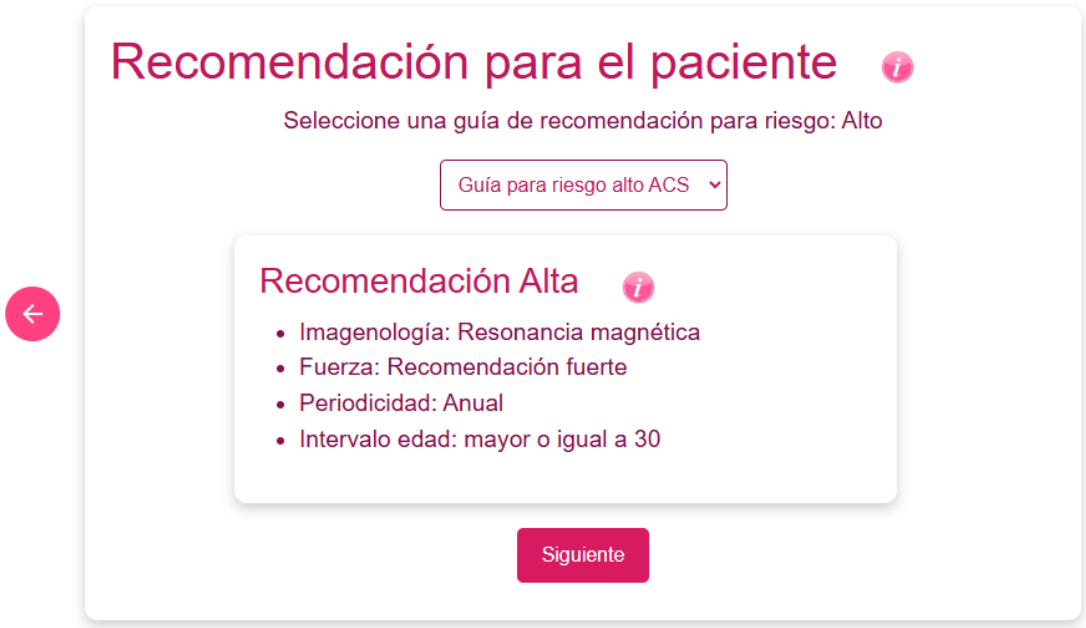

Figura 4.20: Recomendación de estudios ASC HIGH

4.9. Feedback recomendación

Esta pantalla (Figura 4.21) se agregó para ejemplificar el flujo completo en el prototipo. El objetivo es que el médico presente su feedback escrito sobre la recomendación y luego registre si la misma fue aceptada o rechazada.
Estos datos deberán ser guardados con fines de investigación en la versión final de la solución.

Figura 4.21: Pantalla de Feedback sobre recomendación

4.10. Solicitud consentimiento guardar datos

Esta pantalla (Figura 4.22) es la última del flujo y se agregó para ejemplificar el flujo completo en el prototipo. El objetivo es que el médico solicite al paciente el consentimiento sobre el uso de los datos personales para fines de investigación, aceptando o rechazando.
Al otorgar permiso, el individuo ingresado de paciente mujer debe ser persistido en el triplestore en el dataset `Individuals` junto a todas sus relaciones.
Posteriormente se ejecutarían consultas analíticas mediante SPARQL sobre el triplestore para obtener los datos estadísticos sobre el uso de la aplicación y los resultados brindados.

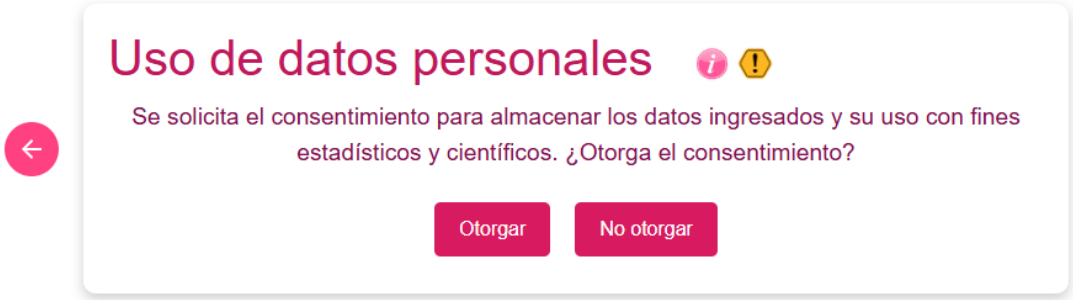

Figura 4.22: Pantalla de solicitud consentimiento datos personales

Capítulo 5

Experimentación

En este capítulo se muestran las diferentes pruebas ejecutadas sobre el sistema en la etapa de verificación, tanto a nivel unitario como pruebas de los casos de uso. En la primera sección se mostrarán los distintos escenarios realizados para verificar que los servicios de OntoForms funcionan como se espera frente a distintas ontologías. En la segunda sección se describe un ejemplo con una ontología de relaciones entre personas en donde se recorre toda la funcionalidad de OntoForms demostrando su uso. Por último, se incluye el detalle de una encuesta desarrollada para relevar el uso y aplicabilidad de la solución realizada a médicos sobre el uso y la aplicabilidad de la aplicación OntoBreastScreen.

5.1. Pruebas de correctitud en los servicios OntoForms

El objetivo de estas pruebas es verificar que todos los servicios base de OntoForms funcionan como se espera, retornando el conjunto esperado de propiedades, clases e individuos. Para esto se elaboró un conjunto de test de integración parametrizados según ontologías para poder ejecutar las pruebas con varias ontologías en cada caso. El diseño de esta solución se puede ver en la siguiente Figura 5.1

Estos test son de integración ya que se clasifican como `@SpringBootTest` levantando todas las clases de la aplicación y generando el contexto de Spring. La idea es tener todos los componentes disponibles tal cual el ambiente productivo solamente `Mocked` el triplestore para que levante la ontología usando el archivo almacenado en los recursos de test del proyecto. En el fragmento de código 5.1 se puede visualizar esta definición.

