INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

Unión De Ontologías Usando Propiedades Semánticas

TESIS
QUE PARA OBTENER EL GRADO DE
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

M. en C. Alma Delia Cuevas Rasgado

Director:

Dr. Adolfo Guzmán Arenas

Codirector:
Alexandre Guelboukh Kahn

México, D. F. **Noviembre de 2006**

SIP-14

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D. F. siendo las 14:00 horas del día 9 del mes de Noviembre de 2006 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis de grado titulada:

"UNIÓN DE ONTOLOGÍAS USANDO PROPIEDADES SEMÁNTICAS"

Presentada por el alumno:

CUEVAS	RASGADO	ALMA DELIA
Apellido paterno	materno	nombre(s)

Con registro: A 0 4 0 2 0 7

aspirante al grado de: ***DOCTOR EN CIENCIAS DE LA COMPUTACIÓN***

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***SU APROBACIÓN DE LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Presidente

DR. IGOR BOLSHAKOV MIRONOVA

Secretario

DR. SERGUEI LEVAHCKINE

Primer vocal
(Director de tesis)

DR. ADOLFO GUZMÁN ARENAS

Segundo vocal
(Co-director)

DR. ALEXANDRE GUELBOUKH KAHN

Tercer vocal

DR. GRIGORI SIDOROV

Suplente

DR. JESÚS MANUEL OLIVARES CEJA

EL PRESIDENTE DEL COLEGIO

DR. HUGO CÉSAR COYOTE ESTRADA

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION
DIRECCION

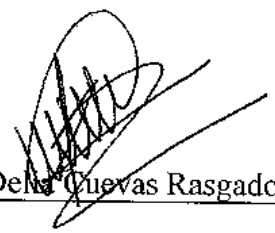
INSTITUTO POLITECNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESION DE DERECHOS

En la Ciudad de México, D. F. el día 24 del mes Noviembre del año 2006, el (la) que suscribe M. en C. Alma Delia Cuevas Rasgado alumno (a) del Programa de Doctorado en Ciencias de la Computación con número de registro A040207, adscrito a Centro de Investigación en Computación del IPN, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del Dr. Adolfo Guzmán Arenas (Director) y Dr. Alexander Gelbukh (Codirector) y cede los derechos del trabajo intitulado Unión de Ontologías Usando Propiedades Semánticas, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección almadeliacuevas@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.



Alma Delia Cuevas Rasgado

Nombre y firma

Resumen

El conocimiento de un ser humano se va acumulando conforme a lo que sucede en su entorno, las fuentes de información tienen un papel importante en este proceso; no se aprende de cero, inclusive un animal nace con conocimiento previo. El aprendizaje sucede agregando nuevos conceptos o asociándolos a los ya existentes. Aunque existe información del exterior que puede contradecir o confundir a un ser humano, éste cuenta con las herramientas que le permite resolverlo de alguna manera. A éste cúmulo de información se le puede llamar su **ontología**.

Las ontologías también se pueden estructurar y definir en las computadoras. Este trabajo se centra en la **unión de ontologías** entre computadoras, durante ésta unión pueden suceder los mismos casos que en una persona; la diferencia es que las máquinas carecen de sentido común y los desafíos son hacer la fusión de manera automática, que no se detenga ante los problemas (redundancias, distinto nivel de descripción…) que se presenten y que el resultado sea lo más cercano a la fusión natural de conocimiento del ser humano.

Existen trabajos [11, 13, 28 y 40] que realizan la unión de ontologías pero lo hacen de manera semiautomática, otros [25 y 34] unen ontologías expresadas en un lenguaje formal, pero son incapaces de unir ontologías mutuamente inconsistentes, como lo son la mayoría de las ontologías reales.

Este trabajo presenta un proceso de unión de ontologías de forma *automático* y *robusto*. *Automático* porque la computadora detecta y resuelve los problemas que se presentan durante el proceso de la unión y *Robusto* porque realiza la unión pese a que las ontologías son mutuamente inconsistentes o representan la información desde distintos ángulos. Se demuestra la eficiencia del algoritmo de fusión a través de varios ejemplos reales con documentos obtenidos de Internet cuyas ontologías se construyeron manualmente y se fusionaron de manera automática. Los resultados tuvieron un ligero margen de error en comparación con la fusión manual de un usuario experto en el tema del documento.

Abstract

A person's knowledge increases as more information is obtained from his environment; information sources play an important role in this process. One does not learn from zero, even an animal is born with innate knowledge. Learning happens by adding new concepts or linking them to already existing ones. Although information from outside sources can contradict or confound a person, he has the tools to solve somehow this problem. The knowledge accumulates in what we can call his **ontology.**

Ontologies can also be structured and defined in computers. This work focuses on **ontology fusion**; during the fusion the same cases arises as those occurring to a person. The difference is that machines have no common sense, so the challenges are to automate the fusion, to perform it in spite of problems (redundancies, descriptions at different detail levels), and that the result be as close as possible to the result obtained by a person.

Previous works [11, 13, 28 y 40] perform ontology fusion in a semi-automatic, computer-assisted manner. Others [25 y 34] fuse ontologies expressed in a formal notation, but are incapable of fusing mutually-inconsistent ontologies, as most of the real-life ontologies are.

This work presents a process for ontology merging which is *automatic* and *robust*. *Automatic* since the computer detects and solves the problems arising during the fusion and *robust* because merging occurs in spite of ontologies being mutually inconsistent and present information from different viewpoints. The efficiency of our algorithm is shown by converting by hand several documents in Internet to ontologies in our notation, and then automatically fusing them. Results show a slight error margin in comparison with manual fusion performed by an expert.

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1 Ubicación

Una persona recolecta lo que sabe de diferentes fuentes de información y para saber más, necesita *combinar* información de muchas fuentes.

Sin embargo, la información que viene de diferentes fuentes puede contener *repeticiones*, referencias a los mismos hechos pero formuladas de diferente manera, diferente nivel de detalle o incluso *contradicciones*.

Por ejemplo, una noticia de El Universal (Cultura) dice: - *El óleo "Jugadoras de cartas II" de Fernando Botero fue adquirido por 1,696 millones por un comprador privado estadounidense no identificado, en un remate de arte latinoamericano que superó los pronósticos de Sotheby's. El cuadro de Botero de 1.6 por 2 metros de 1989 encabezaba la colección de obras de reconocidos artistas* -, mientras que en una noticia de Yahoo México (Entretenimiento) dice: - *El cuadro de Botero "Jugadoras de cartas II", se vendió en un millón 700 mil dólares durante la subasta de arte latinoamericano de la Casa Sotheby's. Fue adquirido por un comprador estadounidense no identificado. Además de la obra de Botero, se vendieron algunas piezas de Francisco Zúñiga, Wilfredo Lam, Gunther Gerzo y Claudio Bravo* -.

En el ejemplo se observan algunas repeticiones (ambas noticias hablan de *la venta del cuadro de Botero "Jugadoras de cartas II" por un comprador estadounidense no identificado*), expresiones diferentes (*remate* vs. *subasta; óleo vs. cuadro*), diferente nivel de detalle (*Fernando Botero* vs. *Botero*), contradicciones (*1,696 millones* vs. *un millón 700 mil dólares*), así como la información mencionada en una noticia y no en otra (*El cuadro de Botero de 1.6 por 2 metros de 1989 encabezaba la colección de obras de reconocidos artistas* y *Además de la obra de Botero, se vendieron algunas piezas de Francisco Zúñiga, Wilfredo Lam, Gunther Gerzo y Claudio Bravo*). El lector que ha interpretado ambas noticias, tendrá en su mente una combinación *consistente* de la información que viene de ambas: que *el óleo o cuadro "Jugadoras de cartas II" de Fernando Botero fue adquirido por un comprador estadounidense y que además de éste, se vendieron los cuadros de otros artistas*. En cuanto a la cantidad que pagó, se puede deducir que El Universal (Cultura) es más específico (y representado en pesos mexicanos). Yahoo México (Entretenimiento) es más general (está redondeado y representado en dólares), aún así, la diferencia es pequeña y el margen de error es aceptable, quizá el lector debería consultar esta cantidad en otra fuente o creer más a El Universal (Cultura) o a Yahoo México (Entretenimiento) o dejar la pregunta abierta.

Mientras los lectores humanos cotidianamente resuelven este problema, no es tan fácil para la computadora. En el ambiente computacional, muchas personas

u organizaciones diferentes desarrollan enormes masas de conocimiento (tan sólo en la Web existen más de mil millones de páginas diferentes). El combinar manualmente esta información de manera consistente está fuera de las capacidades humanas; se tiene que hacer *automáticamente*.

Los métodos de combinación automática de conocimiento se desarrollan dentro de la lógica formal. Sin embargo, los paradigmas lógicos actuales no son *robustos* en cuanto a las contradicciones: al encontrar una contradicción, usualmente se detienen y no pueden continuar; no están diseñados para ignorar las contradicciones y proceder con otra información, ni resolverlas de alguna manera.

En esta tesis, se propone un método para combinar la información proveniente de diferentes fuentes de manera automática, consistente, sin repeticiones y de manera robusta a las contradicciones detectadas. Se limita a la combinación de sólo *dos* fuentes, ya que varias fuentes se pueden fusionar de manera consecutiva, combinando a la vez la fuente n + 1 con el resultado de la combinación de las primeras n fuentes.

El tratamiento automático de información requiere de una representación formal y precisa de la misma. Uno de los formalismos para tal representación, el que se usa en esta tesis, define la *ontología* como un conjunto de información formalmente estructurada de cierta manera, (más adelante se profundizará esta definición). La representación formal de la información en forma de ontologías permite usar los algoritmos precisos para su manipulación computacional. Una ontología se define usando un lenguaje formal, llamado *lenguaje de definición de ontologías*. Diferentes lenguajes de definición de ontologías proporcionan diversas facilidades para el manejo computacional de su contenido; en esta tesis se propone un lenguaje mejorado que facilita la tarea de la combinación de la información en forma de ontologías.

1.1.1 Marco teórico

Hay varias definiciones de ontología, la clásica es la de Gruber[1] que la define como una *representación o especificación formal* (una estructura sintáctica) *de una conceptualización* (conjunto de conceptos) *compartida* [18] (común a varias personas u otros programas).

En Wikipedia[2], *el término ontología en informática hace referencia al intento de formular un exhaustivo y riguroso esquema conceptual dentro de un dominio dado,*

[1] Tom Gruber es uno de los pioneros en la administración de información representada con ontologías, sabe que la definición de ontologías siempre ha causado controversia, doctor en Ciencias de la computación egresado de la Universidad de Massachussets, su área de investigación es administración y representación del conocimiento, conocimiento compartido, comunicación asistida por computadora, por citar algunos, ver: ksl-web.stanford.edu/people/gruber/index.html.

[2] Wikipedia es una enciclopedia libre escrita por voluntarios de todo el mundo, ver más en: es.wikipedia.org/wiki/Ayuda:Contenidos.

con la finalidad de facilitar la comunicación e intercambio de información entre diferentes sistemas.

La definición de *ontología* que se usará para los fines de esta tesis es la siguiente: *Es una representación formal de información de cierto tipo, simbolizada a través de un grafo dirigido donde cada vértice es un concepto que se encuentra enlazado a través de aristas que aquí se llaman relaciones*. Los conceptos y relaciones se identifican mediante etiquetas. Para incrementar el contenido de una ontología es preciso añadirle nuevos conceptos y relaciones una forma es: Adicionándole otra ontología, este proceso se puede realizar manualmente o de forma automática.

El proceso de *unir dos ontologías* consiste en tomar cada concepto de una ontología A y verificar si existe en otra ontología B si no existe se adiciona, si existe se completa su definición. La unión debe ser cuidadosa ya que se pueden presentar repeticiones, expresiones diferentes, distinto nivel de detalle y contradicciones, este trabajo de tesis proporciona soluciones a algunos de estos casos de tal manera que la ontología unida estará libre de relaciones redundantes, relaciones con contradicción o sin inconsistencias y exenta de conceptos repetidos.

Una *inconsistencia* surge entre dos relaciones. Una relación en una ontología A es inconsistente con otra relación en una ontología B si la etiqueta de la primera es incompatible con la etiqueta de la segunda.

Una *incompatibilidad* surge en el contenido (etiqueta o nombre) de las relaciones y los conceptos. Tiene que ver con el conjunto al que pertenece las relaciones y conceptos, se dice que hay compatibilidad cuando una etiqueta es subconjunto de otra. Por ejemplo si en una dice “mexicano” y en otra “oaxaqueño” éstas son compatibles, mientras que “mexicano” y “francés” no lo son (pertenecen al mismo conjunto persona, pero a diferentes conjuntos de nacionalidad), “mexicano” y “polinomio” tampoco. Las etiquetas incompatibles se conocen como relaciones contradictorias o inconsistentes.

El *motivo central de este trabajo* es la unión de ontologías, durante ésta unión pueden haber: *Agregación de conceptos y relaciones*; es decir, los conceptos y relaciones de A que no están en B y los conceptos y relaciones de B que no están en A, *Repetición de conceptos y relaciones*, esto es, los conceptos y relaciones de A que también están en B y contienen la misma información (éstos no se copian a la ontología C que contiene la fusión), *Relaciones contradictorias*, es decir, las relaciones de A que contradicen a las de B y las relaciones de B que contradicen a las de A (éstas tampoco se copian a la C).

Existen *trabajos que hacen unión de ontologías*, tales como: [11, 13, 28 y 40] pero lo hacen de manera semiautomática, otros [25 y 34] requieren ontologías con información ordenada y consistente, pero en la realidad la mayoría de las

ontologías en la Web no son ordenadas ni consistentes, [26] es un notable avance hacia la unión automática de ontologías, ya que prescinde de la ayuda del usuario durante el proceso de fusión, (en comparación a [11, 13, 28 y 40]) para el proceso de alineación[3] usa WordNet e Indexación de direcciones semántica latente, LSI [26] (Latent Semantic Indexing) para asociar las definiciones de los conceptos de las ontologías. En la unión usa mecanismos de Análisis formal de conceptos [16].

Se propone un proceso de unión de ontologías de forma automático y robusto. Un *proceso automático de unión de ontologías* es aquél donde la máquina detecta y resuelve los problemas que se presentan durante el proceso. Un *algoritmo robusto de unión de ontologías* realiza la unión pese a las ontologías desordenadas e inconsistentes.

1.2 El problema a resolver

Desarrollar un algoritmo para fusionar dos ontologías A y B, construyendo así una nueva ontología C = A ∪ B[4] que contendrá la información contenida en B que no está en A más la información en A que no está en B, además sin repeticiones ni contradicciones. Para las contradicciones, se plantea una forma de detectarlas y darles un tratamiento adecuado.

1.3 Justificación del problema y la solución

El algoritmo para la fusión de dos ontologías contribuye al desarrollo de ontologías mejores y más grandes. Más aún, permite para una tarea específica generar automáticamente ontologías a demanda, según las necesidades de la aplicación y los datos disponibles en ese momento. Eso llevará a las mejoras en todos los campos donde se usan las ontologías, lo que justifica el *área* de investigación (construcción de ontologías) seleccionada para esta tesis. Ejemplos de estos campos a los cuales indirectamente contribuye la presente investigación se dan en la sección 1.3.1.

La necesidad de unir dos ontologías para la obtención de otras ontologías más grandes se explica en la sección §1.3.2, lo que justifica el *tema* (unión de ontologías) seleccionado para esta tesis.

Las ventajas de la solución propuesta en cuanto al lenguaje de definición de ontologías y la fusión se exponen en la sección §1.3.3.

Existen maneras triviales de resolver el problema como se muestra en §1.3.4 y maneras formales y rigurosas (Apéndice A) que no funcionan en la vida real. La

[3] La alineación consiste en relacionar las ontologías fuente, buscando el concepto más similar de cada uno de los elementos de una ontología a otra, para saber el grago de similitud de ambas, previo a la fusión.

[4] El símbolo ∪ en la unión de ontologías hace más que una unión de conjuntos, es una unión "cuidadosa" considerando la semántica de los conceptos.

solución que presenta este trabajo no contempla ambas maneras (triviales y formales).

La diferencia entre el algoritmo propuesto y los algoritmos existentes que fusionan ontologías consiste en que es a la vez automático y robusto, las dos cualidades cuya combinación no se ha logrado en los trabajos anteriores, este método (automático y robusto) para unir ontologías y un lenguaje de definición de ontologías se explica en la sección §1.3.5, con lo cual se queda justificada la *solución* específica propuesta en este trabajo. Esta tesis se sustenta en otros trabajos de investigación que se explica en §1.3.6 en el cual también se da a conocer el grupo de investigación que se está ocupando de estos proyectos.

1.3.1 Aplicaciones de ontologías

La unión automática de ontologías tiene utilidad en algunas áreas importantes que a continuación se exponen:

Web Semántica: La necesidad en esta área es que las máquinas (programas, robots, "bots", "crawlers", "arañas", agentes) y no solamente los humanos, puedan entender[5] la gran cantidad de información en la red (Web). Por lo tanto, aquí se centra la importancia de deducir información para añadirla al conocimiento de una ontología. La unión automática de ontologías (que se presenta en este trabajo) parte de la ontología A que puede estar formada de un documento en la Web (actualmente este paso se hace a mano) y tomar otro documento en la Web que será la ontología B uniéndolas con cuidado, devolviendo una ontología C con información de A y B muy útil para responder preguntas de un usuario.

Por ejemplo, con la ayuda del algoritmo OM resultado de esta tesis y de otro algoritmo analizador[69] que convierte documentos de texto a ontologías, sería posible que una araña o "crawler" (como las que posee BiblioDigital [20]) visitara páginas relevantes de Internet (quizá localizadas inicialmente por Google[6]) y recopilara la información sobre cierto tema, por ejemplo *Historia de la computación en México* y la entregará en una ontología consistente y sin redundancia. Esta araña sería "el recolector automático de los conocimientos que la Web contiene sobre cierto tema *t*".

Actualmente, ya se cuenta con Google una especie de recolector automático de información sobre algún tema *t*, pero la interpretación o fusión de la información contenida en los documentos recolectados queda a cargo del lector o usuario. El algoritmo OM automatiza esta interpretación y fusión.

[5] Por "entender" se quiere decir: procesar, aprovechar, hacer deducciones inteligentes. Pragmáticamente se define que un programa "entiende" cierto texto, información o situación, cuando la puede aprovechar para avanzar en sus metas y objetivos.
[6] Empresa cuyo principal producto es el motor de búsqueda del mismo nombre.

Comercio electrónico: Un agente A puede enriquecer el conocimiento de su ontología A (conocer sinónimos de sus productos, tomar nuevas características o propiedades de sus productos) uniendo su ontología A con la de otros agentes B con la ontología B, C con C, etc. que vendan el mismo producto, entonces logra hacerse "entender mejor" con otro agente y su ontología X que compre los productos que vende el agente A.

Recuperación de información: Una ontología A puede enriquecer sus nodos con otras ontologías B, C, etc. y responder preguntas complejas que se deriven de A y de las ontologías que haya copiado. Por ejemplo si en A dice que el delincuente es hombre, B dice que es de raza negra y C dice que es joven entonces la extracción del conocimiento resulta ser una información interesante.

Educación virtual: Un libro virtual A puede unir su ontología A (donde los nodos son temas discutidos en el libro) con la ontología de otro libro virtual B que trata del mismo tema o similares. Esta ontología enriquecida será mejor aprovechada por los estudiantes que aprenden de A. También A puede unir su ontología con una ontología "predecesora", por ejemplo, el libro A que trata de *Finanzas Internacionales* puede encontrar útil fusionar su ontología con B que trata de *Economía* o el libro A que trata de la *Administración en los negocios* encontrará útil fusionar su conocimiento con la ontología *Gerencia*. Esto ayudará a los estudiantes a recapitular conceptos previos.

Bases de datos heterogéneas: Se puede obtener una ontología A de una base de datos, definiendo cada entidad como un nodo y sus atributos como relaciones y unirla con otras ontologías B, C, etc. resultando bases de datos similares. Tal fusión debe hacerse cuidadosamente, verificando los dominios y tipos de datos de los atributos, la sinonimia, evitando información redundante y contradictoria. El resultado de la fusión será una base de datos más útil a cualquier corporación que desea administrar y explotar mejor su información.

Por ejemplo, la ontología de WalMart[7] se puede fusionar con ontologías de otras firmas, para generalizar y complementar la descripción de sus productos y para encontrar los productos similares que Walmart no vende pero lo hacen otras firmas.

1.3.2 Relevancia

Unir dos ontologías de forma automática y consistente requiere de un procesamiento semántico muy fuerte. La idea es: Tomar ontologías y tratar de hacer una tercera que contenga la información de ambas, eliminando lo redundante y lo contradictorio.

Si una máquina de búsqueda o "araña" (poseedora de su ontología local) al visitar una página exógena, pudiese deducir cómo se relaciona su ontología a la

[7] Cadena de hipermercados Americana fundada por Sam Walton en 1962, ver más: en.wikipedia.org/wiki/Wal-Mart.

de la página visitada, entonces podría entenderla y copiarla a la suya. Este es el camino que siguen mapeadores de un esquema (de bases de datos) a otro [36], cuando buscan integrar bases de datos que fueron construidas independientemente [5].

1.3.3 Ventajas de la solución propuesta

La unión de ontologías se realiza de forma automática, no hay intervención del usuario en el proceso, otros trabajos en la Web tales como: [11, 13, 28 y 40] necesitan de una persona que resuelve los problemas generados en la fusión, ver §2.4.1 y otros como en [25 y 34] necesitan ontologías bien definidas y ordenadas para realizar la unión.

1.3.3.1. Ventajas de la unión de ontologías

1. Un algoritmo de unión de ontologías que realice el proceso de manera automática (eliminando las redundancias, resolviendo algunas inconsistencias y los datos imprecisos, por ejemplo: "*Nació en México*" y "*Nació en Guelatao*"), el usuario de la máquina donde este algoritmo trabaje solo espera el resultado de la unión.

2. Un algoritmo que fusione ontologías sin un orden y consistencia perfectos, aunque se sabe que si las ontologías son ordenadas y consistentes se obtendrá mejores resultados.

