Extensión de Hermit para bases de conocimiento con metamodelado y aplicación al dominio de contabilidad

Bruno Di Bello

Tutora: Edelweis Rohrer
Co-tutora: Regina Motz

Proyecto de Grado
Ingeniería en Computación
Universidad de la República
2021

Resumen

Las ontologías son artefactos ampliamente utilizados para representar un dominio con sus conceptos más relevantes, las relaciones que existen entre dichos conceptos, y afirmaciones o axiomas acerca de los conceptos, relaciones y de las instancias o elementos del dominio. En particular, OWL es un lenguaje estándar de W3C para la representación de ontologías, basado en lógica descriptiva.

Hermit es una implementación Java del algoritmo de razonamiento Tableau, que dada una ontología en OWL, devuelve si es consistente y también infiere nuevo conocimiento a partir del conocimiento declarado en la ontología. Sin embargo, ni OWL ni los razonadores existentes para OWL, como Hermit, están pensados para representar y asegurar la consistencia de ontologías con metamodelado. La idea de metamodelado consiste en que un concepto A (que puede tener instancias) es a su vez una instancia de otro concepto B, es decir que A es tratado como instancia y como concepto en la ontología.

Este proyecto tiene como principal objetivo extender al razonador Hermit para que procese ontologías con metamodelado, en base a un enfoque que extiende a la lógica descriptiva $\mathcal{SHIQ}$ y al algoritmo Tableau con nuevas reglas para metamodelado. Además, en este proyecto se muestra la utilidad de esta extensión para una aplicación de contabilidad.

Los objetivos planteados se cumplen a partir del desarrollo de nuevos componentes dentro de Hermit, y de una aplicación Java Swing que a través de una interfaz de usuario, hace uso de una ontología con metamodelado, y de la versión extendida de Hermit. Adicionalmente, fue necesario extender la OWL API para Java, e implementar algunas optimizaciones para mejorar el desempeño del razonador extendido.

Índice general

1. Introducción

Las *ontologías* son una "especificación formal y explícita de una conceptualización compartida", definición que es considerada como la más ampliamente aceptada [1]. Actualmente, las ontologías se usan para representar un dominio de interés dentro de lo que se conoce como la web semántica, y también para enriquecer la representación de los dominios de negocio en aplicaciones de diferentes organizaciones [2]. El lenguaje de ontologías más utilizado es OWL, que es un estándar de la World Wide Web Consortium (W3C), y está basado en el formalismo *lógica descriptiva*, que es una familia de fragmentos decidibles de la lógica de primer orden [3]. En particular, $\mathcal{SROIQ}$ es una lógica altamente expresiva de esta familia, siendo la lógica en la que se basa el lenguaje OWL [4, capítulo 5]. El lenguaje OWL permite representar los conceptos (o clases) de un dominio, sus instancias (o individuos) que son los elementos del dominio y relaciones binarias entre ellos (o roles). Sin embargo, los fragmentos decidibles del lenguaje OWL no permiten declarar que un individuo a es el mismo objeto de la realidad que un concepto A, ni que el concepto A es una instancia de otro concepto B, es decir que A, además de ser un concepto es también tratado como un individuo. Esta correspondencia entre individuos y conceptos es lo que se llama *metamodelado*. Una ventaja del metamodelado, que describe su importancia, puede verse en el caso presentado en [5, sección 2], donde por ejemplo, en una ontología del dominio de contabilidad, el individuo "Pago Mensual" es una instancia del concepto "Definición de Asiento", pero también es modelado como un concepto, con individuos representados por distintas instancias de pagos mensuales.

Un *razonador* es una implementación de un algoritmo de razonamiento. Dada una ontología, un algoritmo de razonamiento chequea si es consistente (tiene un modelo satisfacible) y devuelve conocimiento que se infiere a partir de las sentencias o axiomas declarados en la ontología. Los razonadores más conocidos, como Hermit y Pellet, entre otros, implementan algoritmos de razonamiento para la lógica $\mathcal{SROIQ}$ que subyace OWL [6, 7]. Sin embargo, en presencia de relaciones de metamodelado entre individuos y conceptos, ninguno de ellos tiene la capacidad de detectar todas las inconsistencias que se deben a este tipo de relaciones.

A partir de los trabajos "The description logic SHIQ with a flexible meta-modelling hierarchy" y "A description logic for unifying different points of view", se define la lógica $\mathcal{SHIQM}^*$ para representar relaciones de metamodelado, extendiendo la lógica $\mathcal{SHIQ}$ que es un fragmento (también expresivo) de $\mathcal{SROIQ}$[5, 8]. Dado que la lógica $\mathcal{SHIQM}^*$ no está soportada por los razonadores más usados para OWL como Hermit y Pellet, surge la necesidad de implementar una extensión capaz de verificar la consistencia de ontologías con metamodelado, en particular para la lógica $\mathcal{SHIQM}^*$. El objetivo principal del presente trabajo es la implementación de esta extensión.

Para la implementación de una extensión que dé soporte a la lógica $\mathcal{SHIQM}^*$, en este proyecto se seleccionó al razonador Hermit, que al igual que Pellet es uno de los razonadores más usados. Además de ser una implementación standalone, Hermit está implementado como un plugin de la herramienta Protégé, plataforma open-source ampliamente utilizada para crear ontologías. Sin embargo, a diferencia de Pellet, Hermit está más optimizado y hasta el presente está siendo ampliamente utilizado [6].

Se tomó un caso de estudio de contabilidad de manera tal de poder aplicar en un caso real la extensión de Hermit implementada. Bajo los siguientes conceptos se basa la construcción de una aplicación para modelar un escenario de contabilidad. Los asientos contables son conjuntos de anotaciones que se hacen en un libro diario de contabilidad. El propósito de los mismos es registrar un hecho que provoca una modificación cuantitativa o cualitativa en la composición del patrimonio de una empresa y por ende en las cuentas de las mismas. Cada asiento se compone al menos de dos anotaciones, una al debe y otra al haber, las cuales hacen movimientos inversos y cada uno afecta al menos a una cuenta.

En particular, el caso de estudio de contabilidad considerado en este proyecto es concretamente la aplicación de contabilidad correspondiente al Sistema Integrado de Gestión de Garantías de Alquiler de la Contaduría General de la Nación (SIGGA). Básicamente, esta aplicación realiza los registros contables correspondientes a los pagos de alquiler de los inquilinos, ya sea a través de un descuento de sus haberes o directamente por caja o banco. Asimismo, se registran pagos a los arrendadores o dueños de la propiedades arrendadas, entre otros múltiples movimientos contables.

Los principales aportes del presente trabajo son:

- la implementación de la extensión del razonador Hermit para metamodelado,

de acuerdo a la nueva lógica $\mathcal{SHIQM}^*$,

- la aplicación de algunos mecanismos de optimización para lograr un desempeño aceptable del razonador extendido, y

- la implementación de una interfaz de usuario que invoca al razonador extendido, con la finalidad de mostrar la utilidad de las relaciones de metamodelado para la conceptualización de un caso de uso de contabilidad.

La extensión de Hermit implica además la extensión de la OWL API (API Java para manipular ontologías OWL) para los nuevos constructores de metamodelado implementados.

Lo que resta del presente informe está organizado como se describe a continuación. El Capítulo 2 describe los fundamentos básicos de lógica descriptiva, algoritmos de razonamiento, y en particular la implementación Hermit, y finalmente el enfoque de metamodelado en el que se basa el presente trabajo. El Capítulo 3 presenta los requerimientos funcionales y no funcionales que el proyecto debe resolver. En los capítulos 4 y 5 se explica la estrategia de diseño de la extensión de Hermit, y del prototipo Java de la aplicación de contabilidad, así como su implementación. La verificación del cumplimiento de los requerimientos se describe en el Capítulo 6, relacionando los casos de prueba con los escenarios identificados en el Capítulo 3. Finalmente, se presentan algunas conclusiones y trabajos futuros en el Capítulo 7.

2. Fundamentos Básicos

2.1. OWL y lógica descriptiva

OWL es un lenguaje estándar de W3C basado en el formalismo *lógica descriptiva*, que permite representar conceptos, individuos y roles, en otras palabras puede ser utilizado para representar ontologías. En el año 2004 la W3C presentó la primera versión de OWL. Actualmente cuenta con una segunda versión OWL2 publicada en el año 2009 por el mismo grupo, con una segunda edición publicada en el año 2012 [4, capítulo 4]. Si bien la sintaxis de OWL más ampliamente usada es OWL RDF, basada en el estándar RDF, en este informe se utiliza la sintaxis de lógica descriptiva, por ser más concisa. [4, capítulo 5]

Lógica descriptiva es una familia de lógicas que son fragmentos decidibles de la lógica de primer orden, por ejemplo la lógica $\mathcal{SHIQ}$ [4, capítulo 5]. Para definir la sintaxis de $\mathcal{SHIQ}$ se consideran los siguientes conjuntos disjuntos: $a, b, \ldots$ de individuos, $A, B, \ldots$ de conceptos atómicos y $R, S, \ldots$ de roles atómicos, que contiene todos los nombres de roles R y sus inversos R^-.
$\mathcal{SHIQ}$ permite expresar que un rol R es *transitivo* a través de la declaración $Trans(R)$, y además que dados R y S, $R \sqsubseteq S$ significa que el conjunto de pares de individuos que están en la relación R es un subconjunto del conjunto de pares correspondiente a S.
Si $\sqsubseteq^*$ es la clausura transitiva y reflexiva de $\sqsubseteq$, un rol R es un *subrol* de un rol S si $R \sqsubseteq^* S$. Un rol es *simple* si no es transitivo ni tiene subroles transitivos.

La siguiente sintaxis permite construir *conceptos generales* C, D en $\mathcal{SHIQ}$:
$C, D ::= A \mid \top \mid \bot \mid (\neg C) \mid (C \sqcap D) \mid (C \sqcup D) \mid (\forall R.C) \mid (\exists R.C) \mid (\geqslant n\ S.C) \mid (\leqslant n\ S.C)$
donde n es un entero no negativo, S es un rol simple, $\top$ es todo el universo, es decir, todo el conjunto de elementos del dominio, y $\bot$ es el conjunto vacío. $\neg C$ representa al conjunto de elementos que no pertenecen a C, $C \sqcap D$ es el conjunto de elementos que pertenecen a C y a D, y $C \sqcup D$ es la unión de los conjuntos C y D. $\forall R.C$ representa al conjunto de elementos x del dominio, que en caso de estar

vinculados a otros elementos y a través del rol R, se cumple que y pertence al concepto C. $\exists R.C$ representa al conjunto de elementos que están vinculados con al menos un elemento que pertenece a C. El significado de $\geqslant n\ S.C$ y $\leqslant n\ S.C$ es similar a $\exists R.C$, pero imponiendo una restricción adicional en el mínimo y máximo número de elementos relacionados, respectivamente.

Definition 1 (Ontología in $\mathcal{SHIQ}$)
Una ontología $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ en la lógica $\mathcal{SHIQ}$ *está compuesta por una Tbox* $\mathcal{T}$*, una Rbox* $\mathcal{R}$ *y una Abox* $\mathcal{A}$*, que se definen de la siguiente manera:*

1. $\mathcal{T}$ *es un conjunto finito de axiomas* $C \sqsubseteq D$*, donde* C*,* D *son dos conceptos cualesquiera,*

2. $\mathcal{R}$ *es un conjunto finito de axiomas de la forma* $R \sqsubseteq S$ *y declaraciones de roles transitivos* $Trans(R)$*, donde* R*,* S *son roles atómicos,*

3. $\mathcal{A}$ *es un conjunto finito de sentencias* $C(a)$*,* $R(a, b)$*,* $a = b$*, o* $a \neq b$ *donde* C *es un concepto,* R *es un rol, y* a*,* b *son individuos.*

Intuitivamente, $C \sqsubseteq D$ significa que el conjunto de elementos del dominio que pertenecen a C es un subconjunto del conjunto correspondiente a D. $C(a)$ significa que el individuo a pertenece a C, $R(a, b)$ que el par (a, b) pertenece a la relación R, y $a = b$, $a \neq b$ expresan que los individuos a, b son el mismo elemento del dominio o diferentes elementos, respectivamente. La semántica de los conceptos generales y de los axiomas en la lógica $\mathcal{SHIQ}$ se formaliza a través de la noción de *interpretación* que se define a continuación.

Definition 2 (Interpretación en $\mathcal{SHIQ}$)
Una interpretación $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ *de una ontología en* $\mathcal{SHIQ}$ *se define como un* dominio Δ *y una función* $\cdot^{\mathcal{I}}$*, que asocia cada concepto de la ontología a un subconjunto de* Δ*, cada rol a un subconjunto de* $\Delta \times \Delta$ *y cada individuo a un elemento de* Δ*, tal que para todos los conceptos* C*,* D*, roles* R*,* S*, y enteros no negativos* n*, se satisfacen las siguientes ecuaciones, donde* $\sharp X$ *es la cardinalidad de un conjunto* X*:*

$\top^{\mathcal{I}} = \Delta$
$\bot^{\mathcal{I}} = \emptyset$
$(R^{-})^{\mathcal{I}} = \{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}$
$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$

$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$(\neg C)^{\mathcal{I}} = \Delta \backslash C^{\mathcal{I}}$
$(\exists R.C)^{\mathcal{I}} = \{x \mid x \in \Delta, \exists y.(x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
$(\forall R.C)^{\mathcal{I}} = \{x \mid x \in \Delta, \forall y.(x,y) \in R^{\mathcal{I}} \text{ implica } y \in C^{\mathcal{I}}\}$
$(\geqslant n\ R.C)^{\mathcal{I}} = \{x \mid \sharp\{y.(x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$
$(\leqslant n\ R.C)^{\mathcal{I}} = \{x \mid \sharp\{y.(x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$

Definition 3 (Modelo en $\mathcal{SHIQ}$)
Una interpretación $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ *es un* modelo de una ontología $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ in $\mathcal{SHIQ}$ *si cumple con las condiciones siguientes.*

1. *Para todo axioma* $C \sqsubseteq D$ *en* $\mathcal{T}$ *se cumple que* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

2. *Para todo axioma* $R \sqsubseteq S$ *en* $\mathcal{R}$ *se cumple que* $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ *y para todo axioma* $Trans(R)$ *en* $\mathcal{R}$ *se cumple que si* $\{(x,y),(y,z)\} \subseteq R^{\mathcal{I}}$ *entonces* $(x,z) \in R^{\mathcal{I}}$.

3. *Para todo axioma* $C(a)$, $R(a,b)$, $a = b$ *o* $a \neq b$ *en* $\mathcal{A}$ *se cumple que* $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, $a^{\mathcal{I}} = b^{\mathcal{I}}$ *y* $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ *respectivamente.*

Consideremos la realidad de la aplicación de contabilidad del Sistema Integrado de Gestión de Garantía de Alquileres (SIGGA) de la Contaduría General de la Nación, que es el caso de uso considerado para la construcción de un prototipo que muestra la utilidad de la extensión implementada[1]. En dicho escenario, consideremos los conceptos: *AsientoPagoInquilino* que representa a los asientos de contabilidad que se registran cada vez que un inquilino realiza un pago de alquiler, y *AsientoAlquilerMensual* que representa los asientos de cálculo del monto de alquiler mensual para cada inquilino. Consideremos entonces la siguiente ontología $\mathcal{O} = (\mathcal{T}, \mathcal{A})$:

$\mathcal{T} = \{AsientoPagoInquilino \sqcap AsientoAlquilerMensual \sqsubseteq \bot\}$
$\mathcal{A} = \{AsientoPagoInquilino(as1), AsientoAlquilerMensual(as2)\}$

y la interpretación $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$:

$\Delta = \{a1, a2\}$
$as1^{\mathcal{I}} = a1$
$as2^{\mathcal{I}} = a2.$
$AsientoPagoInquilino^{\mathcal{I}} = \{a1\}$

[1]En el Capítulo 3 se describe este escenario en detalle

$AsientoAlquilerMensual^{\mathcal{J}} = \{a2\}$

Dado $a1$ y $a2$ diferentes elementos de Δ, el axioma en $\mathcal{T}$ se cumple ya que $\{a1\} \cap \{a2\} \subseteq \emptyset$. Los axiomas en $\mathcal{A}$ también se satisfacen porque $a1 \in \{a1\}$ y $a2 \in \{a2\}$. Por lo tanto, de acuerdo a la Definición 3 $\mathcal{I}$ es un modelo de $\mathcal{O}$.

Sin embargo, si se considera la interpretación $\mathcal{I}$ cambiando $as2^{\mathcal{J}} = a2$ por $as2^{\mathcal{J}} = a1$, $\mathcal{I}$ no es un modelo de $\mathcal{O}$ porque no satisface el axioma:
$AsientoPagoInquilino \sqcap AsientoAlquilerMensual \sqsubseteq \bot$, ya que la intersección de los conjuntos $AsientoPagoInquilino^{\mathcal{J}}$ y $AsientoAlquilerMensual^{\mathcal{J}}$, que son ambos $\{a1\}$, no es el conjunto vacío.

Se dice que *una ontología* $\mathcal{O}$ *es consistente* si existe al menos un *modelo de* $\mathcal{O}$. La ontología del ejemplo es consistente ya que la primera interpretación es efectivamente un modelo.

2.2. Algoritmos de razonamiento

Dada una ontología, un *algoritmo de razonamiento* devuelve si la ontología es consistente, o sea si tiene un modelo. Además, hace explícito conocimiento no declarado en la ontología, que se infiere de ella.

Existen diferentes algoritmos de razonamiento, entre los cuales uno de los más utilizados para verificar la consistencia de una ontología en OWL es el *algoritmo de Tableau*. Los razonadores más comúnmente utilizados, como Hermit y Pellet implementan el algoritmo de tableau. Este algoritmo tiene como principal ventaja que aplica a todas las lógicas que son fragmentos de OWL.

Dada una ontología $\mathcal{O}$, *algoritmo de tableau* trata de construir un modelo de $\mathcal{O}$. Si logra construirlo, entonces devuelve que $\mathcal{O}$ es *consistente*, de lo contrario devuelve que $\mathcal{O}$ es *inconsistente*. A partir de los elementos del dominio representados por los individuos y el conocimiento declarado en los axiomas de Abox sobre los individuos, el algoritmo construye un grafo. Este grafo inicial es extendido por la aplicación de un conjunto de reglas, las cuales están determinadas por los axiomas de Tbox y Rbox de la ontología. Cuando ya no se pueden aplicar más reglas, y el grafo obtenido no contiene ninguna contradicción, entonces el algoritmo retorna que la ontología es consistente, de lo contrario retorna que es inconsistente. Una contradicción está dada por situaciones como por ejemplo que un individuo a de

$\mathcal{O}$ es instancia de un concepto C y también de su complemento $\neg C$.

Antes de construir el grafo anteriormente mencionado, el algoritmo de tableau transforma a la ontología $\mathcal{O}$ a la *forma normal de negación* (FNN). Una ontología está en FNN si de tener negaciones, estas afectan solamente a conceptos atómicos. En caso de conceptos que no sean atómicos con negación, lo que se hace es mover la negación inmediatamente antes del nombre del concepto, por ejemplo, el concepto $\neg(A \sqcup B)$ se transforma en $\neg A \sqcap \neg B$. Además, se transforman los axiomas de Tbox $C \sqsubseteq D$ en $\neg C \sqcup D$.

Una vez que la ontología $\mathcal{O}$ está en FNN, el algoritmo de tableau ejecuta los siguientes tres pasos[2]:

1. A partir de los axiomas de Abox en $\mathcal{O}$, se *inicializa un grafo* $\mathcal{L}$, que tiene como nodos los individuos declarados en $\mathcal{O}$. A cada nodo a del grafo inicial se le asocia el conjunto de conceptos C, D,..., $\mathcal{L}(a) = \{C, D, \dots\}$, para todos los axiomas $C(a)$, $D(a)$,.. declarados en $\mathcal{O}$, es decir C, D,... es el conjunto de conceptos de los que a es instancia. A cada par de nodos a, b se le asocia un conjunto de roles R, S,..., $\mathcal{L}(a, b) = \{R, S, \dots\}$ para todos los axiomas $R(a, b)$, $S(a, b)$,... declarados en $\mathcal{O}$. Además, para los axiomas $a = b$, el algoritmo selecciona un elemento representativo (por ejemplo a) de todos los individuos iguales a a, como nodo en el grafo $\mathcal{L}$. La Figura 2.1 ilustra el grafo inicial de la ontología $\mathcal{O} = \{A(a), B(b), R(a, b), S(b, c)\}$

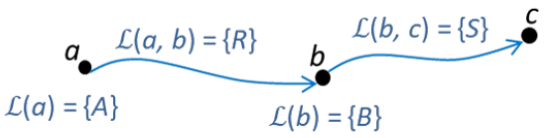

Figura 2.1: Ejemplo de grafo inicial

2. Se *aplica un conjunto de reglas* iterando en el conjunto de nodos del grafo inicial, en forma no determinística, hasta que no sea posible aplicar más reglas. El conjunto de reglas que se aplican están relacionadas con los constructores del lenguaje utilizados en la ontología. A continuación se describen en forma intuitiva algunas de las reglas, relacionadas con los axiomas de Tbox, y los constructores $\sqcup$ y $\exists$.

   La *regla de Tbox* asocia a todos los nodos del grafo los conceptos que se

[2]La especificación formal del algoritmo de tableau se encuentra en [4, capítulo 5]

obtienen de llevar los axiomas de Tbox a FNN, es decir, dado un axioma $C \sqsubseteq D$, el algoritmo asigna $\mathcal{L}(a) = \{\neg C \sqcup D\}$ a todos los nodos del grafo.

La *regla del OR* se aplica a todos los nodos a tales que un concepto de la forma $C \sqcup D$ pertenece a $\mathcal{L}(a)$. En este caso, el algoritmo asocia no determinísticamente o bien C o bien D a $\mathcal{L}(a)$. De forma similar, la *regla del AND* se aplica a todos los nodos a tales que un concepto de la forma $C \sqcap D$ pertenece a $\mathcal{L}(a)$, pero en este caso al algoritmo asocia C y D a $\mathcal{L}(a)$.