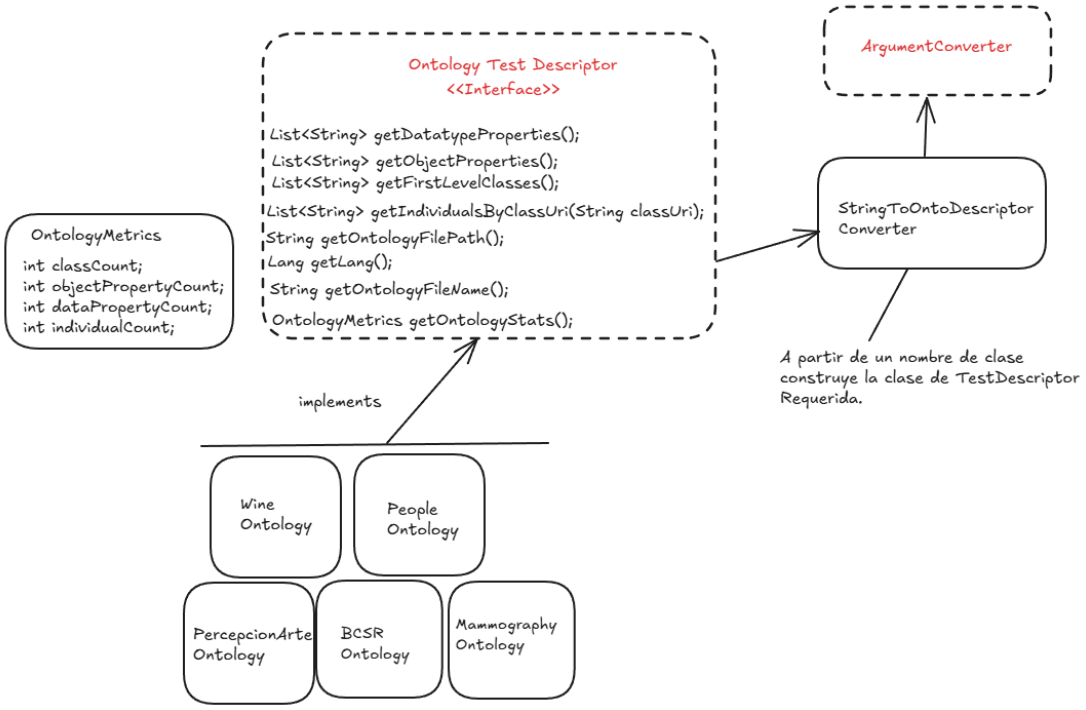

Figura 5.1: Diseño de los tests de integración - Ontoforms

```
1
2  @SpringBootTest
3  public class OntologyRepositoryIT {
4
5      @Autowired
6      private OntologyRepository sut;
7
8      @MockBean
9      private FusekiTripleStoreClient tripleStoreClient;
10
11     private void mockTripleStoreWithLocalImplementation(OntologyTestDescriptor onto) {
12         Model model = RDFDataMgr.loadModel(onto.getOntologyFilePath(), onto.getLang());
13         when(tripleStoreClient.getOntologyByName(onto.getOntologyFileName()))
14         .thenReturn(model);
15     }
16 }
```

Listing 5.1: Definición de clase de test repositorio Ontology

Para generar cada una de las clases que implementan OntologyTestDescriptor se realizó un trabajo manual utilizando `Protégé` en donde se recolectaron todas las URIs referentes a propiedades, clases e individuos. También se vió al-

gunas métricas de ontología. Comparar contra estos resultados asegura que del lado de OntoForms los servicios están devolviendo de forma correcta los datos esperados.

El siguiente test 5.2 está pensado para que valide que todas las dataproperties son recuperadas como se espera según la definición de cada ontología.

```
@ParameterizedTest
@CsvSource({PEOPLE_TTL, WINE_RDF, BREAST_CANCER_RECOMMENDATION_V_25_4_2024_OWL,
       MAMMOGRAPHY_SCREENING_RECOMMENDATION_OWL, PERCEPCION_ARTE_SUBPERCEPCIONES_RDF})
void whenGetDatatypePropertiesThenRetrieveAllExpected(
@ConvertWith(StringToOntoDescriptorConverter.class) OntologyTestDescriptor onto) {
    mockTripleStoreWithLocalImplementation(onto);

    var datatypeProperties =
        sut.getDatatypePropertiesByOntoId(onto.getOntologyFileName());

    assertEquals(onto.getDatatypeProperties().stream().sorted().toList(),
           datatypeProperties.stream().map(Resource::getURI).sorted().toList());
    assertEquals(onto.getOntologyStats().getDataPropertyCount(),
        datatypeProperties.size());
}
```

Listing 5.2: Validación de dataproperties.

El siguiente test 5.3 está pensado para que valide que todas las objectproperites son recuperadas como se espera según la definición de cada ontología.

```
@ParameterizedTest
@CsvSource({PEOPLE_TTL, WINE_RDF, BREAST_CANCER_RECOMMENDATION_V_25_4_2024_OWL,
       MAMMOGRAPHY_SCREENING_RECOMMENDATION_OWL, PERCEPCION_ARTE_SUBPERCEPCIONES_RDF})
void whenGetObjectPropertiesThenRetrieveAllExpected(
@ConvertWith(StringToOntoDescriptorConverter.class) OntologyTestDescriptor onto) {
       mockTripleStoreWithLocalImplementation(onto);

       var objectProperties =
           sut.getObjectPropertiesByOntoId(onto.getOntologyFileName());

       assertEquals(onto.getObjectProperties().stream().sorted().toList(),
              objectProperties.stream().map(Resource::getURI).sorted().toList());
       assertEquals(onto.getOntologyStats().getObjectPropertyCount(),
           objectProperties.size());
}
```

Listing 5.3: Validación de objectproperties.

El siguiente test 5.4 está pensado para que valide que todas las clases de primer nivel son recuperadas como se espera según la definición de cada ontología.

```
@ParameterizedTest
@CsvSource({PEOPLE_TTL, WINE_RDF, BREAST_CANCER_RECOMMENDATION_V_25_4_2024_OWL,
       MAMMOGRAPHY_SCREENING_RECOMMENDATION_OWL, PERCEPCION_ARTE_SUBPERCEPCIONES_RDF})
void whenGetFirstLevelClassesThenRetrieveAllExpected(
 @ConvertWith(StringToOntoDescriptorConverter.class) OntologyTestDescriptor onto) {
       mockTripleStoreWithLocalImplementation(onto);

       var classes = sut.getFirstLevelClassesByOntoId(onto.getOntologyFileName());

       assertEquals(onto.getFirstLevelClasses().stream().sorted().toList(),
              classes.stream().map(Resource::getURI).sorted().toList());
   }
```

Listing 5.4: Validación de clases de primer nivel.

De la misma forma se validan los siguientes datos:

- Los individuos recuperados y su cantidad.
- La cantidad de clases recuperada

5.2. OntoForms: Ontología relaciones entre personas

En esta sección se describe un ejemplo práctico sobre los servicios de generación de formulario de OntoForms Core API para la ontología de relaciones entre personas. Se muestra la aplicación web donde el administrador realiza su gestión y una vista previa del formulario generado a partir de la clase persona. En la figura 5.2 se muestran algunas de las clases y propiedades de la ontología mencionada.