3. Aplicando este algoritmo a las máquinas es como (ambiciosamente se puede decir) hacer "aprendizaje automático". La capacidad de aprender de las personas es finita (también en las máquinas) y limitada (aquí las máquinas pueden concentrar más información procesando muchos documentos) por esta razón el algoritmo de fusión de este trabajo aprovechará esta capacidad de las máquinas.

1.3.3.2. Ventajas del lenguaje de diseño de ontologías

1. Considera un tipo de relación llamada Partición, ver §3.4.2 que no contemplan los lenguajes actuales [3, 6, 10, 17, 27 y 39] expuestos en la Web.

2. Se pueden representar las relaciones como conceptos, aquí una relación es también un concepto, por lo tanto una relación tendrá también otras relaciones, esto le da más semántica a las relaciones.

1.3.4 Formas triviales de unir ontologías

1. Una unión "trivial" sería tomar A y B colgándolas debajo de una raíz ficticia en C, pero si las ontologías tratan del mismo tema habría duplicidad de información (un nodo que ya estaba en A y aparece en B se uniría a través del nodo raíz en C).

2. Una unión “lenta” sería tomar A y B y unirlas a mano, la velocidad y la consistencia de la unión en la ontología resultante irá en función a que el usuario domine o no el tema (quien sepa menos de un tema unirá diferente a aquel que sabe más).

3. Una unión “dirigida” sería tomar A y B uniéndolas mediante un programa de computadora, donde éste (el programa) se detendrá cada vez que encuentre un problema y le preguntará al usuario que hacer (el resultado de la unión será diferente de acuerdo al criterio de solución del usuario, una persona con mejor capacidad de solución hará mejor la unión).

El algoritmo de unión que se expone en §3.2 no es trivial porque hay que dar una solución a cada problema que se presente durante la unión para que no se detenga en el proceso y además el resultado cercano a como lo harían los humanos.

1.3.5 Estado del arte de los lenguajes de definición y de la unión de ontologías

Para representar el conocimiento en una ontología se requiere de un lenguaje de definición, actualmente existen muchos lenguajes en la Web, tales como [3, 6, 10, 17, 27 y 39] pero aún hay deficiencias en la representación del conocimiento (ver §2.3.2 por ejemplo, las relaciones no pueden ser conceptos, clases[8], conjuntos, objetos, instancias. Tampoco existen las relaciones de tipo partición[9] [31] lo más cercano son las listas con restricciones pero no cubren la semántica de las particiones.

Hay trabajos [11, 13, 28 y 40] que realizan la unión de ontologías y lo hacen de manera semiautomática, es decir, el sistema que realiza la unión presenta una serie de sugerencias a un usuario y éste último indica (al primero) las sugerencias a realizar y así sucesivamente hasta acabar la unión, se han hecho pruebas interesantes en cada uno de ellos, pero la unión depende del usuario.

Otros trabajos [25 y 34] requieren de ontologías perfectas y matemáticamente definidas y concentran la unión en una lattice[10], un usuario realiza la poda del lattice para usar la ontología resultante, estos trabajos usan interesantes técnicas de procesamiento de lenguaje natural y Análisis Formal de Conceptos[11], pero es

[8] Las clases y superclases son los conjuntos finitos, no vacíos donde se ubica un nodo en la ontología. Por ejemplo, el nodo Río pertenece a la clase o conjunto de Ríos de un espacio geográfico.

[9] Una partición es una colección de subconjuntos, tales que cualesquiera dos elementos de la colección son mutuamente exclusivos y todos ellos colectivamente exhaustivos. Por ejemplo, la partición Edad que contiene los subconjuntos Infante, Puberto, Adolescente, Joven, Maduro y Anciano, donde una instancia de un subconjunto no puede pertenecer a otro a la vez y todos los subconjuntos forman la partición Edad.

[10] Conjunto de objetos representados gráficamente, donde existe un orden definido entre ellos, en matemática es un conjunto parcialmente ordenado en el cual todo subconjunto finito no vacío tiene un supremo y un ínfimo (ver: es.wikipedia.org).

[11] Análisis Formal de Conceptos, es un método para análisis de datos basados en la teoría de Lattices y Calculo Proposicional, se ubica en la exploración de símbolos de conocimientos (conceptos) contenidos en un contexto formal, tal como una base de datos, ontología (ver mas en: es.wikipedia.org).

minucioso el proceso de comprobar el resultado de la unión y hay pocas ontologías en la Web que son perfectas y matemáticamente definidas.

1.3.6 Pertinencia

OM es parte del proyecto de un grupo de investigación del CIC-IPN en México, que tiene la labor de aplicar la Inteligencia Artificial para representar y manipular el conocimiento, OM se sustenta en algunos trabajos de este equipo, tales como:

- El algoritmo COM [22] toma un concepto C_A en una ontología A y halla el concepto más similar C_B en una ontología B.
- El algoritmo de la teoría de la confusión [21] obtiene el grado de la confusión de usar una palabra *r* en lugar de otra *s* y la confusión de usar *s* en lugar de *r*.

En la Figura 1.1 se presenta gráficamente la interacción de OM con estos trabajos del grupo.

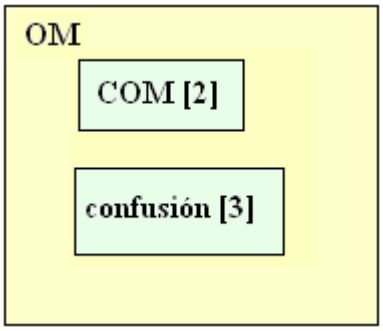


Figura 1.1 El contexto de OM y los trabajos en los que se apoya

Existen otros trabajos del grupo en los cuales OM se puede sustentar en el futuro, estos se describen en §5.1.2, §5.1.3 y §5.1.4.

1.4 Aportaciones

El trabajo supone que existen dos ontologías A y B que habrá que fusionar. Las principales aportaciones de esta tesis son:

1. Un lenguaje para definición de ontologías §3.4.1, con las características que representan las ontologías usadas en este trabajo.
2. Un algoritmo para la fusión de dos ontologías §3.2, que realiza el proceso de forma automática y robusta.
3. Algunos casos §3.4 aplicados al algoritmo COM [7] que relaciona las ontologías a unir, estos casos contribuyen a darle más precisión en su tarea.
4. Un caso §3.8.3.5 aplicado al algoritmo de la teoría de la confusión [21] para apoyar a la solución de relaciones contradictorias o inconsistentes.

1.5 Estructura de la tesis

El capítulo I da una introducción y motivación. En el capítulo II presenta los trabajos relacionados a los lenguajes de definición y unión de ontologías. El capítulo III presenta varios algoritmos que cubren los casos importantes de la fusión explicados mediante ejemplos. El capítulo IV presenta los resultados de las pruebas con documentos traducidos (manualmente) a ontologías. Las conclusiones y casos que se recomiendan para trabajos futuros se presentan en el capítulo V y VI. En el Anexo A se presenta el modelado matemático del problema y la solución.

2. Trabajos previos

En este capítulo se presenta una breve descripción de algunos lenguajes de definición de ontologías, se citan las ventajas y las desventajas para obtener un análisis comparativo de ellos. Asimismo, se describen los trabajos de unión de ontologías con sus algoritmos expuestos de manera general y finalmente un análisis comparativo de estos.

2.1 Lenguajes de definición de ontologías

Hasta ahora, las ontologías se han propuesto para resolver algunos problemas en la Web Semántica[12] (esto es, inducir a la máquina a un mejor *entendimiento* del lenguaje natural); para ello, se usan los lenguajes de definición de ontologías, para estructurar la información. El propósito es que estos lenguajes expresen la semántica que se requiere. Se mostrará que faltan algunos detalles en los lenguajes actuales para cumplir con este propósito.

Para lograr un algoritmo de unión robusto, se requiere de una mejor representación para las ontologías, una de las contribuciones de esta tesis es diseñar una nueva notación, un extracto de éste lenguaje se presenta al final del apartado.

2.1.1 DAML+OIL

DAML + OIL, de las siglas DARPA (Agent Markup Language + Ontology Inference Layer) [6]. Es un lenguaje de etiquetas que proporciona semántica a los recursos de la Web.

El lenguaje fue construido bajo el estándar W3C (World Wide Web Consortium) y el esquema RDF (Resource Description Framework). Proporciona una estructura de la información normalmente vista en lenguajes basados en marcos[13]. Inició del lenguaje original DAML-ONT (DAML Ontology) en octubre de 2000 con el propósito de combinar muchos componentes del lenguaje de OIL (Ontology Inference Layer).

Como una propuesta para la representación de ontologías, OIL usa semánticas formales y razonamiento lógico en la representación de los datos, así como la descripción del significado de términos y deducción de información implícita.

[12] La Web Semántica es una visión o ideal donde las máquinas (programas, robots, "crawlers", "arañas", agentes) y no solamente los humanos, puedan entender la gran cantidad de información en la Web.

[13] Estructura de datos que incluye clases, relaciones, instancias, etc. para representar una situación hipotética, ver: elies.resiris.es/elies19/cap441.html.

OIL tiene una sintaxis bien definida en XML basada en DTD[14] y XML Schema[15], por otra parte OIL es una extensión de RDF[16] y RDFS[17] proporciona dos importantes contribuciones:

1. Una sintaxis estandarizada para representar ontologías y
2. Un conjunto de elementos de modelado, tales como: las relaciones *instancia de* y *subclase de*, que le dan un poder expresivo a la ontología, haciendo viable la inferencia.

Pese a las implementaciones de funcionalidad de OIL, se han definido extensiones en cooperación con DAML+OIL.

Ventajas

El lenguaje está apropiadamente asentado sobre lenguajes de la Web, como: XML Squema y RDF. Ofrece diferentes niveles de complejidad. En cuanto al modelado, OIL refleja cierto consenso entre áreas como: Descripción lógica y Sistemas basados en marcos.

Desventajas

Razonamiento por defecto. Proporciona el mecanismo de heredar valores de superclases, pero tales valores no pueden ser reescritos en una especialización.

Reglas y axiomas. Se pueden expresar solo un número fijo de propiedades algebraicas de ranuras. No existe la facilidad de describir axiomas que cumplan para determinados elementos de la ontología.

Propiedades algebraicas. La carencia de un lenguaje para expresar axiomas podría ser parcialmente compensada por la adición de un conjunto de propiedades que pueden ser especificadas para relaciones en OIL.

2.1.2 RDF/ XML

Resource Description Framework [27] está estrechamente relacionado con XML[18] y por tanto, especificado para usarse con XML. Se usa principalmente para representar Meta datos.

Con RDF y XML es posible describir Ontologías que pueden usarse en la Web semántica. XML es usado para etiquetar el dato, por tanto RDF provee de los datos a las etiquetas [6].

---

[14] DTD de las siglas Document Type Definition es una definición en un documento SGML o XML que especifíca restricciones en la estructura del mismo, ver: es.wikipedia.org/wiki/DTD.

[15] Alternativa basada en XML para DTDs, describe la estructura de un documento XML, ver: w3shools.com/schema/default.asp.

[16] RDF de las siglas Resource Description Framework es un marco de trabajo para metadatos en la WWW, desarrollado por la W3C (World Wide Web Consortium), ver: es.wikipedia.org/wiki/Marco_de_descripción_de_recursos.

[17] RDF Schema, especificación de cómo usar RDF para definir vocabularios RDF, ver: www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[18] XML de las siglas eXtensible Markup Language es un metalenguaje extensible de etiquetas desarrollado por el W3C, ver: es.wikipedia.org/wiki/XML.

RDF es un medio para agregar semántica a un documento. El modelo de datos de RDF provee tres tipos de objetos:

1. *recursos*: es una entidad que puede ser referenciada por un Identificador Único de Recursos (URI)[19].
2. *propiedades*: define una relación binaria entre los recursos y los valores atómicos de los tipos de datos primitivos provistos por XML.
3. *sentencias*: especifica un valor en una propiedad para un determinado recurso.

Si XML también provee un mecanismo para la representación de la información, ¿Porqué RDF?, la razón es que provee un modo estándar de representar metadatos en XML, usando directamente XML podrían obtenerse varias representaciones diferentes. [35].

Ventajas

Proporciona términos consistentes a los metadatos y aporta una descripción semántica rica.

Desventajas

Proporciona un soporte limitado para la especificación del uso de restricciones locales, es decir, restricciones de estructura, de cardinalidad y de tipos de datos.

2.1.3 OWL

Ontology Web Language [3] se diseñó para usarse con RDF/XML solo que OWL permite incrementar la interpretación de la máquina hacia una semántica formal.

OWL cuenta con todas las bondades de RDF y ha sido adoptado por el consorcio (W3C) como el lenguaje estándar para ontologías en la Web semántica[20].

OWL es un lenguaje usado para describir las clases, propiedades y sus relaciones con otras clases y aplicaciones Web.

Para dar mayor funcionalidad al diseño de ontologías, el lenguaje proporciona tres sublenguajes:

[19] URI de las siglas Uniform Resource Identifier, también usado como URL, es un texto corto que identifica unívocamente cualquier recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.) accesible en una red, ver: es.wikipedia.org/wiki/URI.

[20] Web semántica o semantic web es la idea de añadir metadatos semánticos a la WWW. Esas informaciones adicionales –describiendo el contenido, el significado y la relación de los datos- deben ser dadas en forma formal, así que es posible evaluarlas automáticamente por máquinas. El destino es mejorar la WWW para ampliar la interoperabilidad entre los sistemas informáticos y reducir la mediación de operadores humanos necesaria, ver: es.wikipedia.org/wiki/Web_semántica.

1. OWL Lite: creado para usuarios que requieren de la representación de una jerarquía sencilla, con restricciones también sencillas, pero proporciona una rápida migración hacia otras taxonomías[21].
2. OWL DL: es para aquellos usuarios que quieren más expresividad en el lenguaje para los sistemas de razonamiento lógico ya que cuenta con mecanismos de descripción lógica[22] (de ahí el nombre) sin embargo cuenta con otras restricciones por ejemplo; la separación de tipos (una clase no puede ser a su vez una instancia o propiedad, una propiedad no puede ser al mismo tiempo una instancia o clase) y
3. OWL Full: es para dar mayor expresividad y el uso libre de la sintaxis de RDF en la cual una clase simultáneamente se considera como una colección de instancias y cuenta con otras mejoras en el manejo de las funciones, incrementando el significado de los elementos definidos en la ontología.
   Estos sub lenguajes incluyen todos los elementos constructores de OWL.

Ventajas

Concentra las ventajas de DAML+OIL y las de RDF/XML, además de ser reconocido por el consorcio W3C como el estándar para el lenguaje de definición de Ontologías en la Web.

Desventajas

No obstante a las ventajas y mejoras que implementa OWL junto con el lenguaje RDF, dista aun de cubrir con el objetivo tan ambicioso por el que ha surgido, ya que la distancia entre la concepción usada por los humanos y el análisis de datos usado por las computadoras no ha sido resuelta en su totalidad.

2.1.4 KIF

Knowledge Interchange Format [17] es un lenguaje diseñado para el intercambio de conocimientos entre distintos sistemas de cómputo (creado por distintos programadores en distintos tiempos y diferentes lenguajes, etc.).

KIF no pretende ser un lenguaje único para interactuar con los usuarios (humanos, aunque puede usarse para ese propósito pero, muchos sistemas de cómputos pueden interactuar con sus usuarios en cualquier forma que sea más apropiada a sus aplicaciones (por ejemplo las gráficas conceptuales, Prolog, lenguaje natural, etc.); el propósito es interactuar con otras aplicaciones, agentes, etc.

[21] La taxonomía es la ciencia y el arte de la clasificación. Un ejemplo es la taxonomía biológica, esto es, la clasificación de los seres vivos que describen jerárquicamente las relaciones de parentesco y similitud entre los organismos, ver: es.wikipedia.org/wiki/Taxonomía.

[22] Las Lógicas de Descripción (DL por las siglas description Logics) son una familia de lenguajes de representación del conocimiento que pueden ser usados para representar conocimiento terminológico de un dominio de aplicación de una forma estructurada y formalmente bien comprendida, ver es.wikipedia.org/wiki/Lógica_de_descripción.

KIF tampoco pretende ser la representación interna para el conocimiento dentro de los sistemas de cómputo (aunque este lenguaje puede ser usado muy bien para este propósito).

Cuando una computadora lee una base de conocimientos en KIF, convierte el dato en su propia estructura de datos interna (estructuras apuntadores, arreglos, etc.). En realidad, toda la computación para la obtención de la información está hecha usando esas formas internas y cuando el sistema se comunica con otro sistema relaciona la estructura de datos interna al formato KIF.

KIF tiene semánticas declarativas[23]. Es posible entender el significado de las expresiones del lenguaje sin necesidad de usar un interprete para disponer de esas expresiones, esta es la diferencia entre KIF y otros lenguajes que son basados en interpretes específicos, como lo es Prolog.

Las expresiones en KIF se forman acorde a las reglas, los tres tipos de expresiones son:

1. Los *términos* son usados para definir los límites y las cardinalidades
2. Las *sentencias* son usadas para expresar los hechos acerca del mundo y
3. Las *definiciones* son usadas para denotar los objetos del mundo descrito.

Ventajas

KIF es un lenguaje fuertemente expresivo así como la gran habilidad para expresar meta conocimientos, esto es, escribir sentencias de sentencias.

Desventajas

KIF complica el trabajo de construir sistemas. Los resultados de los sistemas tienden a ser pesados (es decir, son extensos y en algunos casos menos eficientes que los sistemas que emplean lenguajes restrictivos[24]).

2.1.5 OCML

Operational Concept Modelling Language [10] es un lenguaje que sirve para la construcción de modelos de conocimientos; permitiendo la especificación y operación de funciones, relaciones, clases, instancias y reglas.

OCML incluye mecanismos para definir ontologías y métodos para resolver problemas en este rubro, como una aportación importante en el área de modelado de conocimientos.

---

[23] Las semánticas declarativas es una representación del conocimiento declarativo contenido en un típico sistema de aplicación basados en el conocimiento, otra definición: Estructura que permite la traducción semi-automática dentro y fuera de un típico lenguaje de representación, ver: 72.14.203.104/search?q=cache:yOWO_OLi8m8J:solar6.ingenieria.uatx.mx/~cperez /cperez /LECTURAS/DIAPOSITIVAS.html.

[24] Un lenguaje restrictivo por ejemplo puede incorporar modelos de representación (UML) como un estándar para especificar detalles adicionales o precisar detalles en la estructura de los modelos, ver: es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/multiple-html/x65.html.

OCML es usado para proporcionar soporte en el modelado a aplicaciones de áreas de administración del conocimiento, diseño de ontologías, comercio electrónico y sistemas basados en conocimiento.

OCML está sustentado por una gran biblioteca de modelos que se pueden reutilizar, que son recursos útiles en la comunidad de modelado de conocimientos.

OCML Soporta la especificación de tres tipos de constructores:

1. *Término funcional*: especifica un objeto en el dominio actual de investigación. Puede ser una constante, variable, cadena o función.
2. *Término de control*: el modelado del problema encierra más que la creación de estructuras y descripción de entidades en el mundo. Los términos de control se requieren para especificar acciones y describen el orden en que serán realizados.
3. *Expresiones lógicas*: OCML también provee de una máquina para especificar expresiones lógicas.

En particular, OCML proporciona los mecanismos para definir las relaciones, funciones, clases, instancias, reglas y procedimientos.

Las relaciones en OCML permiten que el usuario defina etiquetas n-arias (múltiples) entre las entidades.

En OCML se puede representar una semántica formal usando las relaciones opcionales dados en términos de homónimos[25]. Se apoya del lenguaje Lisp en las definiciones de sus clases, funciones, etc. Contempla, la especificación y uso de ranuras y rangos tal y como sucede en los lenguajes basados en marcos. Utiliza el mecanismo de herencia de clases a subclases e instancias.

Ventajas

1. Combina las características de modelado y representación del conocimiento que juntos proporcionan un formalismo híbrido.
2. Proporciona soporte para el modelado orientado a objetos y relacional.
3. Usa expresiones lógicas.
4. Permite a los usuarios definir sus propias reglas, que comprende de cero o mas antecedentes y uno o mas consecuentes.
5. Se realizan mapeos de conocimientos, tales como mapeos de instancias y de relaciones.
6. Proporciona facilidades para definir ontologías. En este proceso de definición puede identificar conflictos tales como: redefinición de conceptos.

[25] Son palabras que sencillamente son similares pero tienen un significado parcial o totalmente diferente, ver: es.wikipedia.org/wiki/Homónimo.

Por defecto, todas las ontologías se construyen en base a las especificaciones para ontologías de OCML. Esta incluye doce subontologías que proporcionan una plataforma de modelado enriquecido a partir del cual se puedan construir otras ontologías y/o resolver problemas de modelado. Puede ser usado como un interpretador para definiciones de Ontolingua [10].

Desventajas

Se dificulta la inclusión de características particulares del usuario.

2.1.6 OKBC

Open Knowledge Base Connectivity [44] es un protocolo para el acceso de bases de conocimientos (KB) almacenados en Sistemas de Representación de conocimientos (KRS), por ejemplo una base de datos orientada a objetos.

OKBC proporciona un conjunto de operaciones para la interfaz de un KRS. La capa de interfaz permite que una aplicación tenga alguna independencia desde su KRS y habilita el desarrollo de herramientas genéricas (por ejemplo, mostradores gráficos y editores) que operan en muchos KRS.

La implementación de OKBC existe para varios lenguajes de programación, incluyendo a Java, C (solo implementación de cliente) y Lisp que proveen acceso a KB de manera local y a nivel de red.

OKBC se enfoca en funciones tales como: operaciones con marcos, ranuras, facetas, verificación de herencia y restricciones de ranuras. Especifica un modelo de conocimientos de KRS (con KB, clases, instancias, ranuras y facetas). Así como un conjunto de operaciones basadas en este modelo (por ejemplo, encontrar un marco más similar a otro, enumerar la ranura de un marco, borrar un marco). Hay aplicaciones que usan estas operaciones para acceder y modificar conocimiento almacenado en un KRS bajo el protocolo OKBC.