La *regla del exists* se aplica a todos los nodos a tales que un concepto de la forma $\exists R.C$ pertenece a $\mathcal{L}(a)$. En este caso, si no existe un nodo x tal que $R \in \mathcal{L}(a, x)$ y $C \in \mathcal{L}(x)$, se extiende el grafo con la creación de un nuevo nodo x, y se asigna $\mathcal{L}(a, x) = \{R\}$ y $\mathcal{L}(x) = \{C\}$.

Por ejemplo, consideremos la ontología $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ donde *AsientoPagoInquilino* representa a los asientos pago de alquiler de un inquilino, *Asiento* representa todo el conjunto de asientos, *Detalle* es el conjunto de todos los detalles (al debe y haber) de los asientos y *detDebe* es el rol que vincula al conjunto de asientos con los detalles que se registran al debe.

$\mathcal{T} = \{AsientoPagoInquilino \sqsubseteq Asiento\}$
$\mathcal{A} = \{AsientoPagoInquilino(as), \exists detDebe.Detalle(as)\}$

El grafo inicial es:
$\mathcal{L}(as) = \{AsientoPagoInquilino, \exists detDebe.Detalle\}$

Si se aplica la regla de Tbox, se agrega a $\mathcal{L}(as)$ la FNN de $AsientoPagoInquilino \sqsubseteq Asiento$:

$\mathcal{L}(as) = \{AsientoPagoInquilino, \exists detDebe.Detalle, \neg AsientoPagoInquilino \sqcup Asiento\}$

Al aplicar la regla del OR, si en la disjunción $\neg AsientoPagoInquilino \sqcup Asiento$ se elige $\neg AsientoPagoInquilino$, se da una contradicción porque $AsientoPagoInquilino$ pertenece a $\mathcal{L}(as)$. Entonces el algoritmo ejecuta un mecanismo llamado *backtracking* por el cual retrocede en la aplicación de las reglas y selecciona el otro elemento de la disjunción, quedando el grafo:

$\mathcal{L}(as) = \{AsientoPagoInquilino, \exists detDebe.Detalle, Asiento\}$

Dado que $\exists detDebe.Detalle \in \mathcal{L}(as)$, es posible aplicar la regla del exists, obteniendo:

$\mathcal{L}(as) = \{AsientoPagoInquilino, \exists detDebe.Detalle, Asiento\}$
$\mathcal{L}(as, x) = \{detDebe\}$
$\mathcal{L}(x) = \{Detalle\}$

En este punto es posible aplicar la regla de Tbox al nuevo nodo x, agregando $\neg AsientoPagoInquilino \sqcup Asiento$ a $\mathcal{L}(x)$, y luego la regla del OR, que por cualquiera de las alternativas no genera una contradicción. Como no es posible aplicar más reglas, se obtiene un modelo canónico de $\mathcal{O}$, representado por el grafo siguiente, e ilustrado en la Figura 2.2.

$\mathcal{L}(as) = \{AsientoPagoInquilino, \exists detDebe.Detalle, Asiento\}$
$\mathcal{L}(as, x) = \{detDebe\}$
$\mathcal{L}(x) = \{Detalle, \neg AsientoPagoInquilino\}$

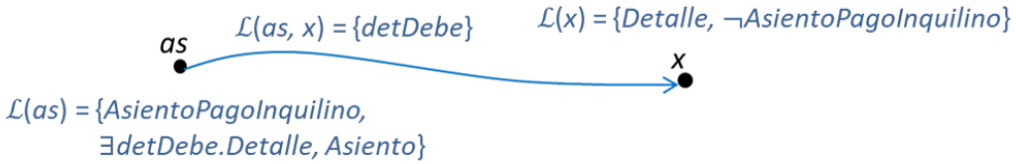

Figura 2.2: Ejemplo de grafo completo

El modelo canónico obtenido es:

$AsientoPagoInquilino^{\mathcal{I}} = Asiento^{\mathcal{I}} = \{as\}$
$detDebe^{\mathcal{I}} = \{as, x\}$
$Detalle^{\mathcal{I}} = \{x\}$

El algoritmo aplica también un mecanismo de *bloqueo* para asegurar la terminación. Por ejemplo, para la ontología $\mathcal{O} = \{A \sqsubseteq \exists R.A, A(a)\}$, el grafo inicial es $\mathcal{L}(a) = \{A\}$. Se aplican las reglas de Tbox y OR obteniéndose el grafo:

$\mathcal{L}(a) = \{A, \exists R.A\}$

Luego, al aplicar la regla del exists se crea un nuevo x, obteniéndose el grafo:

$\mathcal{L}(a) = \{A, \exists R.A\}$ $\mathcal{L}(x) = \{A\}$

En este punto, se aplicarían infinitamente las reglas de Tbox y OR. El bloqueo aplicado difiere dependiendo del fragmento de lógica descriptiva en que está expresada la ontología, pero básicamente, lo que hace es interrumpir la aplicación de las reglas en aquellos nodos que tienen nodos predecesores (a través de un arco del grafo) con el mismo conjunto de conceptos en $\mathcal{L}$.

3. Cuando ya no es posible aplicar más reglas, si no se detectaron contradicciones, se obtiene un modelo canónico de $\mathcal{O}$. En este caso, el algoritmo de Tableau retorna que la ontología es *consistente*. De lo contrario, la ontología es *inconsistente*. Esto implica que por todas las alternativas posibles de aplicación de reglas no determinísticas (por ejemplo la regla del OR) el algoritmo obtiene una contradicción.

Cabe destacar que el no determinismo del algoritmo se da tanto por el orden de aplicación de las reglas, como por la aplicación de reglas que son no determinísticas.

2.2.1. Hermit y el cálculo Hypertableau

Hermit es un razonador open source para ontologías en OWL2, desarrollado en lenguaje Java, que implementa el algoritmo de Tableau optimizado a través de la aplicación del *cálculo hypertableau*. Esto quiere decir que Hermit puede determinar si una ontología es consistente o no, e inferir nuevo conocimiento a partir de una ontología, entre otros. La implementación del *cálculo hypertableau* permite que hermit sea más eficiente que otros razonadores [9].

El cálculo hypertableau es un algoritmo pensado para procesar ontologías en la lógica descriptiva $\mathcal{SROIQ}$ (y todos los fragmentos de ésta, como $\mathcal{SHIQ}$), que busca optimizar al máximo el desempeño de los razonadores basados en el algoritmo de Tableau, reduciendo el no determinismo de este último [9]. Esto se logra a partir del uso de hypertableau e hyperresolution calculi extendidos con una condición de bloqueo para asegurar la finalización. El no determinismo en el algoritmo de Tableau se da principalmente por dos situaciones, *or-branching* y *and-branching*.

El or-branching ocurre al aplicar la *regla del OR* a un nodo x cuando un concepto $C \sqcup D$ pertenece a $\mathcal{L}(x)$. En este caso, el algoritmo de Tableau toma una de las alternativas C o D, y en caso que una de ellas derive en una contradicción, se ejecuta backtracking para tomar la otra alternativa, como se menciona en la sección

anterior. Para ello, se mantiene una estructura de *branching points* con los nodos del grafo a los que debe volver para recorrer los caminos no explorados. La exploración de estas disyunciones deriva en un aumento exponencial de la complejidad del problema. Una fuente importante de disyunciones son los conocidos General Concept Inclusions o GCI, es decir axiomas de la forma $C \subseteq D$, donde C y D son cualquier descripción de concepto, y que al ser llevados a FNN el algoritmo de Tableau transforma estos axiomas en conceptos de la forma $\neg C \sqcup D(s)$. La aplicación de la *regla de Tbox* consiste en agregar conceptos a todos los nodos del grafo, lo que implica la exploración de disjunciones en todos ellos.

El and-branching ocurre al aplicar reglas que agregan nodos al grafo, como la *regla del exists*, descripta en la sección anterior, por lo que el and-branching puede llegar a determinar el tamaño del modelo que se construye al momento de chequear la consistencia de una ontología. Dado que se introducen nuevos nodos, a éstos también se les aplica la *regla de Tbox*, incrementándose el or-branching. Retomando el ejemplo de la sección anterior:

$\mathcal{T} = \{AsientoPagoInquilino \sqsubseteq Asiento\}$
$\mathcal{A} = \{AsientoPagoInquilino(as), \exists detDebe.Detalle(as)\}$

Para el nodo *as*, $\exists detDebe.Detalle$ pertenece a $\mathcal{L}(as)$, por lo que al aplicar la *regla del exists* se crea un nuevo nodo x, y el concepto $\neg AsientoPagoInquilino \sqcup Asiento$ (FNN de $AsientoPagoInquilino \sqsubseteq Asiento$) se agrega a $\mathcal{L}(x)$, introduciéndose una nueva disyunción.

Para evitar el no determinismo producido por el or-branching, el *algoritmo de hypertableau* comienza traduciendo las GCIs de la ontología en cláusulas de llamadas *DL-clauses* que son implicaciones cuantificadas universalmente que contienen conceptos y roles de lógica descriptiva como predicados. Las *DL-clauses* son de la forma:
$\bigwedge U_i \rightarrow \bigvee V_j$
donde los U_i son $R(x, y)$, $A(x)$ y V_j son $R(x, y)$, $A(x)$, $\exists R.C(x)$, $\geqslant nR.C(x)$, $x \approx x$. Por ejemplo, la GCI $\exists detDebe.Detalle \sqsubseteq Asiento$ se transforma en la DL-clause $detDebe(x, y) \wedge Detalle(y) \rightarrow Asiento(x)$. Al igual que en el algoritmo de Tableau descripto en la sección anterior, en el cálculo hypertableau el mecanismo de bloqueo es clave para asegurar la finalización del mismo. Este es el caso de las CGI cíclicas de la forma $C \sqsubseteq \exists R.C$, para las cuales el algoritmo de hyperresolution puede generar infinitos caminos de sucesores. En este tipo de escenarios es que se aplica la técnica de bloqueo denominada pairwise blocking que detecta estos ciclos infinitos [10]. Para reducir el and-branching, el algoritmo de hypertableau hace

una extensión de la técnica de bloqueo de forma que un nodo puede ser bloqueado por otro el cual no necesariamente es su ancestro. A este extensión se la denomina "anywhere pairwise blocking" [9].

El algoritmo de hypertableau tiene dos fases: *preprocessing* e *inferencing*.

El objetivo de la fase de *preprocessing* es transformar una ontología en la lógica $\mathcal{SHIQ}$ en un *ABox normalizado* y una colección de $DL-clauses$ $\mathcal{C}$. Un ABox normalizado es un conjunto de axiomas de las forma $A(x)$, $\geqslant nR.A(x)$, $R(x,y)$ con R un rol atómico. Las GCIs de la Tbox se transforman en $DL-clauses$ como se explicó anteriormente.

El objetivo de la fase de *inferencing* es determinar la consistencia de la ontología en $\mathcal{SHIQ}$, tomando entrada el resultado de la fase de *preprocessing*. En la fase de *inferencing* se aplican las reglas de derivación del cálculo hypertableau para $\mathcal{SHIQ}$ que se muestran en la Figura 2.3[3]. Conceptualmente, el algoritmo va expandiendo el Abox $\mathcal{A}$ de la ontología a medida que aplica estas reglas. La regla más importante es la *Hyp*-rule, en la cual un átomo del encabezado de una $DL-clause$ es derivado solo si todos los átomos del cuerpo de la cláusula se han derivado. σ es un mapeo desde las variables de las $DL-clauses$ a los individuos del $ABox$, y $\sigma(U)$ es el átomo que se obtiene de los U_i remplazando cada variable x por $\sigma(x)$.
La derivación en hypertableau se define como un par (T,λ) donde T es un árbol finito y λ es una función que etiqueta los nodos de T con ABoxes tales que si ϵ es el nodo raíz $\lambda(\epsilon)=A$, y dado un nodo t, si una o más reglas de derivación son aplicables a $\lambda(t)$ y al conjunto de $DL-clauses$ $\mathcal{C}$ de la ontología, entonces t tiene hijos $t_1,...,t_n$ tales que $\lambda(t_1),...,\lambda(t_n)$ son los Aboxes que resultan de aplicar una regla (elegida arbitrariamente) a $\lambda(t)$ y $\mathcal{C}$.
La regla $\bot$-rule se aplica para detectar si existe una contradicción o un *clash*. A contiene un clash si solo si $\bot\in A$, de lo contrario A no contiene contradicciones, es decir que es $clash-free$.

El razonador Hermit es la implementación del algoritmo hypertableau, por lo que entender en profundidad este algoritmo fue fundamental para poder implementar una extensión de este razonador para ontologías con metamodelado, como se plantea en el presente proyecto.

[3] Al aplicar estas reglas se obtiene como resultado las mismas derivaciones que el algoritmo de Tableau clásico anteriormente descripto.

Table 1. Derivation Rules of the Tableau Calculus

Hyp-rule	If 1. $U_1 \wedge ... \wedge U_m \rightarrow V_1 \vee ... \vee V_n \in \Xi(\mathcal{K})$, 2. a mapping $\sigma : N_V \rightarrow N_\mathcal{A}$ exists, for $N_\mathcal{A}$ the set of individuals in $\mathcal{A}$, 3. $\sigma(U_i) \in \mathcal{A}$ for each $1 \leq i \leq m$, 4. $\sigma(V_j) \notin \mathcal{A}$ for each $1 \leq j \leq n$, then if $n = 0$, then $\mathcal{A}_1 = \mathcal{A} \cup \{\bot\}$, otherwise $\mathcal{A}_j := \mathcal{A} \cup \{\sigma(V_j)\}$ for $1 \leq j \leq n$.
$\geq$-rule	If 1. $\geqslant n\, R.C(s) \in \mathcal{A}$, 2. s is not blocked in $\mathcal{A}$, and 3. there are no individuals $u_1, \ldots, u_n$ such that $\{\mathsf{ar}(R, s, u_i), C(u_i) \mid 1 \leq i \leq n\} \cup \{u_i \not\approx u_j \mid 1 \leq i < j \leq n\} \subseteq \mathcal{A}$, then $\mathcal{A}_1 := \mathcal{A} \cup \{\mathsf{ar}(R, s, t_i),\ C(t_i) \mid 1 \leq i \leq n\} \cup \{t_i \not\approx t_j \mid 1 \leq i < j \leq n\}$ where $t_1, \ldots, t_n$ are fresh pairwise distinct successors of s.
$\approx$-rule	If 1. $s \approx t \in \mathcal{A}$ and 2. $s \neq t$ then $\mathcal{A}_1 := \mathsf{merge}_\mathcal{A}(s \rightarrow t)$ if t is named or if s is a descendant of t, $\mathcal{A}_1 := \mathsf{merge}_\mathcal{A}(t \rightarrow s)$ otherwise.
$\bot$-rule	If 1. $s \not\approx s \in \mathcal{A}$ or $\{A(s), \neg A(s)\} \subseteq \mathcal{A}$ and 2. $\bot \notin \mathcal{A}$ then $\mathcal{A}_1 := \mathcal{A} \cup \{\bot\}$.

$$\mathsf{ar}(R, s, t) = \begin{cases} R(s, t) \text{ if } R \text{ is an atomic role} \\ S(t, s) \text{ if } R \text{ is an inverse role and } R = S^- \end{cases}$$

Figura 2.3: Reglas de derivacion del hyperresolution tableau [9]

2.3. Metamodelado

La idea de *metamodelado* tiene que ver con la representación de un conjunto de objetos de un escenario o caso de uso, con diferente nivel de abstracción. Retomando los ejemplos de contabilidad introducidos en la Sección 2.1, el objeto o entidad de la realidad "Asiento de pagos de inquilinos", puede ser modelado en una ontología, como un individuo o como concepto. Es decir, dependiendo de los requerimientos de la aplicación o servicio que se va a implementar, el ingeniero de ontologías adoptará la granularidad más adecuada para su representación. Si lo que se quiere modelar son los diferentes tipos de asiento de la contabilidad de un determinada organización, por ejemplo asientos de pago, asientos cálculo de alquiler mensual, entre otros, entonces será más adecuado representarlos como individuos. Pero si por el contrario, interesa visualizar cada uno de los asientos de pago de inquilinos, como el pago del mes de marzo de 2021 de Juan Pérez, entonces la entidad "Asiento

de pagos de inquilinos" debe modelarse como un concepto. Si en una ontología (o en dos ontologías diferentes) existe un mismo objeto de la realidad que está representado como un individuo a y como un concepto A, la relación o correspondencia entre a y A es lo que se llama *metamodelado*.

Existen diferentes enfoques para representar la relación de metamodelado como una extensión de la lógica descriptiva, cada uno de los cuales extienden también los algoritmos de razonamiento, agregando la capacidad de verificar la consistencia de ontologías con metamodelado. Concretamente, el presente trabajo implementa la extensión del razonador Hermit, de acuerdo a la extensión del algoritmo de Tableau para metamodelado que se describe en los artículos [5, 8]. En este trabajo se define la lógica $\mathcal{SHIQM}^*$ que extiende la lógica $\mathcal{SHIQ}$ con dos nuevos constructores para representar la relación de metamodelado. Con respecto al algoritmo de razonamiento, se agregan cinco nuevas reglas y una verificación adicional (de existencia de ciclos) al algoritmo de Tableau, de forma de asegurar que todas las inconsistencias que surjan de las relaciones de metamodelado en una ontología sean detectadas.

La lógica $\mathcal{SHIQM}^*$ agrega dos nuevos axiomas a la lógica $\mathcal{SHIQ}$ para representar metamodelado:

- $a =_{\mathsf{m}} A$, donde a es un individuo, A es un concepto atómico, y a representa el mismo objeto de la realidad que A,
- $\mathsf{MetaRule}(R, S)$, donde R y S son roles atómicos, y su significado es que por cada axioma $a =_{\mathsf{m}} A$, se cumple que $A \sqsubseteq \forall S.(\sqcup X)$, siendo X el conjunto de todos los conceptos B tales que $R(a, b)$ y $b =_{\mathsf{m}} B$.

En el escenario de contabilidad mencionado anteriormente, consideremos el conjunto de los asientos con sus detalles al debe y al haber, de acuerdo la forma de registro de contabilidad clásico. Por ejemplo, desde la perspectiva del experto que define los asientos a realizar, para representar que el asiento de pago de inquilinos puede tener al debe detalles de caja o banco, se declaran los siguientes axiomas de Abox:

$defDetDebe(asientoPagoInquilino, detPagoInqCaja)$
$defDetDebe(asientoPagoInquilino, detPagoInqBanco)$

siendo *asientoPagoInquilino*, *detPagoInqCaja* y *detPagoInqBanco* individuos que

representan el tipo de asiento de pago de inquilinos, y los detalles de asientos asociados a cuentas o rubros de caja y de banco, respectivamente. Esta definición del experto es una regla relacionada con los detalles del asientos al debe, que debe cumplirse cada vez que se registre un asiento de pago de alquiler para un inquilino. De acuerdo a esta regla, los detalles al debe del asiento pueden estar asociados únicamente a cuentas de caja o de banco.

Desde la perspectiva del operador que registra cada asiento particular, existirá un conjunto de asientos de pago de inquilinos, donde cada uno de ellos tendrá detalles al debe que únicamente puede asociarse a cuentas de caja o de banco, de acuerdo a la definición del experto. Por lo tanto, la granularidad adecuada para representar asientos de pago de inquilinos y a sus detalles consiste en declarar como conceptos *AsientoPagoInquilino*, *DetPagoInqCaja* y *DetPagoInqBanco*, que representan conjuntos. Los nuevos constructores $=_{\mathsf{m}}$ y MetaRule permiten expresar la correspondencia entre individuos y conceptos, y entre las relaciones de individuos y las relaciones entre los conceptos, respectivamente, agregando los siguientes axiomas:

$asientoPagoInquilino =_{\mathsf{m}} AsientoPagoInquilino$
$detPagoInqCaja =_{\mathsf{m}} DetPagoInqCaja$
$detPagoInqBanco =_{\mathsf{m}} DetPagoInqBanco$

$\mathsf{MetaRule}(defDetDebe, detDebe)$

El constructor $=_{\mathsf{m}}$ permite declarar que los individuos *asientoPagoInquilino*, *detPagoInqCaja* y *detPagoInqBanco* son los mismos objetos de la realidad que los conceptos *AsientoPagoInquilino*, *DetPagoInqCaja* y *DetPagoInqBanco*. El constructor MetaRule permite asegurar que la definición de un tipo de asiento con sus detalles permitidos al debe (y al haber) se va a cumplir al momento de registrar los asientos concretos. Por ejemplo, cada uno de los asientos representados por el concepto *AsientoPagoInquilino* solo podrá estar vinculado al debe con los detalles que son instancias de los conceptos *DetPagoInqCaja* y *DetPagoInqBanco*, a través del rol *detDebe*.

Definition 4 (Ontología in $\mathcal{SHIQM}^*$)
Una ontología $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A}, \mathcal{M})$ en la lógica $\mathcal{SHIQ}^*$ *está compuesta por una Tbox* $\mathcal{T}$*, una Rbox* $\mathcal{R}$ *y una Abox* $\mathcal{A}$ *y una Mbox* $\mathcal{M}$*, tal que::*

- $(\mathcal{T}, \mathcal{R}, \mathcal{A})$ *es una ontología en la lógica* $\mathcal{SHIQ}$*,*
- $\mathcal{M}$ *es un conjunto de axiomas de metamodelado de la forma* $a =_{\mathsf{m}} A$ *y*

$\mathsf{MetaRule}(R, S)$, *donde* a *es un individuo,* A *es un concepto atómico, y* R, S *son roles atómicos.*

Intuitivamente, la introducción de axiomas de metamodelado $a =_{\mathsf{m}} A$ que igualan un individuo a un concepto, significa que el individuo a debe interpretarse como el conjunto de elementos que representa el concepto A. Recordando la definición 2 de interpretación, la función de interpretación asocia a cada individuo un elemento del dominio de interpretación Δ. Sin embargo, dado que en una ontología en $\mathcal{SHIQM}^*$ un individuo a que está en un axioma $a =_{\mathsf{m}} A$ se interpreta como un conjunto, el dominio Δ en una ontología con metamodelado va a estar compuesto por elementos atómicos, pero también por conjuntos, y conjuntos de conjuntos, con varios niveles de metamodelado. Por ejemplo, en la siguiente ontología:

$\mathcal{O} = \{C(b), B(a), a =_{\mathsf{m}} A, b =_{\mathsf{m}} B\}$

el individuo a se interpreta como el conjunto representado por el concepto A, y el individuo b se interpreta como el conjunto representado por el concepto B, que contiene al conjunto A.