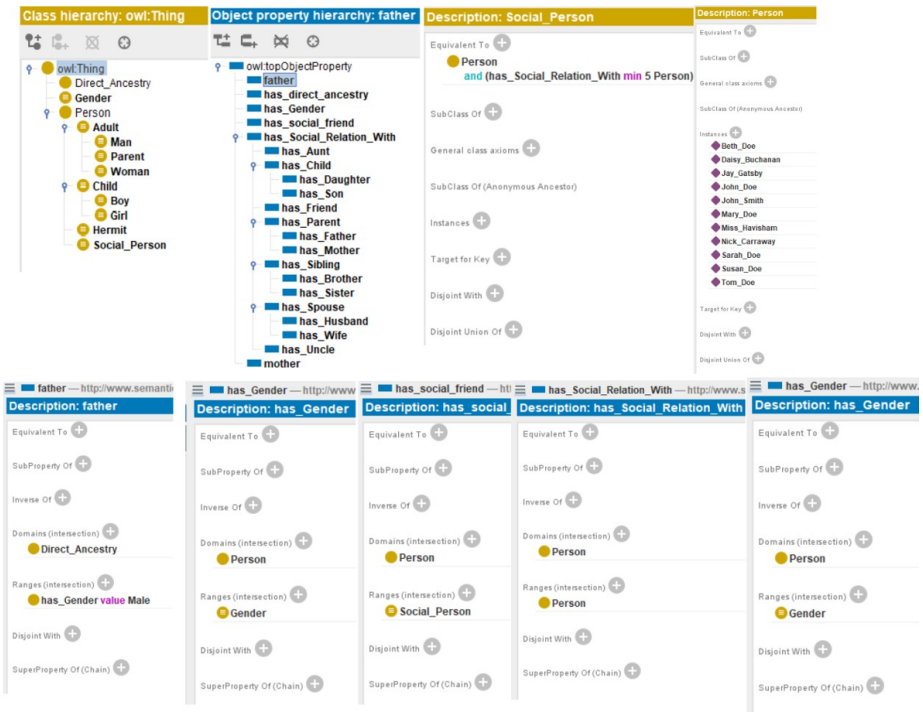

Figura 5.2: Vista previa of the People Ontology

Para visualizar un formulario y poder poblar de instancias una clase de la ontología, el primer paso es cargar la ontología en el sistema de OntoForms a través de la página web del administrador (accesible en `http://ontoforms-admin-ui.web.elasticloud.uy/`), como muestra la Figura 5.3. Una vez que la ontología fue cargada, puede ser explorada al presionar el CTA que indica ver detalles. Esto puede ser de ayuda para reconocer todas las clases en su jerarquía y las propiedades e individuos por cada una que se seleccione (ver figura 5.4).

Después de eso, el usuario administrador puede ir al menú de formularios y seleccionar una ontología junto a una clase principal desde el árbol de clases para