Ventajas

La meta de OKBC es servir como interfaz para diferentes KRS y al igual que otros protocolos, tiene su propia terminología. Esto es, pretende ser capaz de interpretar la semántica de diversos KRS.

Desventajas

Al tratar de llevar a cabo su meta al cien por ciento, le afecta la falta de acuerdos en el manejo de las terminologías en el campo de la representación del conocimiento, porque los investigadores usan diferentes términos que significan la misma cosa y usan el mismo término para cosas diferentes. Esto es, términos diferentes para las clases, instancias, relaciones entre conceptos e instancias, etc.

2.1.7 El lenguaje OM

OM de las siglas Ontology Merging es un lenguaje desarrollado en esta tesis, con la finalidad de diseñar ontologías con conceptos y relaciones que proporcionan más semántica a las operaciones de búsqueda de conocimiento.

La estructura semántica de las ontologías definidas en este lenguaje, es a través de un conjunto de etiquetas (como en XML) que identifican el concepto y sus relaciones, por ejemplo, <*concept*> que indica el nombre del concepto. Esta etiqueta permite el anidamiento de conceptos, <*language*> que representa el lenguaje del concepto, <*word*> donde se encuentran las palabras que definen al concepto y <*relation*>que representa el tipo de relación que conecta al concepto.

Las relaciones en OM pueden ser implícitas y explícitas:

Implícitas: son las relaciones expresadas por el anidamiento, el concepto externo se reconoce como antecesor mientras que el interno como sucesor. Estas relaciones son: *member*, *part*, *part** y *subset*.

Explícitas: son las que se encuentran definidas entre las etiquetas <*relation*></*relation*>, puede ser una actividad (eats), propiedad (color) o atributo del concepto. No se permiten relaciones anidadas.

Existe otro tipo de relación llamada Partición, esta se diferenciará con la palabra "*Partition*" y contiene una estructura diferente a las relaciones explícitas. Una partición es un conjunto cuyos subconjutos están bien definidos por un rango o intervalo. Cada instancia de un subconjunto no puede ser instancia de otro dentro de la misma partición, es decir cada subconjunto de la partición son mutuamente exclusivos y colectivamente exhaustivos.

Ventajas

El lenguaje permite que un concepto pueda tener relaciones n-arias (múltiples).
Una relación puede ser un concepto.
Se permiten las relaciones de tipo partición.
Proporciona más semántica a la interpretación de los conceptos de la ontología al usar sinónimos en la definición de los conceptos.

Desventajas

Pese a que cuenta con su propio lenguaje de definición de ontologías y su propia estructura de datos hace falta más pruebas para demostrar la riqueza de su representación.

2.1.8 Análisis comparativo de los lenguajes de definición de ontologías expuestos

El lenguaje OM no es compatible con los demás lenguajes porque es más rico en su representación (al considerar las particiones y que las relaciones pueden ser

también nodos, por citar algunos) por tanto, se puede convertir una ontología diseñada en los otros lenguajes a OM en este caso habría ventajas de OM que no se aprovecharían en el diseño. Por otro lado, no se puede convertir un diseño en OM a los lenguajes anteriores porque habría elementos que no se podrían representar en la notación de los otros lenguajes. Solo OM y OCML representan relaciones n-arias.

Se han comparado las características generales de los cinco lenguajes para definición de ontologías, con los resultados que a continuación se presentan en la Tabla **2.1**.

Tabla 2.1 Características generales de los lenguajes para definir ontologías, donde la diferencia importante es que OM es el único lenguaje que representa las particiones

Características	DAML+OIL	RDF/XML	OWL	KIF	OCML	OKBC	OM
Particiones	-	-	-	-	-	-	√
Conceptos, Relaciones, Instancias, Rangos, Marcos	Clases, Slots, Instancias, Expresión	Recurso, Propiedad, Sentencia	√	√	√	Clases, Slots, Facetas, Herencia	Conceptos, Relaciones, Instancias, Rangos,
Basados en marcos con semánticas formales y razonamiento lógico	√	No describe la semántica entre conceptos y relaciones más allá de los mecanismos heredados	√	√	√	√	Basados en marcos con semánticas formales
Representación de metadatos	X	√	√	√	√	√	√
Formato para interactuar con los usuarios	√	√	√	X	√	√	√

2.2 Métodos actuales de mapeo y unión de ontologías

Se presentan unos métodos que realizan la unión de ontologías, los algoritmos que se dedican a este proceso, finalmente se muestra una conclusión.

2.2.1 PROMPT

El algoritmo PROMPT, [13] cuyo antecesor es SMART, [13] ha sido desarrollado en el área de Informática Médica de la Universidad Stanford. Este algoritmo está implementado en la herramienta interactiva Protégé 2000 [43] que se ocupa de la adquisición de conocimientos y del diseño de ontologías, cuya base de conocimientos es compatible con OKBC (Open Knowledge-Base Connectivity) [38].

El algoritmo PROMPT realiza la unión de ontologías previamente cargadas en el editor de ontologías de Protégé 2000.

Las operaciones que PROMPT realiza son: la copia de clases, ranuras e instancias, además hace una copia superficial y profunda de las clases [15].

La unión de ontologías se efectúa de forma semiautomática ya que permite al usuario intervenir tomando las decisiones más importantes durante este proceso.

Los problemas o las posibles inconsistencias generadas de la unión son resueltas por el usuario, estas son: las referencias sin conectar, los conflictos con nombres de clases, las ranuras con nombres repetidos, por citar algunos.

El modelo de conocimientos de PROMPT está basado en marcos, conocidos como los bloques principales para construir ontologías. Cada marco tiene un nombre único y está compuesto por:

1. *Clases*: Llamados también como Tipos, es el conjunto de entidades, las clases forman una taxonomía jerárquica con múltiples herencias.
2. *Ranuras*: Es la relación binaria entre clases e instancias, puede tomar el valor de un rango (dominio de la clase), el rango de la ranura restringe los valores que éste puede tomar.
3. *Instancias*. Son los elementos que forman un conjunto de clases.

PROMPT trabaja con el protocolo de conectividad abierta de base de datos OKBC. El modelo también considera a las Facetas[26].

PROMPT sugiere al usuario las clases, ranuras e instancias a unir, identificando inconsistencias y problemas que resulten de la unión además sugiere las estrategias para resolver estos problemas.

El editor de ontologías de Protégé puede importar ontologías de DAML+OIL.

Procedimiento del algoritmo

1. PROMPT crea una lista de casamientos entre las similitudes de clases, ranuras e instancias, muestra al usuario esta lista.
2. El usuario elige la operación de esta lista de sugerencias.
3. PROMPT realiza la operación elegida y genera una lista de sugerencias (inconsistencias y problemas) en base al resultado de esta operación.
4. El usuario resuelve las inconsistencias e indica a PROMPT lo que debe unir.
5. regresa a 3.

Se realizó un experimento en el área médica, con la finalidad de obtener la representación del conocimiento acerca de la estructura del cuerpo humano [14].

PROMPT reconoce las *metaclases*[27]. La búsqueda de similitud entre dos conceptos se basa en los nodos vecinos; es decir, patrones estructurales.

2.2.2 Chimaera

Proyecto diseñado en el Laboratorio de Sistemas de Conocimiento (KSL) de la Universidad Stanford [28]. La herramienta interactiva Chimaera se usa para la

[26] Facetas son la relación ternaria entre una clase, una ranura y un valor
[27] Meta clases son las clase que tiene instancias que también son clases

unión de ontologías, utiliza el editor de ontologías de Ontolingua [38] las ontologías pueden tener distintos formatos.

La relación taxonómica que Chimaera considera durante la unión es la de tipo estructural (subclase-superclase, tales como "*es un*", "*un tipo de*") [38] así mismo, permite al usuario elegir el nivel de sugerencia, por ejemplo el nivel más alto es la verificación de los anacronismos [38].

Al igual que en PROMPT, Chimaera requiere de la intervención del usuario en el proceso de unión, por esa razón se dice que es semiautomática.

Procedimiento del Algoritmo

1. el usuario solicita a la herramienta, un análisis para el proceso de la unión (se piensa añadir reglas que permitan al usuario personalizar el diagnóstico a su ambiente particular).
2. la herramienta lo ubica en el lugar de la ontología donde se requiere la atención, permitiendo que el usuario decida realizar la unión.
3. continúa con la lista de sugerencias al usuario.

La lista de sugerencias contiene:

a) prueba de no completitud y verificación sintáctica: la existencia de términos no definidos,
b) verificación semántica: los términos con rangos contradictorios,
c) análisis taxonómico: la detección de ciclos

Chimaera es compatible con el protocolo OKBC, acepta más de 15 tipos de archivos, algunos de ellos son: ANSI KIF, Ontolingua, Protégé, Classic, iXOL, RDF, DAML, etc. también considera las relaciones en términos sintácticos por ejemplo: dadas las relaciones entre las clases X y Y supone que las subclases de éstas también se encuentran relacionadas por lo tanto, al hacer nuevos enlaces a la clasificación verifica la semántica considerando la relación de transitividad, así como el nombre de relaciones entre las ranuras (atributos, propiedades).

Chimaera fue usado en un importante proyecto de Base de conocimientos para analizar ontologías, actualmente está siendo analizado por compañías importantes como: VerticalNet[28] y Cisco[29]. Los objetivos de Chimaera son: diseñar y unir ontologías [38].

Chimaera guía al usuario en los puntos (clases, ranuras, facetas) en los que se requiere la unión, pero no verifica los conflictos o estrategias como efecto de esa unión [15].

[28] www.verticalnet.com/home.asp.
[29] www.cisco.com/.

2.2.3 OntoMerge

OntoMerge [11] surgió en el Departamento de Ciencias de la Computación de la Universidad Yale. Esta herramienta une dos ontologías usando un mecanismo de búsqueda de similitud entre los elementos (clases, facetas, ranuras) de las ontologías, a este proceso se le conoce como *puenteo de axiomas*.

Procedimiento del algoritmo

1. Traduce las ontologías de una representación sintáctica común en formato DAML+OIL (§2.1.1); este traductor se llama PDDAML (Planning Domain DAML), el lenguaje interno que usa es Web-PDDL (Planning Domain Definition Language) es un tipo de lenguaje lógico de primer orden muy poderoso para aplicaciones de Web que usa la sintaxis de LISP con XML y algunas notaciones flexibles para los axiomas [11]. Este lenguaje se usa para realizar la traducción sintáctica.
2. Analiza (mapea) todas las instancias de una ontología para hacer sugerencias de la unión.
3. Un usuario experto está presente a cada paso de la unión eligiendo cada una de las sugerencias.
4. Una máquina de inferencia llamada OntoEngine realiza la traducción semántica mediante la unión de términos (conceptos) y axiomas, en este proceso se realiza el *puenteo de axiomas* o la alineación de conceptos y ranuras a través de su similitud. La máquina de inferencias procesa afirmaciones y consultas en sintaxis Web-PDDL, ejecutándose en un manejador de datos o manejador de peticiones, a través de la aplicación de reglas deductivas como un componente de inferencia importante en los sistemas de traducción, el Web-PDDL usa sintaxis como Lisp y tipos poderosos de lenguaje lógico de primer orden [11]; es decir la búsqueda de similitud entre los elementos de la ontología se realiza a través de un algoritmo de inferencia.
5. Una vez unidas las ontologías las traduce a su formato inicial DAML+OIL.

OntoMerge puede usarse conjuntamente con PROMPT (§2.2.1) para apoyar al usuario en el *puenteo de axiomas*.

Para evitar ciclos infinitos se pone un límite a la complejidad de términos que OntoEngine genera y por supuesto, la máquina deductiva se detiene cuando llega a la conclusión (o a la meta). El lenguaje interno PDDAML también usa el lenguaje OWL.

OntoMerge sirve como una conexión semiautomática entre agentes y humanos para encontrar maneras de copiar ontologías con distintas notaciones tanto de sintaxis como de semántica [42].

2.2.4 FCA-Merge

El método FCA-Merge [34] del Instituto para la Informática Aplicada de la Universidad de Karlsruhe, Alemania. Realiza la unión de ontologías a través de un conjunto de documentos basándose de la técnica Bottom-up. El algoritmo FCA-Merge está implementado en el ambiente OntoEdit[30].

FCA-Merge sigue un proceso de unión estructural, donde se aplican técnicas de procesamiento de lenguaje natural y Análisis Formal de Conceptos[31] para obtener una lattice de estos conceptos (conjunto de objetos representados gráficamente, donde existe un orden definido entre ellos). Este proceso de unión también requiere de la intervención humana. La ontología resultante surge de la lattice de conceptos. Los resultados de la unión son comprobados manualmente [34].

Procedimiento del algoritmo

1. Se extraen dos documentos de texto de los cuales serán procesados usando la técnica *Botton-up* y mediante técnicas de lenguaje natural, obteniendo dos ontologías conocidas como ontologías fuente y destino.
2. Se extraen instancias de la ontología fuente, de un determinado dominio realizando el mapeo entre las instancias de esta fuente a otra ontología destino, durante este proceso de mapeo se aplican técnicas de lenguaje natural y técnicas matemáticas tomadas del análisis formal de conceptos.
3. Se realizan de forma automática el proceso de extracción de documentos y el mapeo, no así la unión de términos estrechamente relacionados que se realiza en proceso paralelo con el mapeo pero requiere de la intervención de un usuario.
4. Se crea una lattice de conceptos como resultado de la unión.
5. Se realiza la poda de la lattice por medio del usuario y de forma manual.

Los conceptos considerados como diferentes aparecerán en ambos documentos, pero no aquellos similares que mapearán a un mismo concepto.

Durante el proceso de unión se enfrenta con el problema de la ambigüedad de conceptos, estos son tratados de forma diferente, pero no aquellos que están en una ontología y no en la otra porque serán unidos. El algoritmo de unión contempla la similitud entre conceptos y sus vecindades.

[30] OntoEdit es un ambiente de Ingeniería de ontologías para desarrollar y dar mantenimiento de ontologías gráficas, ver: www.ontoknowledge.org/tools/ontoedit.shtml.

[31] Análisis Formal de Conceptos o Lattice de Galois, es un método para análisis de de datos basados en la teoría de Lattices y Calculo Proposicional, se ubica en la exploración de símbolos de conocimientos (conceptos) contenidos en un contexto formal, tal como una base de datos u ontología.

Al finalizar la unión, todos los conceptos de la ontología fuente estarán en la ontología destino, así como sus relaciones, los posibles conflictos y duplicaciones hallados durante este proceso serán resueltos por el usuario.

2.2.5 IF-Map

El Proyecto AKT, [37] es el resultado de una colaboración interdisciplinaria de investigación (IRC), patrocinada por la ingeniería británica y el Consejo de Investigación en Ciencias Físicas. En el proyecto se escriben programas para rastrear la red y buscar recursos en RDF para construir o poblar ontologías. Este proyecto usa el algoritmo IF-MAP (Information Flow-based Method For Ontology Mapping) desarrollado en la Universidad de Southamptom que requiere de la base de conocimientos de Protégé y el traductor de Ontolingua para encontrar la similitud entre ontologías. Este traductor convierte de formato KIF, bases de conocimiento de Ontolingua y Protégé a cláusulas de Prolog. El algoritmo IF-MAP tiene cierta influencia del método FCA-Merge [34].

En el proyecto AKT [25], se encuentran rutinas para rastrear la red y obtener ontologías codificadas en el formato RDF (Resource Description Frame Word). Construye semi automáticamente una ontología como resultado de ese mapeo [25].

El algoritmo IF-Map se aplicó en un experimento a gran escala, mapeando cinco ontologías del departamento de ciencias de la computación en cinco universidades del reino unido. El mapeo y el unión están basados en heurísticas[32] y usan la sintaxis para determinar la correspondencia o equivalencia entre los conceptos de las ontologías, pero rara vez usan la semántica; la metodología que usa en el proceso de mapeo y unión es parecido al método FCA-Merge.

IF-Map usa el IF (Information Flow-based) como un fundamento matemático para establecer mapeos entre dos ontologías, se formalizan estos mapeos en términos de lógica isomorfismo.

Procedimiento del algoritmo

1. *Cosecha de ontologías*: se pueden descargar ontologías de servidores, tales como: Ontolingua, WebOnto y editarlos por ejemplo en el editor de ontologías de Protégé. Actualmente se están diseñando programas para rastrear ontologías de la red en el formato RDF con la finalidad de poblar ontologías, se han tratado con formatos en KIF, OCML, RDF, Prolog y bases de conocimiento nativos en Protégé.
2. *Traducción*: consiste en interpretar los formatos recopilados en el paso 1 y ejecutarlos en el motor de Prolog, se traducen estos formatos a las cláusulas

[32] Heurística es la capacidad de un sistema para realizar de forma inmediata innovaciones positivas para sus fines. La capacidad heurística es un rasgo característico de los humanos, desde cuyo punto de vista puede describirse como *el arte y la ciencia del descubrimiento y de la investigación* o de resolver problemas mediante la creatividad y el pensamiento lateral o divergente, ver: es.wikipedia.org/wiki/Heurística.

de Prolog, el traductor de IF-Map se personaliza para la traducción a la taxonomía de clases, relaciones e instancias y la unión de estos elementos.

3. *Infomorfismo*: para la generación infomorfismo el proceso traduce las ontologías y genera infomorfismo, los enlaces formales entre constructores similares en ambas ontologías, realizando la unión de estos conceptos.
4. *Presentación de resultados*: se procede a mostrar el resultado de la unión en el formato RDF, este paso incluye la traducción de Prolog a RDF y se almacenan los resultados en una base de conocimientos para referencias futuras y mantenimiento [25].

La unión de un conjunto de conceptos tiene una estructura adicional que puede deducir la manera en que se conectan las instancias clasificadas respecto a esos conceptos. A través de técnicas de análisis formal de conceptos por ejemplo, se puede hacer explicitas tales estructuras en forma de una lattice de conceptos, la jerarquía de conceptos representado en esta lattice depende de las instancias elegidas y la clasificación [25], [37].

2.2.6 ISI

El proyecto ISI [40] del instituto de Ciencias de la Información de la Universidad del Sureste de California, realiza la unión de ontologías extremadamente grandes, por ejemplo: CYC[33] [32] y SENSUS[34] [45].

Procedimiento del algoritmo

1. El algoritmo crea una lista de sugerencias que presenta al usuario, en la que no solo se consideran los nombres de las clases sino aquellos conceptos cuyos nombres y definiciones comparten muchas palabras conocidas y una gran similitud entre hermanos, hijos y ancestros; sin embargo, el algoritmo no considera ranuras (nombres de las relaciones).
2. El usuario indica la sugerencia a realizar.
3. El algoritmo realiza la unión y regresa a 1.

El trabajo desarrollado en USC/ISI se usó para la construcción de la ontología SENSUS. Esta unión consideró un proceso de 8 pasos una parte manualmente y otra por un algoritmo, para crear una nueva ontología unida que contiene 2, 431 conceptos. En este proceso se encontraron pequeñas inconsistencias [40].

[33] CYC es un servidor de conocimientos muy extenso y multicontextual basado en máquinas de inferencia desarrolladas por Cycorp, ver: www.cyc.com.

[34] SENSUS es una ontología con 70 000 conceptos, como un marco de trabajo en el cual puedan ser ubicados nuevos conocimientos, es una extensión y reorganización de WordNet (construido por la Universidad de Princeton por George Millar y sus colaboradores), ver: www.isi.edu/natural-language/projects/ONTOLOGIES.html

2.2.7 HCONE-merge

Este método de alineación y unión de ontologías ha sido desarrollado en el Departamento de Información e Ingeniería de Sistemas y Comunicaciones de la Universidad de Aegean en Karlovassi, Samos, Grecia [26]. Es un trabajo de unión de ontologías que se esfuerza por salir del conjunto de algoritmos que funcionan semi-automáticamente, usa la base de conocimiento de WordNet [12] (como ontologías intermediaria entre las dos ontologías a fusionar) y el método LSI (Latent Semantic Index conocida como una técnica de vector de espacio originalmente propuesto en el área de recuperación de información e indexación) en la alineación de ontologías y además la Lógica Descriptiva.

Una vez tomadas dos ontología bajo el formato DAML-OIL (§2.1.1) HCONE automáticamente alinea éstas con WordNet[35] (usando el método LSI), un usuario verifica la correcta alineación y HCONE une las ontologías (se presume que este proceso pueda ser automático), finalmente un usuario verifica la unión correcta.

La meta de HCONE es realizar la unión eficientemente y con la mínima participación del ser humano. Actualmente, HCONE requiere de un usuario para validar la semántica de los conceptos en la ontología, depurando los posibles errores derivados de este proceso, por lo pronto se está investigando una serie de técnicas heurísticas para realizar el proceso automáticamente

Procedimiento del algoritmo

1. Alinea las ontologías A y B fuentes con WordNet buscando el morfismo semántico entre cada una de estas y WordNet en este proceso se aplica el método LSI asociando los conceptos de las ontologías con los sentidos de WordNet.
2. Realiza la unión de las ontologías A y B de tal manera que el resultado sea la mínima unión de vocabularios ontológicos y axiomas con respecto a la ontología intermediaria (donde han sido alineados).
3. Los conceptos de la ontología A y la B que han casado (casan si los dos conceptos han mapeado con el mismo sentido en Wordnet) los une (su significado, su clasificación, renombra los conceptos si es necesario). El método de unión ha sido probado con ontologías reales construidas a mano y comprobadas también de forma manual.

Al aplicar el Análisis Formal de Conceptos y encontrar un morfismo entre las ontologías a unir, este proceso se restringe a la unión de ontologías correctamente descritas, por lo que las ontologías inconsistentes que la mayoría de las veces se encuentran en el Web serían descartadas en el proceso de unión.

Puede decirse que HCONE es un esfuerzo hacia la resolución de sinónimos y en general de desambiguación con ayuda de WordNet.

[35] Con una ontología derivable de Wordnet, para ser más precisos [26].

2.2.8 El algoritmo OM

El proyecto OM (Ontology Merging) se ha estado desarrollando en el Centro de Investigación en Computación del Instituto Politécnico Nacional (CIC-IPN) en México desde enero de 2004 hasta la fecha. Éste es un algoritmo totalmente automático (no usa la participación del usuario) y robusto (se pueden unir ontologías no definidas correctamente o inconsistentes). En el proceso intervienen dos ontologías A y B para formar una tercera ontología C.