Una ontología $\mathcal{O}$ en la lógica $\mathcal{SHIQM}$ es una ontología en $\mathcal{SHIQ}$ cuya Mbox contiene únicamente axiomas $a =_{\mathsf{m}} A$. Al agregar los axiomas $\mathsf{MetaRule}(R, S)$ se obtiene la lógica $\mathcal{SHIQM}^*$.

Definition 5 (Modelo de una ontología en $\mathcal{SHIQM}^*$)
Una interpretación $\mathcal{I}$ *es un* modelo de una ontología $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A}, \mathcal{M})$ en $\mathcal{SHIQM}^*$ *si se cumple lo siguiente:*

1. *el dominio de interpretación* Δ *es un subconjunto de algún conjunto* S_n *para* $n \in \mathbb{N}$ *natural, tal que* S_n *se define inductivamente a partir de un conjunto no vacío* S_0 *de objetos atómicos, como* $S_{n+1} = S_n \cup \mathcal{P}(S_n)$.

2. $\mathcal{I}$ *es un modelo de la ontología* $(\mathcal{T}, \mathcal{R}, \mathcal{A})$ *en* $\mathcal{SHIQ}$.

3. $a^{\mathcal{I}} = A^{\mathcal{I}}$ *se cumple para cada axioma* $a =_{\mathsf{m}} A$.

4. $A^{\mathcal{I}} \subseteq (\forall S.(\sqcup X))^{\mathcal{I}}$ *se cumple para cada axioma* $\mathsf{MetaRule}(R, S)$ *y cada axioma* $a =_{\mathsf{m}} A$, *donde* $X = \{B \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \text{ and } b =_{\mathsf{m}} B \in \mathcal{M}\}$.

Con respecto al chequeo de consistencia de las ontologías en la lógica $\mathcal{SHIQM}^*$,

el algoritmo de razonamiento para $\mathcal{SHIQ}$ se extiende con cinco reglas que consideran los axiomas de metamodelado, y se agrega también un chequeo de ciclos en los axiomas de tipo $a =_{\mathsf{m}} A$. Los pasos del algoritmo extendido son los mismos tres pasos que se describen en la sección 2.2, excepto que en el paso 2) también se aplican las nuevas reglas y en el paso 3), además de verificar que no existen contradicciones, se chequea que no existan ciclos. Las nuevas reglas para metamodelado son $\approx$-rule, $\not\approx$-rule, close-rule, Close-Meta-rule y MetaRule(R, S)-rule, las cuales se presentan a continuación, para una ontología $\mathcal{O} = (\mathcal{T}, \mathcal{R}, \mathcal{A}, \mathcal{M})$.

$\approx$-rule:
Sean $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$ que pertenecen a $\mathcal{M}$. Si $a \approx b$, y $A \sqcup \neg B, B \sqcup \neg A$ no pertenecen a $\mathcal{T}$, entonces se agrega $A \sqcup \neg B, B \sqcup \neg A$ a $\mathcal{T}$.

$\not\approx$-rule:
Sean $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$ que pertenecen a $\mathcal{M}$. Si $a \not\approx b$ y no existe ningún nodo raíz z tal que $(A \sqcap \neg B \sqcup B \sqcap \neg A) \in \mathcal{L}(z)$ entonces se crea un nodo raíz z y se asigna: $\mathcal{L}(z) = \{A \sqcap \neg B \sqcup B \sqcap \neg A\}$

close-rule:
Sean $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$ que pertenecen a $\mathcal{M}$ donde $a \approx x$, $b \approx y$, siendo x e y sus respectivos representantes de las clase de equivalencia de a y b. Si no se cumple que $x \approx y$ ni $x \not\approx y$, entonces se agrega o bien $x \approx y$ o $x \not\approx y$. En el caso $x \approx y$, también se realiza lo siguiente:
1. se agrega $\mathcal{L}(y)$ a $\mathcal{L}(x)$,
2. para todos los arcos dirigidos en $\mathcal{L}$ que van desde y a algún w, se crea un arco desde x a w si ya no existe, tal que $\mathcal{L}(x, w) = \emptyset$,
3. se agrega $\mathcal{L}(y, w)$ a $\mathcal{L}(x, w)$,
4. para todos los arcos dirigidos desde algún w a y, se crea un arco desde w a x si ya no existe, tal que $\mathcal{L}(w, x) = \emptyset$,
5. se agrega $\mathcal{L}(w, y)$ a $\mathcal{L}(w, x)$,
6. se asigna $\mathcal{L}(y) = \emptyset$ y se eliminan todos los arcos desde/a y.

Close-Meta-**rule**:
Sean $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$ and MetaRule(R, S) que pertenecen a $\mathcal{M}$, donde $a \approx x$, $b \approx y$, siendo x, y representantes de las clases de equivalencia de a y b. Si no se cumple que $R \in \mathcal{L}(x, y)$ ni $\sim R \in \mathcal{L}(x, y)$, entonces se agrega o bien R a $\mathcal{L}(x, y)$ o $\sim R$ a $\mathcal{L}(x, y)$.

MetaRule(R, S)-**rule**:
Sean $a =_{\mathsf{m}} A$ y MetaRule(R, S) que pertenecen a $\mathcal{M}$, y se asume la Close-Meta-rule no puede ser aplicada. Si $\neg A \sqcup \forall S.(\sqcup X)$ no pertenece a $\mathcal{T}$, entonces se agrega

$\neg A \sqcup \forall S.(\sqcup X)$ a $\mathfrak{T}$, siendo $X = \mathsf{Image}_{\mathcal{L}}(R, a)$ donde $\mathsf{Image}_{\mathcal{L}}(R, a) = \{B \mid P \in \mathcal{L}(x, y),\ b =_{\mathsf{m}} B,\ P \sqsubseteq^{*} R,\ a \approx x,\ b \approx y$, siendo x e y los representantes de las clases de equivalencia de a y $b\}$.

En las reglas para metamodelado anteriormente presentadas, las relaciones $\approx$ y $\not\approx$ se introducen para registrar las igualdades y desigualdades entre nodos del grafo, tanto para los nodos raíz (declarados en la ontología) como para los nodos creados por el algoritmo (ver descripción de la regla del $\exists$ en la Sección 2.2). Los arcos del grafo están etiquetados con roles R, al igual que para la lógica $\mathcal{SHIQ}$, y además con construcciones $\sim R$, los cuales indican que los nodos de inicio y fin no están conectados por el rol R.

La regla $\approx$-rule transfiere la igualdad $a \approx b$ a los conceptos corrsepondientes, cuando $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$. Para ello, se agregan dos axiomas de Tbox, que son equivalentes a $A \equiv B$.

The $\not\approx$-rule transfiere la desigualdad $a \not\approx b$ a los conceptos correspondientes, cuando $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$. Para ello, se crea un nuevo elemento z, y se asocia el concepto $A \sqcap \neg B \sqcup B \sqcap \neg A$ a z para asegurar que existe al menos un nodo que es instancia de uno de los conceptos A o B, pero no de ambos.

La regla close-rule agrega o bien $a \approx b$ o $a \not\approx b$ en caso que no exista ninguno de estos axiomas vinculando los individuos a y b, y se necesita por completitud.

La regla $\mathsf{MetaRule}(R, S)$-rule se crea para que cuando existen pares de individuos a, b tales que $R(a, b)$, esta relación entre individuos se traslade a la relación, dada por el rol S, entre las instancias de los conceptos A, B, tales que $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$. Para ello, esta regla cambia la Tbox agregando axiomas $A \sqsubseteq \forall S.C$ tal que el concept A está en un axioma $a =_{\mathsf{m}} A$, y donde C es una disjunción de conceptos con metamodelado relacionados con a via R.

La regla Close-Meta-rule crea arcos en forma no determinística etiquetados con R o $\sim R$. Dado que la regla $\mathsf{MetaRule}(\mathsf{R}, \mathsf{S})$-rule solo se aplica cuando la regla Close-Meta-rule ya no puede ser aplicada, esto garantiza que el concepto C en $A \sqsubseteq \forall S.C$ es siempre el mismo.

Finalmente, el chequeo de ciclos se agrega para detectar situaciones como la que se muestra a continuación.

$\mathcal{O} = \{C(b), B(a), A(c), a =_{\mathsf{m}} A, b =_{\mathsf{m}} B, c =_{\mathsf{m}} C\}$

Se observa que el concepto C tiene una instancia que corresponde al concepto B (por metamodelado), y B tiene una instancia que corresponde al concepto A, que tiene una instancia que corresponde al concepto C, por lo que C pertenece a sí mismo. En el grafo que construye el algoritmo de Tableau, el algoritmo extendido para $\mathcal{SHIQM}^*$ verifica que no exista un ciclo, de acuerdo a la definición que se presenta a continuación.

Definition 6 (Ciclo)
El grafo $\mathcal{L}$ tiene un ciclo con respecto a una Mbox $\mathcal{M}$ si existe una secuencia de axiomas de metamodelado $A_0 =_{\mathsf{m}} a_0$, $A_1 =_{\mathsf{m}} a_1$, ... $A_n =_{\mathsf{m}} a_n$ all in $\mathcal{M}$ tal que

$$\begin{array}{ll} A_1 \in \mathcal{L}(x_0) & x_0 \approx a_0 \\ A_2 \in \mathcal{L}(x_1) & x_1 \approx a_1 \\ \vdots & \vdots \\ A_n \in \mathcal{L}(x_{n-1}) & x_{n-1} \approx a_{n-1} \\ A_0 \in \mathcal{L}(x_n) & x_n \approx a_n \end{array}$$

La Figura 2.4 ilustra un ejemplo de un grafo conteniendo un ciclo, en el que se observa que el conjunto representado por el concepto A (e individuo a) pertenece a C, que pertenece a B, que pertenece a A.

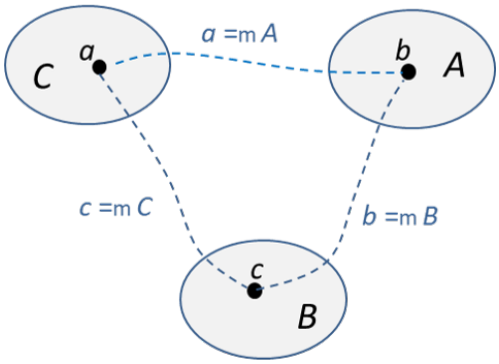

Figura 2.4: Ejemplo de grafo con un ciclo

3. Requerimientos

En este capítulo se describen los requerimientos funcionales y no funcionales de este proyecto, tanto para la implementación de la extensión del razonador Hermit, que es la parte central de trabajo, como para el desarrollo del prototipo del caso de uso de contabilidad.

Antes de describir los requerimientos funcionales y no funcionales se establece el alcance del proyecto y se describe brevemente el proceso de desarrollo seguido para la obtención de un producto que resuelve los requerimientos planteados.

3.1. Propósito y alcance

El propósito principal del proyecto es extender el razonador Hermit agregándole la capacidad de evaluar la consistencia de ontologías con metamodelado. El alcance de esta extensión consiste en agregar cinco nuevas reglas para el tratamiento de los constructores de metamodelado de la lógica $\mathcal{SHIQM}^*$ y la verificación de la no existencia de ciclos en el grafo como condición adicional para concluir que una ontología en la lógica $\mathcal{SHIQM}^*$ es consistente. [5].

Como beneficio de este desarrollo se identifica la posibilidad de evaluar modelos más complejos y más precisos de la realidad. El caso de uso de contabilidad de este proyecto es un claro ejemplo de un caso de uso con requerimientos relevantes que pueden ser expresados explícitamente con la extensión desarrollada. Mostrar estas ventajas para una aplicación de contabilidad también forma parte del alcance de este proyecto.

3.2. Proceso de desarrollo

La Figura 3.1 muestra el proceso de desarrollo seguido en este proyecto.

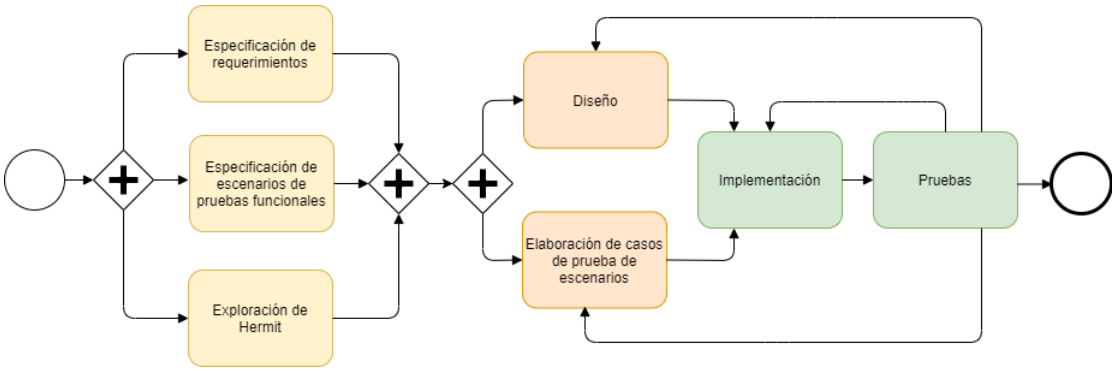

Figura 3.1: Proceso de desarrollo del proyecto

En el proceso se distinguen claramente dos fases, con un enfoque de desarrollo basado en pruebas. Como se observa en la figura 3.1, hay una primera fase de exploración de Hermit, definición de los requerimientos y principales escenarios funcionales, y una segunda fase de diseño, implementación y pruebas.

En la primera fase, a medida que se fueron definiendo los requerimientos funcionales, se elaboraron los escenarios de prueba relevantes que permitieron asegurar un buen cubrimiento de la casuística. Además se establecieron requerimientos no funcionales y se analizó en detalle la arquitectura e implementación de Hermit, como preparación para la siguiente fase. En la segunda fase, se procedió al diseño de la extensión y paralelamente a la elaboración de un conjunto de casos de prueba concretos para cada escenario identificado en la fase anterior. A partir del diseño, se ejecutaron varias iteraciones de implementación y pruebas por escenario y por cada nueva regla. En cada iteración se ajustaron también funcionalidades transversales a todas las reglas, como el backtracking. Una vez finalizada la implementación y prueba de la mayor parte de los requerimientos funcionales, se ejecutaron algunas iteraciones para la implementación de optimizaciones, lo que permitió resolver el requerimiento de desempeño que se describe en este capítulo. A su vez, en algunas de estas iteraciones fue necesario ajustar el diseño de la extensión, a partir de los resultados obtenidos en las pruebas.

3.3. Requerimientos funcionales

A continuación se presentan los requerimientos funcionales del proyecto a través de la especificación de un conjunto de escenarios de verificación de la consistencia de una ontologías que incluyen algún subconjunto de los constructores de la lógica $\mathcal{SHIQM}^*$. Adicionalmente se incluye una descripción general del caso de uso de contabilidad, junto con los requerimientos que una ontología en la lógica $\mathcal{SHIQM}^*$ permite resolver.

Ontología $\mathcal{SHIQ}$ consistente. Una ontología en la lógica $\mathcal{SHIQ}$ (sin axiomas de metamodelado) que es consistente al ejecutar el razonador Hermit, debe resultar consistente al ejecutar Hermit extendido para $\mathcal{SHIQM}^*$, y debe inferir los mismos axiomas.

Ontología $\mathcal{SHIQ}$ inconsistente. Una ontología en la lógica $\mathcal{SHIQ}$ que es inconsistente al ejecutar el razonador Hermit, debe resultar inconsistente al ejecutar Hermit extendido para $\mathcal{SHIQM}^*$.

Ontología $\mathcal{SHIQM}$ consistente. Una ontología en la lógica $\mathcal{SHIQM}$ (con axiomas $a =_{\mathsf{m}} A$) que es consistente de acuerdo a la semántica definida para $\mathcal{SHIQM}$, debe resultar consistente e inferir los axiomas correspondientes, al ejecutar Hermit extendido para $\mathcal{SHIQM}^*$.

Ontología $\mathcal{SHIQM}$ inconsistente. Una ontología en la lógica $\mathcal{SHIQM}$ que es inconsistente de acuerdo a la semántica definida para $\mathcal{SHIQM}$, debe resultar inconsistente al ejecutar Hermit extendido para $\mathcal{SHIQM}^*$.

Ontología $\mathcal{SHIQM}^*$ consistente. Una ontología en la lógica $\mathcal{SHIQM}^*$ (con axiomas $a =_{\mathsf{m}} A$ y $\mathsf{MetaRule}(R, S)$) que es consistente de acuerdo a la semántica definida para $\mathcal{SHIQM}^*$, debe resultar consistente e inferir los axiomas correspondientes, al ejecutar Hermit extendido para $\mathcal{SHIQM}^*$.

Ontología $\mathcal{SHIQM}^*$ inconsistente. Una ontología en la lógica $\mathcal{SHIQM}^*$ que es inconsistente de acuerdo a la semántica definida para $\mathcal{SHIQM}^*$, debe resultar inconsistente al ejecutar Hermit extendido para definida para $\mathcal{SHIQM}^*$.

Ontología $\mathcal{SHIQM}^*$ consistente con solo MetaRule. Una ontología en la lógica $\mathcal{SHIQM}^*$ que contiene axiomas $\mathsf{MetaRule}(R, S)$ y no contiene axiomas $a =_{\mathsf{m}} A$, tal que la ontología que resulta de excluir los axiomas $\mathsf{MetaRule}(R, S)$ es consistente al ejecutar el razonador Hermit, debe resultar consistente al ejecutar Hermit exten-

dido para $\mathcal{SHIQM}^*$. Además, debe inferir únicamente los axiomas que se deducen de la ontología en $\mathcal{SHIQ}$ que resulta de excluir los axiomas $\mathsf{MetaRule}(R, S)$.

Ontología $\mathcal{SHIQM}^*$ inconsistente con solo MetaRule. Una ontología en la lógica $\mathcal{SHIQM}^*$ que contiene axiomas $\mathsf{MetaRule}(R, S)$ y no contiene axiomas $a =_{\mathsf{m}} A$, tal que la ontología que resulta de excluir los axiomas $\mathsf{MetaRule}(R, S)$ es inconsistente al ejecutar el razonador Hermit, debe resultar inconsistente al ejecutar Hermit extendido para $\mathcal{SHIQM}^*$.

Caso de uso de contabilidad.

En el escenario de contabilidad planteado, se identifican dos tipos de usuarios: *usuarios expertos* en contabilidad que definen qué tipo de asiento (con sus cuentas al debe y haber) se debe registrar para cada movimiento financiero, y los *usuarios operadores* que registran asientos concretos (por ejemplo un asiento de pago de un inquilino en particular) que deben respetar las deficiones realizadas por los usuarios expertos. En esta aplicación, así como en todas las aplicaciones en las que existen determinados perfiles de usuarios a cargo de definir reglas que refieren a las tareas realizadas por perfiles más operativos, sucede que los mismos objetos de la realidad, por ejemplo, asientos, se visualizan con diferente granularidad por usuarios expertos y por usuarios operadores. Por ejemplo, para un usuario experto el "asiento de pago de inquilinos" es un tipo de asiento que se percibe como una instancia o entidad atómica, mientras que un usuario operador visualiza uno a uno los asientos que se registran, o sea que percibe la misma entidad como un conjunto de instancias.

En el escenario planteado, un *requerimiento funcional* relevante principalmente para los usuarios expertos, es que la aplicación permita visualizar en forma explícita *restricciones y reglas que se aplican a las definiciones* de los diferentes tipos de asientos, que son muy importantes ya que evitan implementar controles sobre los datos de los asientos concretos, los cuales generalmente quedan escondidos en el código de la aplicación. Existe también un requerimiento no funcional muy importante, que consiste en poder *representar los diferentes niveles de abstracción* con que los diferentes perfiles visualizan los objetos de la realidad, y poder unificar esas visiones haciendo explícita la correspondencia entre la instancia, por ejemplo, "el asiento de pago de inquilinos" que visualiza el experto, y el conjunto de instancias, "todos los asientos de pagos de inquilinos" que visualiza el operador.

El prototipo que se desarrolla en este proyecto debe proveer una interfaz para que usuarios expertos ingresen definiciones de asientos y operadores ingresen asien-

tos concretos que cumplen con dichas definiciones. La aplicación debe interpretar los datos ingresados y poblar con ellos una ontología, la cual es evaluada por el razonador Hermit extendido para la lógica $\mathcal{SHIQM}^*$.

3.4. Requerimientos no funcionales

A continuación se presentan los requerimientos no funcionales del proyecto, básicamente relacionados a estándares de calidad del código, modularidad y desempeño.

Modularidad y estándares de calidad del código. El código de la extensión debe ser modular, cumplir con los principios básicos del diseño orientado a objetos como alta cohesión, bajo acoplamiento y mantener el criterio de separación de responsabilidades adoptado en la arquitectura de Hermit.

Cumplir con la especificación del algoritmo para $\mathcal{SHIQM}^*$. La implementación de la extensión se basa en el enfoque de metamodelado descripto en la Sección 2.3. Este enfoque propone una extensión modular del algoritmo de Tableau a través de la incorporación de cinco nuevas reglas. El código de la extensión debe cumplir con la especificación de estas reglas, así como con la detección de ciclos y la definición de ontología consistente que propone el enfoque [5].

Desempeño. El tiempo de ejecución del razonador Hermit extendido para la lógica $\mathcal{SHIQM}^*$, desde que es invocado a través de la interfaz de usuario hasta que devuelve el resultado, no debe superar los 20 segundos, para ontologías de pequeño porte, como la ontología utilizada en el prototipo de contabilidad.