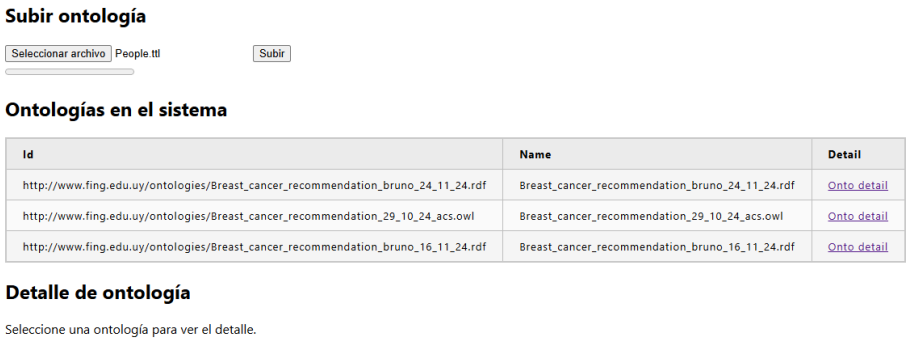

Figura 5.3: Selección y carga de ontología

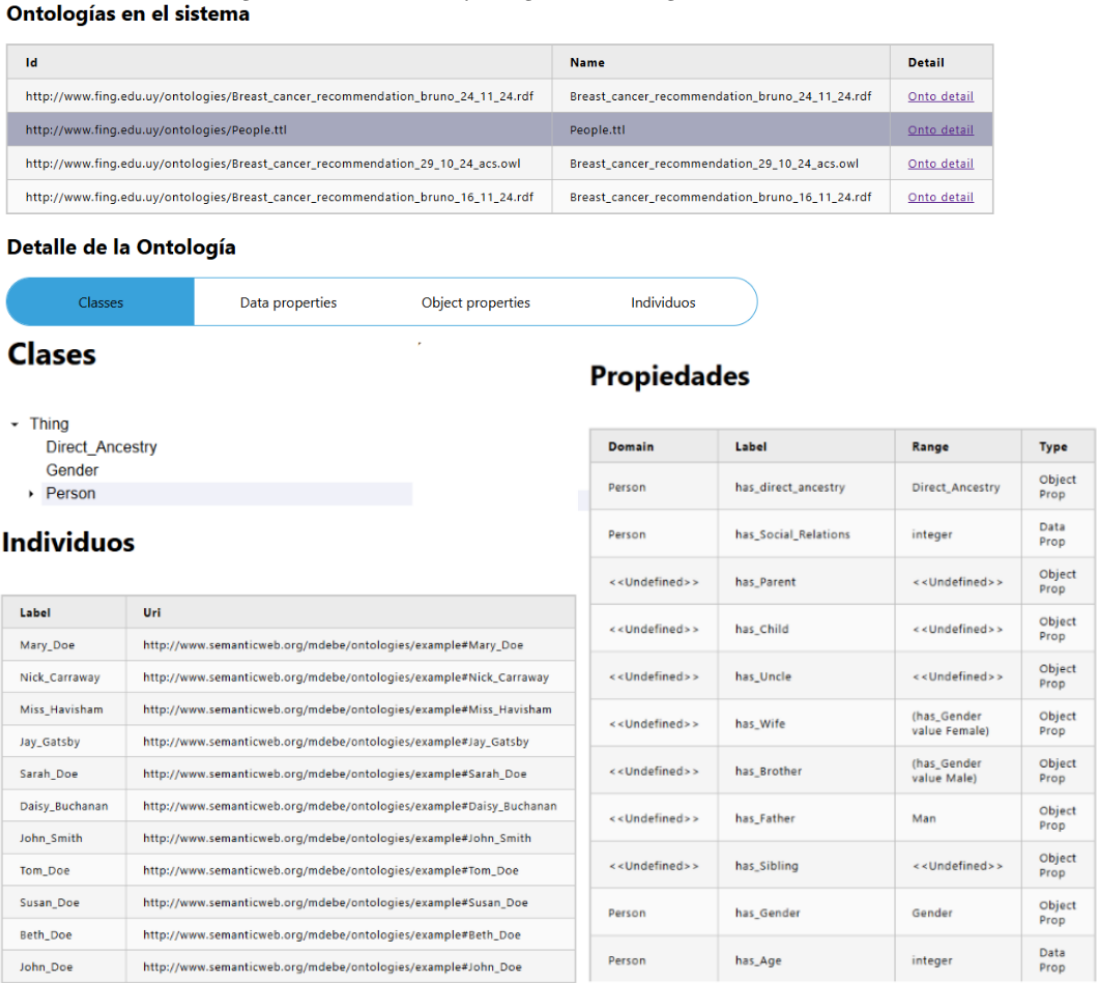

Figura 5.4: Explorar ontología

desplegar el formulario por defecto para la ontología relaciones entre personas y la clase *Person* como se muestra en la Figura 5.5

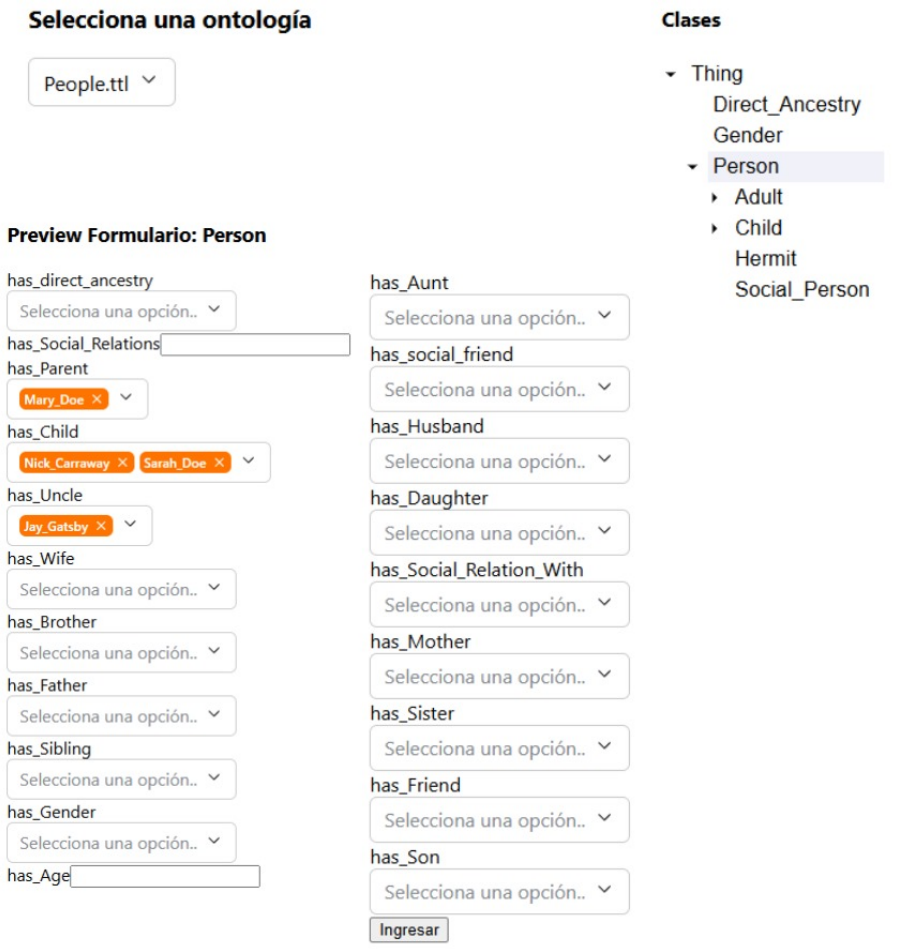

Figura 5.5: Previsualización de formulario por defecto para clase People

Una vez que el formulario por defecto es visualizado, el usuario administrador puede introducir las diferentes configuraciones. Como fue mencionado anteriormente, una de las contribuciones de OntoForms es la posibilidad de configurar clases intermedias como transparentes, incrustando el formulario de ellas como una sección dentro del formulario principal. En este caso, la clase intermedia *DirectAncestry* juega el rol de conectar a cada instancia de persona con dos instancias a través de las propiedades $father$ y $mother$. La Figura 5.6 y 5.7 demuestran el ejemplo. Además, es posible ocultar las propiedades que no son esperadas que se instancien por introducción de datos por el usuario, sino que

por razonamiento.

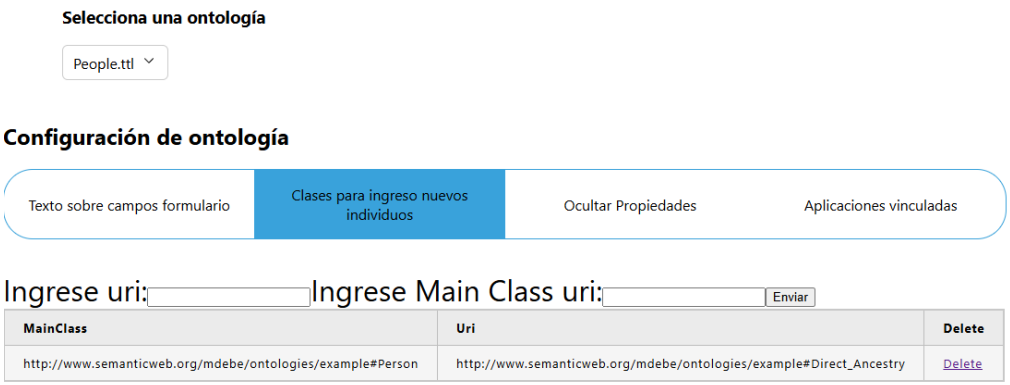

Figura 5.6: Configurar la clase DirectAncestry como transparente

Otra contribución relevante de OntoForms es el uso de servicios de razonamiento para considerar las inferencias en el formulario generado. En este ejemplo, las propiedades $father$ y $mother$ tienen rango $hasGender\ value\ Male$ y $hasGender\ value\ Female$ respectivamente. Dado que no hay en la ontología individuos de la clase Persona que estén explícitamente declarados como una instancia de los rangos anteriores, no sería posible presentar instancias para que el usuario seleccione. Al ejecutar el razonador, OntoForms puede presentar al usuario las instancias de Person que cumplen con dichas restricciones. La Figura 5.8 ilustra este ejemplo. Del lado izquierdo, para la propiedad $father$ podemos ver que solo $TomDoe$ puede ser seleccionado, y del lado derecho podemos comprobar la consulta de DL que verifica que solamente la instancia $TomeDoe$ cumple con la condición $hasGender\ value\ Male$. Lo mismo pasa para la propiedad $mother$.