Procedimiento del algoritmo

1. Copia la ontología A hacia C.
2. Partiendo del concepto raíz $C_{Raíz}$ en C.
3. Busca en B su concepto más similar C_B (usando COM [7]), el concepto más similar también es conocido como *cms*.
4. Si hay un *cms* en B se adicionan nuevas relaciones, nuevos conceptos, se verifican sinónimos, detectan y resuelven algunas inconsistencias (usando la teoría de la confusión [21]), se verifica e impide la copia de las relaciones redundantes.
5. Si no hay un *cms* accede al siguiente nodo C_C (hijo de la raíz) y regresa a 3

Si en el paso 4 no se resuelven las inconsistencias se conserva la relación "inconsistente" en C (la ontología resultante). Más detalles sobre este algoritmo se explica en §3.2.1.

2.2.9 Análisis comparativo de los trabajos de unión de ontologías expuestos

Una manera de clasificar las herramientas de fusión de ontologías es por el tipo de entrada en el cual basa su análisis: por ejemplo:

La diferencia entre el proceso de unión.

La principal diferencia entre OM y los demás trabajos de unión es que todos los anteriores usan un proceso semi-automático mientras que en OM es totalmente automático. Otra diferencia es que OM puede unir ontologías carentes de infomorfismo (véase anexo A) lo que [25 y 34] no podrían.

Diferencias en la búsqueda del concepto más similar de una ontología a otra

1. ISI usa el nombre de clases y compara a través de mecanismos de lenguaje natural.
2. Chimaera lo hace a través de la jerarquía de las clases.
3. PROMPT lo hace a través de la jerarquía de las clases, ranuras y facetas.
4. FCA-Merge trabaja con las Instancias de las clases.

5. OntoMerge, FCA-Merge e IF-Map lo hacen a través de las descripciones de las clases (usando mecanismos de lenguaje natural).
6. OM lo hace a través de COM [7] que compara las descripciones de los nodos, descripciones de las relaciones (que también son nodos), descripciones de las instancias y identificación se sinónimos. Es decir, a través de un proceso de verificación estructural (nodos padre e hijo) y a través de la semántica de cada nodo y relación.

Se presenta una tabla comparativa sobre los datos generales y una clasificación de estas herramientas por la forma de realizar el análisis de la información (véase la Tabla 2.2) aplicados a cada algoritmo expuesto.

Tabla 2.2 Características básicas en el mapeo de ontologías de los trabajos de unión

Características	**PROMPT**	**Chimaera**	**Onto Merge**	**FCA Merge**	**IF Map**	**ISI**	**HCONE**	**OM**
Nombre de clases y lenguaje natural	x	x	x	√	x	√	√	√
Jerarquía de las clases	x	√	x	x	x	x	√	√
Jerarquía de las clases, ranuras y facetas	√	x	x	x	√	x	√	√
Instancias de las clases	x	x	x	√	√	x	√	√
Descripción de las clases, basadas en descripción lógica	x	x	√	√	√	x	√	x
Relaciones semánticas	x	√	√	√	√	x	√	x
Análisis formal de conceptos	x	x	x	√	√	x	√	x

Diferencias en el proceso de fusión

OntoMerge [11], Chimaera [28], Prompt [13] e ISI [40] confían en el usuario para resolver los problemas más importantes encontrados en el proceso y son considerados como fusionadores semiautomáticos. FCA-Merge [34], IF-Map [25] y HCONE [26] requieren ontologías consistentes que son expresados en una notación formal empleado en Análisis Formal de Conceptos [16], el cual limita su uso. Sin embargo, HCONE [26] es un notable avance hacia la unión automática.

El Algoritmo OM realiza la fusión de manera:

1. Robusta (resuelve la mayoría de los problemas de la fusión, no se detiene ni se cicla en el proceso)
2. Automática (sin intervención del usuario)
3. Completa (el resultado contiene todo el conocimiento disponible de las fuentes fusionadas, sin redundancia, detectando los sinónimos, entre otras tareas)
4. Consistente (sin contradicciones)

Funcionamiento de PROMPT, Chimaera y OM

PROMPT y Chimaera son herramientas parecidas en su funcionamiento en el conjunto de herramientas que fusionan ontologías. Se han usado para unir las mismas ontologías y se realizaron la misma secuencia de pasos en la unión en cada herramienta. Las operaciones realizadas incluyeron unión de relaciones y clases. Después de cada paso, se compararon el conjunto de nuevas sugerencias que los dos sistemas generaron. Un usuario experto unió dos ontologías manualmente para conocer si las sugerencias que los sistemas producían eran las correctas [13].

PROMPT tuvo un 30% más de sugerencias correctas que Chimaera. Las sugerencias de Chimaera fueron un subconjunto de las sugerencias de PROMPT. El 20% de las sugerencias correctas de Chimaera fueron exactamente las mismas de PROMPT. El 80% restante fueron mucho menos específicos que las sugerencias de PROMPT: Chimaera señaló la clase (concepto) en la ontología donde se requería la acción y PROMPT sugirió una acción específica (o acciones alternativas), que requería el marco (frame) [13].

OM requiere dos ontologías definidas en el lenguaje OM y no proporciona sugerencias, sino con su base de conocimiento §3.2.1 realiza la fusión de forma automática.

Herramientas PROMPT, Chimaera y OM en el proceso de fusión

Los resultados demostraron que un experto humano coincidió con una gran cantidad de sugerencias de ambas herramientas y las estrategias para la solución de conflictos que PROMPT sugirió. PROMPT fue capaz de realizar un gran número de operaciones de unión con la aprobación del experto humano, ahorrando el tiempo y el esfuerzo del experto [13].

Existe una gran diferencia en la manera en que Chimaera y PROMPT guían al usuario en el proceso de unión: PROMPT presenta una lista de operaciones específicas que se sugieren al usuario. Chimaera se ubica en la clase donde requiere la atención del usuario, pero no indica que se necesita hacer exactamente ahí. Por ejemplo, en la Figura 2.1, Chimaera sugiere que la clase `Ontology` requiere de la atención del usuario porque algunas de las relaciones (ranuras) de esa clase vino de diferentes ontologías fuente. No especificó cuales relaciones (ranuras) necesita considerar el usuario y que necesita hacer (el usuario). En cambio PROMPT sugirió que la relación (ranura) `axioms` (que vino de dos ontologías diferentes), debía ser unido. En ontologías extensas la gran cantidad de sugerencias específicas que muestra PROMPT pueden abrumar al usuario, mientras que la forma de indicar las clases (conceptos) que requieren atención de Chimaera puede ser mejor [13].

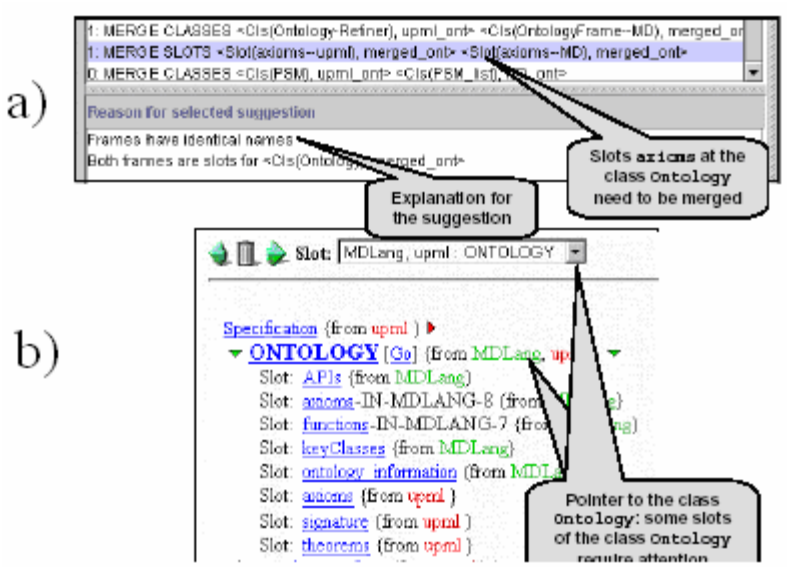


Figura 2.1 Diferencias entre las sugerencias de a) PROMPT y b) Chimaera. PROMPT sugiere que el usuario fusione dos relaciones (ranuras) específicas `axioms` de la clase (concepto) `Ontology`. Chimaera indica al usuario la clase `Ontology` que tiene dos relaciones originados de dos ontologías diferentes sin especificarle cuales relaciones necesita considerar

OM pide a un usuario que ingrese las dos ontologías a fusionar. En la Figura 2.2 el usuario elige el área de trabajo a) (Panel A) donde se ubicará la ontología A, presiona el botón Ontología A y señala el archivo con extensión *ont* b), luego elige el segundo área de trabajo donde se alojará la ontología B (igual que en a) pero elige el Panel B), presiona el botón Ontología B y señala el archivo *ont*, elige el área de trabajo de la ontología resultante C (Panel R), presiona el botón Ontología R y solo OM trabaja en la fusión de ambas ontologías A y B, la ontología resultante aparece en el Panel R, éstas son las únicas participaciones del usuario en la fusión: 1) para elegir las ontologías y 2) esperar el resultado en el Panel R.

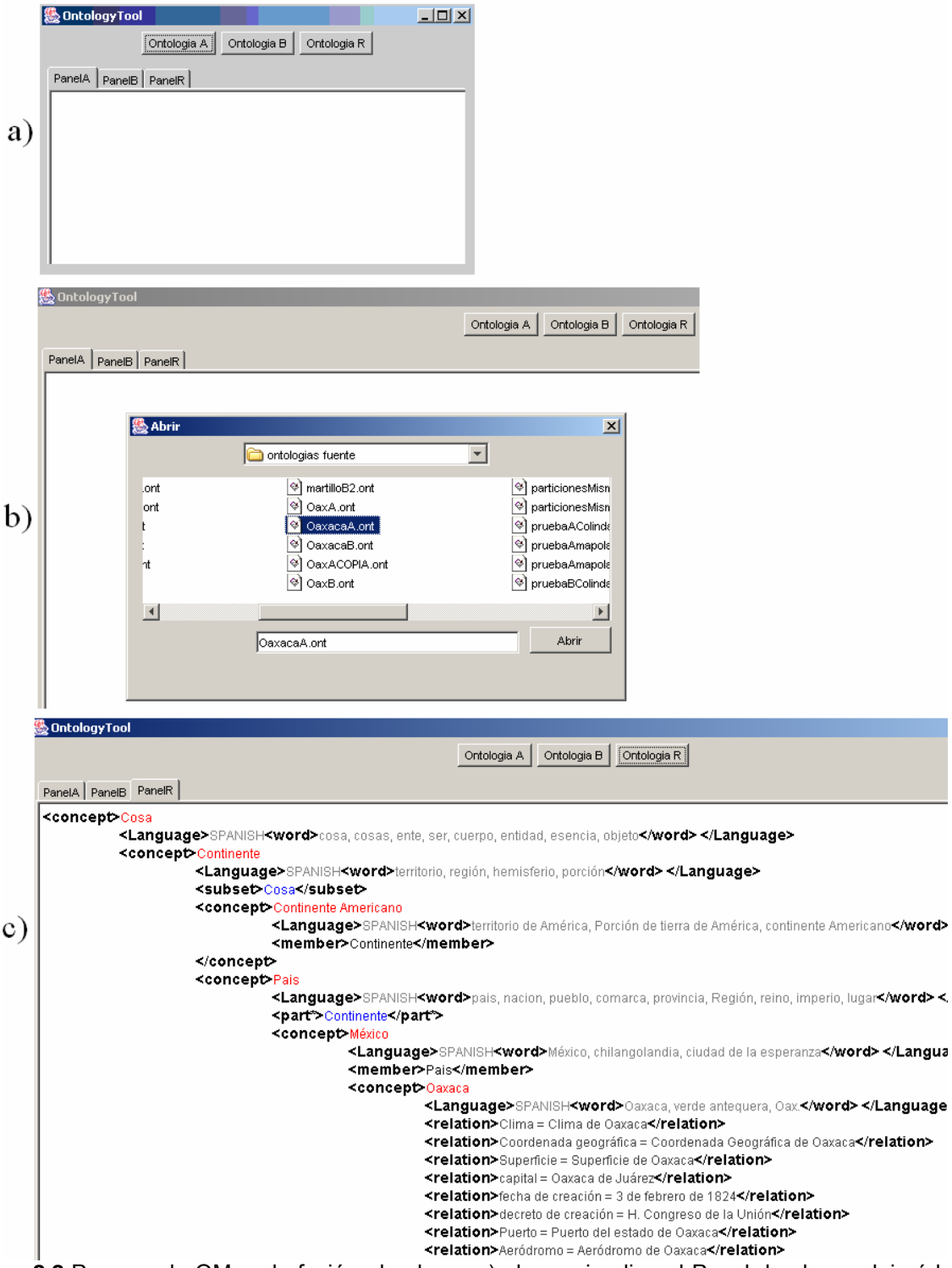


Figura 2.2 Proceso de OM en la fusión, donde en a) el usuario elige el Panel donde se alojará la ontología, en b) elige la ontología fuente y en c) aparece el resultado de la fusión (en el Panel R)

3. El algoritmo OM para fusionar ontologías

La fusión de ontologías consiste en tomar dos ontologías y producir una tercera que conserve el conocimiento de las ontologías formantes, este resultado de la fusión debe conservar la semántica de las ontologías unidas. A continuación se detalla el concepto de ontología, posteriormente se centra en el algoritmo de fusión y los de apoyo para realizar esta tarea.

3.1 Definición de ontología

Una **ontología** es una tupla O = (C, R) donde:

C es un conjunto de nodos (que denotan conceptos) de los cuales algunos de ellos son relaciones.

R es un conjunto de restricciones, de la forma (r c_1 c_2 … c_k) entre la relación r y los conceptos c_1 hasta c_k. Se dice que la **aridad** de r es k. Ejemplos: (corta `tijera papel`), (transcribe `impresora documento tinta`). En ese ejemplo, los conceptos que también son relaciones son cortar, transcribir o color. color tiene aridad 3.

3.1.1 Definiciones adicionales

En Computación, una ontología es una estructura de datos donde la información se almacena como nodos y relaciones. Normalmente, la información almacenada en una ontología se denomina *conocimiento*. Las relaciones también son conceptos y en su mayoría son binarias. Existen relaciones n-arias que son difíciles de representar gráficamente ya que intervienen más elementos vinculados a la relación, por ejemplo: (préstamo, `Juan, Ana, $3,000, 15%, 1º de diciembre 2006`).

Algunos trabajos (ver anexo A) dividen a los conceptos en clases o tipos (como *`abogado`* o *`conejo`*) e instancias (individuos, como *`Juan Pérez`* o *`Conejo Bugs`*); es decir, Los conjuntos I (instancias) y T (tipos) forman una partición de C.

Tipos e instancias. Algunas definiciones de ontologías contienen una función de *clasificación* [Anexo A], que en la definición de esta tesis se representa con la relación *miembro de*. Así, si `Juan Pérez`[36] (una instancia) es del tipo `Persona` (un Tipo), es decir, está clasificado como `Persona`, se dice que Juan Pérez ∈ `Persona` (∈ se interpreta como pertenece); se piensa de igual forma que el conjunto `Persona` representa a todas las personas (instancias) de la comunidad.

[36] A partir de este momento los conceptos, las instancias y las relaciones que denoten conceptos se identificarán con el tipo de letra `Courier New`, otras veces y para los efectos de diferenciar a las relaciones, se mostrarán en el tipo Arial Narrow.

Concepto. Formalmente un concepto es un miembro de I ∪ T (el símbolo ∪ es la unión de conjuntos). Informalmente, un concepto es la representación concreta de una idea, acción, propiedad o individuo que existe en el mundo real.

Existen conceptos que están pobremente definidos en una ontología, pues todo lo que se sabe de ellos es su nombre (palabra o frase en un lenguaje natural asociada al concepto y que lo describe). Por ejemplo: el concepto `verde` (véase la Figura 3.1); todo lo que se sabe de ese concepto es su nombre: `verde` y que es un color del fruto de la amapola. A estos conceptos los llamamos *pre-conceptos* y los identificamos a lo largo de este documento con un círculo con línea delgada y discontinua. A los otros conceptos, con más información, se les identifica con un círculo en línea continua. Todos los conceptos, inclusive los pre-conceptos, se identifican en el texto en letra `Courier New`. Ejemplo: `Oaxaca` (véase la figura 3.2).

Nombre de un concepto. Es útil tener una relación nombre (`Benito Juárez`, `Benito Juárez García`, `Licenciado Juárez`, etc.) que liga un concepto `Benito Juárez` con una(s) palabra(s) o frases temáticas en un lenguaje, (español) que representa a su(s) nombres.

Nota sobre Relación. Muchas relaciones son binarias, como *padre de*, por lo que los lenguajes de representación usan árboles o redes para representar ontologías, colocando los conceptos en los nodos de la red y mostrando las relaciones como arcos. Por conveniencia, y cuando no cause confusión, en este trabajo de tesis se seguirá esta costumbre.

Las relaciones en general no son conmutativas, por ejemplo, (préstamo `Juan Pérez`, `Ana Cruz`) donde no se puede intercambiar al prestamista `Juan Pérez` con el deudor `Ana Cruz`. Las relaciones en la gráfica se representan por flechas y en el texto en letra Arial Narrow.

Valor de una relación. Otra costumbre tolerada es identificar en una relación binaria como (sueldo `Juan Pérez`, `20,000`), al segundo concepto como “el dueño de la relación” y al tercero como “su valor”. Así, se dice que el valor de la relación sueldo para `Juan Pérez` es `20,000` y se dice que el sueldo de `Juan Pérez` es de `20,000`.

Nota sobre las Restricciones. En esta tesis una restricción es una tupla donde el primer elemento es una relación “entre” los otros conceptos de la tupla, por ejemplo (come `Juan_Pérez carne`). Una restricción puede ir más allá, por ejemplo, la restricción “*todo conejo es un mamífero*”, que se expresa como una ecuación lógica. La notación de esta tesis está concentrada en la fusión de ontologías, por lo que no usa ecuaciones lógicas.

Nota sobre Aridad. Puesto que una relación es un concepto, puede tener

relaciones. Una relación importante es la aridad. De hecho, es una función *a*: R → N. La aridad asigna a cada relación un número natural, como ya se definió. Una relación es *monovaluada* cuando su aridad es 1. En otro caso, la relación es *multivaluada* (§3.9). Ejemplo: a(mi padre) = 1; *a*(mi hijo) > 1, donde *a* es la función que identifica la aridad de la relación.

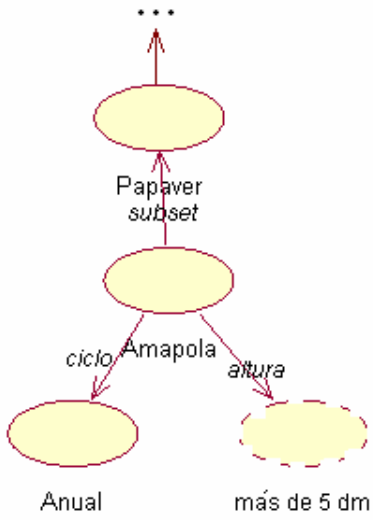


Figura 3.1 Ejemplo de un pre-concepto llamado `más de 5 dm`, donde solo se sabe que es la altura de la `Amapola`.

Se presenta la Figura 3.2 con un ejemplo de conceptos relacionados.

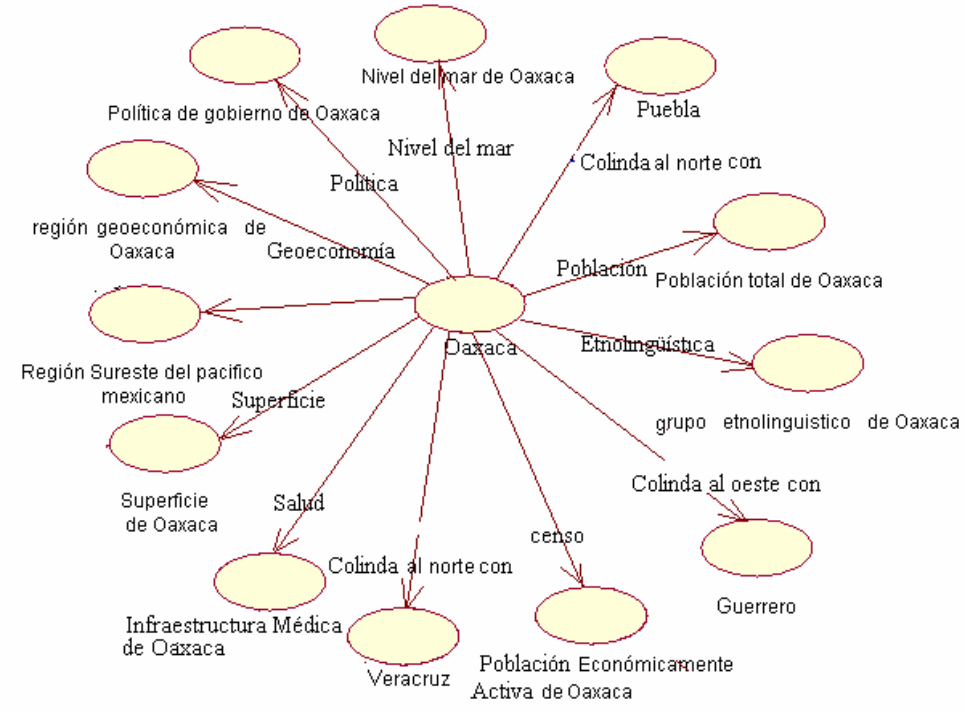


Figura 3.2 Ejemplo de un concepto llamado `Oaxaca`, con varios enlaces a conceptos, es decir, relaciones explícitas (ver §3.7.2)

Una **Ontología sin poblar** solo tiene Tipos, carece de instancias.
Una **Ontología poblada** tiene ambos.

Formalmente, el trabajo de fusionar dos ontologías para formar una tercera puede verse como establecer un infomorfismo entre dos lógicas locales. Este enfoque se da en el Anexo A. Sin embargo, en este trabajo no se sigue esa línea, porque ésta presenta problemas y dificultades en su mapeo o apego al mundo real.