Dado que el atributo de desempeño no es el foco de este proyecto, no se establece un umbral de tiempo demasiado exigente. Este requerimiento se impone en el proyecto con el objetivo de asegurar que esta primera versión de la extensión de Hermit para $\mathcal{SHIQM}^*$ se ejecute en un tiempo razonable. Queda para otro proyecto la optimización de esta extensión para que pueda ser aplicada a ontologías de mayor porte.

4. Diseño

En este capítulo se presenta el diseño de la extensión del razonador Hermit, que es una extensión modular, ya que se agregan las cinco nuevas reglas de derivación para metamodelado descriptas en la Sección 2.3 respetando los criterios y patrones de diseño aplicados en la implementación original. Para completar el desarrollo de esta extensión, se debe extender la funcionalidad de Hermit y la OWL API. En la Figura 4.1 se ve cómo Hermit interactúa con la OWL API para el chequeo de consistencia de una ontología. Hermit utiliza la API para interpretar la ontología y así construir su propia representación de la misma como se describe en el algoritmo de hypertableau.

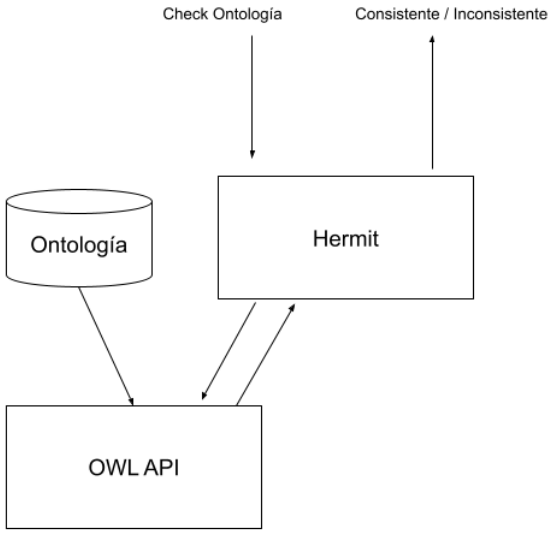

Figura 4.1: Interaccion Usuario-Hermit-OWl API

También en una tercera parte de esta sección se describe el diseño de la solución de la interfaz de usuario de contabilidad.

4.1. OWL API con metamodelado

La OWL API es una aplicación implementada en Java que brinda una interfaz para manipular ontologías OWL, permitiendo a los desarrolladores trabajar con las mismas con un nivel apropiado de abstracción [11]. Permite separar algunos problemas como el parseo y la serialización de estructuras de datos, que resuelve la OWL API, de la verificación de la consistencia e inferencia de conocimiento, que es el foco de los razonadores como Hermit. La OWL API se distribuye bajo las licencias LGPL y Apache.

Las principales funcionalidades que provee la OWL API se describen brevemente a continuación [12].

- Serialización: traduce la ontología a alguna sintaxis como OWL RDF, a partir de estructuras o representaciones internas [13].
- Modelado: Provee de estructuras de datos que representan los elementos de una ontología.
- Parseo: A partir de una ontología serializada en OWL RDF por ejemplo, obtiene una representación interna de la misma.
- Manipulación: Provee una representación y funcionalidades para la manipulación de ontologías.
- Inferencias: Provee una representación para implementar la semántica formal del lenguaje OWL.

Dado que la OWL API es usada por razonadores para la lectura y manipulación de ontologías OWL, para que el razonador Hermit extendido pueda chequear consistencia y realizar inferencias en ontologías con metamodelado, se debe extender también a la OWL API para que esta pueda interpretar los nuevos contructores $=_{\mathsf{m}}$ y $\mathsf{MetaRule}$ introducidos en la Sección 2.3.

En este proyecto se extiende la versión de OWL API 4.5.7 utilizada por la versión de Hermit 1.4.3.456, que es la versión del razonador usada por el editor de ontologías Protégé.

En la Figura 4.2 se puede ver cómo la OWL API modela una ontología como un conjunto de axiomas y anotaciones, es por eso que se dice que proporciona una

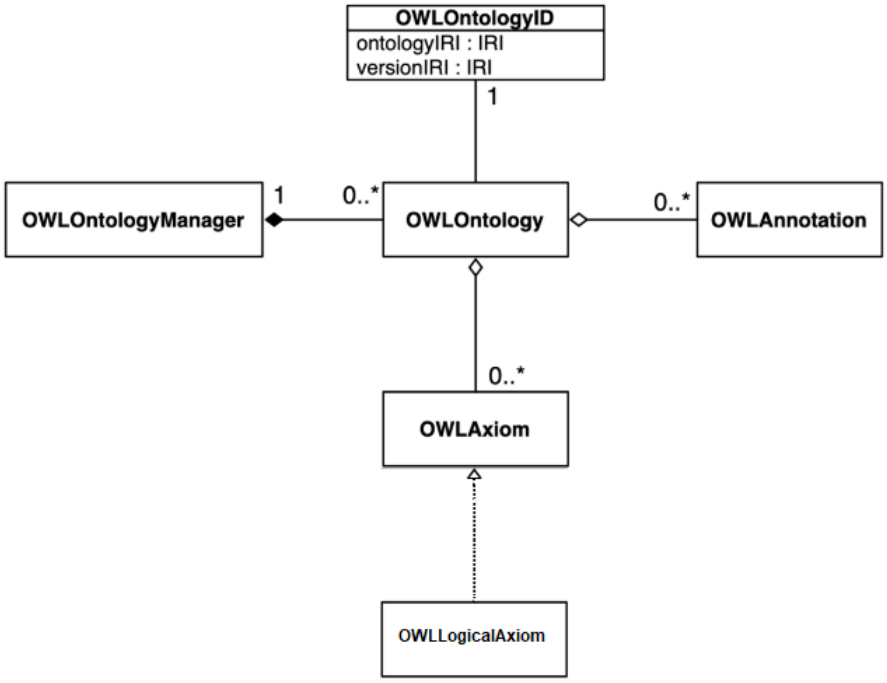

Figura 4.2: Diagrama UML que muestra el manejo de ontologías en la OWL API

vista de alto nivel centrado en axiomas [14].

Las modificaciones realizadas a la OWL API en este proyecto deben proveer a la OWL API de la capacidad de leer e interpretar los dos nuevos tipos de axiomas para metamodelado: axiomas $a =_{\mathsf{m}} A$ y axiomas $\mathsf{MetaRule}(R, S)$. Para la introducción de estos axiomas, y teniendo en cuenta lo representado en el diagrama de la Figura 4.2, se deduce que el componente OWLLogicalAxiom, que representa a una interfaz de aquellos axiomas que no son ni de definición ni de anotaciones, debe ser extendido con los nuevos tipos de axiomas. Por lo tanto, se plantean dos implementaciones de dicha interfaz, una para cada tipo de axioma, y en consecuencia se realizan los cambios que la propia arquitectura de la OWL API requiera. Por ejemplo, se modifican los componentes que usan el patrón visitor y necesitan definir un comportamiento para los nuevos tipos de axiomas agregados. En la Figura 4.3 se presentan las dos nuevas interfaces para los axiomas OWLMetamodellingAxiom y OWLMetaRuleAxiom.

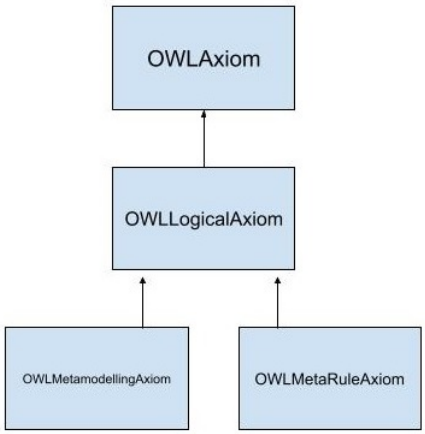

Figura 4.3: Diseño de extensión de OWL API

4.2. Extensión de Hermit

Hermit es un razonador de ontologías en OWL 2. Es una aplicación Java que cuenta con dos interfaces para poder interactuar con ella: OWLReasoner y CommandLine, como se ilustra en la Figura 4.4. La interfaz OWLReasoner permite su integración con otras aplicaciones como Protégé, mientras que CommandLine es una interfaz de usuario en consola para ejecutar comandos simples del razonador sobre ontologías. Ambas Interfaces acceden a servicios expuestos por la clase Reasoner. Este componente funciona de fachada desacoplando al cliente del sistema complejo que se oculta en Hermit, haciendo que el mismo pueda ser más reutilizable y portable.

Como se ve en la Figura 4.4 se destacan cuatro conjuntos de componentes que exponen servicios a la clase Reasoner, relacionados cada uno a un servicio especifico. Estos conjuntos son Loading, Classification, Realisation y Reasoning. El conjunto de componentes *Loading* es el encargado de leer la ontología a ser procesada para normalizarla y crear el conjunto de DL-Clauses. El conjunto de componentes de *Reasoning* toma esta transformación de la ontología para determinar si la misma es consistente o no. En estos dos conjuntos de componentes se identifican cambios a realizar al razonador Hermit para que sea capaz de verificar la consistencia de ontologías con metamodelado.

Los cambios en los componentes de *Loading* o carga de ontología son sencillos, ya que utilizando la versión de la OWL API que soporta metamodelado, descripta

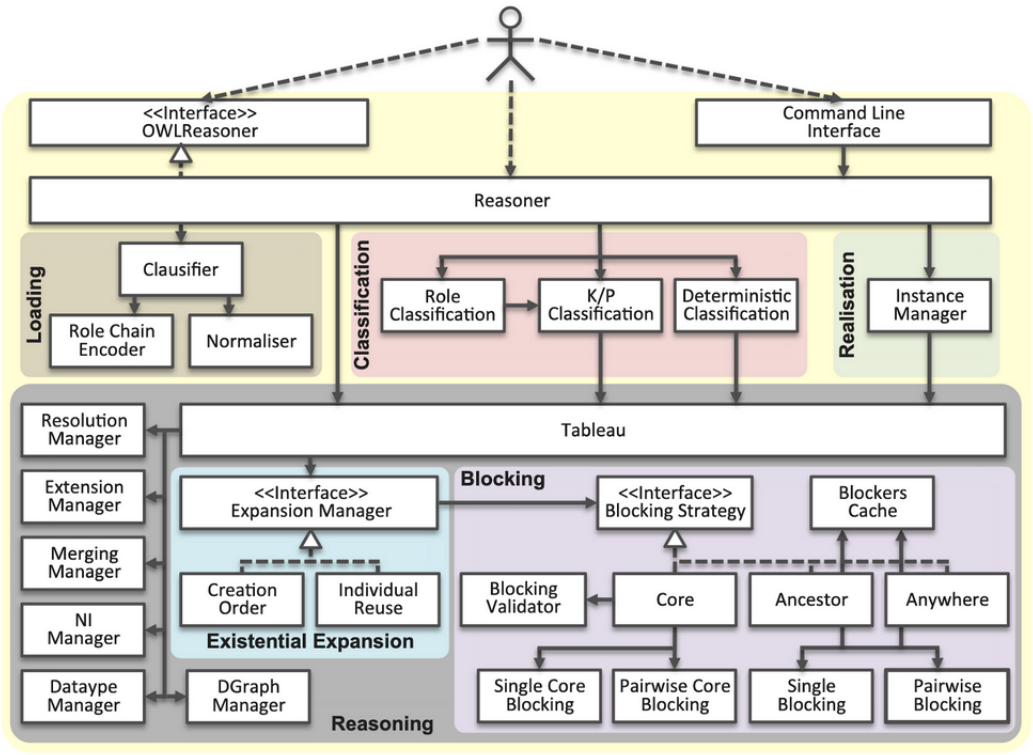

Figura 4.4: Arquitectura de Hermit [15]

en la sección anterior, la mayor parte del trabajo estaría realizado. Además de las modificaciones en la OWL API, desde Hermit se deben agregar algunos comportamientos en algunas implementaciones del patrón visitor para los nuevos tipos de axiomas, es decir, OWLMetamodellingAxiom y OWLMetaRuleAxiom.

En la Figura 4.5 se muestran los componentes que exponen servicios a la clase Tableau que pertenece al conjunto de componentes *Reasoning*, y se representan con borde más grueso los nuevos componentes que se implementaron para extender las funciones de razonamiento para ontologías con metamodelado. Estos nuevos componentes son *MetamodellingManager*, *BranchedHyperresolutionManager* y *MetamodellingAxiomHelper*. Como se puede ver en la Figura 4.4, dentro del conjunto de componentes de Reasoning se destaca el componente Tableau, el cual es el encargado de manejar la ejecución del algoritmo de hypertableau haciendo uso de distintos datos y servicios de los componentes Managers. En función a esto es que se agregaron el MetamodellingManager, con funciones relacionadas a la aplicación de reglas de metamodelling y control de ciclos y el BranchedHyperresolutionManager, el cual cumple funciones relacionadas al backtracking dentro de la extensión de

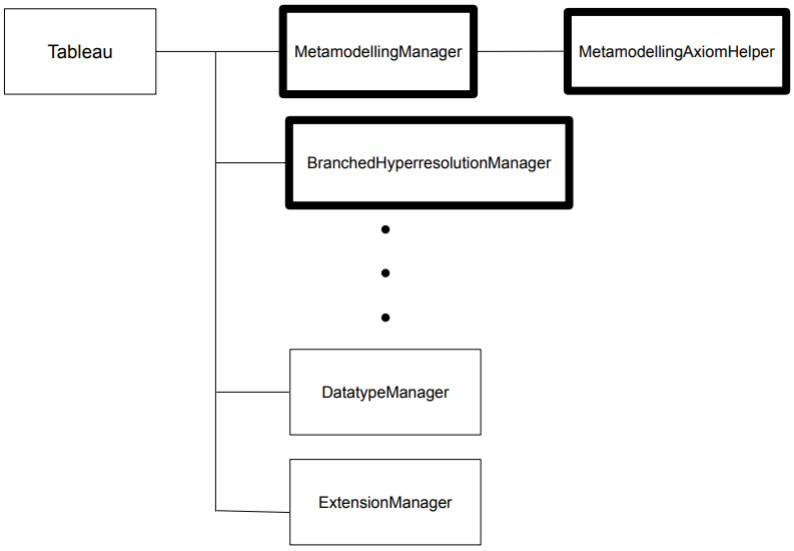

Figura 4.5: Tableau y componentes asociados para metamodelado (borde grueso).

Hermit. Para reducir la cantidad de código y abstraer al MetamodellingManager del nivel de manejo de axiomas y DL-clauses se crea el MetamodellingAxiomHelper, el cual asiste al MetamodellingManager en tareas relacionadas a la creación de axiomas y DL-clauses, y en la búsqueda de los mismos dentro de una ontología.

4.3. Prototipo de contabilidad

Con el objetivo de mostrar la utilidad de la extensión de Hermit en un caso real, se consideran los requerimientos del caso de uso de contabilidad descriptos en la Sección 3.3. Se diseña un prototipo que permite la definición de diferentes tipos de asientos por parte del usuario experto, y el ingreso de asientos concretos por parte del usuario operador, los cuales deben respetar las definiciones del usuario experto. La capa de presentación de esta aplicación resuelve la interfaz para el usuario experto y la interfaz para el usuario operador en una única pantalla[1]. Esta pantalla consta de dos paneles, un panel superior para la definición de asientos con las cuentas válidas al debe y al haber, y un panel inferior para el ingreso de asientos con datos a qué tipo de asiento corresponde (definido por el experto), la fecha, y las cuentas e importes al debe y al haber. La capa de aplicación se encarga

[1]En la aplicación real, estos usuarios con diferentes perfiles tiene diferentes permisos y obviamente acceden a pantallas separadas.

de interpretar los datos ingresados por el usuario (experto u operador dependiendo del panel en el cual son ingresados) y de agregar axiomas a la ontología que se muestra en la Figura 4.6.

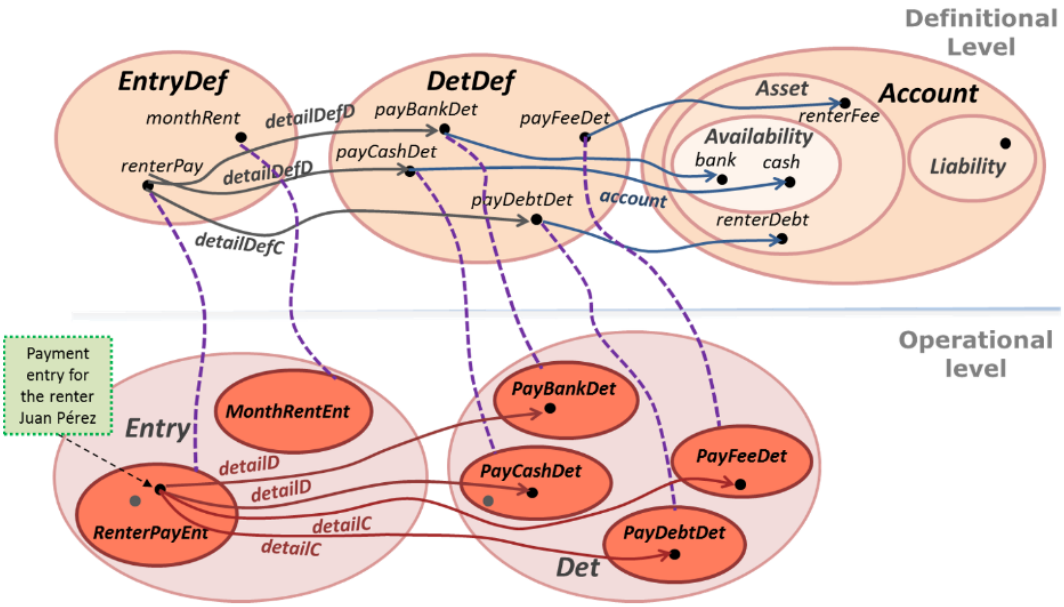

Figura 4.6: Ontología de contabilidad

Los conceptos *EntryDef*, *DetDef*, *Account*, *Entry* y *Det*, así como los roles *detailDefD*, *detailDefC*, *account*, *detailD* y *detailC*, forman parte de la estructura básica de la ontología, es decir, ya existen al momento del ingreso de datos por parte del experto y del operador. Las subclases del concepto *Account* y sus instancias (las cuentas del plan de cuentas) también han sido previamente cargadas en la ontología. Con los datos ingresados por el usuario experto, la aplicación crea las instancias de los conceptos *EntryDef* y *DetDef*, es decir las diferentes definiciones de asientos y sus correspondientes detalles, y además las subclases de los conceptos *Entry* y *Def*, que representan los conjuntos de asientos que corresponden a estas definiciones, es decir, las definiciones se vinculan a las subclases de Entry y Def por la relación de metamodelado. Con los datos ingresados por el usuario operador, la aplicación puebla las subclases de *Entry* y *Def*, es decir los asientos concretos clasificados de acuerdo al tipo de asiento definido por el usuario experto.

La Figura 4.7 muestra la arquitectura de la aplicación. Para la interfaz gráfica se propone la utilización de Java Swing debido a la facilidad de uso e integración con Hermit y la OWL API. El componente controlador recibe los datos ingresados por

el usuario desde la interfaz y los interpreta de manera que utilizando los servicios de la OWL API con la extensión de metamodelado, extiende la ontología de la Figura 4.6. Una vez actualizada la ontología, el controlador invoca los servicios de razonamiento de Hermit con la extensión de metamodelado para determinar si la ontología es consistente, y comunica el resultado a la interfaz. La consistencia de la ontología asegura que se cumplan restricciones en las definiciones, en los asientos, y que estos últimos respetan las definiciones.

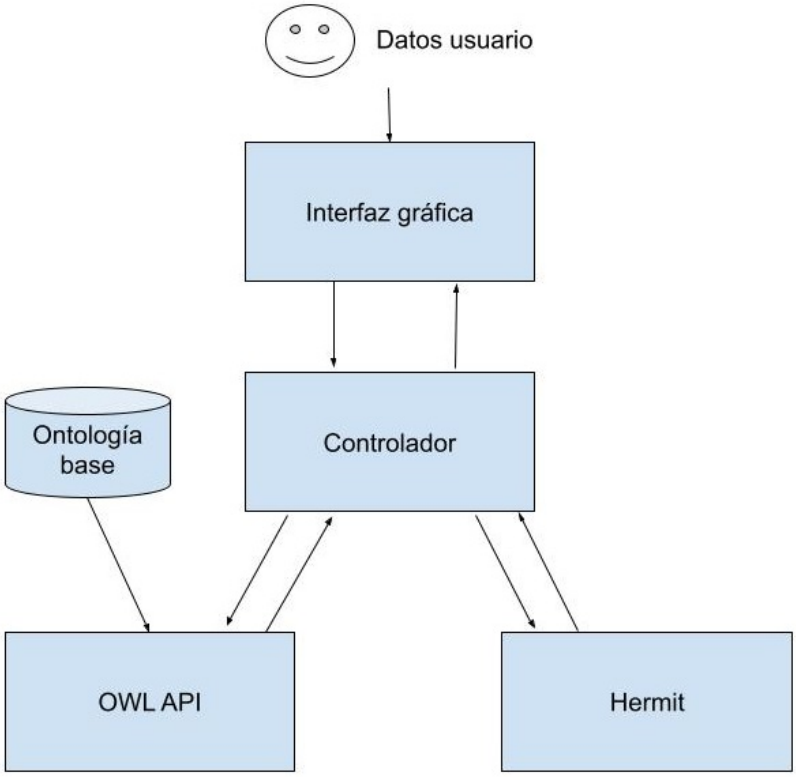

Figura 4.7: Arquitectura del prototipo de contabilidad

5. Implementación

En este capítulo se describen los cambios que se realizaron a las implementaciones de la OWL API, y de los componentes de Loading y Reasoning de Hermit, que se describen en las Secciones 4.1 y 4.2.