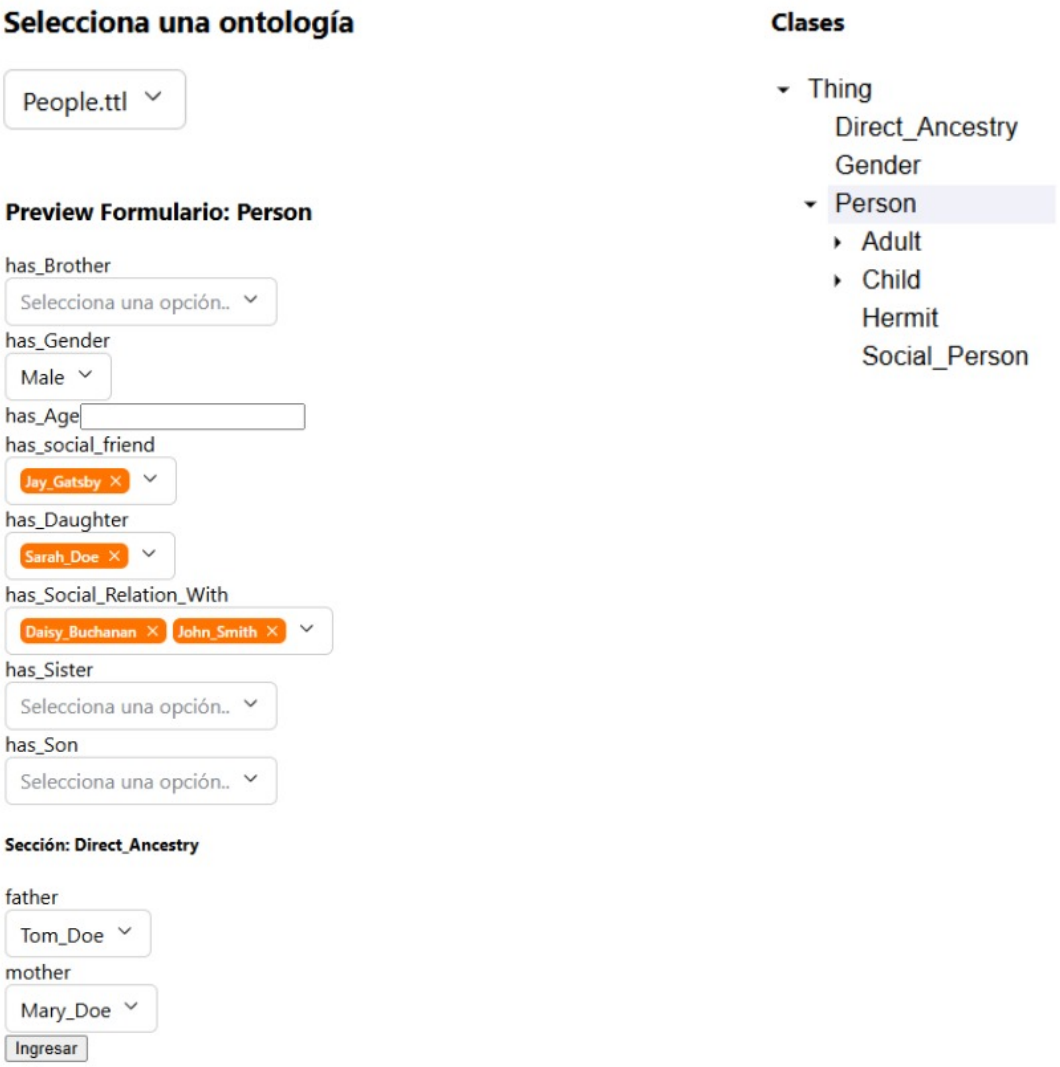

Figura 5.7: Previsualiza el formulario con la clase DirectAncentry oculto

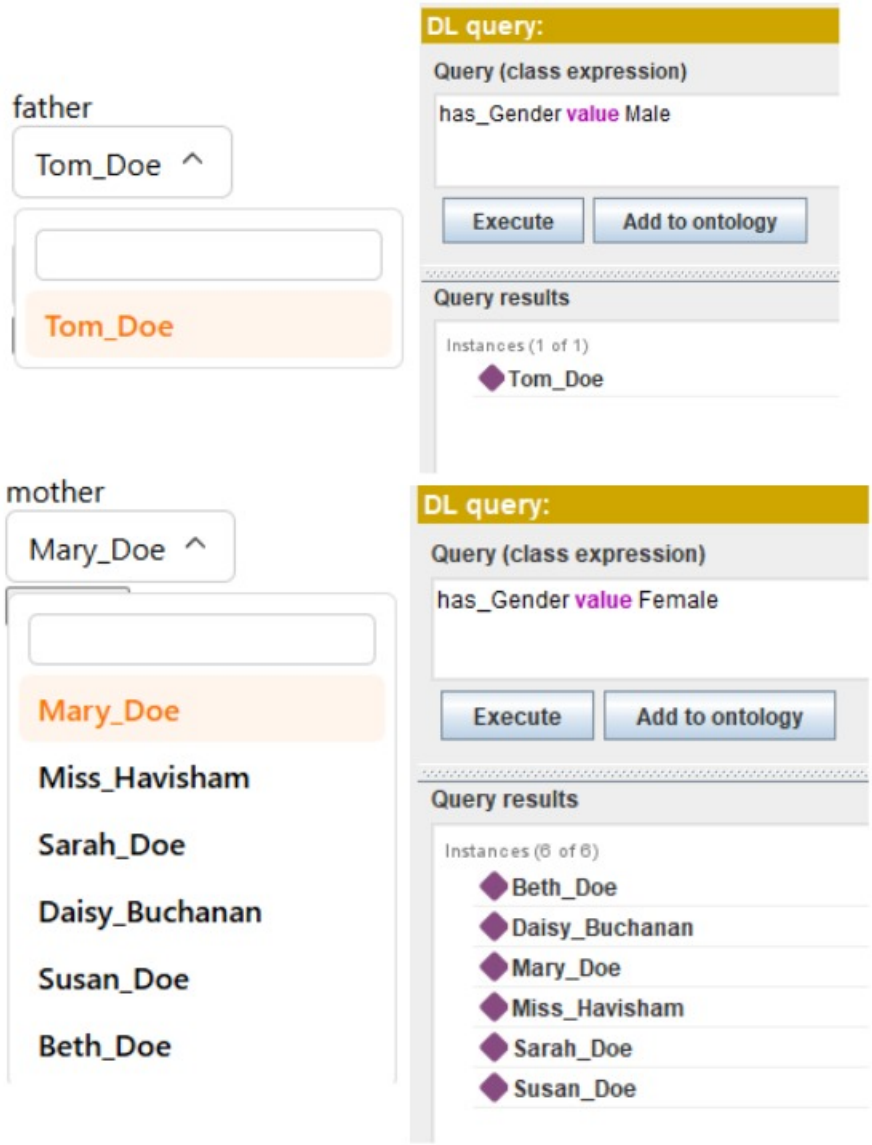

Figura 5.8: Previsualize form with the class DirectAncestry hidden

5.3. Evaluación OntoBreastScreen

Se desarrolló una encuesta que tiene como objetivo obtener retroalimentación de profesionales de la salud acerca del uso de la aplicación para la prevención del cáncer de mama, OntoBreastScreen. Junto con la encuesta, se les brinda a los médicos el acceso a la plataforma y el siguiente documento introductorio al proyecto, brindando un contexto de la problemática https://docs.google.com/document/d/1gS0hOq9ftINaAZXUyrjUFSc4cnp241Wx

La encuesta está disponible en https://docs.google.com/forms/d/1qL5lgHyG01ilTYDXS5uYdnajuh-NJFkO7_vw87lypi0/prefill y se almacenan los resultados en la planilla https://docs.google.com/spreadsheets/d/1eqcuEdbokzlRz-GUOW1Dkx9H0DVh4OYowIVM2VJPxdQ. A continuación, describo las secciones de la encuesta:

Información general del profesional consultado:

- ¿En qué ámbito desempeña su labor? Opciones "Ámbito privado", "Ámbito público", .ªmbos".
- En los centros de atención donde desempeña su labor, ¿Tiene acceso a una computadora? Opciones: "Si, en todos", .ªn ninguno", .ªn algunos".
- ¿En qué especialidad se desempeña mayoritariamente? Opciones: "Ginecología", .ºncología", "Medicina familiar", "Medicina de primer nivel de atención".

Información del profesional consultado, acerca de su labor en relación al screening de cáncer de mama:

- En cuanto a su labor, ¿Realiza recomendaciones de screening mamográfico? Opciones: "Si", "No", .ºtro:".
- Durante la consulta con el paciente, acerca del screening mamográfico: ¿Determina el nivel de riesgo de la paciente en relación al cáncer de mama? Opciones: "Si", "No", .ºtro:".
- ¿Cómo determina el nivel de riesgo de cáncer de mama de la mujer al momento de la consulta? Opciones: .ª través de un cuestionario simple acerca de factores de riesgo básicos", .ª través de uso de aplicación de calculadoras de riesgo basadas en modelos de evaluación como IBIS, Gail u otras", "No se realiza evaluación alguna del riesgo de contraer la enfermedad", .ºtros".
- ¿Cómo realiza hoy en día la recomendación de estudios de screening mamográfico? Opciones: .ªn un marco de toma de decisión compartida con la paciente, con evidencia en mano", Recomienda a la paciente tomar la decisión de acuerdo a la evidencia", .ªplicando el mismo protocolo para todas las pacientes", "según consenso del lugar de trabajo", .ºtro".

Evaluación de la aplicación OntoBreastScreen

- En un rango del 1 al 5, ¿Considera que la aplicación OntoBreastScreen provee facilidades para aplicar estrategias de prevención (Screening) de cáncer de mama al momento de la consulta acerca del screening?
- En un rango del 1 al 5, ¿Qué tan simple le resulta utilizar OntoBreastScreen?