3.1.2 Definición del problema a resolver

Considerando dos ontologías pobladas A y B, se trata de unirlas en una tercera ontología C. Se forma una nueva ontología, de manera general se muestra como:

$$C = A \cup \{C_C \mid C_C = ext(R_A, R_B) \ \forall \ C_A \cup A\ , C_B \cup B\ ,\ C_c \cup C,\ R_A, R_B \cup R\}$$

(La ontología resultante C es la ontología original A añadida de ciertos conceptos y relaciones de B que la función *ext* extrae, ver §3.2).

Donde:

C_A es un concepto de la ontología A, R_A son las relaciones de C_A que existen en A, R_B son las relaciones de C_B que existen en B y C_B es el concepto más similar *cms* en B a C_A;

$\cup$. indica una unión de ontologías (es una unión "cuidadosa") y no unión de conjuntos.

ext(R_A, R_B) es el algoritmo que complementa las relaciones R_A que ya están en C con aquéllas de C_B (que están en B) que no contradicen el conocimiento de A, según abajo se explica.

Al aplicar *ext* a cada concepto C_A de A, el algoritmo OM (Ontology Merging, o fusión de ontologías) en realidad lo que hace es extraer de B el conocimiento "adicional" que no estaba presente en A y agregarlo al resultado C. Esta extracción debe hacerse con cuidado, para no introducir inconsistencias, contradicciones o información redundante en C.

Se presenta el proceso general de *ext* que incluye la función *com* (que también se explica de manera general)[37]:

1) Para cada concepto $C_A \in A$, obtiene $C_B = com(C_A, B)$, el concepto en B más similar a C_A, así como un mensaje *msg* con el caso (A, B, C, D) usado para encontrar el concepto más similar.

[37] Explicado en detalle en 3.5

2) Si *msg* = “Caso A”, en la unión de C_A y C_B se usarán las relaciones R_B del concepto C_B obtenido en 1) para calcular C_C = *ext*(R_A, R_B), que se explica detalladamente en 3.2.1.

3) Si *msg* <> “Caso A” se conserva C_A en C_C, sin ningún agregado.

La ontología C es el resultado de la fusión, particularmente se muestra como:

$C = \{C_C \mid C_C \text{ se obtiene en (2)}\}\ \{C_A : com(C_A, B)=nulo\}\ \{C_B : com(C_B, A=nulo\}$

La fusión C tiene tres componentes:

- Los elementos C_C que se obtienen (con cuidado) en el punto (2) de analizar qué se debe agregar a C_A para nodos que existen tanto en A como en B (primer conjunto de la fórmula).
- Los elementos de C_A que no tienen parecido con ningún concepto en B (segundo conjunto en la fórmula arriba);
- Los elementos de B que no tienen parecido con ningún concepto de A (tercer conjunto);

Reafirmando, la función *ext*(R_A, R_B) fusiona las relaciones R_A de C_A con las relaciones R_B de C_B, añadiendo a C las relaciones R_B que le hagan falta y considerando las relaciones R_A de C_A (que ya se habían copiado a C) con sus sinónimos[38] extraídos de las relaciones R_B del concepto equivalente C_B. En este proceso, se detectan las inconsistencias entre los nombres de las relaciones. Una inconsistencia o confusión es un hecho de la B que contradice un hecho de A (más detalles de esto en §3.8.1).

La fusión en la ontología C será:

Simétrica, es decir, $A \cup B = B \cup A$ (el símbolo $\cup$ representa a la fusión de las dos ontologías) si durante el proceso de fusión no existieron inconsistencias (§3.8.1) entre las relaciones o **Asimétrica**, en caso contrario,[39] es decir, se queda con el conocimiento de A. El algoritmo OM imita el comportamiento de los humanos en la fusión del conocimiento y en caso de duda la gente prefiere su conocimiento que el de las otras personas.

Existen otras opciones (que podrían considerarse a futuro) para evitar elegir el conocimiento de A ante las inconsistencias no resueltas por medio de la teoría de la confusión [21], estas son:

1. *Preguntar a Internet*. Cuando la relación inconsistente entre A y B no se resolvió queda la posibilidad de consultar el número de páginas en Internet que

[38] Explicado más adelante, dos conceptos son sinónimos si comparten la misma o parecida descripción (conjunto de palabras).

[39] En caso de inconsistencia, el algoritmo de unión privilegia el conocimiento de A (la primera de las dos ontologías a fusionar), por lo que C resulta asimétrico. Podría inventarse (trabajo futuro) un algoritmo simétrico OM’ que recurriera a una fuente externa (por ejemplo, la Web) en caso de inconsistencia.

mencionen información acerca de la relación inconsistente. Por ejemplo, si A tiene la creencia que: *Agustín Lara nació en la ciudad de México* y la creencia de B dice: *Agustín Lara nació en Tlacotalpan, Veracruz.* Se contabilizará el número de páginas en Internet que mencionen la creencia de A y la de B, eligiéndose aquella que tenga un mayor número de páginas. Esta opción es *medianamente acertada*, porque los buscadores no analizan que las oraciones efectivamente contengan la creencia buscada sino traen las palabras de cada creencia no importando si entre estas hay mucha distancia de por medio en todo el documento.

2. *Aplicar un enfoque estadístico*. Cuando se haya elegido la creencia de A se guardaría la creencia de B (en otra parte de A) contabilizándose como un voto y si se fusionara A nuevamente con otra ontología B y esta tuviese la misma creencia de la B anterior, A estaría cambiando su creencia (porque serían dos votos contra uno de A). En el ejemplo de 1 sería como encontrarse a dos ontologías B que dijeran que *Agustín Lara nació en Tlacotalpan, Veracruz* y así sucesivamente*.* Esta no sería una solución inteligente porque la ontología A estaría cambiando de creencia tantas veces como el número de fusiones se realice, la cantidad de votos no asegura la veracidad de la creencia.
3. *Elegir a la ontología que tenga más conceptos relacionados*. Se adopta la creencia de la ontología que tenga más relaciones conectados al concepto que forma parte de la creencia. Por ejemplo se adoptaría la postura de B si tiene más relaciones conectadas al concepto *Agustín Lara*, significa que *sabe más* acerca del tema. Aunque es posible que sepa más acerca de sus obras pero podría estar errónea acerca de ciertos eventos de su vida como es su nacimiento.

Cuando un hecho o varios afirman algo sobre una realidad, es difícil saber cuál es el hecho verídico, cuál tiene la razón. Aunque no es el alcance de esta tesis descubrir la verdad, al menos en ella se trata de dar una respuesta ante los problemas de inconsistencia, evitando que la fusión se detenga y requiera de la intervención del usuario.

3.2 El Algoritmo para la unión de ontologías Ontology Merging (OM)

El algoritmo de unión de ontologías de forma automática y robusta tiene el siguiente funcionamiento de manera general:

$$OM = ((C \leftarrow A) \cup B)$$

El símbolo $\cup$ indica la unión de ontologías, se trata de una unión cuidadosa de relaciones de un nodo. El símbolo $\leftarrow$ indica la copia total de A a la ontología resultante C.

Este proceso general se presenta como sigue:

1. C $\leftarrow$ A. Copia toda la ontología A hacia C (ahora todos los conceptos y

relaciones de C serán los mismos que los de A excepto que, C progresivamente será diferente a A, en función a los nodos de B que no estén en C)

2. Busca en B cada concepto C_C de C. La búsqueda inicia del concepto raíz en C, tomando cada uno de los hijos de éste hasta terminar de visitar todos los conceptos de C. Aquí hay dos opciones (está o no el concepto):

    a. **C_C encuentra en B a un concepto más similar *cms***.
        i. Se enriquecen las relaciones de C_C que son sinónimas con las de *cms*.
        ii. Se reciben en C_C las nuevas relaciones que *cms* tiene en B.
            1. Se copian los conceptos[40] (los que no tiene C) que encuentre en las nuevas relaciones que vienen de *cms*.
        iii. Se detectan inconsistencias entre las relaciones de C_C y *cms*.
        iv. Se resuelven algunas inconsistencias detectadas y en C_C se queda la relación consistente.
        v. Si no se resuelve la inconsistencia, en C_C se queda la relación original (la que ya estaba en C, la que recibió de A).
        vi. Se copian nuevos conceptos hijos[40] de *cms* a C.
    b. No se encuentra el concepto C_C en B.
        i. Toma el siguiente concepto en la C.

En el proceso general de la unión no se indica la copia de las particiones, porque una partición es un tipo de relación y éstas están incluidas en la copia de las relaciones.

En la Figura 3.3 se muestra la unión a otro nivel de detalle; en el inciso a) A se copia a la C y en la b) el concepto C_C se ubica en la raíz de la C a partir de esta ubicación se tomarán cada uno de los hijos de la raíz (hasta que no haya más hijos cesa la copia) luego en c) se usa el algoritmo COM [7] que busca C_C en B, obteniéndose un concepto más similar *cms* cuyas relaciones se añadirán a C.

En la misma Figura 3.3 aún se conservan pasos importantes de la unión de manera general, tales como: la copia de las relaciones implícitas (véase § 3.7.1), las relaciones explícitas (véase § 3.7.2), y la copia de las particiones (véase §3.6.1), estos pasos se contemplan en los algoritmos que más adelante se presentan.

[40] Al copiar los nuevos nodos en C, copiará también todos los antecesores de éste que estén en B y que no encuentre en C, solo se copiarán las etiquetas o nombres y las relaciones implícitas (ver §3.7.1 y 3.7.2) de cada concepto que no encuentre, esta copia se llama "copia superficial". Cuando se copien las relaciones explícitas (ver §3.7.2) de estos conceptos (los que se copiaron superficialmente) se dirá que el concepto ha sido copiado de manera completa o se ha realizado la "copia profunda" del concepto.

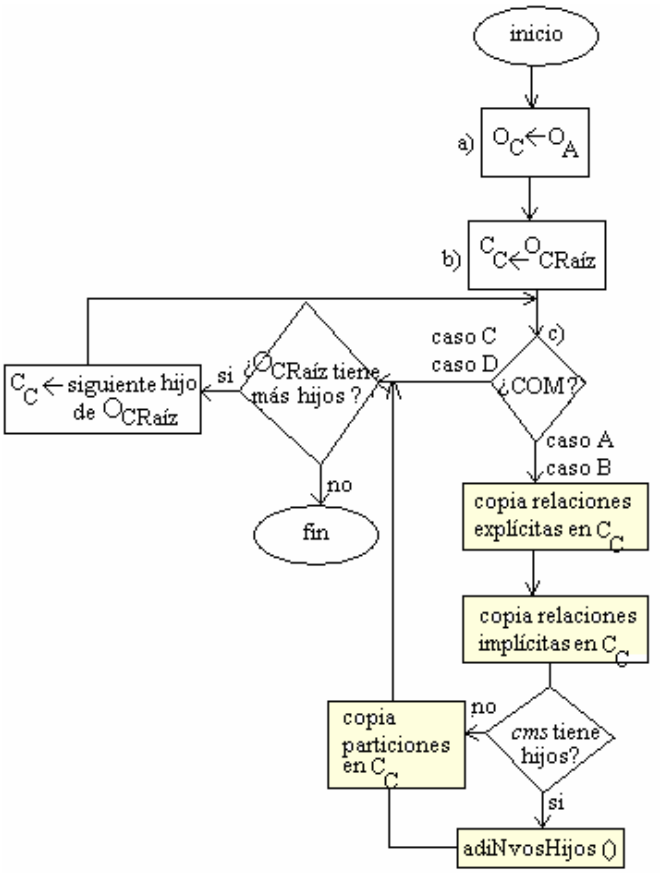


Figura 3.3 Diagrama general de flujo de datos de la unión

En la Figura 3.4 se muestra los pasos de la copia de las relaciones explícitas. Donde:

1. En d) se realiza una comparación de las palabras o frases que definen tanto a C_C como a *cms*, añadiendo las nuevas palabras que haya en *cms* y no en C_C.
2. En e) se verifican todas las relaciones de C_C, si la relación es ("sin", "excepto") indica que el valor no se añadirá como concepto con argumento en C (ver §3.1), estos conceptos "prohibidos" se verifican en el algoritmo *chec*() en el inciso i).
3. En f) se pregunta si el nombre de la relación se encuentra en la lista de relaciones de C_C, o si el algoritmo *extrae*() devuelve "verdadero", es decir si al extraer los artículos y conectores de las relaciones (ver §3.4.4.1) se ha hallado la relación actual en la lista de relaciones de *cms*.
4. En g) se realiza la depuración de los valores de cada relación (en *cms* y C_C) eliminando los conceptos antecesores y conservando los conceptos más específicos (Esta es una de las aportaciones de OM al algoritmo de la teoría de la confusión [21], ver §3.8.3.6).
5. En h) se realiza la aplicación de la confusión entre los valores de las relaciones (actualmente copiándose), ver §3.8.3.4.

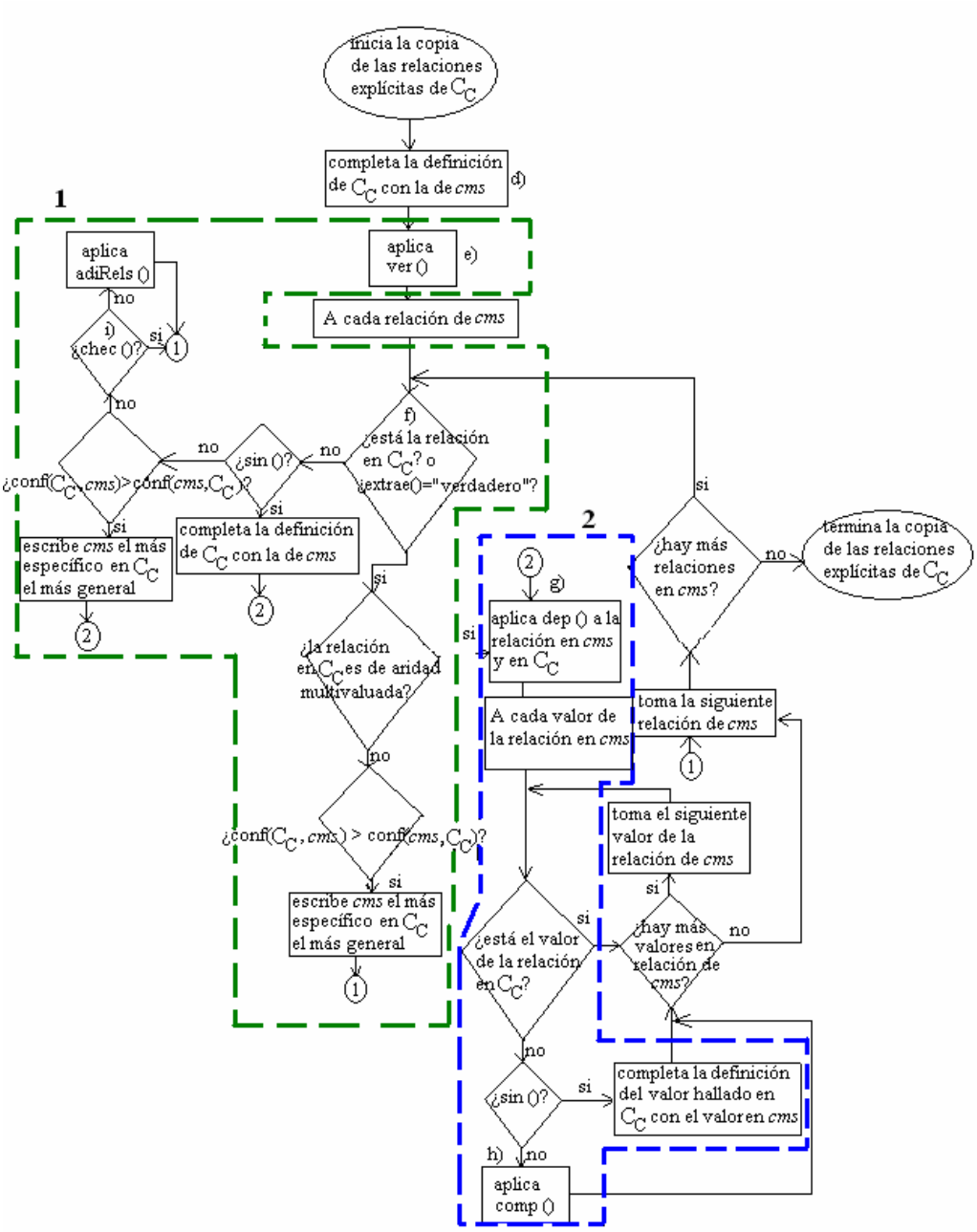


Figura 3.4 Diagrama de flujo de datos de la copia de las relaciones explícitas de un concepto *cms* en la B hacia C_C en la C, donde el rectángulo 1 con líneas discontinuas indica los cambios en la relación y el 2 indica los cambios en el (los) valor (es) de la relación

Después de las relaciones explícitas se copian las relaciones implícitas, en la Figura 3.5 se presenta el diagrama de flujo general. Donde:

1. En j) se muestra el algoritmo que añade (en C) los antecesores del concepto (en B) que se indica en la relación implícita, por ejemplo en la relación implícita <member> Flor</member> el proceso (j) añade los papás de `Flor`, ver un ejemplo más detallado en: §4.4.1.

2. En k) se verifica si la relación causa redundancia en las relaciones implícitas de C y de B de no ser redundante, el algoritmo *verRedun*() devuelve “falso” de lo contrario devuelve “verdadero”, de ser verdadero se añade la nueva relación implícita en C.
3. El inciso l) significa que ha provocado redundancia en C, por lo que o) copia los hijos de *cms* en la C véase §3.7.3.1.
4. En m) si no hay redundancia por C, entonces lo hay por B por lo que se adiciona la nueva relación implícita, se añaden los nuevos papás del concepto que se indica en la relación (se añaden los papás de `Flor` en el ejemplo anterior) y se elimina la relación implícita que es redundante en C, ver §3.7.3.2.
5. En n) se busca la sinonimia de relación (si hay un concepto que es sinónimo de la relación de *cms*, en la lista de relaciones de C_C), véase §3.4.6.

Lo más delicado de la copia de las relaciones implícitas en la detección y tratamiento de las relaciones redundantes.

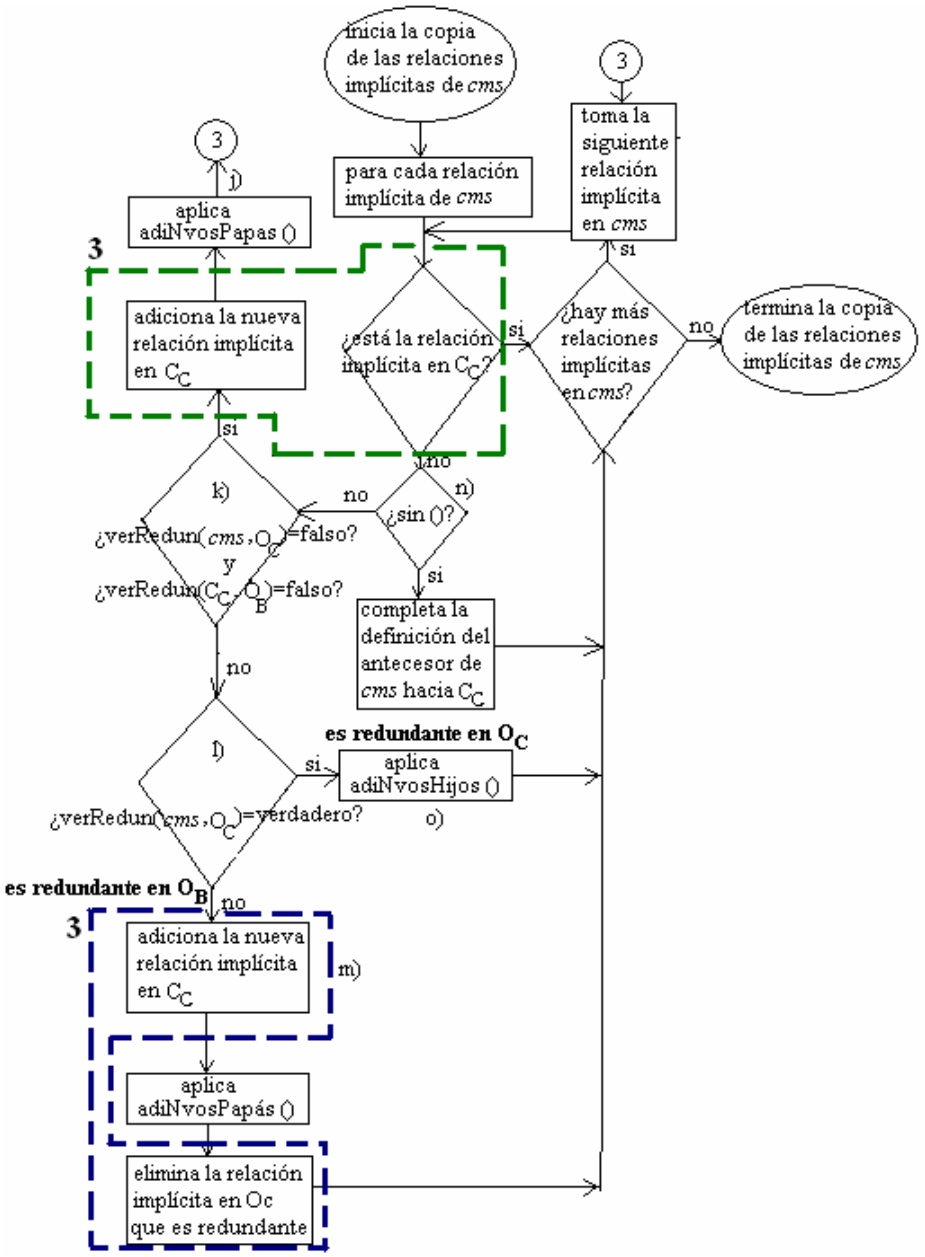


Figura 3.5 Diagrama de flujo de la copia de las relaciones implícitas de *cms* C_C, donde el rectángulo 3 con líneas discontinuas indica los cambios en las relaciones implícitas

Posterior a la copia de las relaciones implícitas se realiza la copia de los hijos de *cms* que no se encuentren en C_C, pero si todos los hijos ya se encuentran continúa con la copia de las particiones que no estén en C_C. En la Figura 3.6 se observan los pasos de la copia de las particiones. Donde:

En r) si no se encuentran los valores de la partición en *cms*, se aplica la búsqueda de sinónimos, véase §3.4.6 si no se han encontrado sinónimos se aplica el algoritmo de la teoría de la confusión [21] que se incluye en p) este algoritmo modifica la lista de valores de la partición a copiar (donde se eligen los conceptos más particulares sobre los más generales, por ejemplo: entre los conceptos

Chiapas y República mexicana, se elige el primero por ser más particular), véase §3.8.3.6.
En q) se verifica si existe un concepto donde se podrá copiar la partición actual (la que se está analizando), es decir, si hay un concepto subconjunto que puede contener la partición para clasificar de mejor manera sus subconjuntos, véase §3.10 si no existe una organización de subconjunto partición, ésta se añade al concepto C_C, si existe, ésta se adiciona al concepto *papá común* encontrado aplicando el algoritmo *subConjPart* (),véase §3.10.