5.1. OWL API con metamodelado

Como se menciona en la Sección 4.1 sobre el diseño, para extender la OWL API y proporcionarle la capacidad de manipular ontologías con metamodelado se extendió la interfaz OWLLogicalAxiom, la cual es extensión de la interfaz OWLAxiom. Un ejemplo de extensión de la interfaz OWLLogicalAxiom es OWLPropertyAssertionAxiom que representa un axioma de la forma $R(a, b)$ con R siendo un rol, y a y b dos individuos. Para este proyecto se implementaron dos nuevas extensiones, *OWLMetamodellingAxiom* y *OWLMetaRuleAxiom* como se muestra en la Figura 5.1.

La Figura 5.2 muestra la clase *OWLMetamodellingAxiom* que representa al axioma de metamodelado $a =_{\mathsf{m}} A$, que expresa la correspondencia entre un individuo a y un concepto A, los cuales se refieren a un mismo objeto de la realidad y lo modelan de distinta manera. Para la implementación de este axioma se extiende la interfaz OWLLogicalAxiom y se agregan 2 atributos, un individuo y una clase.

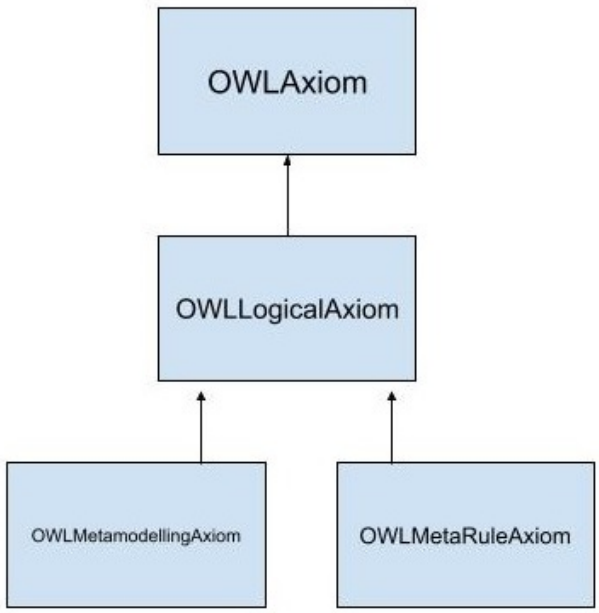

Figura 5.1: Diseño de extensión de OWL API

```java
public interface OWLMetamodellingAxiom extends OWLLogicalAxiom {

    /**
     * Gets the ModelClass in this axiom
     * @return The class expression that represents the Model in this axiom.
     */
    OWLClassExpression getModelClass();

    /**
     * Gets the MetamodelIndividual in this axiom.
     * @return The individual that represents the Metamodel in this axiom.
     */
    OWLIndividual getMetamodelIndividual();
```

Figura 5.2: OWLMetamodellingAxiom.java

La Figura 5.3 muestra la clase *OWLMetaRuleAxiom* que representa al axioma de metamodelado $\mathsf{MetaRule}(R, S)$ que expresa la correspondencia entre dos roles

R y S, tal que si un par de individuos a, b con metamodelado (axiomas $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$) están en la relación R, entonces los conceptos correspondientes A, B deben estar en la relación S. Para su implementación, además de extender la interfaz OWLLogicalAxiom se agregan 2 atributos, uno para el rol R y otro para el rol S.

```java
public interface OWLMetaRuleAxiom extends OWLLogicalAxiom {

    /**
     * Gets the property R in this axiom
     * @return The property for the individuals from the metamodelling axiom
     */
    OWLObjectPropertyExpression getPropertyR();

    /**
     * Gets the property S in this axiom
     * @return The property for the elements of the classes from the metamodelling axiom
     */
    OWLObjectPropertyExpression getPropertyS();
```

Figura 5.3: OWLMetaRuleAxiom.java

Tanto OWLMetamodellingAxiom como OWLMetaRuleAxiom son interfaces, por lo tanto se debieron agregar a las modificaciones sus respectivas implementaciones.

La OWL API hace un extenso uso del patrón de diseño visitor, el cual ayuda a la separación entre funcionalidad y las estructuras de datos. El patrón facilita agregar funcionalidades, pero cuenta con la desventaja de que los cambios en las estructuras o agregar nuevas estructuras, como en el caso de esta implementación, implican cambios en cada una de las implementaciones del visitor. Esto hace que una gran cantidad de archivos deban ser modificados por agregar estas nuevas estructuras que representan a los axiomas de metamodelado.

Para representar al concepto de MBox, que engloba al conjunto de axiomas $a =_{\mathsf{m}} A$ y $\mathsf{MetaRule}(R, S)$, se debe agregar los nuevos tipos de axiomas como un atributo, a la clase AxiomType, como se muestra en la Figura 5.4. También se agrega el MBox como atributo de la clase OWLOntology, estructura que representa una ontología y que previamente ya tenía atributos ABox, TBox y RBox.

Por ultimo, se modificaron las clase que tienen la responsabilidad del manejo de las diferentes sintaxis en las que se escribe una ontología, como RDF o XML. En cada clase se debió especificar cómo es la nomenclatura de un axioma de *Metamo-*

```
@Nonnull public static final Set<AxiomType<?>> MBoxAxiomTypes = CollectionFactory.createSet(
        (AxiomType<?>) METAMODELLING, METARULE);
```

Figura 5.4: AxiomType.java

delling ($a =_{\mathsf{m}} A$) o*MetaRule* ($\mathsf{MetaRule}(R, S)$) en cada lenguaje y cómo se deben comportar los distintos handlers al leer los axiomas, esto es, cómo la API detecta un axioma de este tipo al leer la ontología y cómo debe construir el axioma.

```
/** METAMODELLING                    */  METAMODELLING                    ("MetaModelling"),
/** METARULE                         */  METARULE                         ("MetaRule");;
```

Figura 5.5: OWLXMLVocabulary.java

Por ejemplo, un axioma de Metamodelling en un documento de OWL en XML se debe especificar con el tag $< MetaModelling >$, como se presenta en la figura 5.6.

```
<MetaModelling>
    <NamedIndividual IRI="#a"/>
    <Class IRI="#A"/>
</MetaModelling>
```

Figura 5.6: Especificación de un axioma de Metamodelling

El detalle de todos los archivos modificados y agregados se encuentran en el apéndice de este documento.

5.2. Extensión de Hermit

Los cambios involucrados en la extensión de Hermit se separan en dos categorías: cambios a los componentes de Loading o carga de la ontología y cambios a los componentes de Reasoning. Los componentes de carga de la ontología, que interoperan con la OWL API, se encargan de crear la estructura de la ontología basada en DL-Clauses, la cual es manipulada por Hermit. Los componentes de Loading, que se encargan de las tareas de razonamiento, implementan la reglas del algoritmo de Tableau, adaptadas para mejorar su desempeño con el cálculo Hypertableau, como se menciona en la Sección . Por lo tanto, en esta extensión se debe incluir la implementación de las cinco nuevas reglas para metamodelado descriptas en la Sección 2.3. Además, se realizan cambios para adaptar algunas funcionalidades ya existentes a ontologías cn metamodelado.

5.2.1. Cambios en los componentes de Loading

En lo que refiere a la carga de ontologías, la mayor parte de los cambios están dentro de la OWL API, descriptos en la Sección 5.1, por lo que el trabajo a realizar en el conjunto de componentes de Loading de Hermit es menor al de los componentes de Reasoning. Los cambios a los componentes de Loading se reducen a pequeños cambios en la clase OWLNormalization.

La clase OWLNormalization es utilizada para realizar una transformación estructural de la ontología, en la cual primero se simplifican los axiomas y luego se pasan a su forma normal de negación (FNN), en la que deben estar los axiomas de la ontología para ser procesados por los componentes de Reasoning. Dado que esta transformación a FNN solo se aplica a axiomas de TBOX, las implementaciones agregadas en esta extensión para los axiomas de metamodelado devuelven el axioma tal cual está, no ejecutándose transformación alguna.

Los cambios realizados a la clase OWLNormalization implican extender el uso del patrón visitor para los nuevos tipos de axiomas agregados en la OWL API, OWLMetamodellingAxiom y OWLMetaRuleAxiom. Esto se debe a que ambas clases implementan la interfaz de OWLAxiomVisitorEx de la OWLAPI, la cual tiene dos métodos visit para los nuevos tipos de axiomas introducidos en la extensión, que se deben implementar.

En otras palabras, los cambios a los componentes de Loading se realizan estrictamente con el fin de respetar los patrones de diseño establecidos por Hermit y por la OWL API, más específicamente el patrón visitor. Al no requerirse una normalización de los axiomas de metamodelado, no se realizan modificaciones para estos nuevos axiomas.

5.2.2. Cambios en los componentes de Reasoning

Los cambios en los componentes de Reasoning son los de mayor complejidad y conforman la principal dificultad del proyecto. A partir del diseño de la extensión, que se describe en la Sección 4.2, la fase de implementación consistió de dos actividades bien diferenciadas, que se detallan a continuación.

- Extender las capacidades de Hermit con las siguientes funcionalidades:

- aplicación de las nuevas reglas de razonamiento para metamodelado,
- control de la existencia de ciclos en le grafo, y
- extensión del backtracking para las nuevas reglas no determinísticas.

- Agregar un conjunto de optimizaciones para mejorar el rendimiento y la velocidad de ejecución de Hermit, que se vieron degradados por la implementación de las funcionalidades anteriormente mencionadas.

Las secciones que se presentan a continuación describen la implementación de estos cambios.

5.2.2.1. Reglas de metamodelado

En la arquitectura de Hermit, que se muestra en la Figura 4.4, el componente Reasoner se encarga de exponer los servicios de razonamiento a las interfaces de usuario. Pero la complejidad detrás de estos servicios se encuentra en el componente Tableau, el cual es invocado por el componente Reasoner para ejecutar el algoritmo, que se encarga de devolver el resultado entregado al usuario. Los cambios referentes a las reglas de metamodelado presentadas en la Sección 2.3 se encuentran dentro del componente Tableau y en los componentes de tipo Manager que son usados por el mismo, en particular, MetamodellingManager y MetamodellingAxiomHelper.

En la figura 5.7 se presenta un pseudocodigo que representa en alto nivel de abstracción cada iteración del algoritmo de hypertableau, que se encuentra implementado en el componente Tableau. Cada iteración contiene a su vez un loop en donde hay iteraciones siempre y cuando se generen nuevos axiomas aplicando las reglas determinísticas de expansión del hypertableau. Luego se procesan las disyunciones, generándose una rama por cada alternativa, y se toma uno de los caminos, volviendo a aplicarse las reglas de hypertableau. En caso de encontrar contradicciones, se hace backtracking y se toma por otro camnino o se establece que la ontología es inconsistente, en caso que se hayan explorado todos los caminos.

En la Figura 5.8 se presenta el pseudocódigo del hypertableau presentado previamente en la Figura 5.7 pero con la extensión para ontologías en la lógica $\mathcal{SHIQM}^*$, que invoca las nuevas reglas para metamodelado.

```
IteracionTableau() {
    while (Se generaron nuevos axiomas) {
        Aplicar reglas de expansion
        deterministicas de hypertableau
    }
    if (si no se encuentran contradicciones) {
        Si hay ramificaciones tomar un camino
    } else {
        if (hay otro camino posible sin recorrer) {
            hacer backtracking
        } else {
            ontologia inconsistente
        }
    }
}
```

Figura 5.7: Pseudocódigo de cada iteración del hypertableau

En cada iteración, las reglas de $\approx$-rule y $\not\approx$-rule se ejecutan dentro del loop que evalúa e infiere DL-Clauses, asegurando así, la aplicación de las reglas para todos los nuevos axiomas correspondientes de la ontología a medida que estos son inferidos. Lo mismo sucede con el control de ciclos, el cual se ejecuta en cada iteración, ya que puede haberse generado un ciclo a partir de alguna inferencia. Las reglas close-rule y Close-Meta-rule son no determinísticas, y la regla MetaRule(R, S)-rule se aplica luego que no se puede aplicar más la regla Close-Meta-rule). Estas tres reglas se aplican una vez que en el loop que se ejecuta al comienzo de cada iteración, ya no se pueden aplicar más reglas de expansión, es decir que ya no se infieren más DL Clauses hasta la próxima iteración. Esto se debe a que se pueden crear disyunciones innecesarias, las cuales luego pueden perjudicar la performance del algoritmo, por la ejecución de iteraciones o la generación de datos innecesarios. Por ejemplo, si al aplicar las reglas de expansión se infiere una desigualdad, pero previo a generar dicha inferencia se aplica la regla close-rule que es no determinística, se genera una disyunción que va a evaluar la igualdad o la desigualdad entre un par de nodos. Hermit entonces procederá a evaluar la igualdad, se inferirá la desigualdad aplicando las reglas de expansión, y se encontrará una inconsistencia. Por lo tanto, se aplicará backtracking y recién en ese momento se procederá a evaluar la desigualdad. Con este ejemplo se muestra que todos los pasos ejecutados al aplicar la regla close-rule se pueden haber ahorrado al ejecutar todas las reglas de expansión antes de aplicar las reglas no determinísticas.

A continuación se describe la implementación de cada una de la reglas de metamodelado definidas en las Sección 2.3. Estas reglas se implementan en el componente

```
IteracionTableauMetamodelling() {
    while (Se generaron neuvos axiomas) {
        Aplicar reglas de expansion
        deterministicas de hypertableau
        Aplicar Regla Equal y Not Equal
        Chequeo adicional de Close Meta Rule
        if (hay ciclos) {
            contradiccion
        }
    }
    if (si no se encuentran contradicciones) {
        Aplicar Close Metamodelling Rule
        if (Se puede aplicar Close MetaRule) {
            Aplicar Close MetaRule
        } else {
            Aplicar MetaRule
        }
        if (si hay ramificaciones) {
            tomar un camino
        }
    } else {
        if (hay otro camino posible sin recorrer) {
            hacer backtracking
        } else {
            ontologia inconsistente
        }
    }
}
```

Figura 5.8: Pseudocódigo de cada iteración del hypertableau extendido para metamodelado

MetamodellingManager.

Regla ≈-rule

Sean $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$ que pertenecen a $\mathcal{M}$. Si $a \approx b$, y $A \sqcup \neg B, B \sqcup \neg A$ no pertenecen a $\mathcal{T}$, entonces se agrega $A \sqcup \neg B, B \sqcup \neg A$ a $\mathcal{T}$.

Para implementar la aplicación de esta regla se hizo una división de las distintas tareas que la misma debe realizar, para reducir la complejidad del trabajo. Primero se deben tomar pares de individuos que estén en axiomas de metamodelado $a =_{\mathsf{m}} A$ y sean iguales (ya sea porque así fueron declarados en el Abox

de la ontología o porque hermit deriva que corresponden al mismo elemento del dominio). Luego se verifica que los axiomas $A \sqcup \neg B$, $B \sqcup \neg A$ no estén en la TBox, y en ese caso se agregan.

Para asegurar que la regla se aplique a todos los pares de individuos que cumplen la condición anteriormente mencionada, se ejecutan dos iteraciones, una dentro de la otra, ambas sobre la colección de axiomas $a =_{\mathsf{m}} A$, tomando cada uno de sus individuos.

Si existe igualdad entre los individuos de un par (a, b), se procede a tomar las clases A, B asociadas a estos por los axiomas $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$. Se itera ahora entre los conjuntos de clases asociadas para tomar pares de clases. Si existen los axiomas $A \sqcup \neg B, B \sqcup \neg A$ para el par de clases (A, B) se sigue iterando. Si no existe, se agregan los axiomas.

Los axiomas de metamodelado $a =_{\mathsf{m}} A$ se leen directamente de la ontología haciendo uso de la OWL Api. Para saber si dos individuos son iguales, se toman los nodos que representan a esos individuos y se revisa si fueron mergeados y cuáles son sus representantes de la clase de equivalencia. Hermit no almacena un axioma de igualdad cuando lo infiere, sino que conserva la información de las igualdades dentro de las estructuras de los nodos.

Dado que Hermit guarda los axiomas de Tbox en forma de DL-Clauses, antes de agregar los axiomas $A \sqcup \neg B, B \sqcup \neg A$ (si ya no existen, como indica la regla) es necesario traducirlos a DL-Clauses. El equivalente a $A \sqcup \neg B$, $B \sqcup \neg A$ son dos cláusulas de inclusión atómica de conceptos entre A y B y entre B y A, $A(x) \rightarrow B(x)$ y $B(x) \rightarrow A(x)$.

Cómo guardar los axiomas de Tbox $A \sqcup \neg B$, $B \sqcup \neg A$ traducidos a DL-Clauses, no fue una tarea trivial. A diferencia de los axiomas de ABox, los cuales Hermit guarda dentro del ExtensionManager a medida que son derivados, los axiomas de TBox no se conservan en ningún lugar más que en la propia estructura de la ontología, transformada inicialmente en DL Clauses. Esto se debe a que la versión de Hermit sin extender no deriva axiomas de Tbox a medida que aplica las reglas. Es por ello que en este trabajo se tomó la decisión de guardar los axiomas de Tbox (como DL-Clauses) inferidos por las reglas de metamodelado[1] junto a las DL Clauses iniciales de la TBox de la ontología. Por lo tanto, también es en esa estructura donde se hace una búsqueda del axioma, previo a su posible inserción. En el caso en que no se encuentre el axioma y se deba agrega al conjunto de DL Clauses de la ontología,

[1]Las reglas de metamodelado que derivan axiomas de Tbox son $\approx$-rule y $\mathsf{MetaRule}(R, S)$-rule.

aún se debe realizar un paso más. Al inicio, Hermit genera una estructura dentro del componente HyperresolutionManager, necesaria para la implementación del hyperresolution calculus, en la cual guarda los axiomas de Tbox como DL-Clauses. Esta estructura se mantiene incambiada durante toda la ejecución de Hermit. Sin embargo, la extensión de Hermit para $\mathcal{SHIQM}^*$ requiere que nuevos axiomas de TBox sean derivados, e incluso eliminados (en caso que luego de aplicar la regla $\approx$-rule se detecte una contradicción y se deba hacer backtracking). Por lo tanto, es necesario reconstruir esta estructura cada vez que el conjunto de axiomas de Tbox cambia. Afortunadamente, no es una operación costosa, pero se deben tener algunas consideraciones, que se explican a continuación.

Para la creación de un nuevo HyperresolutionManager con un nuevo conjunto de axiomas de TBox, se modifica la ontología inicial, agregando o removiendo axiomas, y se vuelve a llamar al constructor del componente con esta nueva versión de la ontología. Luego se sustituye la versión anterior del componente por esta nueva versión y se continua con el algoritmo. El motivo por el cual se crea un componente nuevo y no se modifica el anterior, es porque al momento de hacer backtracking se debe reestructurar todo el componente (removiendo axiomas), entonces simplemente se vincula al componente de tableau con el HyperresolutionManager del branching point de donde surge la otra ramificación.

Regla $\not\approx$-rule

Sean $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$ que pertenecen a $\mathcal{M}$. Si $a \not\approx b$ y no existe ningún nodo raíz z tal que $(A \sqcap \neg B \sqcup B \sqcap \neg A) \in \mathcal{L}(z)$ entonces se crea un nodo raíz z y se asigna: $\mathcal{L}(z) = \{A \sqcap \neg B \sqcup B \sqcap \neg A\}$

Para la implementación de esta regla, se implementa una solución similar a la de $\approx$-rule, salvo algunas diferencias.

Al igual que en la $\approx$-rule se hace una división de tareas, con el mismo fin de reducir la complejidad. Se toman todos los pares posibles de nodos pertenecientes a axiomas de metamodelling $a =_{\mathsf{m}} A$. Luego, a diferencia de la regla $\approx$-rule se consulta si dado un par de individuos estos son diferentes.

Para saber si dos nodos son diferentes, tanto para desigualdades que se cargan con la ontología como para aquellas inferidas, podemos buscar estas desigualdades en el $ExtensionManager$. Específicamente este tipo de axiomas que se tratan de una relación y dos individuos, se encuentran en la $TernaryExtensionTable$, el cual es un array de $Object$ donde podemos encontrar la desigualdad de la forma

$A_{n_0} =\neq, A_{n_1} = a, A_{n_2} = b$ para dos nodos diferentes a y b. Por lo que con una iteracion sobre ese array basta para determinar si hay desigualdad entre ellos. Sin embargo como parte de las optimizaciones que se detallan mas adelante este comportamiento fue modificado.

Si los nodos del par en el cual se esta iterando no son diferentes, se consulta por el siguiente, pero si lo son, se itera sobre las clases relacionados a esos individuos por medio de axiomas de metamodelling, de tal manera de verificar todos los pares posibles, siendo cada par (A, B) con $A/a =_{\mathsf{m}} A$ y $B/b =_{\mathsf{m}} B$. Para cada par de clases se consulta si el axioma $A \sqcap \neg B \sqcup B \sqcap \neg A \in \mathcal{F}(z)$ para algún nodo z pertenece al $ABox$. Si pertenece, se procede con el próximo par, de lo contrario se crea un nuevo individuo z y se agrega los axiomas $A \sqcap \neg B \sqcup B \sqcap \neg A$ al $TBox$ y $\mathcal{F}(z) = \{A \sqcap \neg B \sqcup B \sqcap \neg A\}$ al $ABox$.

Los axiomas se agregan a la TBox de igual manera que en la regla anterior. Primero a la ontología y luego se crea una nueva instancia del $HyperresolutionManager$. Para el ABox Hermit ya cuenta con métodos que agregan axiomas nuevos y crean nuevos individuos. Estos se utilizan para crear el nodo z y agregar $\mathcal{F}(z) = \{A \sqcap \neg B \sqcup B \sqcap \neg A\}$ al $ABox$. Los axiomas inferidos de $ABox$ en Hermit, se guardan en el $ExtensionManager$. En particular el axioma creado en este caso iría a parar a la $binaryExtensionTable$ arreglo que almacena axiomas del tipo $A(x)$.

Regla close-rule

Sean $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} B$ que pertenecen a $\mathcal{M}$ donde $a \approx x$, $b \approx y$, siendo x e y sus respectivos representantes de las clase de equivalencia de a y b. Si no se cumple que $x \approx y$ ni $x \not\approx y$, entonces se agrega o bien $x \approx y$ o $x \not\approx y$.