- En un rango del 1 al 5, ¿Qué tan últil le resulta contar con la información referente a los modelos de evaluación de riesgo y guías de recomendación, accesible desde una única plataforma?
- En caso de identificar funcionalidades relevantes que OntoBreastScreen debería brindar y no lo está haciendo, ¿Qué funcionalidades deberían incorporarse a la aplicación?
- ¿Recomendaría a otro profesional OntoBreastScreen?
- ¿Considera que esta aplicación (y los datos generados de la misma) podrían aportar a estrategias de prevención del cancer de mama? Opciones: Si, No.
- ¿Considera que esta aplicación (y los datos generados de la misma) podrían aportar a estrategias de control del cancer de mama? Opciones: Si, No.
- ¿Considera que esta aplicación (y los datos generados de la misma) podrían aportar a estrategias de investigación del cancer de mama? Opciones: Si, No.

Por el momento no se cuenta con los datos sobre las respuestas.

Capítulo 6

Conclusiones y Trabajo Futuro

Uno de los aportes principales de este trabajo es el diseño y desarrollo del componente de software OntoForms, el cual permite generar de forma automática una estructura que describe la interfaz gráfica de un formulario para ingresar individuos a una ontología de dominio. Las aplicaciones basadas en ontologías se benefician de OntoForms, dado que pueden basarse en el servicio de generación de formularios para el desarrollo de la interfaz gráfica donde el usuario interactúa con la ontología.

La estructura que se genera representa a un formulario para insertar y actualizar individuos de la ontología. También instancia el objeto y las data properties asociadas con el individuo. El mayor desafío con la generación del algoritmo de OntoForms fue la gestión de propiedades pertenecientes a una clase de la ontología en el paradigma de mundo abierto, optando por una solución basada en *RDF as Frames* de Jena. La funcionalidad clave que distingue a OntoForms es la capacidad de usar los servicios de razonamiento basados en la semántica formal de lógica de descripciones y el uso de la estrategia de RDF as Frames con el marco de trabajo Jena para definir las propiedades de una clase, asegurando que tanto los axiomas declarados como los inferidos son utilizados para generar la interfaz de usuario.

OntoForms utiliza una estrategia de convención sobre configuración, por lo que la generación de formularios a partir de clases de una ontología funciona desde el momento en que la ontología está cargada en el sistema. Para personalizaciones más avanzadas, también se ofrece una interfaz de administración en donde el usuario administrador puede seleccionar las clases de la ontología y propiedades que deberían excluirse del formulario, traducciones de los campos y marcar clases como transparentes acorde a las necesidades del dominio.