Por ejemplo: el concepto `Persona` con una partición definida `Edad`{0<edad<=1:`bebé`; 1<edad<=10:`niño`; 10<edad<=13:`adolescente`; 13<edad<18:`joven`; 18<=edad<40: `adulto`; 40<=edad<60: `maduro`; 60<=edad: `adulto mayor`}. Un rango es: 0<edad<=1, su valor es: `bebé` y así con los demás rangos y valores de la partición. En la Figura 3.6 el rectángulo 4 con línea discontinua indica los cambios en los conceptos, por ejemplo: se completa la definición del concepto `Persona` en C_C con la de *cms* y se completa la definición de valor `bebé` o `niño`, etc. con su sinónimo en la partición de *cms*, el rectángulo 5 con línea discontinua encierra los cambios que se realizan a nivel de las particiones (en el ejemplo: Edad), el rectángulo 6 señala los cambios en los rangos (en el ejemplo: 0<edad<=1)y el 7 indica los cambios a nivel de valores de los rangos (en el ejemplo: `bebé`, `niño`, etc.).

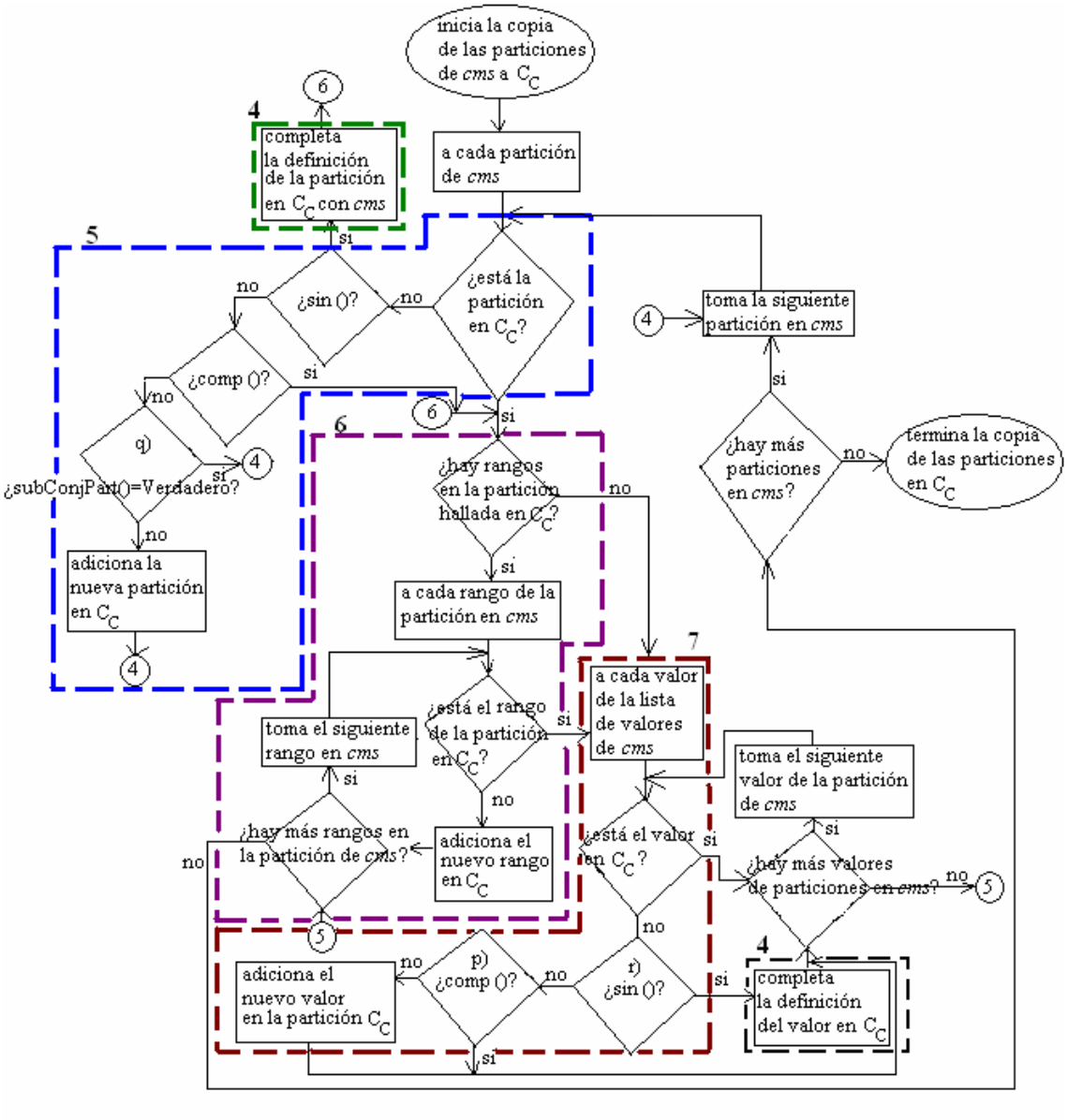


Figura 3.6 Diagrama de la copia de las particiones de un concepto *cms* hacia C_C

3.2.1 Soporte de conocimiento de OM

OM usa algunas bases y recursos de conocimiento incorporados, el cual le ayuda a detectar contradicciones, encontrar sinónimos y similares. Estos son:

1. Una lista de artículos y conectores (el, los, para, y, o, esos, etc.) que se ignoran en el nombre o etiqueta de la relación.
2. Una lista de palabras (sin, excepto, no contiene y otras similares) que cambian (niegan) el significado de una relación, por ejemplo, la relación sin `Espina Tallo` como parte del `Girasol`, significa que no existirá un concepto `Espina` como parte del `Tallo` del `Girasol`.
3. Una jerarquía de conceptos para facilitar la computación de las inconsistencias,

mientras más conceptos tenga la jerarquía más posibilidades habrá de resolver más inconsistencias. Esta jerarquía actualmente se incrementa de forma manual.

En un futuro cercano,

4. OM consultará algunos recursos externos tales como: WordNet [12], diccionarios o tesauros electrónicos. Podrá usarse el algoritmo HCONE [26] para relacionar las ontologías fuente con WordNet.
5. OM contará con una "ontología base" inicial que de seguro ayudará a hacer una mejor fusión. Esta ontología puede ir creciendo mediante el sencillo recursos de usar las ontologías que OM mismo vaya produciendo!

3.2.2 Descripción de los algoritmos

El algoritmo OM está compuesto por trece algoritmos que realizan un papel particular en la fusión de dos ontologías. A continuación se definen cada uno de ellos.

- El algoritmo *com*(C , O) busca el *concepto más similar* llamado *cms* a C (un concepto) en la ontología O, este algoritmo devuelve:

  a) El concepto *cms* en O,
  b) *vs,* un número entre cero y uno, que refleja el valor de la similitud entre C y *cms*. *vs* = 0 indica que no hay similaridad de ningún concepto en O con C, en este caso *cms* no existe.
  c) El mensaje llamado *msg* conteniendo una palabra (ejemplo: "caso A") que corresponde al caso (de COM [7]) que se usó para encontrar este *cms*.

  Una explicación más detallada se encuentra en §3.3.

- El algoritmo *conf*(*r*, *s*, J) usa la jerarquía J (que contiene a los conceptos *r* y *s*) para obtener el grado de confusión [21] de usar *r* en vez de *s.* Este valor es un número entre cero y uno. *conf* no es una función simétrica. Un ejemplo se detalla en §3.8.2.

- El algoritmo *ver*(lis_{Rels}) verifica cada relación r_C de una lista de relaciones lis_{Rels} (las relaciones del concepto que se está copiando). Si el nombre $C_{nombreC}$ de r_C contiene palabras que denotan ausencia tales como "sin", "excepto", "con ausencia de", "no permitido", "prohibido", por citar algunas, entonces los valores de la relación r_C se añadirán a una lista de conceptos no permitidos $lis_{ConceptosNoPermitidos}$ y estos no serán copiados como conceptos con argumentos en C. Este algoritmo devuelve $lis_{ConceptosNoPermitidos}$. Ejemplo: r_C = (sin `amapola`, `peciolo`). El nombre de r_C es "sin", por lo que `peciolo` se añade a la lista de conceptos $lis_{ConceptosNoPermitidos}$ y no se copiará a C. Se presentan detalles descritos en §4.4.3.

- El algoritmo *chec*(r_{Fuente}, O) checa en la relación r_{Fuente} (la que se quiere copiar a la ontología destino O –la que será el resultado de la unión) si su nombre $C_{nombreFuente}$ contiene alguna palabra que denota ausencia, como: "sin", "excepto", "con ausencia de", "no permitido", "prohibido" por citar algunas, entonces se busca cada elemento de la lista lis_{Fuente} (son los valores de r_{Fuente}) en O, si se encuentran devuelve "verdadero" (significa "no los copiarás a O"); si no, devuelve "falso" (significa "los copiará a O"). Un ejemplo se presenta en §4.4.3.

- El algoritmo *adiRels*(r_{Fuente}, lis_{RelsC}, O_{Fuente}, $O_{Destino}$) adiciona la relación r_{Fuente} (la que se quiere copiar) de la ontología O_{Fuente} a la lista de relaciones lis_{RelsC} de la ontología $O_{Destino}$. (la ontología resultante).
  1) Si el nombre $C_{nombreFuente}$ de la relación fuente r_{Fuente} es un concepto entonces:
     a. Realiza *com*($C_{nombreFuente}$, $O_{Destino}$).
     b. Si *com* devuelve *msg* <> "*Caso A"* entonces,
        (i) Añade $C_{nombreFuente}$ a $O_{Destino}$. (no tenía el nombre en $O_{Destino}$, lo añade)
        (ii) Toma los papás del concepto $C_{nombreFuente}$ en la ontología O_{Fuente} y realiza la función *adiNvosPapas*($lis_{PadresCnombreFuente}$, $C_{nombreFuente}$, $O_{Destino}$).[41]
        (iii) Escribe en $C_{nombreFuente}$ la referencia[42] al concepto añadido, ve a (d)
     c. Si *msg* = "*Caso A"* entonces, escribe en $C_{nombreFuente}$ la referencia al concepto hallado[43]
     d. A cada valor $C_{valorFuente}$ de la lista lis_{Fuente} en la relación r_{Fuente},
        (i) Si $C_{valorFuente}$ es un concepto entonces
            (1º) Aplica *com*($C_{valorFuente}$, $O_{Destino}$)
            (2º) Si *msg* <> "*Caso A"* entonces,
                (i) Toma los papás de $C_{nombreFuente}$ en O_{Fuente} y realiza *adiNvosPapas*($lis_{PadresCvalorFuente}$, $C_{valorFuente}$, $O_{Destino}$).
  2) Si no, adiciona la relación r_{Fuente} a la lista de relaciones lis_{RelsC} [44]

  Un ejemplo detallado y graficado se presenta en §4.2.3 en las figuras 5.9 y 5.10 y en §3.7.3.2

[41] El valor de la relación, que es un concepto, no lo tiene C. Por tanto, se copia a C pero necesita colgarlo de algún lugar. Se buscan los papás de ese concepto y los copiará a C, siempre y cuando no los tenga.

[42] Ejemplo: se esta copiando Hidrología (Oaxaca, Hidrología de Oaxaca) de B a C. La relación es Hidrología. Su valor es Hidrología de Oaxaca. En el paso 1, *com* no da "caso A". Entonces, se añade Hidrología a C (añade un nuevo concepto, Hidrología, a C). Se estaba copiando a C **Hidrología** (Oaxaca, Hidrología de Oaxaca). Entonces, en el nombre azul, se hace referencia a este nuevo nodo que se acaba de añadir a C. Es decir, **Hidrología** ya no es un pre-concepto, sino que es un concepto "completo", con argumentos.

[43] El caso A significa que el concepto (Hidrología en nuestro ejemplo) ya se encuentra en C, por lo que solo es necesario al copiar la relación, indicar que esa relación **Hidrología** *es* o se refiere a ese concepto (o sea, no es un conceptos sin argumentos).

[44] Esta adición se hace si se cumple (1) o no.

- El algoritmo *dep*(lis_{valor}, O) depura una lista de conceptos lis_{valor} , eliminando de ella los conceptos que son antecesores de algún otro que también se encuentre en esa lista. Es decir, conserva solo los más particulares. *dep* verifica si en la lista existen antecesores y sucesores buscados en O, es decir si cada concepto C_{valor1} de la lista lis_{valor} es antecesor de cada concepto C_{valor2} de la misma lista, el primero será eliminado de lis_{valor}. Este algoritmo devuelve la lista lis_{valor} depurada. Un ejemplo se presenta en §3.8.3.5.

- El algoritmo *comp*($lis_{valorFuente}$, $lis_{valorDestino}$) toma dos listas de conceptos: $lis_{valorFuente}$ de B y $lis_{valorDestino}$ de C. (Estas listas son valores de alguna relación). Si algún concepto en $lis_{valorFuente}$ es más específico (es hijo, nieto…) que otro concepto en la lista $lis_{valorDestino}$, lo copia a $lis_{valorDestino}$, y borra de $lis_{valorDestino}$ el concepto más general. Es decir, *comp* regresa una lista de conceptos "lo más específicos posibles". Un ejemplo se presenta en §3.8.3.6. Este proceso se realiza de la siguiente manera:

  1) Toma el $C_{valorFuente}$
  2) Si *conf*($C_{valorFuente}$, $C_{valorDestino1}$, *J*) = *conf*($C_{valorDestino1}$, $C_{valorFuente}$, *J*) entonces ve a (5)
  3) Si *conf*($C_{valorFuente}$, $C_{valorDestino1}$, *J*) < *conf*($C_{valorDestino1}$, $C_{valorFuente}$, *J*) entonces ve a (5)
  4) Si *conf*($C_{valorFuente}$, $C_{valorDestino1}$, *J*) > *conf*($C_{valorDestino1}$, $C_{valorFuente}$, *J*) entonces
     a. Almacena temporalmente la posición del concepto $C_{valorDestino1}$ en la lista.
     b. Almacena como menor este valor de la confusión *vc*.
  5) Toma el siguiente elemento de la lista $lis_{valorDestino}$.
  6) Si se ha acabado de recorrer la lista $lis_{valorDestino}$ entonces toma el menor valor de *vc* y re-escribe $C_{valorFuente}$ en la posición guardada temporalmente $C_{valorDestino1}$.
  7) Devuelve la lista $lis_{valorDestino}$.

  Un ejemplo se detalla en §3.8.3.6 y se representa en el cuadro 4.1.

- El algoritmo *sin*(C_{fuente}, O) devuelve (si existe) un concepto sinónimo de C_{fuente} (el concepto en B que se quiere copiar,) en la ontología O (que es la ontología destino). Este algoritmo se sobre-escribe[45] con los siguientes:
  1) *sin*(C_{fuente}, lis_{elem},O) devuelve un concepto sinónimo de C_{fuente} buscado en lis_{elem} de la O,
  2) *sin*(C_{fuente}, $C_{destino}$,O) devuelve verdadero si C_{fuente} y $C_{destino}$ son sinónimos en O,
  3) *sin*(ri_C, $lis_{RelsImpC}$, O) busca el sinónimo de ri_C en $Lis_{RelsImpC}$ de la ontología O devolviendo un objeto sinónimo $ri_{sinónimo}$ que contiene $lis_{PadreSinónimo}$ ($C_{PadreSinónimo1}$, $C_{PadreSinónimo2}$, … , $C_{PadreSinónimoN}$) y

[45] Un algoritmo *alg* sobre-escribe o redefine a otro algoritmo *alg2*, dependiendo de la lista de argumentos que se le pasen.

4) *sin*(C_{fuente}, O_{fuente}, $C_{destino}$,$O_{destino}$) devuelve verdadero si C_{fuente} en la O_{fuente} y $C_{destino}$ en la $O_{destino}$ son sinónimos, si no devuelve falso.

La sinonimia se obtiene comparando las definiciones de los conceptos C involucrados. Más detalles sobre el funcionamiento de este algoritmo se encuentra en §3.4.6 y un ejemplo práctico está en §4.2.2.

- El algoritmo *extrae*(C_{nombre}, lis_{Rels}) toma un concepto C_{nombre}, a la etiqueta de éste le extrae los artículos, conectores y demás elementos tales como: "a","la", "el", "con", "los", "sin", "para", "por", "y", por citar algunos. Por ejemplo si C_{nombre} contiene la frase: "colinda de norte a sur", al aplicar este algoritmo la frase resultante es: "colinda norte". Este proceso se realiza para cada uno de los conceptos de la lista lis_{Rels}, luego va comparando la frase resultante del concepto C_{nombre} con cada una de las frases resultantes de cada concepto de la lista lis_{Rels} devolviendo verdadero si son iguales o falso si no lo son. El proceso se realiza de la siguiente manera:

  1) Extrae los artículos y conectores en la etiqueta del concepto C_{nombre}.
  2) Extrae los artículos y conectores en la etiqueta de cada concepto $C_{nombrelis}$ de la lista de conceptos lis_{Rels}.
  3) Busca C_{nombre} en la lista de relaciones lis_{Rels}. Si se encuentra devuelve "verdadero", si no, devuelve "falso"

  Un ejemplo del funcionamiento de este algoritmo aplicado a un caso práctico se presenta en §3.4.4.1.

- El algoritmo *adiNvosPapas*(lis_{Padres}, C, O) añade nuevos papás (antecesores directos) que están en la lista de conceptos lis_{Padres} a un concepto C (el que se encuentra copiando) en una ontología O (la ontología destino). El procedimiento es el siguiente. Los papás en lisPadres vienen del concepto *cms* más similar a C.

  A cada concepto padre C_{Padre} de la lista de conceptos padres lis_{Padres}.

  a. Aplica *com*(C_{Padre}, O)[46]
  b. Si *msg* = "*Caso A*" o "*Caso C"* (el papá se ha encontrado en la ontología destino O) entonces, coloca la referencia[47] del concepto C (actualmente copiando) al concepto $C_{PadreHallado}$ en la O, procesa el siguiente concepto padre C_{Padre}.
  c. Si *msg* <> "*Caso A"* o "*Caso C"* entonces

[46] Busca (usando COM [22]) el papá en la ontología destino, si se encuentra "cuelga" de él, el concepto C que se está copiando, sino busca el abuelo de C (en la ontología destino) y aplica lo mismo, si se encuentra "cuelga" de él, el papá, luego el concepto C, sino toma el bisabuelo y así sucesivamente hasta llegar a la raíz. Si no encontró a un solo antecesor, "cuelga" toda la ascendencia (papás, abuelos, bisabuelos, etc) de C en el concepto raíz.

[47] El padre se encuentra en la ontología destino, ahora el concepto C que se está copiando será hijo de este padre encontrado, C podrá tener uno o varios padres. Entonces, este proceso hace que C "señale" (como hijo) al papá hallado en la ontología destino.

(i) Si $C_{Padre} = O_{Raíz}$ entonces
(1º) Añade C como hijo de C_{Padre}
(2º) Coloca la referencia[48] del concepto C (actualmente copiando) al concepto C_{Padre} en la ontología destino O, procesa el siguiente concepto padre C_{Padre}.
(ii) (cuando C_{Padre} no es raíz): Toma los abuelos[49] de cada concepto C_{Padre} de la lista lis_{Padres}, procesa el siguiente concepto padre C_{Padre}.

Un ejemplo práctico se presenta en §4.5.2 en la Figura **4.36** donde el concepto C se ha copiado como hijo de la raíz y otro ejemplo se detalla en §4.4.1 en la Figura 4.23 en la cual, se ha encontrado el abuelo del concepto C y debajo de éste se ha copiado el papá de C.

- El algoritmo *adiNvosHijos*(C, O) añade nuevos hijos del concepto C (actualmente copiándose) a la ontología destino O:

  A cada hijo del concepto C:

  a. Aplica *com*($C_{HijoFuente}$, O)[50] [$C_{HijoFuente}$ es el hijo del concepto C]
  b. Si *msg* = "*Caso D"* o "*Caso B"* entonces (el hijo no se ha encontrado en la ontología destino O)
     (i) Si $C_{HijoFuente}$ se encuentra en la lista de conceptos que no se deben copiar $lis_{ConceptosNoPermitidos}$ entonces ve a (1) [toma el siguiente hijo]
     (ii) Si no se encuentra en $lis_{ConceptosNoPermitidos}$ (se debe copiar a O) entonces, añade el concepto $C_{HijoFuente}$ no hallado a la ontología destino O y procesa el siguiente hijo $C_{HijoFuente}$.
  c. Si *msg* = "*Caso A"* o *"Caso C"* (se ha encontrado) entonces, adiciona nuevos papás del concepto que se ha hallado de esta forma: *adiNvosPapas*($lis_{PadresHijoHallado}$, O).

Un ejemplo se detalla en §3.7.3.1.