En el caso $x \approx y$, también se realiza lo siguiente:
1. se agrega $\mathcal{L}(y)$ a $\mathcal{L}(x)$,
2. para todos los arcos dirigidos en $\mathcal{L}$ que van desde y a algún w, se crea un arco desde x a w si ya no existe, tal que $\mathcal{L}(x, w) = \emptyset$,
3. se agrega $\mathcal{L}(y, w)$ a $\mathcal{L}(x, w)$,
4. para todos los arcos dirigidos desde algún w a y, se crea un arco desde w a x si ya no existe, tal que $\mathcal{L}(w, x) = \emptyset$,
5. se agrega $\mathcal{L}(w, y)$ a $\mathcal{L}(w, x)$,
6. se asigna $\mathcal{L}(y) = \emptyset$ y se eliminan todos los arcos desde/a y.

Esta regla difiere mucho de las anteriores, desde su definición, hasta su implementación en Hermit. También como se mencionó previamente es aplicada en otro

bloque de código.

Para su implementación al igual que en las reglas previas se toman de a pares de individuos pertenecientes a axiomas de metamodelling. Luego se chequea que dado un par de individuos, los representantes de la clase de equivalencia de estos no sean ni iguales ni diferentes utilizando las mismas funciones que en las reglas previas. De ser así se procede a crear una disyunción en donde estos individuos, los representantes de las clases de equivalencia, o bien son iguales o bien son distintos. Por ejemplo siendo a y b los nodos representantes de la clase de equivalencia la disyunción seria de esta forma: $a = b \vee a \neq b$.

Obtener los representantes de equivalencia de cada nodo no presentó dificultad ya que las estructuras de Hermit para los nodos presentan una función que retorna el nodo canónico o en otras palabras el representante de la clase de equivalencia del mismo.

A diferencia de previas implementaciones, para esta regla como resultado se tiene que agregar una disyunción. Hermit mantiene un conjunto de disyunciones donde cada una tiene una *previousdisjunction* y una *nextdisjunction*, con las cuales Hermit genera un orden de ejecucion. Llegado su momento y en caso de tener que retroceder para tomar otro camino (backtracking) sabe que disyunciones tiene que volver a procesar. Hermit ya tiene un método para agregar una nueva disyunción por lo que no se tuvo que implementar nada nuevo para eso. Lo mismo con la evaluación de la misma y la generación de ramas, por tanto solo hay que construir la disyunción y agregarla a la estructura correspondiente utilizando funciones ya definidas.

Regla Close-Meta**-rule**

Sean $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$ and $\mathsf{MetaRule}(R, S)$ que pertenecen a $\mathcal{M}$, donde $a \approx x$, $b \approx y$, siendo x, y representantes de las clases de equivalencia de a y b. Si no se cumple que $R \in \mathcal{L}(x, y)$ ni $\sim R \in \mathcal{L}(x, y)$, entonces se agrega o bien R a $\mathcal{L}(x, y)$ o $\sim R$ a $\mathcal{L}(x, y)$.

Al igual que la regla close-rule, la Close-Meta-rule presenta una disyunción y por eso se aplican de manera consecutiva en el mismo bloque de código.

Se comienza tomando de a pares de individuos que pertenecen a axiomas de metamodelling. Para cada individuo del par, utilizando el método proporcionado por Hermit obtenemos sus representantes de las clases de equivalencias. Luego con

este nuevo par de individuos representantes de clases de equivalencia se obtienen aquellas relaciones R que relacionan a ambos nodos, es decir $R(a, b)$ con a y b los representantes de la clase de equivalencia de los nodos del par que esta siendo analizado. Una vez se tenga ese conjunto de propiedades, se itera entre los axiomas de MetaRule de la ontología y se chequea que cada R dentro de cada $MetaRule(R, S)$ en la ontología tenga una instancia de $R(a, b)$ o $\backsim R(a, b)$ siendo a y b.. Si no se encuentra ninguna de estas instancias, se debe crear una disyunción nueva en donde existe la instancia de $R(a, b)$ o $\backsim R(a, b)$.

Para el paso en el que dado un par de individuos se obtienen aquellas relaciones o propiedades que contienen una instancia con ambos individuos, al igual que en la $\neq -rule$ se busca esas instancias dentro de la $TernaryExtensionTable$ del $ExtensionManager$, ya que los axiomas de $ABox$ iniciales y los inferidos se encuentran ahí, aunque esto posteriormente terminó siendo modificado debido a las optimizaciones que se presentan más adelante.

Como adicional a este axioma, se debe agregar un chequeo en donde dados dos individuos a y b los cuales están relacionados por una relación R perteneciente a algún axioma de $MetaRule(R, S)$, estos tampoco pueden estar relacionados por R por definición. Para eso se hace un chequeo dentro de la iteración del tableau, en donde se toma de a pares de individuos pertenecientes a axiomas de metamodelling y se toman las relaciones por las cuales estos están relacionados, tomando esta información de la $TernaryExtensionTable$ del $ExtensionManager$ de $Hermit$ y si dentro de ese conjunto de clases se encuentran tanto R, como R se encuentra una inconsistencia y el modelo que estaba siendo construido no es valido. Para esto Hermit también proporciona una función que inserta un *clash* en el $ExtensionManager$ que luego hace que Hermit haga backtracking, de haber otro camino posible, de lo contrario concluirá que la ontología es inconsistente.

Regla $\mathsf{MetaRule}(R, S)$**-rule**

Sean $a =_{\mathsf{m}} A$ y $\mathsf{MetaRule}(R, S)$ que pertenecen a $\mathcal{M}$, y se asume la $\mathsf{Close\text{-}Meta}$-rule no puede ser aplicada. Si $\neg A \sqcup \forall S.(\sqcup X)$ no pertenece a $\mathcal{T}$, entonces se agrega $\neg A \sqcup \forall S.(\sqcup X)$ a $\mathcal{T}$, siendo $X = \mathsf{Image}_{\mathcal{L}}(R, a)$ donde $\mathsf{Image}_{\mathcal{L}}(R, a) = \{B \mid P \in \mathcal{L}(x, y),\ b =_{\mathsf{m}} B,\ P \sqsubseteq^* R,\ a \approx x,\ b \approx y$, siendo x e y los representantes de las clases de equivalencia de a y $b\}$.

Por definición la MetaRule(R,S)-rule debe ser aplicada luego de que la Close-MetaRule ya no pueda ser aplicada. Para asegurar esto, dentro de la iteración de Hermit, la $\mathsf{MetaRule}(R, S)$-rule es aplicada luego de que se intenta aplicar la

Close-MetaRule y esta ya no produce ninguna disyunción.

Para la implementación de esta regla se itera sobre los axiomas de metamodelling $a =_m A$ y para cada nodo que representa el individuo del axioma, en este caso a, se hace una nueva iteración pero entre los axiomas de $MetaRule(R, S)$, donde se buscan todos aquellos nodos relacionados con a por R, es decir los nodos b con $R(a, b)$. Si el conjunto de los nodos relacionados a a por R contiene al menos un individuo, entonces se procede a obtener el representante de la clase de equivalencia de cada nodo del conjunto y así obtener las clases relacionadas por un axioma de metamodelling a los individuos representados por esos nodos. Esto es, si tenemos el conjunto de nodos $\{b_1, ..., b_n\}$ como representante de las clases de equivalencia de los nodos relacionados a a por R, entonces B_i pertenece al conjunto de clases relacionados por axioma de metamodelling a esos nodos si $b_i =_{\mathsf{m}} B_i$ para algun $b_i \in \{b_1, ..., b_n\}$. Luego se verifica si el axioma $\neg A \sqcup \forall S.(\sqcup X)$ donde $Image_F(R, a) = \{B | P \in F(x, y), b =_m B, P \sqsubseteq R, a \approx x, b \approx y$ siendo x e y los representantes de las clases de equivalencia de a y $b\}$, se encuentra en el $TBox$. De no ser así se agrega a la $TBox$.

5.2.2.2. Chequeo de ciclos

Como se describe en la Sección 2.3, para que el algoritmo de Tableau devuelva que la ontontología es consistente, se debe verificar que no se pueden ejecutar más reglas de expansión, no se deben encontrar contradicciones y el grafo que se construye a partir de la ontología inicial y de la aplicación de las reglas no contiene ciclos. Por lo tanto, en adición a la implementación de las nuevas reglas de metamodelado se debe agregar un chequeo de ciclos. Este chequeo evita que en el modelo canónico representado por el grafo, un elemento del dominio pertenezca a sí mismo, de acuerdo a la Definición 6 de ciclo, que se ilustra en la Figura 2.4, la cual por claridad se incluye nuevamente a continuación.

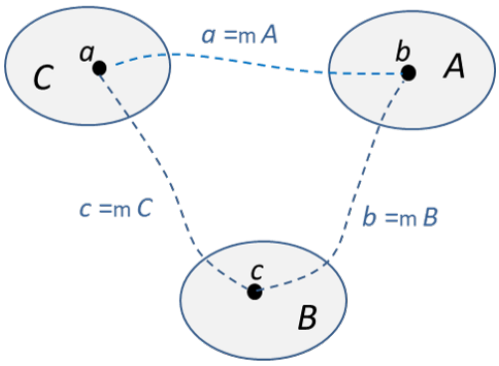

Figura 5.9: Ejemplo de grafo con un ciclo

Para encontrar ciclos en el grafo construido por Hermit se utilizó el algoritmo para encontrar ciclos en un grafo dirigido [16]. Las Figuras 5.10 y 5.11 muestran el código implementado.

```
public static boolean findCyclesInM(Tableau tableau) {
    flaggedNodes = new ArrayList<Node>();
    nodeStack = new Stack<Node>();
    existCycle = false;
    for (Node metamodellingNode : tableau.getMetamodellingNodes()) {
        if (!flaggedNodes.contains(metamodellingNode)) {
            controlCycle(metamodellingNode, tableau);
            if (existCycle) {
                return true;
            }
        }
    }
    return false;
}
```

Figura 5.10: Algoritmo para encontrar ciclos

El algoritmo es recursivo y utiliza un stack de nodos para marcar a aquellos nodos que no finalizaron la llamada recursiva. Básicamente lo que se hace es invocar a la función *controlCycle* para cada nodo a perteneciente a un axioma de metamodelado $a =_{\mathsf{m}} A$, siempre y cuando ese nodo no esté marcado. Cada uno de estos nodos, y todos los nodos equivalentes, son marcado y, agregados al stack, invocándose recursivamente a la función *controlCycle* para las instancias de los conceptos A en los axiomas de metamodelado. En caso que algunas de estas instancias ya se encuentre en el stack, el algoritmo encuentra un ciclo.

```
private static void controlCycle(Node node, Tableau tableau) {
    nodeStack.push(node);
    flaggedNodes.add(node);
    for (Node instance : getInstancesFromMetamodellingEqualClasses(node, tableau)) {
        if (existCycle) {
            return;
        } else {
            if (!flaggedNodes.contains(instance)) {
                controlCycle(instance, tableau);
            } else {
                if (nodeStack.contains(instance)) {
                    existCycle = true;
                }
            }
        }

    }
    nodeStack.pop();
    return;
}
```

Figura 5.11: Función auxiliar al algoritmo para encontrar ciclos

5.2.2.3. Backtracking

Una de las implementaciones que presentó más complejidad fue la extensión de la funcionalidad de backtracking.

Hermit por sí mismo ya hace backtracking sobre el ABox (representado por el grafo con nodos y arcos etiquetados), pero no sobre los axiomas de TBox. En la extensión para metamodelado fue necesario implementar backtracking para la Tbox debido a que Las nuevas reglas no determinísticas Close-rule y Close-Meta-rule disparan la ejecución de las reglas $\approx$-rule y MetaRule(R, S)-rule, las cuales agregan axiomas de Tbox. Por ejemplo, para los axiomas $a =_{\mathsf{m}} A$ y $b =_{\mathsf{m}} b$, si a y b no están declarados iguales ni diferentes, la Close-rule introduce $a = b$ o $a \neq b$; por $a = b$ agrega axiomas de Tbox equivalentes a $A \equiv B$, y si posteriormente se detecta una contradicción, esos axiomas se deben eliminar para tomar la alternativa $a \neq b$.

Como fue mencionado previamente, los axiomas de TBox se mantienen en un aestructura del HyperresolutionManager. Para implementar backtracking para la Tbox, se agregó una estructura nueva que almacena las distintas versiones de los HyperresolutionManager. Esta estructura es usada en caso de hacer backtracking y tener que volver a una versión anterior de la Tbox. La forma de saber qué HyperresolutionManager vincular al componente Tableau al realizar backtracking es guardar cada versión del HyperresolutionManager junto con el BranchingPoint, es decir con el identificador de la ramificación a donde se debería volver en caso de

encontrarse una inconsistencia. Por lo tanto, en el momento en que Hermit encuentra una contradicción y determina que debe hacer backtracking (ya que aún tiene ramas por recorre)r, vuelve a un BranchingPoint y en este punto la extensión de Hermit vincula al componente de Tableau la nueva versión de HyperresolutionManager sin los axiomas de TBox que pudieron haberse derivado de reglas de metamodelado (como la ≈-rule) en el recorrido de esa rama.

5.2.2.4. Optimizaciones

Una vez que se finalizó la implementación de los componentes anteriormente descriptos (nuevas reglas, control de ciclos del grafo, entre otros) y se ejecutaron los casos de prueba, se observó que el tiempo de ejecución de los mismos, en algunos casos, era bastante superior al requerimiento de desempeño de 20 segundos planteado en la Sección 3.4. Si bien la mayoría de casos se ejecutan en menos de un segundo, en esta primera versión de Hermit extendido para metamodelado, la complejidad del modelo o grafo que se construye impactó de forma exponencial en su tiempo de ejecución. En este caso, la complejidad del modelo se mide en términos de la cantidad de disyunciones que se crean al aplicar las reglas close-rule y Close-Meta-rule, a partir de individuos con metamodelado los cuales no están explícitamente igualados, diferenciados o relacionados.

Poniendo foco en aquellos casos con peor desempeño, se compararon los tiempos de respuesta del razonador extendido, con los tiempos de respuesta de Hermit sin la extensión para los mismos casos quitando los axiomas de metamodelado. De esta forma, fue posible comprobar el desempeño de Hermit sin la extensión era razonable, quedando claro que la causa de la degradación del desempeño se debió a la implementación de la extensión.

Como segunda investigación para encontrar aquellas implementaciones que consumían más tiempo, se registró el tiempo de ejecución de cada regla utilizando funciones de java y logs. Se encontró que aquellas reglas que recorrían arreglos, sobre todo los de ABox, eran las que insumían más tiempo.

Por último, se encontró que en algunas iteraciones se estaba invocando al componente MetamodellingManager para aplicar reglas de metamodelado, cuando en realidad en varias de estas invocaciones no se cumplían las condiciones para derivar nuevos axiomas (por ejemplo axiomas de TBox al aplicar la regla ≈-rule) o no se realiza backtracking. Por lo tanto, evitando realizar siempre esa invocación en cada iteración, se podría mejorar el desempeño.

Teniendo como insumo la investigación anteriormente descripta, se procedió a la implementación de tres optimizaciones que lograron disminuir el tiempo de ejecución a unos 30 segundos aproximadamente. Estas optimizaciones consistieron en:

1. mejorar el chequeo que devuelve si dos nodos son diferentes,

2. mejorar el chequeo que devuelve si dos nodos están relacionados por un rol y

3. habilitar y deshabilitar la invocación de las reglas $\approx$-rule y $\neq$-rule.

Las optimizaciones 1 y 2 surgen de la necesidad de evitar la iteración sobre el array de axiomas del ABox. La solución propuesta fue utilizar la estructura Map de Java en su implementación con Hash, para pasar a tener operaciones de orden n a un orden de complejidad menor. Las optimizaciones consisten en que al inicio del algoritmo de Tableau se guardan en los respectivos mapas todas las desigualdades y axiomas que relacionan dos nodos a través de alguna relación. Además, como al momento de realizar backtracking algunos axiomas o desigualdades previamente derivados deben ser descartados, éstos son removidos de los mapas. aldades inferidos deben ser descartados, estos se deben remover de los mapas. De esta forma, en la aplicación de las reglas $\neq$-rule, Close-rule, Close-Meta-rule y MetaRule(R, S)-rule, al momento de chequear si dos nodos son diferentes o si se relacionan por una relación R, se sustituye la iteración en el array de axiomas de ABox por un par de accesos en un mapa y una pequeña iteración.

Esta pequeña iteración sería, por ejemplo, en el caso de la desigualdad, el Mapa tiene como key los nodos y como valor la lista de nodos de los cuales el nodo key es diferente. Para el caso de relaciones la key siguen siendo los nodos, pero el valor es otro mapa, con key siendo los nodos nuevamente y el valor una lista de relaciones por los cuales los dos nodos key son relacionados y en ese orden en particular. En la figura 5.12 se muestra un ejemplo de las estructuras utilizadas para esta optimización. En el primer ejemplo de la figura, el cual describe un mapa de desigualdad, se muestra como el mapa tiene 2 elementos, uno con key a y valor una lista de nodos con b y c ya que $a \neq b$ y $a \neq c$. En el ejemplo del mapa de relaciones se muestra que tiene también 2 elementos en el mapa. El primero de ellos tiene como key a y como valor otro mapa con 2 elementos. El primero tiene como key b y como valor una lista con dos relaciones R y S, esto significa que a esta relacionado con b por R y S o en otras palabras $R(a, b)$ y $R(a, c)$.

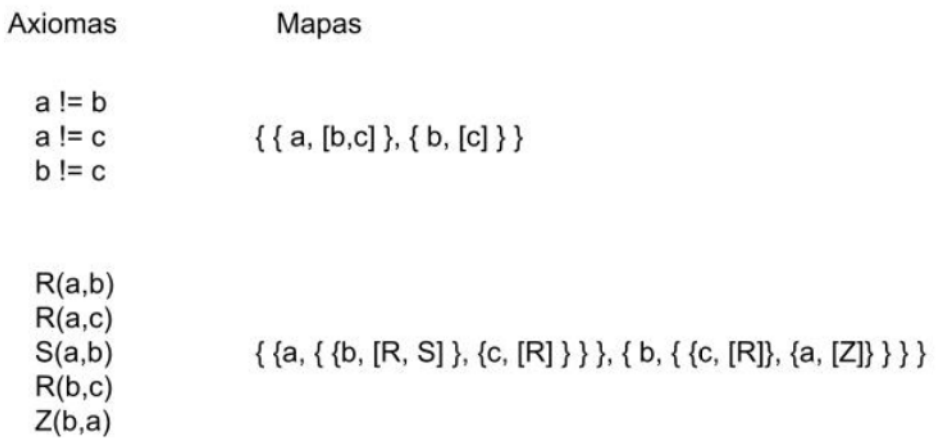

Figura 5.12: Ejemplo de los mapas usados para las optimizaciones

La tercera optimización se implementó para evitar llamadas innecesarias al MetamodellingManager al aplicar las reglas $\approx$-rule y $\neq$-rule. Como se mencionó anteriormente, hay iteraciones en las que se sabe de antemano que no se derivará ningún axioma como resultado de la invocación, por lo que se pueden evitar esos costos de tiempo. La solución que se implementó consiste en activar una bandera cada vez que se deriva una igualdad, desigualdad o se hace backtracking, y desactivarla luego de invocar a las reglas de $\approx$-rule y $\neq$-rule.

La Figura 5.13 muestra cómo mejora el desempeño del razonador Hermit extendido para tres casos de prueba.

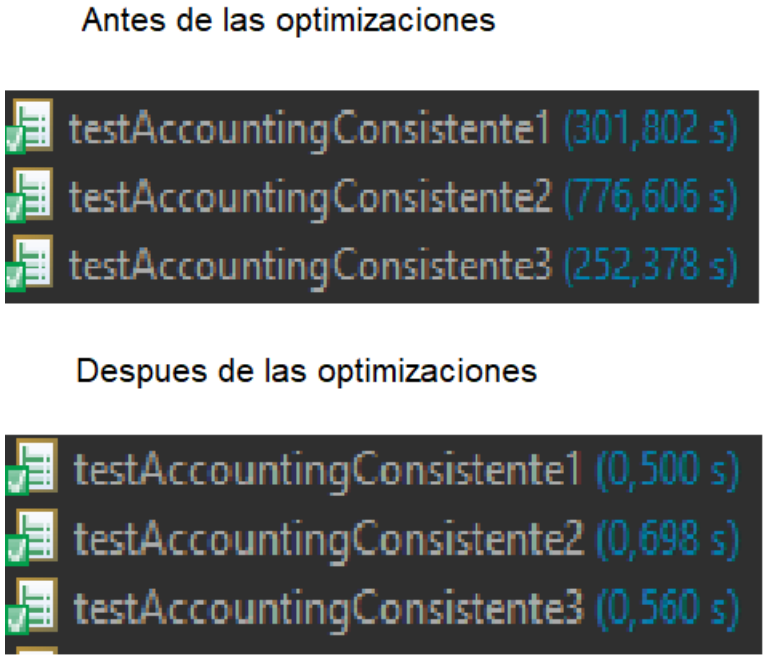

Figura 5.13: Resultado de las optimizaciones en ejecución casos de prueba

5.3. Prototipo de contabilidad

El prototipo de contabilidad se implementó usando la librería de Java Swing. La elección por Swing surge a partir de que es fácil de usar y rápidamente se puede crear una aplicación de escritorio. Además, se integra fácilmente con Hermit y la OWL API ya que todo es parte del ambiente Java.

A nivel gráfico, la interfaz de esta mini aplicación consiste en dos paneles, uno para el ingreso de las definiciones de asientos y la otra para el ingreso de los asientos propiamente dichos. La Figura 5.14 muestra cómo luce la interfaz implementada. En ambos paneles, cada fila provee campos para que el usuario ingrese los datos de las definiciones, en el panel superior, y de los asientos, en el panel inferior. También se agregan botones que permiten agregar filas y un botón para verificar si los datos ingresados son consistentes, en otras palabras, si la ontología que se alimenta con los datos ingresados por el usuario es consistente o no.