Se desarrolló un ejemplo práctico del uso de OntoForms al implementar la aplicación OntoBreastScreen, una aplicación basada en la ontología BCSR-ONTO sobre prevención en cáncer de mama, en donde el usuario médico hace uso de la ontología para ingresar los datos de una paciente y la aplicación con esos datos y modelos de cálculo de riesgo, genera una sugerencia de estudios preventivos.

El desarrollo del sistema OntoBreastScreen culminó en un prototipo que los usuarios finales pueden utilizar para los casos de uso propuestos. Este prototipo demuestra la flexibilidad del sistema, ya que permite realizar cambios en la ontología que se reflejan automáticamente en el formulario. Además, se puede utilizar un razonador para hacer inferencias y generar recomendaciones útiles para los médicos. Se integraron tres modelos de cálculo de riesgo, utilizando guías para los modelos ASC y MSP UY. Para el modelo IBIS, se creó una integración personalizada mediante un web-scraper que ejecuta un formulario en línea.

Para reflejar la opinión de los usuarios finales acerca del prototipo, se realizó una encuesta donde se buscó medir la valoración del mismo en cuanto a la utilidad de la aplicación, la simpleza, la relevancia y la completitud de la herramienta. El procesamiento de esta encuesta se encuentra en proceso y será un insumo muy valioso para futuras decisiones sobre el sistema OntoBreastScreen.

La solución OntoForms se desarrolló con las últimas tecnologías de la industria, lo que garantiza un fácil mantenimiento y despliegue en diferentes entornos. Se implementaron los requisitos para la generación de formularios a partir de ontologías, aunque se identificaron varios aspectos que pueden mejorarse en el futuro.

Posibles extensiones a OntoForms como trabajo futuro:

- Brindar servicio de Persistencia en el triplestore de las tripletas obtenidas de los formularios.
- Proveer un sistema de manejo de usuarios y seguridad para el uso de la herramienta.
- Extender la representación de los formularios para poder tener preguntas condicionales.
- Look&Feel del administrador web.

Respecto a la aplicación OntoBreastScreen, podemos concluir que hemos creado un prototipo funcional que destaca por la facilidad de su desarrollo en co-diseño con la experta en el dominio, Lic. Yasmine Anchén. Esto fue posible gracias a que el sistema se basó en una ontología del dominio, que sirvió como un

modelo conceptual enriquecido semánticamente. Además, hemos integrado diversas calculadoras y recomendaciones para pacientes en una única herramienta. Para impulsar su evolución, se han identificado los siguientes puntos de mejora:

- Persistir los comentarios de los médicos junto a la decisión de si aceptó o no la recomendación.
- Persistir los datos de la mujer si ella acepta participar de estudio de investigación estadístico.
- Obtener licencia de calculadoras e integrarse vía API.
- Mostrar botón para realizar la justificación de las inferencias asociadas a las recomendaciones.

Finalmente, señalar que la demo del sistema OntoBreastScreen se encuentra disponible en: http://risk-prevention-studies.web.elasticloud.uy/

OntoForms se encuentra disponible en:
https://github.com/brunoszilagyiibarra/ontoforms-backend| y
https://github.com/brunoszilagyiibarra/ontoforms-admin

y se puede probar con distintas ontologías en:
http://ontoforms-admin-ui.web.elasticloud.uy/

Referencias

Anchén, Y., Rohrer, E., y Motz, R. (2023). An ontology for breast cancer screening. En *International conference on conceptual modeling* (pp. 5–14).

Anchén, Y., Rohrer, E., y Motz, R. (2024). *Breast cancer screening recommendation ontology.* https://bioportal.bioontology.org/ontologies/BCSR-ONTO/.

Apache jena fuseki servidor sparql. (s.f.). Descargado 2024-12-19, de https://jena.apache.org/documentation/fuseki2/

Cardoso, F., Kyriakides, S., Ohno, S., Penault-Llorca, F., Poortmans, P., Rubio, I., ... Senkus, E. (2019). Early breast cancer: Esmo clinical practice guidelines for diagnosis, treatment and follow-up. *Annals of oncology*, *30*.

Fuseki server protocol. (s.f.). Descargado 2024-12-19, de https://jena.apache.org/documentation/fuseki2/fuseki-server-protocol.html#datasets-and-services

Gonçalves, R. S., Tu, S. W., Nyulas, C. I., Tierney, M. J., y Musen, M. A. (2017, agosto). An ontology-driven tool for structured data acquisition using Web forms. *Journal of Biomedical Semantics*, *8*(1), 26. Descargado 2023-09-18, de https://doi.org/10.1186/s13326-017-0133-1 doi: 10.1186/s13326-017-0133-1

Ibis v8 calculator. (s.f.). Descargado 2024-12-23, de https://ems-trials.org/riskevaluator/

Kim, G., y Bahl, M. (2021). Assessing risk of breast cancer: A review of risk prediction models. *Journal of Breast Imaging*, *3*.

Liu, F. (2009, diciembre). An ontology-based approach to Automatic Generation of GUI for Data Entry. *University of New Orleans Theses and Dissertations*. Descargado de https://scholarworks.uno.edu/td/1094

Peng, Y., y Kang, Y. (2013). Ontology-Based Dynamic Forms for Manufacturing Capability Information Collection. En V. Prabhu, M. Taisch, y D. Kiritsis (Eds.), *Advances in Production Management Systems. Sustainable Production and Service Supply Chains* (Vol. 415, pp. 468–476). Berlin, Heidelberg: Springer Berlin Heidelberg. Descargado 2023-10-04, de http://link.springer.com/10.1007/978-3-642-41263-9_58 doi: 10.1007/978-3-642-41263-9_58

Project lombok. (s.f.). Descargado 2024-12-19, de https://projectlombok.org/

Rdf as frames. (s.f.). Descargado 2024-12-19, de https://jena.apache.org/documentation/notes/rdf-frames.html

Rocchetti Martínez, N. P., y Labandera, G. (2016). Generación automática de formularios para el ingreso de datos en ontologías. Aplicación en la implementación de una ontología de percepciones de piezas de arte. *Universidad de la República.Facultad de Ingeniería. Tesis de Grado en Ingeniería en Computación.*. Descargado de https://hdl.handle.net/20.500.12008/27334

Rutesic, P., Radonjic-Simic, M., y Pfisterer, D. (2021). An Enhanced Metamodel to Generate Web Forms for Ontology Population. En B. Villazon-Terrazas, F. Ortiz-Rodriguez, S. Tiwari, A. Goyal, y MA. Jabbar (Eds.), *Knowledge Graphs and Semantic Web* (pp. 109–124). Cham: Springer International Publishing. doi: 10.1007/978-3-030-91305-2_9