- El algoritmo *verRedun*(ri_{fuente}, $lis_{PadresDestino}$, O) verifica si la relación implícita (definida en §3.7.1) ri_{fuente} del concepto C que se está copiando, genera una relación redundante (esto es, se encuentra entre la lista de papás $lis_{PadresDestino}$ de C) en la ontología destino O; es decir, se busca la relación implícita ri_{fuente} en la lista de conceptos $lis_{PadresC}$ devolviendo "verdadero" si se encuentra o "falso" si no se encuentra. El proceso a seguir es:

[48] Esta parte del algoritmo indica que el concepto papá del concepto C es la raíz (hijo del concepto raíz).
[49] Toma cada uno de los abuelos de C, los busca en la ontología destino, si no los encuentra, sigue con los bisabuelos y así hasta encontrar un antecesor y de ahí copia toda la descendencia de C en la ontología destino. Esta copia se llama *copia recursiva de antecesores* donde cada uno de estos antecesores se copia en la ontología destino de manera *superficial* es decir, solo se copian sus definiciones (palabras que lo describen) y sus relaciones implícitas (miembro de, parte de, subconjunto, tipo de), posteriormente se copiarán de forma *completa*, es decir, sus relaciones explícitas (propiedades, características, acciones, etc).
[50] Busca cada hijo del concepto C (que se encuentra copiando) en la ontología destino, el hijo que no esté lo copia a la ontología destino, el que esté le pone una referencia (señalamiento, apuntador) indicando que éste es hijo del concepto C.

A cada concepto padre C_{Padre} de la lista de padres $lis_{PadresDestino}$ del concepto C que se encuentra en pleno proceso de copiado,

a. Busca la relación implícita ri_{fuente} del concepto C en la lista de relaciones implícitas del concepto padre C_{Padre}[51].
b. Si se encuentra, devuelve "verdadero".
c. Si no se encuentra
   (i) Si $C_{Padre} = O_{Raíz}$ (ya ha llegado al concepto raíz $O_{Raíz}$, por lo tanto, no se encontró una relación redundante) entonces devuelve falso.
   (ii) Si no, toma la lista de padres (del padre C_{Padre} recién analizado) $lis_{PadresC}$ y procesa la siguiente relación implícita ri_{fuente}.

Un ejemplo detallado se presenta en §4.3.1.

- El algoritmo *subConjPart*(P_{fuente}, O) busca en la ontología destino O los conceptos de la partición (definido en §4.2.6) P_{fuente} del concepto C (actualmente copiándose). Si todos los elementos de P_{fuente} son hallados en O y además son hermanos entre sí (tienen el mismo papá) entonces copia la partición p_{fuente} del concepto C en el papá de todos los hermanos y devuelve verdadero, de lo contrario devuelve falso (si al menos un concepto no comparte el mismo papá). El procedimiento es el siguiente:

  A cada concepto de la partición P_{fuente} de C

  a. Inicia ConteoHnos[52] a 0
  b. Aplica *com*(C_{valorC}[53], O)
  c. Si *msg* = "*Caso A"* o "*Caso C"* entonces
     (i) Si *cms*[54] se encuentra en la lista de hermanos encontrados en O $lis_{HermanosC}$ entonces incrementa ConteoHnos en 1, toma el siguiente concepto de la partición P_{fuente} y búscalo en O (paso b).
     (ii) Si la lista de conceptos de la partición P_{fuente} se ha terminado entonces
        (1º) Obtiene el $C_{PapáComun}$ de todos los conceptos hermanos $lis_{HermanosC}$

[51] Ejemplo: suponiendo que la relación implícita a buscar es: "miembro de" `Herramienta` del concepto `Martillo de carpintero` si el la ontología fuente hay un concepto `Martillo` que es papá de `Martillo de carpintero` y su abuelo es: `Herramienta` ahí se podrá encontrar la relación implícita buscada, porque `Martillo` es un miembro de `Herramienta` pero si se ha definido como "tipo de" `Herramienta` no se considera una relación redundante; es decir hay un concepto `Martillo` que es una instancia de `Herramienta` y también es un tipo de `Herramienta` es aceptable.

[52] Esta variable llevará la cuenta de los hermanos encontrados en la ontología destino, para después comparar esta cantidad con el número total de hijos del papá de todos los hermanos. Si coincide, el algoritmo devolverá verdadero, si no devolverá falso, es decir, si todos los conceptos de la partición se encontraron en la ontología destino y tienen un papá en común pero este papá tiene un (o varios) hijo(s) que no está(n) en la partición entonces devuelve falso.

[53] C_{ValorC} es el concepto de la partición que se quiere encontrar en la ontología destino.

[54] La presencia del concepto más similar *cms* indica que halló uno de los conceptos de la partición.

(2º) Si todos los conceptos hermanos de $lis_{HermanosC}$ son hijos de $C_{PapáComun}$ entonces, añade la partición P_{fuente} en el concepto $C_{PapáComun}$ y devuelve "verdadero"

d. Si no, devuelve "falso".

Un ejemplo detallado se presenta en §3.10 y en §4.2.6.

- El algoritmo *ext*($lis_{Relsc\mathit{ms}}$, lis_{RelsC}) de OM copia y complementa las relaciones explícitas (definida en §3.7.2) de la lista de relaciones $lis_{Relsc\mathit{ms}}$ del concepto *cms* (concepto más similar en la ontología fuente B, de C la ontología destino) hacia las relaciones explícitas lis_{RelsC} del concepto destino C_C. Este algoritmo devuelve una lista de relaciones lis_{RelsC} complementadas[55] o añadidas[56] en la ontología destino C. El proceso de fusión de OM lo realiza este algoritmo *ext*. En lo sucesivo este algoritmo se llamará también OM:

1) C← A (se copia toda la ontología A hacia C, ahora C será la ontología destino, al principio es idéntica a A.).
2) $C_{Raíz} \leftarrow C_C$ (inicialmente el concepto C_C es la raíz de la ontología C luego, C_C será cada uno de los descendientes de C_C, de modo que cada nodo en la ontología C se convierta en C_C.
   a. Realiza *com*(C_C, B) para buscar el *cms* de Cc (en la ontología B).
   b. Si *msg* = "*Caso A"* o "*Caso C"* entonces [C_C será complementado[55] o añadido[56] como sigue]; de otra forma ve a 3).
      (i) Completa[55] la definición de *cms* con la definición de C_C (el concepto que se está enriqueciendo)
      (ii) Llena la lista de conceptos no permitidos $lis_{ConceptosNoPermitidos}$ realizando la función: *ver*(lis_{RelsC}).
      (iii) A cada **relación explícita** de la lista de relaciones $lis_{Relsc\mathit{ms}}$ de *cms* se le aplica lo siguiente:
         (1º) Si el nombre $C_{nombre\mathit{cms}}$ de la relación explícita se encuentra en la lista lis_{RelsC} entonces
            (01) Si este nombre $C_{nombreC}$ señala[57] a un concepto con aridad multivaluada entonces
               (Uno) Realiza: *dep*($lis_{valor\mathit{cms}}$, B) y *dep*(lis_{valorC}, C).[58]
               (Dos) A cada concepto en la lista $lis_{valor\mathit{cms}}$ de los valores de la relación encontrada de *cms*:
                  I. Busca el concepto $C_{valor\mathit{cms}}$ en la lista lis_{valorC}.
                  II. Si lo encuentra entonces toma otro concepto valor $C_{valor\mathit{cms}}$, ve a (2.b.1º.01.Dos).

[55] Complementar consiste en enriquecer cada relación en la lista lis_{RelsC} con su relación equivalente en la lista de relaciones $lis_{RelsCMS}$ incluyendo los conceptos nuevos que no se hallaron en la ontología destino (éstos fueron copiados *superficialmente*, es decir solo se copió su descripción y sus relaciones implícitas).

[56] Añadir nuevas relaciones consiste en crear nuevas relaciones de la lista de relaciones fuente $lis_{Relsc\mathit{ms}}$ hacia la lista de relaciones destino lis_{RelsC} incluyendo los conceptos nuevos.

[57] El nombre de la relación no solo es una etiqueta sino apunta o señala a un concepto.

[58] Se depuran las listas de conceptos (valores de una relación en *cms* y C_C) eliminando los antecesores que se encuentren en cada relación, conservando únicamente los conceptos más específicos, hijos, nietos, etc.

III. Si no lo encuentra

A. Si el algoritmo *sin*($C_{valor cms}$, lis_{valorC}, C) devuelve un concepto sinónimo entonces

- Complementa la definición del concepto $C_{valor cms}$ con la de C_{valorC} hallado, toma otro concepto y ve a (2.b.iii.1º.01.Dos.I).

B. Si no devuelve un sinónimo entonces realiza: *comp*($lis_{valor cms}$,lis_{valorC}), para conservar en lis_{valorC} solo los conceptos más específicos, toma el siguiente concepto y ve a (e.b.iii.1º.Dos.I).

(Tres) Cuando no haya más conceptos en la lista $lis_{valor cms}$, toma la siguiente relación en la lista de relaciones $lis_{Relscms}$ de *cms* y regresa a (2.b.iii.1º)

(02) Si el nombre $C_{nombreC}$ de la relación considerada señala a un concepto con aridad monovaluada (§3.9) entonces

(Uno) Si *conf*(C_{valorC}, $C_{valor cms}$, *J*) > *conf*($C_{valor cms}$, C_{valorC}, *J*) [donde J es la jerarquía §3.2.1] entonces re-escribe el concepto más específico $C_{valor cms}$ en el concepto más general C_{valorC}. En todo caso, toma la siguiente relación en la lista de relaciones $lis_{Relscms}$ de *cms* y regresa a (2.b.iii.1º).

(2º) Si el nombre de la relación $C_{nombrecms}$ de la relación considerada no se encuentra en la lista de relaciones lis_{RelsC} de C_C entonces

(01) Si el algoritmo: *extrae*($C_{nombrecms}$,lis_{RelsC}) [para eliminar los artículos y conectores de la frase §3.4.4.1] devuelve “verdadero” entonces regresa a (2.b.iii.1º.01)

(02) Si no [*extrae*($C_{nombrecms}$,lis_{RelsC}) devuelve “falso”]:

(Uno) Si el algoritmo *sin*($C_{nombrecms}$, lis_{RelsC},C) devuelve un sinónimo, entonces complementa la definición de la relación $C_{nombrecms}$ con la de la relación $C_{nombreC}$ hallada y regresa a (2.b.iii.1º.01)

(Dos) Si no devuelve sinónimo,

I. Si *conf*($C_{nombreC}$, $C_{nombrecms}$, *J*) > *conf*($C_{nombrecms}$, $C_{nombreC}$, *J*) entonces re-escribe el concepto más específico $C_{nombrecms}$ sobre el concepto más general $C_{nombreC}$ ve a (2.b.iii.1º.01).

II. Si no es mayor,

A. Si el algoritmo *chec*(r_{cms}, C), devuelve “verdadero” entonces toma la siguiente relación de la lista de relaciones $lis_{Relscms}$ de *cms*, regresa a (2.b.iii.1º).

B. Si no [*chec*(r_{cms}, C) devolvió “falso”], agrega la relación, así: *adiRels*(r_{cms}, lis_{RelsC}, B, C) y regresa a (2.b.iii.1º).

(iv) Una vez terminada la lista de relaciones explícitas, a cada **relación implícita** ri_{cms} de la lista de relaciones implícitas $lis_{RelsImpcms}$ de *cms* se le aplica lo siguiente:

(1º) [Copia las relaciones implícitas de *cms* que no estén en C_C así:] Busca esta relación implícita ri_{cms} en la lista de relaciones implícitas $lis_{RelsImpC}$ de C_C.

(Uno) Si encuentra la relación implícita ri_{cms} en la lista de relaciones implícitas $lis_{RelsImpC}$ de C entonces, toma la siguiente relación implícita y ve a (2.b.iv.1º).

(Dos) Si no está la relación implícita ri_{cms} entonces

I. Si el algoritmo *sin*(ri_{cms}, $lis_{RelsImpC}$, C) devuelve una relación implícita sinónima entonces, completa la definición del sinónimo encontrado $ri_{sinónimo}$ con la definición de $ri_{cms,}$ procesa la siguiente relación implícita ri_{cms} y regresa a (2.b.iv.1º).

II. Si no halló una relación implícita sinónima entonces

- Verifica si hay relaciones redundantes: Si al realizar *verRedun*(ri_{cms}, $lis_{PadresC}$, C) y *verRedun*(ri_C, lis_{Padres}*cms*, B) [donde ri_C es la relación implícita de C_C] devuelven "falso" entonces [la relación implícita que se desea añadir no provoca redundancia en C].

  * Adiciona la relación implícita ri_{cms} a la lista de relaciones implícitas $Lis_{RelsImpC}$ de $C_{C.}$

  * Adiciona los nuevos papás de *cms* a C, así: *adiNvosPapas*(lis_{Padres}*cms*, C), toma la siguiente relación implícita y regresa a (2.b.iv.1º).

- Si el algoritmo: *verRedun*(ri_{cms},$lis_{PadresC}$,C) devuelve "verdadero" entonces, (*Es transitivo en A*) copia los hijos así: *adiNvosHijos*(C_{cms}, C), toma la siguiente relación implícita y regresa a (2.b.iv.1º).

- Si el algoritmo: *verRedun*(ri_C,lis_{Padres}*cms*,B) devuelve "verdadero" entonces (*Es transitivo en B*),

  * Adiciona la relación implícita ri_{cms} a la lista de relaciones implícitas $lis_{RelsImpC}$ de C_C.

  * Adiciona los nuevos papás, así: adiNvosPapas($lis_{Padrescms}$,C)

  * Elimina la relación implícita ri_C de la lista de relaciones implícitas $lis_{RelsImpC}$ en C_C, toma la siguiente relación implícita ri_{cms} de *cms* y regresa a (2.b.iv.1º).

(v) Si se ha terminado la lista de relaciones implícitas $lis_{RelsImpcms}$ de *cms*, adiciona **los nuevos hijos** de *cms* que no tenga C_C así:

(1º) Si el concepto *cms* tiene hijos entonces aplica el algoritmo *adiNvosHijos*(C_{cms}, lis_{HijosC}, C) [Copia nuevos hijos de *cms* que no estén en C_C]; en todo caso ve a (2.b.v.2º.01).

(2º) Si no tiene hijos entonces

(01) A cada **partición** p_{cms} de la lista de particiones lis_{Part}*cms* de *cms*: (Copia las particiones de *cms* que no estén en C_C)

(Uno) Si la partición p_{cms} de *cms* se encuentra en la lista de particiones lis_{PartC} de C_C entonces

I. Si existen rangos $lis_{Rango cms}$ en la partición p_{cms} entonces
   A. A cada rango de la lista $lis_{Rango cms}$ de la partición p_{cms} de *cms*
      - Si se encuentra el rango $C_{Rango cms}$ en la lista de rangos lis_{RangoC} de C_C entonces busca cada valor $C_{valorRango cms}$ de la lista de valores $lis_{valorRango cms}$ en la lista de rangos $lis_{valorRangoC}$ de C_C.
        1) Si se encuentra, procesa el siguiente valor $C_{valorRango cms}$ y buscalo en la lista $lis_{valorRangoC}$ de C_C.
        2) Si no se encuentra
        * Si el algoritmo *sin*($C_{valorRango cms}$, $lis_{valorRangoC}$,C) **devuelve un sinónimo** entonces complementa la definición del valor $C_{valorRango cms}$ y el valor sinónimo encontrado $C_{valorRangoSinónimo}$, procesa el siguiente valor $C_{valorRango cms}$ y búscalo en $lis_{valorRangoC}$. Al terminar todos los valores de la lista $lis_{valorRango cms}$, procesa el siguiente rango y regresa a (2.b.v.2º.01.Uno.I.A).
        * Si **no devuelve un sinónimo** entonces realiza el algoritmo *comp*($lis_{valorRango cms}$, $lis_{valorRangoC}$) [aquí se añaden valores más específicos sobre valores más generales], procesa el siguiente valor $C_{valorRango cms}$ y búscalo en $lis_{valorRangoC}$. Al terminar todos los valores de la lista $lis_{valorRango cms}$, procesa el siguiente rango y regresa a (2.b.v.2º.01.Uno.I.A), -- Si al realizarse el algoritmo *comp*($lis_{valorRango cms}$, $lis_{valorRangoC}$), no se añadieron valores más específicos sobre valores más generales en la lista $lis_{valorRangoC}$ entonces, añade el valor $C_{valorRango cms}$ a la lista $lis_{valorRangoC}$ en C_C, procesa el siguiente rango. En todo caso, regresa a (2.b.v.2º.01.Uno.I.A).
      - Si no se encuentra el rango $C_{Rango cms}$ en la lista de rangos lis_{RangoC} de C_C entonces, añade el rango $C_{Rango cms}$ a la lista de rangos lis_{RangoC} de C_C.
      - Si se ha terminado la lista de rangos $lis_{Rangocms}$ de *cms* entonces, toma la siguiente partición p_{cms} de la lista $lis_{Partcms}$ de *cms* y regresa a (2.b.v.2º.01)

II. Si no existen rangos entonces en la partición p_{cms} entonces
   A. Busca cada valor $C_{valorcms}$ de la lista de valores $lis_{valor cms}$ de *cms* en la lista de valores lis_{valorC} de C_C.
      - Si se encuentra $C_{valor cms}$, toma el siguiente valor $C_{valor cms}$ y regresa a (2.b.v.2º.01.Uno.II.A)
      - Si no se encuentra $C_{valor cms}$
        * Si el algoritmo *sin*($C_{valor cms}$, lis_{valorC},C) **devuelve un sinónimo** entonces
        * * Complementa las definiciones del valor $C_{valor cms}$ y

el valor sinónimo $C_{valorSinónimo}$,

* * Procesa la siguiente partición P_{cms} de la lista de particiones $lis_{Partcms}$ de *cms* y regresa a (2.b.v.2º.01.Uno).

* Si **no devuelve un sinónimo** entonces

* * Realiza el algoritmo *comp*($lis_{valorcms}$,lis_{valorC}) [para añadir valores más específicos sobre valores más generales].

* * Si al realizarse el algoritmo *comp*($lis_{valorcms}$,lis_{valorC}), no se añadieron valores más específicos sobre valores más generales en la lista lis_{valorC} entonces, añade el valor $C_{valorcms}$ a la lista lis_{valorC} en C_C. En cualquier caso, procesa la siguiente partición P_{cms} de la lista $lis_{Partcms}$ de particiones de *cms* y regresa a (2.b.v.2º.01.Uno).

B. Si se ha terminado de procesar la lista $lis_{valorcms}$, toma la siguiente partición p_{cms} de la lista $lis_{Partcms}$ de *cms* y regresa a (2.b.v.2º.01).

(Dos) Si la partición p_{cms} de *cms* no se encuentra en la lista de lis_{PartC} de C_C entonces

I. Si el algoritmo *sin*(p_{cms}, lis_{PartC},C) **devuelve una partición sinónimo** entonces complementa las definiciones de la partición p_{cms} con las de la partición sinónimo $p_{sinónimo}$, procesa la siguiente partición p_{cms} de la lista $lis_{Partcms}$ de *cms* y ve a (2.b.v.2º.01.Uno.I)

II. Si **no devuelve una partición sinónimo**, ejecuta el algoritmo *comp*($lis_{Partcms}$, lis_{PartC}) [para cambiar valores más específicos sobre valores generales]

A. Si al realizar el algoritmo *comp*($lis_{Partcms}$, lis_{PartC}) se cambiaron valores específicos sobre valores generales, regresa a (2.b.v.2º.01.Uno.I).

B. Si no se se cambiaron valores específicos sobre valores generales entonces,

- Si al ejecutar el algoritmo *subConjPart*(p_{cms}, B) [que organiza subconjuntos en particiones §3.10] devuelve "falso", entonces * añade la partición P_{cms} a la lista de particiones lis_{PartC} de C_C, * procesa la siguiente partición P_{cms} de la lista $lis_{Partcms}$ de *cms* , * regresa a (2.b.v.2º.01.Uno).
- Si al ejecutar el algoritmo *subConjPart*(p_{cms}, B) [que organiza subconjuntos en particiones §3.10] devuelve "verdadero", entonces * procesa la siguiente partición P_{cms} de la lista $lis_{Partcms}$ de *cms* y regresa a (2.b.v.2º.01.Uno).

(02) Si la lista $lis_{Partcms}$ de particiones de *cms* se ha terminado entonces

3) C_C se coloca en el siguiente hijo de C_C y regresa a (2.a).
4) Si todos los nodos de C se han recorrido a profundidad primero, concluyó la fusión de B a C.

3.3 El Comparador de Ontologías Mixtas COM

Se describe el funcionamiento del algoritmo Comparador de Ontologías Mixtas COM original [22] usado para mapear ontologías diferentes. COM permite encontrar los conceptos más similares entre dos ontologías mediante el uso de palabras que pueden ser ambiguas.

En lo que sigue, A y B son ontologías, C_A es un concepto en A (el concepto a buscar en B) y C_B es un concepto en B (el concepto *cms* en B más similar a CA). *pa* son las palabras (o frases en español) asociadas a C_A, que definen a C_A.

P_A es el nodo en A que es papá de C_A. P_B es el nodo o concepto en B que es papá de C_B.

En [22], COM también se llama *hallar.* COM consta de cuatro casos:

3.3.1 Caso A: C_A casa con C_B y P_A casa con P_B

En B se buscan dos conceptos C_B y P_B, de manera que la definición de P_B coincida con la mayoría de las palabras que definen P_A y la mayoría de las palabras que definen C_B coincidan en la definición de C_A el algoritmo devuelve:

* El C_B (conocido también como *cms*) en B,
* El valor *vs* (*valor de similitud*) = entre 0 y 1.

Los sub casos son los siguientes:

1) *Caso A Papá*: C_A casa con C_B y P_A con P_B.
2) *Caso A Abuelo*: C_A casa con C_B pero P_A con A_B, el abuelo de C_B.
3) *Caso A Abuelo simétrico*: C_A casa con C_B pero A_A, el abuelo de C_A con P_B.
4) *Caso A Bisabuelo*: C_A casa con C_B pero P_A con B_B, el bisabuelo de C_B.
5) *Caso A Bisabuelo simétrico*: C_A casa con C_B pero B_A, el bisabuelo de C_A con P_B.

En [22] se encuentran detallados los casos del algoritmo COM original, sin embargo, se dará una explicación de cada uno de los Casos A de forma gráfica en la Figura 3.7.