Tanto la creación de la interfaz como las integraciones con Hermit y la OWL API fueron sencillas gracias a que todo es parte de Java y está integrado como si fuese

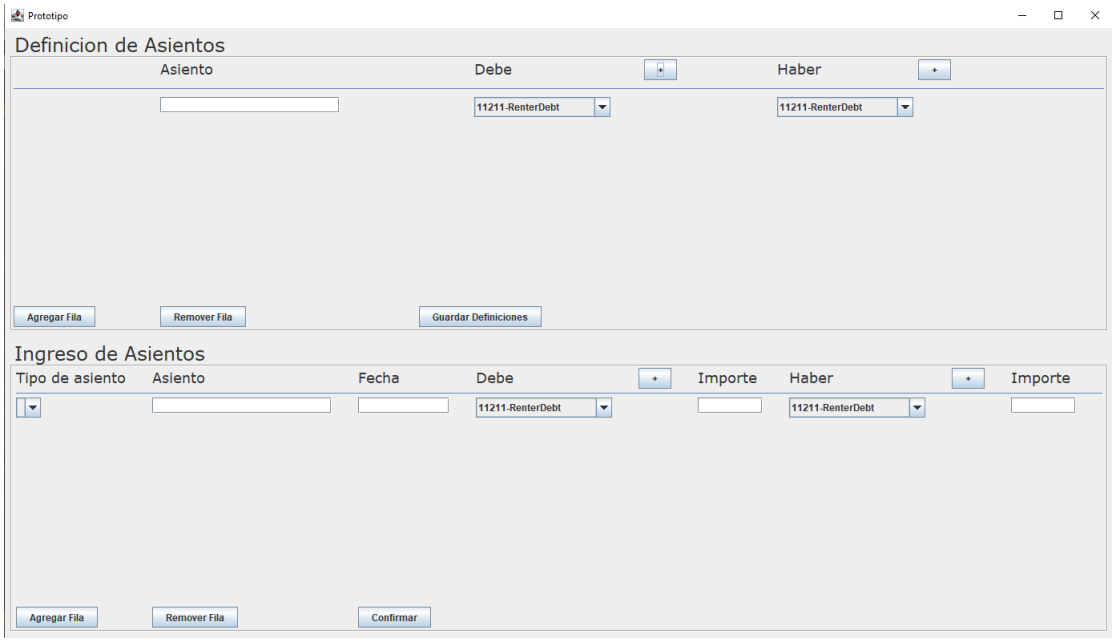

Figura 5.14: Aplicación de contabilidad

el mismo proyecto. La mayor complejidad en la implementación de esta aplicación consistió en la construcción de la ontología a partir de los datos ingresados. Cada fila en los paneles de definición o ingreso de asientos representa un conjunto de axiomas que se deben modelar en la ontología, que luego será procesada por el razonador. Por lo tanto, en el momento en el que el usuario presiona el botón *confirmar*, se leen los datos ingresados, se agregan los axiomas a la ontología y se ejecuta el razonador Hermit extendido para metamodelado, con esa ontología como entrada.

A continuación, se describe brevemente cómo se va construyendo la ontología con metamodelado a partir de los datos ingersados.

En el panel superior de la figura 5.14 se visualiza la definición del *asiento mensual de generación de la deuda* de alquiler y cuotas (en caso de convenios) de los inquilinos, que está ya cargado en la ontología. Los axiomas que representan esta definición son:

$EntryDef(monthRent)$
$DetDef(monthDebtDet)$ $DetDef(monthFeeDet)$ $DetDef(monthLandlordDet)$
$detailDefD(monthRent, monthDebtDet)$ $detailDefD(monthRent, monthFeeDet)$
$detailDefC(monthRent, monthLandlordDet)$

$monthRent =_{\mathsf{m}} MonthRentEnt$
$monthDebtDet =_{\mathsf{m}} MonthDebtDet \quad monthFeetDet =_{\mathsf{m}} MonthFeeDet$
$monthLandlordDet =_{\mathsf{m}} MonthLandlordDet$

$\mathsf{MetaRule}(detailDefD, detailD) \qquad \mathsf{MetaRule}(detailDefC, detailC)$

Como se describe en la Sección 4.3, las instancias *monthRent*, *monthDebtDet*, *monthFeeDet* y *monthLandlordDet* representan al asiento mensual de generación de la deuda con sus detalles, desde la perspectiva del usuario experto. Para cada una de estas instancias, se crean los conceptos *MonthRentEnt*, *MonthDebtDet*, *MonthFeeDet* y *MonthLandlordDet* que representan estos mismos asientos y detalles desde la perspectiva del usuario operador. Los axiomas de metamodelado como $monthRent =_{\mathsf{m}} MonthRentEnt$ vinculan las instancias a los conceptos correspondientes, y los axiomas como $\mathsf{MetaRule}(detailDefD, detailD)$ expresan que la correspondencia entre asientos y detalles definida por el usuario experto deben mantenerse al nivel de los conceptos correspondientes. Como se explica en la Sección 2.3, al ejecutar el razonador se infiere:

$MonthRentEnt \sqsubseteq \forall detailD.(MonthDebtDet \sqcup MonthFeeDet)$
$MonthRentEnt \sqsubseteq \forall detailC.MonthLandlordDet$

Una de las ventajas del enfoque de metamodelado en este caso de estudio es que las restricciones que se agreguen en el nivel de definición evitan que se ingresen asientos incorrectos al nivel del usuario operador. Una posible restricción (tomada del caso real) es que los asientos con cuentas de disponibilidad (Caja, Banco) al debe, solo pueden tener al haber cuentas de activo que no sean de disponibilidad[2]. Esta restricción está representada en la ontología por el siguiente axioma:

$\exists detailDefD.(\exists account.Availability) \sqsubseteq \forall detailDefC.(\exists account.(Asset \sqcap \neg Availability))$

Si el usuario experto ingresa la definición del *asiento de pago de inquilinos*, con detalles al debe asociados a las cuentas de Caja y Banco, y detalles al haber asociados a las cuentas de deuda de inquilinos, deudas por cuotas y arrendadores, los axiomas que se agregan a la ontología son:

$EntryDef(renterPay)$
$DetDef(payCashDet) \quad DetDef(payBankDet)$

[2]La Figura 4.6 muestra la clasificación de las cuentas en disponibilidad (Availability), Activo (Asset) y Pasivo (Liability).

$DetDef(payDebtDet)$ $DetDef(payFeeDet)$ $DetDef(payLandlordDet)$
$detailDefD(renterPay, payCashDet)$ $detailDefD(renterPay, payBankDet)$
$detailDefC(renterPay, payDebtDet)$ $detailDefC(renterPay, payFeeDet)$
$detailDefC(renterPay, payLandlordDet)$

$renterPay =_{\mathsf{m}} RenterPayEnt$
$payCashDet =_{\mathsf{m}} PayCashDet$ $payBankDet =_{\mathsf{m}} PayBankDet$
$payDebtDet =_{\mathsf{m}} PayDebtDet$ $payFeeDet =_{\mathsf{m}} PayFeeDet$
$payLandlordDet =_{\mathsf{m}} PayLandlordDet$

Si se ejecuta el razonador Hermit extendido, va a devolver que la ontología es inconsistente, ya que de acuerdo a la restricción ingresada, los asientos con detalles al debe asociados a cuentas de disponibilidad deben tener al haber solo cuentas de activo que no son de disponibilidad. Sin embargo, la definición de asiento *renterPay* tiene al haber al detalle *payLandlordDet* que está asociado a una cuenta de pasivo. Como los detalles están asociados a una única cuenta (el rol *account* es funcional) y los conceptos *Asset* y *Liability* son disjunto, se genera una contradicción. La Figura 5.15 muestra la respuesta que recibe el usuario al confirmar la definición ingresada.

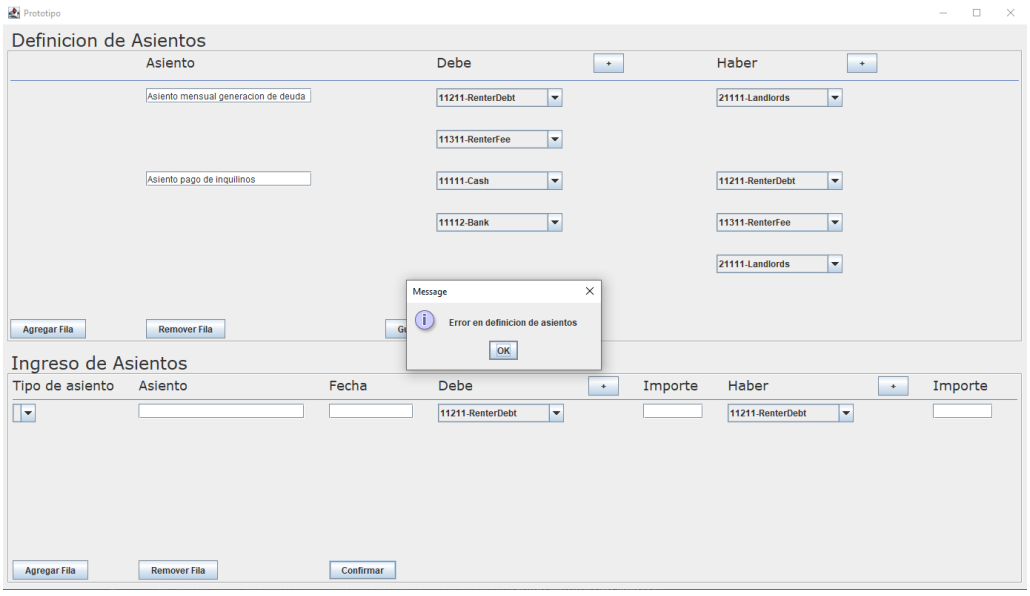

Figura 5.15: Aplicación de contabilidad: error en definición de asiento

En el ingreso de los asientos por el usuario operador, la mayor ventaja que brinda el enfoque de metamodelado es asegurar que los asientos cumplan con las definicio-

nes de los expertos. Si al ingresar un asiento concreto de pagos del inquilino Juan, el usuario introduce al haber un detalle asociado a una cuenta de desperfectos que no es ni de deuda de alquiler ni de cuotas, como indica la definición del asiento de pago de inquilinos, el razonador devuelve que la ontología es inconsistente cuando es ejecutado al confirmar el asiento. Esto sucede porque a partir del axioma $\mathsf{MetaRule}(detailDefC, detailC)$ se infiere:

$$RenterPayEnt \sqsubseteq \forall detailC.(PayDebtDet \sqcup PayFeeDet)$$

Los conceptos correspondientes a diferentes detalles son todos disjuntos, por lo que el detalle de desperfectos no puede pertenecer a *PayDebtDet* ni a *PayFeeDet*. La Figura 5.16 muestra la respuesta que recibe el usuario al confirmar el asiento ingresado.

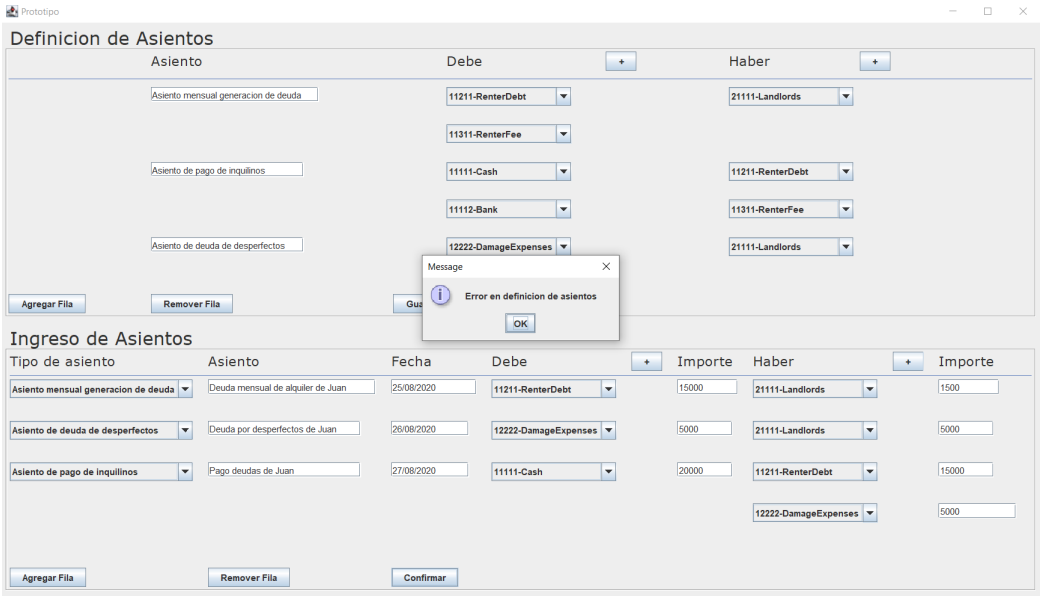

Figura 5.16: Aplicación de contabilidad: asiento no respeta definición

6. Verificación

En esta sección se presentan los métodos de verificación de la calidad de la implementación de la extensión del razonador Hermit para metamodelado. La aplicación de estos métodos permite asegurar el cumplimiento de los requerimientos presentados en el Capítulo 3. Se plantean dos métodos de verificación: revisión estática y revisión dinámica. A continuación se describe cómo se llevaron adelante las revisiones estáticas y dinámicas, y qué requerimientos fueron cubiertos por cada una de ellas.

6.1. Revisión estática

Para asegurar el cumplimiento de los requerimientos de modularidad y estándares de calidad del código, y de la especificación del algoritmo para $\mathcal{SHIQM}^*$, descriptos en la Sección 3.4, se realizaron revisiones estáticas que se describen a continuación.

6.1.1. Modularidad y estándares de calidad del código

Uno de los puntos a verificar es el hecho de que se cumpla con la arquitectura de Hermit, la cual es una arquitectura modular. Mas específicamente, tomando como referencia el componente de tableau y todos aquellos componentes relacionados a las tareas de razonamiento, en la Figura 6.1 se ve cómo el modulo de tableau, que tiene la responsabilidad de ejecutar las iteraciones para aplicar las reglas, accede a distintos componentes de tipo Manager, que tienen diferentes responsabilidades. Respetando esta arquitectura, en la implementación de la extensión se agregó un componente de tipo manager, el MetamodellingManager, el cual tiene la responsabilidad de aplicar las reglas de metamodelado. En la Figura 4.5 de la Sección 4.2 que describe el diseño de la extensión, se representa la integración de este nuevo componente a la arquitectura de Hermit. Además del MetamodellingManager, se

crea una clase auxiliar que funciona como parte del módulo, MetamodellingAxiomHelper, con la finalidad de no cargar al MetamodellingManager de tanto código y de tareas de más bajo nivel como la construcción de nuevas DL-Clauses o la comparación entre ellas. En la Figura 6.2 se puede ver un ejemplo de cómo desde el componente Tableau se accede a las funciones expuestas por el MetamodellingManager para la aplicación de reglas de metamodelling.

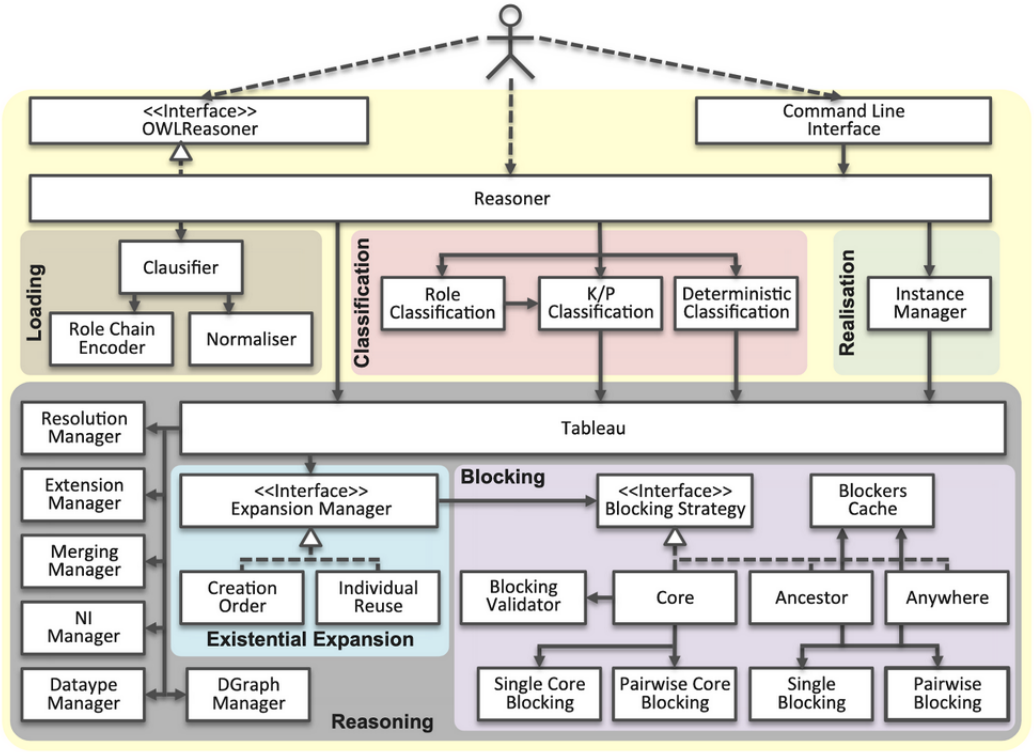

Figura 6.1: Arquitectura de Hermit [15]

```
this.m_metamodellingManager.checkCloseMetamodellingRule();
if (!this.m_metamodellingManager.checkCloseMetaRule()) {
    this.m_metamodellingManager.checkMetaRule();
}
```

Figura 6.2: Ejemplo de uso del MetamodellingManager

Otro de los puntos a verificar es el cumplimiento de los estándares de programación orientada a objetos, como lo son el bajo acoplamiento y la alta cohesión.

El principio de alta cohesión nos dice que la información que almacena y maneja una clase debe ser coherente, y sus responsabilidades deben estar fuertemente relacionadas. Dentro del nuevo componente MetamodellingManager se encuentran

métodos que representan la aplicación de las nuevas reglas del algoritmo para $\mathcal{SHIQM}^*$, así como colecciones de datos utilizadas por estos métodos, como por ejemplo la estructura que guarda los nodos con metamodelado que son diferentes entre sí, mencionada al describir las optimizaciones en la Sección 5.2.2.4. Claramente, tanto los métodos como los atributos de la clase son coherentes y están relacionados con funciones asociadas a la relación de metamodelado, por lo que podemos afirmar que se cumple con el principio de alta cohesión.

El principio de bajo acoplamiento refiere a la idea de tener pocas dependencias entre las clases, de modo de que si se realiza una modificación en una de ellas, tenga poca repercusión en el resto. Para cumplir con este principio, la interacción entre los componentes Tableau y MetamodellingManager se redujo a invocaciones desde Tableau para la aplicación de reglas. Por lo tanto, se minimiza el acoplamiento entre componentes, logrando cumplir con el requisito.

6.1.2. Especificación de algoritmo para $\mathcal{SHIQM}^*$

El presente proyecto se basa en el algoritmo de razonamiento para ontologías en la lógica $\mathcal{SHIQM}^*$ propuesto en el enfoque presentado en [5, 8]. Dado que Hermit chequea la consistencia de ontologías en la lógica $\mathcal{SROIQ}$, no detecta inconsistencias que surgen de los axiomas de metamodelado presentados en la Sección 2.3, por lo que en este proyecto se implementaron las reglas $\approx$-rule,$\neq$-rule, close-rule, Close-Meta-rule y MetaRule(R, S)-rule. En la Sección 5.2.2 se especifica cómo fueron implementadas estas reglas, así como también el control de existencias de ciclos en el grafo que construye el razonador, de acuerdo a la Definición 6 presentada en la Sección 2.3. Como la extensión está basada en el algoritmo propuesto en [5, 8] se puede afirmar que se cumple con la especificación del mismo.

6.2. Revisión dinámica

Para asegurar el cumplimiento de todos los requerimientos funcionales y el requerimiento no funcional de desempeño, descriptos en el Capítulo 3, se aplicó el método de revisión dinámica mediante la ejecución de casos de prueba, como se describe a continuación.

6.2.1. Requerimientos funcionales

Para diseñar los casos de prueba se consideraron los escenarios planteados en la Sección 3.3 que describe los requerimientos funcionales, para asegurar el cubrimiento de los mismos. Cada escenario cubre un conjunto de ontologías de acuerdo a un criterio que combina las siguientes clasificaciones:

- lógicas $\mathcal{SHIQ}$ (o un fragmento de ella), $\mathcal{SHIQM}$, $\mathcal{SHIQM}^*$, $\mathcal{SHIQM}^*$ con solo axiomas $\mathsf{MetaRule}(R, S)$,
- consistente, inconsistente

Además, en los escenarios para ontologías con metamodelado inconsistentes, se incluyeron casos de prueba con existencia de ciclos. Por lo tanto, se utilizó una estrategia de testing basado en requerimientos y funcionalidades.

Los distintos casos de prueba en cada escenario junto a sus especificaciones, resultados esperados y resultados obtenidos se adjuntan en el Anexo A.

6.2.2. Requerimiento de desempeño

Los casos de prueba de los diferentes escenarios también permitieron hacer una evaluación del tiempo de ejecución de Hermit extendido para metamodelado. Una vez implementadas las optimizaciones que se describen en la Sección 5.2.2.4, ninguno de los casos de prueba tiene un tiempo de ejecución superior a los 20 segundos, límite planteado en el requerimiento de desempeño especificado en la Sección 3.4.

7. Conclusiones y trabajos futuros

El principal objetivo del presente proyecto es la implementación de una extensión de Hermit capaz de procesar ontologías con metamodelado y determinar si éstas son consistentes o no. Además, para mostrar la utilidad de la extensión implementada, se incluye en el alcance de este proyecto la implementación de un prototipo para simular un caso de uso de contabilidad, que integra el razonador Hermit extendido para determinar si los asientos de contabilidad ingresados por el usuario cumplen con un conjunto de reglas definidas por un usuario experto.