Vcelak, P., Kryl, M., Kratochvil, M., y Kleckova, J. (2017, octubre). Ontology-based web forms — Acquisition and modification of structured data. En *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)* (pp. 1–5). Shanghai: IEEE. Descargado 2023-09-13, de http://ieeexplore.ieee.org/document/8302291/ doi: 10.1109/CISP-BMEI.2017.8302291

Anexo A

Implementaciones

```
    @PostConstruct
    private void initializeDatasets(){

        restClient = RestClient.builder().requestInterceptor(new
            BasicAuthenticationInterceptor(username, pass)).build();

        var datasets = restClient.get().uri(URI.create(host + "/$/datasets"))
                .retrieve()
                .body(Map.class);

        List<Map<String, Object>> o = (List<Map<String, Object>>)
            datasets.get("datasets");
        List<String> dsNames = o.stream().map(m -> (String) m.get("ds.name")).toList();

        generateDataSetIfNotExists(ONTOLOGIES_TBOX_DATASET, dsNames);
        generateDataSetIfNotExists(ONTOLOGIES_ABOX_DATASET, dsNames);
        generateDataSetIfNotExists(ONTOLOGIES_CONFIG_DATASET, dsNames);
    }

    private void generateDataSetIfNotExists(String dsName, List<String> dsAvailable) {
        if(dsAvailable.contains(dsName)) {
            log.info("Dataset {} ya esta disponible", dsName);
        } else {
            MultiValueMap<String, String> bodyPair = new LinkedMultiValueMap();
            bodyPair.put("dbType", List.of("tdb2"));
            bodyPair.put("dbName", List.of(dsName));

            ResponseEntity<Void> bodilessEntity = restClient.post().uri(URI.create(host
                + "/$/datasets"))
                    .contentType(MediaType.APPLICATION_FORM_URLENCODED)
                    .body(bodyPair)
                    .retrieve()
                    .toBodilessEntity();

            if(bodilessEntity.getStatusCode().is2xxSuccessful()) {
                log.info("Se genero con exito el dataset {}", dsName);
            } else {
                log.info("hubo un fallo {} al intentar crear el dataset {}",
                    bodilessEntity.getStatusCode(), dsName);
            }
        }
```

```
    }
}
```

Listing A.1: Inicialización datasets

```
   /**
    * Se obtienen todos los nombres de las ontologias guardadas en el triplestore,
       dataset ontologies.
    * @return
    */
    public List<OntologyName> getAllOntologyNames() {
        try (RDFConnection conn = RDFConnectionFuseki.connect(host +
            ONTOLOGIES_TBOX_DATASET)) {
            Iterable<String> iterableNames = () -> conn.fetchDataset().listNames();

            return StreamSupport.stream(iterableNames.spliterator(), false)
                    .map(graphName -> new OntologyName(graphName,
                        graphName.replace(NAMED_GRAPH_PREFIX, "")))
                    .toList();
        }
    }
```

Listing A.2: Listar todas las ontologías guardadas

```
{
"classUri": "http://www.semanticweb.org/mdebe/ontologies/example#Person",
"sectionName": "Person",
"fields": [
{
    "classType": "object-field",
    "label": "has_Brother",
    "uri":
        "http://www.semanticweb.org/mdebe/ontologies/example#has_Brother",
    "order": null,
    "options": [
        {
            "label": "Tom_Doe",
            "uri":
                "http://www.semanticweb.org/mdebe/ontologies/example#Tom_Doe"
        }
    ],
    "singleOption": false,
    "type": "multi-option"
},
{
    "classType": "object-field",
    "label": "has_Gender",
    "uri":
        "http://www.semanticweb.org/mdebe/ontologies/example#has_Gender",
    "order": null,
    "options": [
        {
            "label": "Male",
            "uri":
                "http://www.semanticweb.org/mdebe/ontologies/example#Male"
        },
        {
            "label": "Female",
            "uri":
                "http://www.semanticweb.org/mdebe/ontologies/example#Female"
```

```
32          }
33      ],
34      "singleOption": true,
35      "type": "single-option"
36  },
37  {
38      "classType": "datatype-field",
39      "label": "has_Age",
40      "uri": "http://www.semanticweb.org/mdebe/ontologies/example#has_Age",
41      "order": null,
42      "datatype": "integer",
43      "type": "integer"
44  },
45      "subForms": [
46  {
47  "classUri":
        "http://www.semanticweb.org/mdebe/ontologies/example#Direct_Ancestry",
48  "sectionName": "Direct_Ancestry",
49  "fields": [
50  {
51      "classType": "object-field",
52      "label": "father",
53      "uri": "http://www.semanticweb.org/mdebe/ontologies/example#father",
54      "order": null,
55      "options": [
56          {
57              "label": "Tom_Doe",
58              "uri":
                    "http://www.semanticweb.org/mdebe/ontologies/example#Tom_Doe"
59          }
60      ],
61      "singleOption": true,
62      "type": "single-option"
63  },
64  {
65      "classType": "object-field",
66      "label": "mother",
67      "uri": "http://www.semanticweb.org/mdebe/ontologies/example#mother",
68      "order": null,
69      "options": [
70          {
71              "label": "Mary_Doe",
72              "uri":
                    "http://www.semanticweb.org/mdebe/ontologies/example#Mary_Doe"
73          },
74          {
75              "label": "Miss_Havisham",
76              "uri":
                    "http://www.semanticweb.org/mdebe/ontologies/example#Miss_Havisham"
77          },
78          {
79              "label": "Sarah_Doe",
80              "uri":
                    "http://www.semanticweb.org/mdebe/ontologies/example#Sarah_Doe"
81          },
82          {
83              "label": "Daisy_Buchanan",
84              "uri":
                    "http://www.semanticweb.org/mdebe/ontologies/example#Daisy_Buchanan"
85          },
86          {
```

```
87              "label": "Susan_Doe",
88              "uri":
                    "http://www.semanticweb.org/mdebe/ontologies/example#Susan_Doe"
89          },
90          {
91              "label": "Beth_Doe",
92              "uri":
                    "http://www.semanticweb.org/mdebe/ontologies/example#Beth_Doe"
93          }
94      ],
95      "singleOption": true,
96      "type": "single-option"
97  }
98      ],
99      "subForms": []
100  }
101 ]
102 }
103 }
```

Listing A.3: Salida de la ejecución de algoritmo generación formularios