Como resultado del trabajo realizado, se logra cumplir con el objetivo principal del proyecto, obteniéndose un producto que cumple con los requerimientos de forma satisfactoria, de acuerdo a los resultados obtenidos de la fase de verificación de la calidad del producto. Como trabajo futuro resta realizar algunos ajustes que se describen en la siguiente sección.

En cuanto a los conocimientos adquiridos, este proyecto permitió profundizar en un gran conjunto de conocimientos, ya sea sobre base de conocimientos, base de conocimientos con metamodelado, razonadores, algoritmos complejos como tableau e hypertableau, y otros tipos de algoritmos como el de chequeo de ciclos en un grafo.

Todo lo documentado en este informe así como lo implementado en este proyecto forman una base sólida y resuelve la parte esencial del trabajo a la hora de implementar una versión final de Hermit extendido para ontologías con metamodelado.

7.1. Trabajo futuro

Se identifican un conjunto de mejoras a realizar con el fin de obtener una versión del producto para liberar a la comunidad. Estas mejoras consisten en: optimizar tiempos de ejecución y realizar ajustes para resolver casos de prueba puntuales que con la versión del producto obtenida no siempre devuelven el resultado esperado.

Con respecto a optimizar tiempos de ejecución, a pesar de que en este proyecto se cumple con el requerimiento planteado de 20 segundos, los tiempos de respuesta obtenidos no se asemejan a los de la versión de Hermit sin metamodelado. El límite para esta mejora debería pasar de los 20 segundos a estar en el orden del segundo o un poco más para ontologías extremadamente complejas. La implementación de las optimizaciones sobre la versión extendida de Hermit que se realiza en el presente proyecto, son una demostración de que se pueden lograr considerables mejoras a partir de pequeños cambios.

Con respecto a la segunda mejora planteada, del total de casos de prueba ejecutados (documentados en el Anexo A), hay un par de casos que generaron resultados distintos en distintas ejecuciones. Esta situación se dio de manera aislada, concretamente ocurrió que al ejecutar estos casos reiteradas veces se puede obtener un resultado no deseado en alguna de las ejecuciones. Se identifica entonces como mejora resolver el defecto relacionado con estas ejecuciones fallidas.

Para este trabajo a futuro se deja la versión extendida de la OWL API [17] y de Hermit [18] en dos repositorios de Github.

Bibliografía

[1] N. Guarino, D. Oberle y S. Staab. "What is an ontology?" En: *Handbook on Ontologies*. Ed. por S. Staab y R. Studer. Springer Publishing Company, Incorporated, 2009, págs. 1-17.

[2] R. Studer, V.R. Benjamins y D. Fensel. "Knowledge engineering: Principles and methods". En: *Data & Knowledge Engineering* 25(1) (1998).

[3] G. Antoniou y F. van Harmelen. "Web Ontology Language: OWL". En: *Handbook on Ontologies*. Ed. por S. Staab y R. Studer. Springer Publishing Company, Incorporated, 2009, págs. 112-131.

[4] P. Hitzler, M. Krötzsch y S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.

[5] P. Severi, E. Rohrer y R. Motz. "A Description Logic for Unifying Different Points of View". En: *Knowledge Graphs and Semantic Web - First Iberoamerican Conference, (KGSWC)*. Springer, 2019.

[6] I. Horrocks, B. Motik y Z. Wang. "The HermiT OWL Reasoner". En: *1st International Workshop on OWL Reasoner Evaluation (ORE-2012)*. CEUR-WS.org, 2012.

[7] E. Sirin y col. "Pellet: A practical OWL-DL reasoner". En: *Web Semantics: Science, Services and Agents on the World Wide Web* 5(2) (2007).

[8] R. Motz, E. Rohrer y P. Severi. "The description logic SHIQ with a flexible meta-modelling hierarchy". En: *J. Web Semant.* 35 (2015).

[9] B. Motik, Rob Shearer e I. Horrocks. "A Hypertableau Calculus for SHIQ". En: *Description Logics* (2007).

[10] I. Horrocks, U. Sattler y S. Tobies. "Reasoning with Individuals for the Description Logic SHIQ". En: *Proceedings of the 17th International Conference on Automated Deduction* (2000).

[11] *OWL API*. URL: `http://owlapi.sourceforge.net/`.

[12] Sean Bechhofer, Raphael Volz y Phillip Lord. "Cooking the Semantic Web with the OWL API". En: *The Semantic Web - ISWC 2003* (2003), págs. 659-675.

[13] *RDF*. URL: https://www.w3.org/RDF/.

[14] Matthew Horridge y Sean Bechhofer. "The OWL API: A Java API for OWL Ontologies". En: *Semantic Web journal* 2 (2011), págs. 11-21.

[15] Birte Glimm y col. "HermiT: An OWL 2 Reasoner". En: *Journal of Automated Reasoning* 53 (2014), págs. 245-269.

[16] R. Sedgewick y K. Wayne. *Algorithms*. Addison-Wesley, 2011.

[17] *Versión extendida OWL API código*. URL: https://github.com/brunodibello/owlapi_4.5.7_metamodelado.

[18] *Versión extendida Hermit código*. URL: https://github.com/brunodibello/Hermit_143456_metamodelado.

A. Anexo - Casos de prueba

Escenario ontología $\mathcal{SHIQ}$ consistente.

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestConservativity1	$A_1(p)$, $\neg A_2(p)$, $a_1 = a_2$	Consistente	Consistente
TestConservativity2	$A_1 \equiv A_2$, $A_1(p)$, $a_1 \neq a_2$	Consistente	Consistente
TestConservativity3	$B(a)$, $A(b)$	Consistente	Consistente
TestCycles4	$A = A_1 \sqcap A_2$, $A = A_3 \sqcap \exists R.A_2$, $A_3(a)$, $\exists R.A_2(a)$	Consistente	Consistente
TestCycles6	$A = A_1 \sqcap A_2$, $A = A_3 \sqcap \exists R.A_2$, $B = B_3 \sqcap \forall R.B_2$, $B = B_1 \sqcup B_2$, $B_3(p)$, $\forall R.B_2(p)$, $A_3(q)$, $\exists R.A_2(q)$, $a = p$, $b = q$	Consistente	Consistente
TestCycles9	$A = A_1 \sqcap A_2$, $X = A_3 \sqcap \exists R.A_2$, $X \subseteq A$, $A_3(a)$, $\exists R.A_2(a)$	Consistente	Consistente
TestCycles11	$A = A_1 \sqcap A_2$, $B = B_1 \sqcup B_2$, $X = A_3 \sqcap \exists R.A_2$, $Y = B_3 \sqcap \forall R.B_2$, $X \subseteq A$, $Y \subseteq B$, $B_3(p)$, $\forall R.B_2(p)$, $A_3(q)$, $\exists R.A_2(q)$, $a = p$, $b = q$	Consistente	Consistente
TestEquality20	$A = A_1 \sqcap (A_2 \sqcup \forall R.A_3)$, $A = A_4 \sqcup \not A_5$, $B = B_1 \sqcap \exists R.B_2$, $B = B_4 \sqcap \exists R.B_3$, $A_5(p)$, $B_4(p)$, $\not A_4(p)$, $\exists R.B_3(p)$, $a = p$, $b = q$, $p = q$	Consistente	Consistente

Escenario ontología $\mathcal{SHIQ}$ inconsistente.

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestConservativity4	$A \equiv B$, $C \subseteq \forall R.A$, $C \subseteq \forall R.B$, $A(p)$, $B(q)$, $C(r)$	Inconsistente	Inconsistente

Escenario ontología $\mathcal{SHIQM}$ consistente.

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestDifference14	$A \equiv X$, $X \subseteq B$, $owl : Thing(p)$, $a \neq b$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Consistente	Consistente
TestEquality2	$A_1(p)$, $A_2(p)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Consistente	Consistente
TestEquality4	$A_1(p)$, $\neg A_2(p)$, $P(q, a_1)$, $P(q, a_2)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Consistente	Consistente
TestEquality6	$A_1 \sqcap A_2 \sqcap \bot$, $A_1(p)$, $P(q, a_2)$, $P(q, a_1)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Consistente	Consistente
TestEquality8	$A_3 \sqcap A_4 \sqcap \bot$, $A_3(p)$, $P(a_1, a_3)$, $P(a_2, a_4)$, $PFunc$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$, $a_3 =_{\mathsf{m}} A_3$, $a_4 =_{\mathsf{m}} A_4$	Consistente	Consistente
TestEquality10	$A_1 \equiv \exists R.A_2$, $A_1 \equiv \forall R.\neg A_1$, $P(p, a_2)$, $P(p, a_1)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Consistente	Consistente
TestEquality12	$A_1 \equiv \exists R.A_2$, $A_2 \equiv \forall R.\neg A_1$, $A_1(q)$, $P(p, a_1)$, $P(p, a_2)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Consistente	Consistente
TestEquality14	$A_1(p)$, $(A_3 \sqcup \neg A_2)(p)$, $a_1 = a_2$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Consistente	Consistente
TestEquality15	$A_1(p)$, $(A_3 \sqcup \neg A_2)(p)$, $a_1 = a_2$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Consistente	Consistente
TestEquality16	$A_1 \subseteq A_2$, $owl : Thing \subseteq \leq 2.R$, $A_2 \sqcap A_3 \sqcap \bot$, $A_1(s)$, $A_2(t)$, $A_3(u)$, $R(p, a_2)$, $R(p, a_1)$, $R(p, a_3)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$, $a_3 =_{\mathsf{m}} A_3$	Consistente	Consistente
TestEquality18	$A_2 \subseteq A_3$, $owl : Thing \subseteq \leq 2.R$, $owl : Thing \subseteq \leq 1.S$, $A_1 \sqcap A_2 \sqcap \bot$, $A_1(s)$, $A_2(t)$, $A_3(u)$, $S(a_1, a_2)$, $S(a_2, a_3)$, $R(p, a_2)$, $R(p, a_3)$, $R(p, a_1)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$, $a_3 =_{\mathsf{m}} A_3$	Consistente	Consistente

Escenario ontología $\mathcal{SHIQM}$ inconsistente

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestCycles1	$A(a)$, $a =_{\mathsf{m}} A$	Inconsistente	Inconsistente
TestCycles2	$B(a)$, $A(b)$, $a =_{\mathsf{m}} B$, $b =_{\mathsf{m}} A$	Inconsistente	Inconsistente
TestCycles3	$B \subseteq A$, $B(a)$, $a =_{\mathsf{m}} A$	Inconsistente	Inconsistente
TestCycles5	$A \equiv (A_1 \sqcap A_2)$, $A \equiv (A_3 \sqcap \forall R.A_2)$, $A_3(a)$, $\exists R.A_2(a)$, $a =_{\mathsf{m}} A$	Inconsistente	Inconsistente
TestCycles7	$A \equiv (A_1 \sqcap A_2)$, $A \equiv (A_3 \sqcap \exists R.A_2)$, $B \equiv (B_3 \sqcap \forall R.B_2)$, $B \equiv (B_1 \sqcup B_2)$, $B_3(p)$, $\forall R.B_2(p)$, $A_3(q)$, $\exists R.A_2(q)$, $a = p$, $b = q$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestCycles8	$B \subseteq A$, $B(a_1)$, $P(p, a_1)$, $P(p, a)$, $PFUNC$, $a =_{\mathsf{m}} A$	Inconsistente	Inconsistente
TestCycles10	$A \equiv (A_1 \sqcap A_2)$, $X \equiv (A_3 \sqcap \exists R.A_2)$, $X \subseteq A$, $A_3(a)$, $\exists R.A_2(a)$, $a =_{\mathsf{m}} A$	Inconsistente	Inconsistente
TestCycles12	$A \equiv (A_1 \sqcap A_2)$, $B \equiv (B_1 \sqcup B_2)$, $X \equiv (A_3 \sqcap \exists R.A_2)$, $Y \equiv (B_3 \sqcap \forall R.B_2)$, $X \subseteq A$, $Y \subseteq B$, $B_3(p)$, $\forall R.B_2(p)$, $A_3(q)$, $\exists R.A_2(q)$, $a = p$, $b = q$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestCycles13	$A(b)$, $B(c)$, $a =_{\mathsf{m}} A$, $c =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestCycles14	$A \equiv A_1$, $B(a_1)$, $A(b)$, $a =_{\mathsf{m}} A$, $a_1 =_{\mathsf{m}} A_1$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestCycles15	$A \equiv X$, $A_1 \equiv X$, $B(a_1)$, $A(b)$, $a =_{\mathsf{m}} A$, $a_1 =_{\mathsf{m}} A_1$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestCycles16	$A_1 \equiv X$, $X \equiv (A_2 \sqcup A_3)$, $A \subseteq (A_2 \sqcup A_3)$, $A_2 \subseteq A$, $A_3 \subseteq A$, $B(a_1)$, $A(b)$, $a =_{\mathsf{m}} A$, $a_1 =_{\mathsf{m}} A_1$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestDifference2	$A_1 \equiv A_2$, $A_1(p)$, $a1 \neq a_2$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestDifference4	$A_1 \subseteq B$, $A_2 \subseteq (A_1 \sqcap C)$, $B \subseteq A_2$, $A_1(p)$, $a_2 \neq a_3$, $P(q, a_3)$, $P(q, a_1)$, $PFUNC$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestDifference8	$A_1 \equiv A_2$, $owl : Thing \subseteq (\leq 1.R.C \sqcap \leq 1.R.D)$, $owl : Thing \subseteq \geq 1.R.(C \sqcup D)$, $C(a_1)$, $D(a_2)$, $R(c, a_1)$, $R(c, a_2)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestDifference10	$X \equiv \exists R.A_3$, $X \subseteq ((A_1 \sqcup \neg A_2) \sqcap (A_2 \sqcup \neg A_1))$, $owl : Thing \subseteq \exists R.A_3$, $A_1(p)$, $A_2(q)$, $a_1 = a_2$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestDifference11	$X \equiv (\neg A_4 \sqcup \exists R.A_3)$, $A_4 \subseteq \exists R.A_3$, $X \subseteq ((A_1 \sqcup \neg A_2) \sqcap (A_2 \sqcup \neg A_1))$, $A_1(p)$, $A_2(q)$, $A_3(s)$, $a_1 \neq a_2$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestDifference13	$A \equiv (A_1 \sqcap (A_2 \sqcup \forall R.A_3))$, $A \equiv ((A_6 \sqcup A_7) \sqcap \exists S.A_4)$, $B \equiv (B_1 \sqcup B_2)$, $B \equiv (\exists R.B_3 \sqcup \forall R.B_3)$, $owl:Thing \equiv (A_6 \sqcup A_7)$, $owl:Thing \equiv \exists S.A_4$, $A(p)$, $a = p$, $b = q$, $p \neq q$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestEquality1	$A_1(p)$, $\neg A_2(p)$, $a =_{\mathsf{m}} A_1$, $a =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestEquality3	$A_1(p)$, $\neg A_2(p)$, $a_1 = a_2$, $a =_{\mathsf{m}} A_1$, $a =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestEquality5	$A_1(p)$, $\neg A_2(p)$, $P(q, a_1)$, $P(q, a_2)$, $PFUNC$, $a =_{\mathsf{m}} A_1$, $a =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestEquality7	$A_1 \neq A_2$, $A_1(p)$, $P(q, a_1)$, $P(q, a_2)$, $PFUNC$, $a =_{\mathsf{m}} A_1$, $a =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestEquality9	$A_1 \equiv (B \sqcup C)$, $A_2 \equiv (B \sqcup C)$, $A_3 \neq A_4$, $A_3(p)$, $P(a_1, a_3)$, $P(a_2, a_4)$, $FUNCP$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$, $a_3 =_{\mathsf{m}} A_3$, $a_4 =_{\mathsf{m}} A_4$	Inconsistente	Inconsistente
TestEquality11	$A_1 \equiv \exists R.A_2$, $A_1 \equiv \forall R.\neg A_1$, $P(p, a_2)$, $P(p, a_1)$, $FUNCP$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestEquality13	$A_1 \equiv \exists R.A_2$, $A_1 \equiv \forall R.\neg A_1$, $A_1(q)$, $P(p, a_2)$, $P(p, a_1)$, $FUNCP$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$	Inconsistente	Inconsistente
TestEquality17	$A_1 \equiv A_2$, $owl:Thing \subseteq\leq 2.R$, $owl:Thing \subseteq\leq 1.S$, $A_2 \neq A_3$, $A_1(s)$, $A_2(t)$, $A_3(u)$, $S(a_1, a_2)$, $S(a_2, a_3)$, $R(p, a_3)$, $R(p, a_1)$, $R(p, a_2)$, $a_1 =_{\mathsf{m}} A_1$, $a_2 =_{\mathsf{m}} A_2$, $a_3 =_{\mathsf{m}} A_3$	Inconsistente	Inconsistente
TestEquality19	$C \subseteq \exists R.A$, $C \subseteq \forall R.\neg B$, $A(p)$, $B(q)$, $C(r)$, $a = b$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestEquality21	$A \equiv (A_1 \sqcap (A_2 \sqcup \forall R.A_3))$, $A \equiv (A_4 \sqcup \neg A_5)$, $B \equiv (B_1 \sqcap \exists R.B_2)$, $B \equiv (B_4 \sqcap \exists R.B_3)$, $A_5(p)$, $B_4(p)$, $\neg A_4(p)$, $\exists R.B_3(p)$, $a = p$, $b = q$, $p = q$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$, $\neg A(p)$	Inconsistente	Inconsistente
TestEquality22	$A \equiv (A_1 \sqcup A_2)$, $A_1(p)$, $\neg B(p)$, $a = b$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente
TestEquality23	$A \equiv X$, $(X \sqcap \neg B)(p)$, $a = b$, $a =_{\mathsf{m}} A$, $b =_{\mathsf{m}} B$	Inconsistente	Inconsistente

Escenario ontología $\mathcal{SHIQM}^*$ consistente

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestCaseG1	$A_1(p)$, $\neg A_2(p)$, $P(q,a_1)$, $P(q,a_2)$, $R(a_1,a_2)$, $S(p,r)$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $MetaRule(R,S)$	Consistente	Consistente
TestCaseG2	$A_1 \neq A_2$, $A_1(p)$, $P(p,a_2)$, $P(p,a_1)$, $R(a_1,a_2)$, $S(p,r)$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $MetaRule(R,S)$	Consistente	Consistente
TestCaseG3	$A_1 \equiv \exists R.A_2$, $A_1 \equiv \forall R.\neg A_1$, $P(p,a_2)$, $P(p,a_1)$, $S(a_1,a_2)$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $MetaRule(R,S)$	Consistente	Consistente
TestCaseG4	$A_1(p)$, $(A_3 \sqcup \neg A_2)(p)$, $R(a_1,a_2)$, $S(p,r)$, $a_1 = a_2$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $MetaRule(R,S)$	Consistente	Consistente
TestCaseG5	$A_2 \subseteq A_3$, $owl:Thing \leq 2.R$, $owl:Thing \leq 1.S$, $A_1 \neq A_2$, $A_1(s)$, $A_2(t)$, $A_3(u)$, $S(a_1,a_2)$, $S(a_2,a_3)$, $R(p,a_2)$, $R(p,a_3)$, $R(p,a_1)$, $T(s,t)$, $T(s,u)$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $a_3 =_\mathsf{m} A_3$, $MetaRule(S,T)$	Consistente	Consistente

Escenario ontología $\mathcal{SHIQM}^*$ inconsistente

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestCaseH1	$A_1(p)$, $\neg A_2(p)$, $P(q,a_1)$, $P(q,a_2)$, $R(a_1,a_2)$, $S(p,r)$, $FUNCP$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $MetaRule(R,S)$	Inconsistente	Inconsistente
TestCaseH2	$A_1 \neq A_2$, $A_1(p)$, $P(p,a_2)$, $P(p,a_1)$, $R(a_1,a_2)$, $S(p,r)$, $FUNCP$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $MetaRule(R,S)$	Inconsistente	Inconsistente
TestCaseH3	$A_1 \equiv \exists R.A_2$, $A_2 \equiv \forall R.\neg A_1$, $A_1(q)$, $P(p,a_2)$, $P(p,a_1)$, $R(q,r)$, $S(a_1,a_2)$, $FUNCP$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $MetaRule(S,R)$	Inconsistente	Inconsistente
TestCaseH4	$A_1 \subseteq A_2$, $owl:Thing \subseteq \leq 2.R$, $A_2 \neq A_3$, $A_1(s)$, $A_2(t)$, $A_3(u)$, $R(p,a_2)$, $R(p,a_1)$, $R(p,a_3)$, $R(a_1,a_2)$, $S(s,u)$, $a_1 =_\mathsf{m} A_1$, $a_2 =_\mathsf{m} A_2$, $a_3 =_\mathsf{m} A_3$, $MetaRule(R,S)$	Inconsistente	Inconsistente
TestCaseH5	$A \equiv (A_1 \sqcup A_2)$, $A_1(p)$, $\neg B(p)$, $R(a,b)$, $S(p,r)$, $a = b$, $a =_\mathsf{m} A$, $b =_\mathsf{m} B$, $MetaRule(R,S)$	Inconsistente	Inconsistente

Escenario ontología $\mathcal{SHIQM}^*$ consistente con solo MetaRule

Nombre	Ontología	Resultado esperado	Resultado obtenido
HidrografiaMetaRule (Resumido)	$GravedadDePeligro(peligroso)$, $Roca(lasPipas)$, $seHalla(lasPipas, aFlorDeAguaAlDatumDeLaCarta)$, $conGravedad(lasPipas, peligroso)$, $MetaRule(aguasDeTipo, conGravedad)$	Consistente	Consistente

Escenario ontología $\mathcal{SHIQM}^*$ inconsistente con solo MetaRule

Nombre	Ontología	Resultado esperado	Resultado obtenido
TestCaseJ1	$owl : Thing \equiv (A \sqcup \neg B)$, $B \subseteq A$, $B(a)$, $\neg B(a)$, $owl : Thing(b)$, $owl : Thing(c)$, $\neg B(c)$, $owl : Thing(d)$, $R(a, b)$, $R(b, c)$, $R(d, a)$, $R(d, b)$, $SsubpropertyofR$, $MetaRule(R, S)$	Inconsistente	Inconsistente