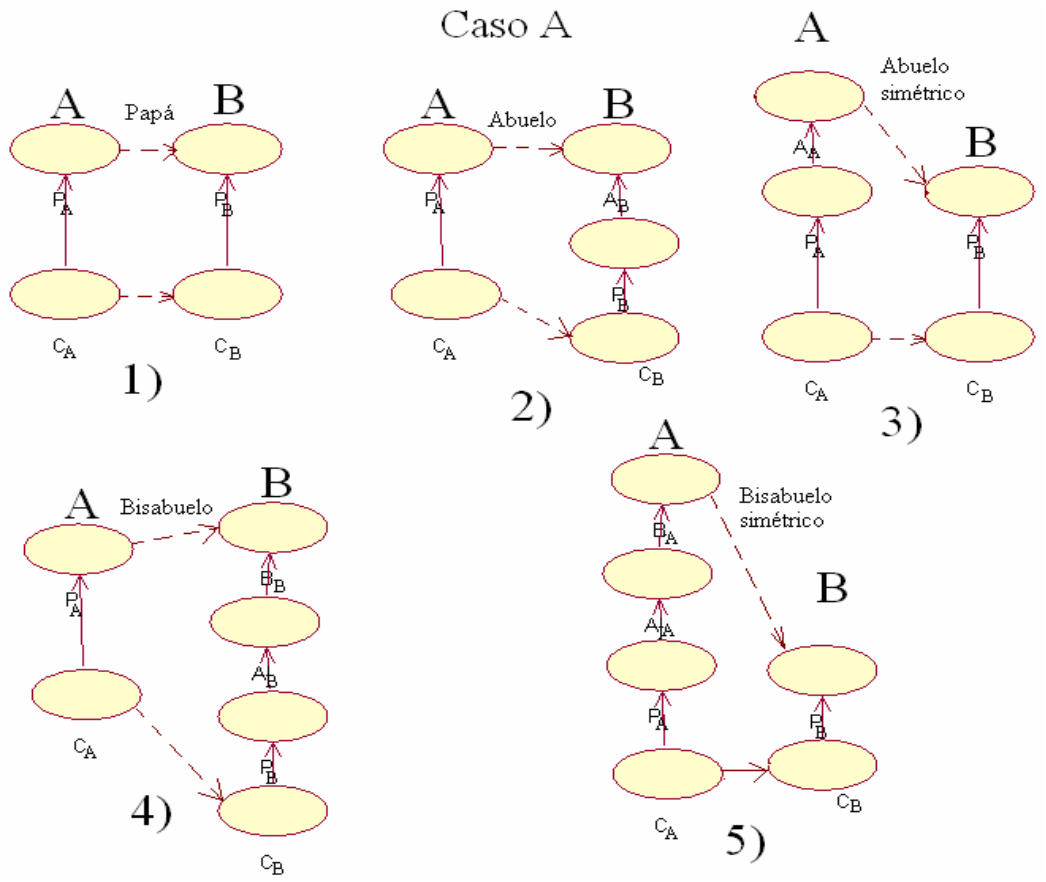


Figura 3.7 Caso A del COM original, donde: se presentan los 5 subcasos del caso A

3.3.2 Caso B: Los papás P_A y P_B coinciden, pero no hay C_B

Se encuentra P_B pero no C_B, se llama recursivamente al proceso *hallar* de la forma P_B' = *hallar*(*pa*) para confirmar que P_B es antecesor de C_A. Si el P_B' hallado es $O_{BRaíz}$ el algoritmo concluye sin éxito; si no, se busca en B para cada hijo de P_B, aquél, cuya mayoría de propiedades y valores, coincidan (recursivamente vía *hallar*) con las correspondientes de C_A. Es decir, se busca en B al hijo de P_B que tenga la mayor cantidad de las propiedades (y valores) de C_A. Si el candidato C_B' tiene además hijos, se ve si éstos coinciden (usando recursivamente *hallar*) con los hijos de C_A. Si se encuentra un C_B' con las propiedades deseadas, el algoritmo concluye con éxito indicando el C_B' encontrado.

En otro caso, se intenta hallar el C_B' entre los hijos del padre (en B) de P_B, es decir, entre los hermanos de P_B; en caso necesario, entre los hijos de los hijos de P_B, o sea, entre los nietos de P_B Si no se encontró un C_B', entonces el más cercano a C_A es P_B, por lo que *hallar* devuelve el mensaje "hijo de P_B " (significa que un hijo de P_B que aún no existe en B es el más similar a C_A) y el algoritmo concluye.

Los sub casos son:

1) *Caso B Hijos*: P_A casa con P_B pero C_A no, se comparan las propiedades de C_A con las de los Hijos de P_B.
2) *Caso B Tíos*: P_A casa con P_B pero C_A no, se comparan las propiedades de C_A con las de los Tíos de P_B si este existiese.
3) *Caso B Nietos*: P_A casa con P_B pero C_A no, se comparan las propiedades de C_A con las de los Nietos de P_B (si éste existiese) y los hermanos de éste en la B, si no coinciden devuelve "Es un hijo de P_B".

En la Figura 3.8 y Figura 3.9 se presentan los casos de COM donde se comparan las propiedades o características de los conceptos, ahí intervienen otros conceptos tales como: Hijos y Tíos. Los nombres y valores de las propiedades deben ser conceptos con argumentos, se comparan los nombres y valores de las relaciones, es decir, las palabras: "Altura" y "altura" son iguales pero "Altura" y "Anchura" son distintos.

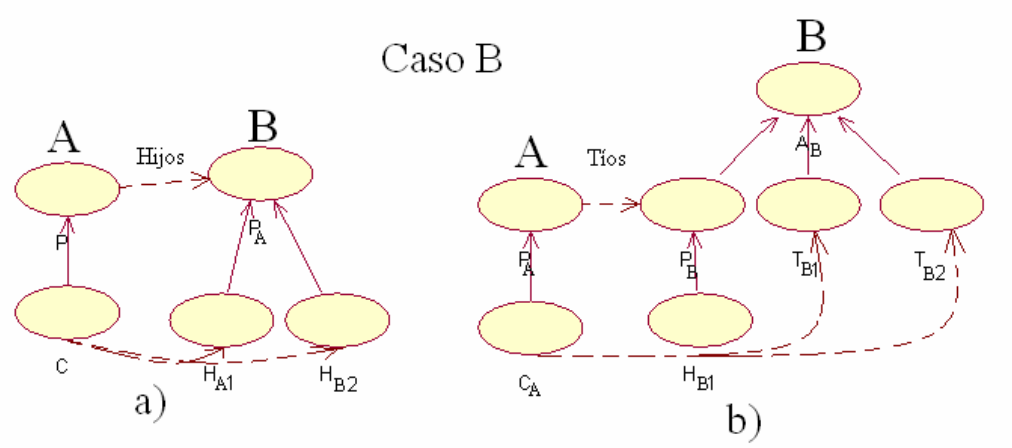


Figura 3.8 Muestra en a) el casamiento de P_A con P_B pero C_A no casó con C_B entonces se comparan las propiedades de C_A con las propiedades de los Hijos del P_B hallado, si coinciden la mayoría de las propiedades de C_A con la mayoría de las propiedades de un Hijo de P_B COM devuelve msg:"Caso B", *cms*=H_{B1} o H_{B2} (el concepto cuya mayoría de propiedades hayan casado con P_A). Lo mismo sucede en b) pero con los hermanos de P_B, es decir los Tíos de C_B si existiesen

Se presenta en la Figura 3.9 el *Caso B* de COM [22] usando la comparación de las propiedades de C_A con las propiedades de los nietos de C_A si éste existiese. Se considera la mayoría cuando coinciden más de la mitad del número de propiedades de C_A con uno de los nietos N_B. Se dice que C_A casa con ese nieto. Si dos nietos N_{B1} y N_{B2} tienen propiedades que casan con la mayoría de las propiedades de C_A, se toma el de mayor casamiento. Si ambas mayorías son iguales, se dice que "C_A casa con un nieto de P_B" y no casa ni con N_{B1} ni con N_{B2}.

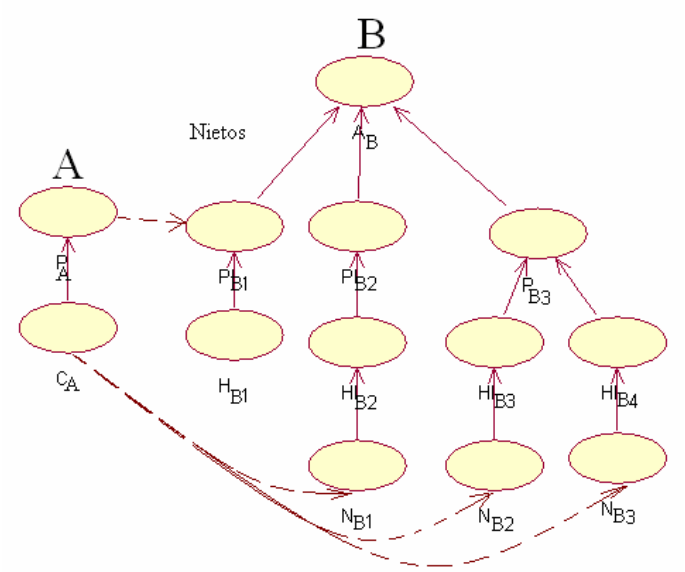


Figura 3.9 Muestra el casamiento de P_A con P_{B1} pero C_A no ha casado con algún concepto en B por lo que se comparan las propiedades de C_A con las propiedades de los Nietos de P_{B1} hallado (si existiesen) y las propiedades de los nietos de los hermanos P_{B1} (si existiesen), si la mayoría de las propiedades de C_A casa con la mayoría de las propiedades de uno de los Nietos de P_{B1}, P_{B2} o P_{B3} COM devuelve: msg = "Nieto de ", *cms* = P_{B1}, P_{B2} o P_{B3} (solo un P_B) cuyo nieto haya casado (la mayoría de las propiedades) con C_A. El *vs* es el mismo que devuelve el COM [22] original

3.3.3 Caso C: coincide C_A con C_B pero no hay P_B

Si se halla C_B pero no P_B, se busca si el abuelo en B de C_B es similar a P_A o si el bisabuelo de C_B en B es similar a P_A (esto ya fue comentado en el Caso A). Si se halla, entonces el concepto en B más similar a P_A es el abuelo o el bisabuelo de C_B y concluye el algoritmo. Si no se halla, se verifica si la mayoría de las propiedades (y sus valores correspondientes) de C_A coinciden con las de C_B y si la mayoría de los hijos de C_A coinciden (usando *hallar*) con la mayoría de los hijos de C_B; si las propiedades y los hijos coinciden, entonces la respuesta es C_B y concluye el algoritmo aunque no se haya hallado en B al P_B que corresponda al concepto P_A en A. Si solamente una parte de propiedades e hijos son similares entonces la respuesta es "probablemente C_B" y concluye el algoritmo. Si ninguna propiedad ni hijos son similares la respuesta es "no existe" y concluye el algoritmo.

Subcaso:

1) Caso C Hijos: C_A casa con C_B pero P_A no, se comparan las propiedades de los hijos de C_A con las propiedades de cada Hijo de C_B si coinciden devuelve C_B.

En la Figura 3.10 se presenta el caso C de COM [22] donde no se ha encontrado el más similar a P_A por lo que la búsqueda se amplía a los hijos del C_B encontrado.

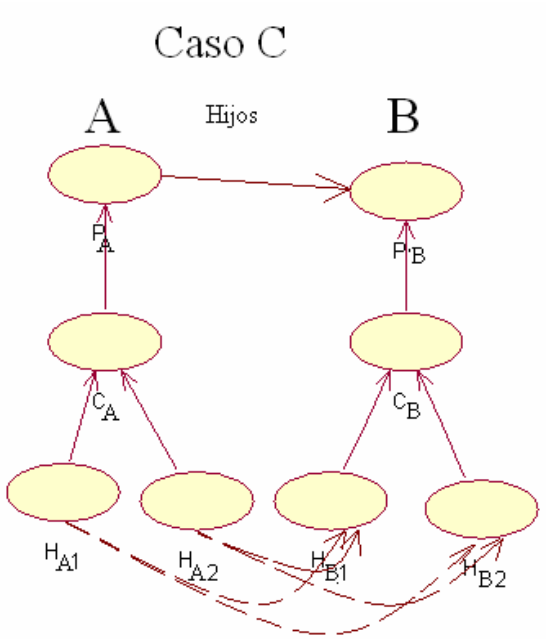


Figura 3.10 Muestra el casamiento de CA con CB pero PA no ha casado con PB, por lo tanto, busca que la mayoría de las propiedades de los Hijos del CA hallado casen con la mayoría de las propiedades de los hijos de CB, COM devuelve msg="Caso C", CMS=CB

3.3.4 Caso D: No hay CB ni PB

Si no se encuentra C_B ni P_B, entonces la respuesta es "no existe" y concluye el algoritmo. En la Figura 3.11 se presenta el caso D donde no existe un *cms* para C_A y tampoco para el antecesor de éste P_A, esta situación es común cuando se tratan de dos ontologías totalmente diferentes, por ejemplo: una ontología con temas de "Sistemas de Información" con otra ontología con temas de "Recursos Naturales".

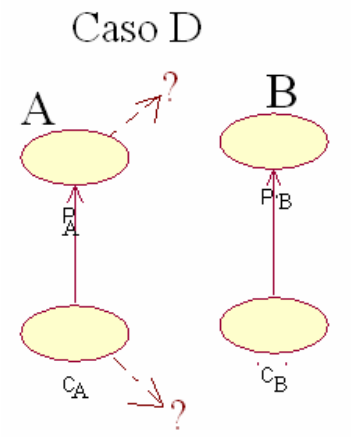


Figura 3.11 muestra el No casamiento de C_A y P_A. COM devuelve msg="caso D Concepto no hallado"

Para cada uno de los casos anteriormente descritos, COM [22] encuentra una respuesta mediante una búsqueda algo restringida.

3.4 Adaptaciones de este trabajo al algoritmo de búsqueda COM

El algoritmo COM [22] se adaptó a la estructura de las ontologías presentada en éste trabajo de tesis (véase §3.1) y además se le hicieron mejoras (descritas en los subtemas de este apartado) para obtener más precisión en la búsqueda del concepto más similar *cms*. Estas aportaciones se han presentado en [7] y también se describen a continuación.

3.4.1 Nueva notación para las ontologías

La nueva notación (véase la Tabla 3.1) consiste en una estructura con etiquetas que identifican la descripción de los conceptos y sus relaciones.

Tabla 3.1 Etiquetas usadas en el lenguaje OM

<concept>*c*</concept>	Donde *c* representa el nombre del concepto.
<language>*l*</language>	Donde *l* representa el lenguaje en el cual las palabras están definidas.
<word> w_1,w_2…w_n </word>	Donde w_1,w_2…w_n representan las palabras que describen al concepto *c*.
<arity>*a*</arity>	Donde *a* es un numero positivo que describe la aridad del concepto *c*.
<relation>*n*=*v*</relation>	Donde *n* representa el nombre y *v* representa el valor de la relación, *n* y *v* son conceptos, *v* puede ser una lista si la relación tiene mas de un valor.
<member>*c*</member>	El concepto que contiene esta relación es *miembro de* el concepto *c*.
<part>*c*</part>	El concepto que contiene esta relación es *parte de* el concepto *c*.
<part*>*c*</part*>	El concepto que contiene esta relación tiene al menos un elemento que forma *parte de* al menos un elemento del concepto *c*.
<subset>*c*</subset>	El concepto que contiene esta relación es *un subconjunto de* el concepto *c*.
<type>*c*</type>	El concepto que contiene esta relación es *un tipo de* el concepto *c*.

Una relación puede estar conectada a más de un valor. Por ejemplo (Color `Manzana Amarillo, Verde, Rojo`) esto es, que el concepto `Manzana` tiene una relación Color que conecta a tres elementos.

Relación implícita. En la notación, se observa que las relaciones member (miembro_de), subset (subconjunto_de), part (parte_de) y part* (parte_de*), se escriben de una forma especial distinta de todas las demás. Véase la Figura 3.12 Se denominan "implícitas" por razones históricas.

Relación explícita. Todas las demás relaciones son explícitas. Por ejemplo: (color `Conejo Bugs, gris`) donde `Conejo Bugs` es el concepto que tiene la relación color con el concepto `gris`. Las relaciones explícitas se llaman también propiedades o características del concepto (§3.7.2).

Las relaciones están declaradas explícitamente, excepto unas cuantas que son implícitas. Por ejemplo en la Figura 3.12 la relación *eat*; excepto la relación de hiponímia[59], que se expresa a través de la anidación de conceptos. Otro ejemplo, se presenta en `plant` como un subconjunto de `physical_object`.

```
<concept>thing
        <language> English <word>thing, something, object, entity </word> </Language>
        <concept>physical_object
                <language> English <word> concrete_object, physical_object</word> </Language>
                <subset>thing </subset>
                <concept>plant
                        <language> English <word>plant, tree</word> </Language>
                        <subset> physical_object </subset>
                        <concept>fruit
                                <language> English<word>fruit, citric</word> </Language>
                        </concept>
                </concept>
                <concept> human being,
                        <language> English
                        <subset> physical_object </subset>
                        <word> person, people, human being </word></Language>
                        <relation>eats=tropical_fruit, citrus</relation>
                        <relation>Partition=age {0<age<=1 : baby; 1<age<=10 : child;10<age<=17 : puberty; 17<age<=29 :
                        young; 29<age<=59 : mature; age>59 : old;}</relation>
                </concept>
        </concept>
        <concept>abstract_object
                <language> English <word>imaginary object, abstract thing</word> </Language>
                <subset> thing </subset>
                <concept>soul
                    <language> English <word>soul, spirit</word> </Language>
                    <subset> abstract_object </subset>
                </concept>
        </concept>
</concept>
```

Figura 3.12 Representación de una ontología con la nueva anotación, donde la etiqueta de cada concepto (por ejemplo *thing*) está después de <concept>; el lenguaje en que está definido el concepto (por ejemplo *English*) va entre <language> y </language>; la definición del concepto (por ejemplo concrete_object, physical_object) va entre <word> y </word>; las relaciones del concepto (por ejemplo *eats* del concepto human being) va entre <relation> y </relation>. La descripción de un concepto termina con </concept>. Los conceptos anidados (como thing y physical_object) indican que physical_object es subordinado (o hipónimo) de thing, el significado preciso de esta subordinación está indicado por <*subset*> thing </subset>. Las relaciones expresadas por el anidamiento se llaman relaciones implícitas (actualmente, son: miembro de [*member*], *parte de* [*part*], *subconjunto* [*subset*] *y parte** [*part**]), las otras, tal como *eats,* se llaman relaciones explícitas. Las relaciones Explícitas de un concepto en otros trabajos se conocen como propiedades o atributos del concepto

Ventajas

i) Las etiquetas definen claramente cada concepto,
ii) Se permiten las particiones, las superclases múltiples, las extensiones, relaciones simplificadas y valores.

[59] Hipónimo es la palabra que posee todos los rasgos semánticos de otra más general (su hiperónimo), ver: es.wikipedia.org/wiki/Hiponimia. Es la relación de inclusión de significado entre dos términos. Un término es hipónimo de otro si el significado del segundo está incluido (implicado) en el primero. Por ejemplo, el significado de "animal" está incluído en el de "conejo": "conejo" es hipónimo de "animal". De "animal" se dice en este caso que es hiperónimo de "conejo", ver: www2.udec.cl/~prodocli/semántica1/hiponimia.htm.

iii) Los conceptos de una ontología pueden ser enriquecidos en cualquier momento; de lo cual carecen otros formatos. Por ejemplo: una ontología define que Canadá es un país, y después de otra ontología se añade la relación Canadá es un mercado.

En la Figura 3.13 se representa la ontología gráficamente.

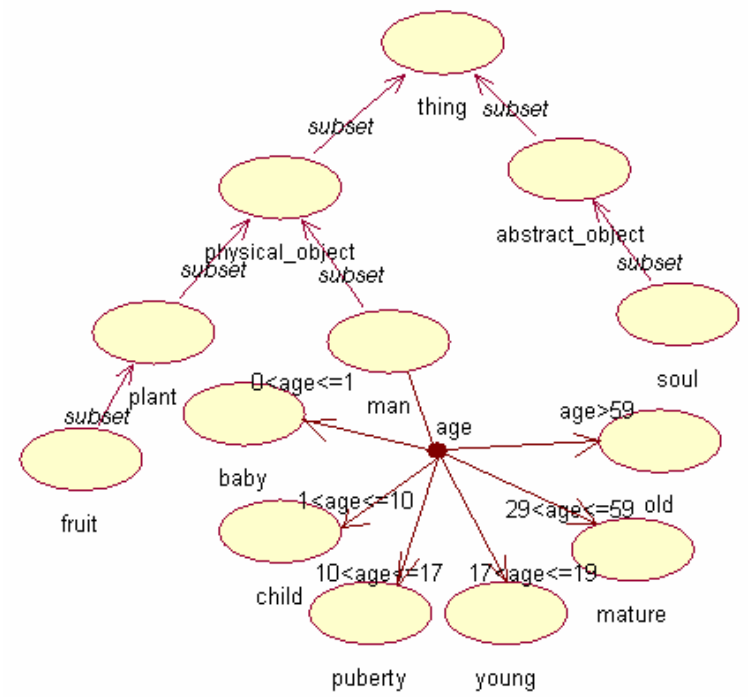


Figura 3.13 Presentación de la ontología bajo la anotación de OM donde cada concepto está relacionado con otro a través de la relación implícita *subset*, excepto la partición en el concepto man (identificado por el círculo pequeño)

3.4.2 Representación de particiones

Las particiones son un tipo particular de relación. Una partición de un conjunto es una colección de subconjuntos, tales que cualesquiera dos elementos de este conjunto son mutuamente exclusivos y todos ellos colectivamente exhaustivos. Gráficamente se representan con un círculo de color oscuro.

Ventajas:

i) Una partición tiene más información que solo una colección de subconjuntos. Los lenguajes OWL [3], DAML+OIL [39], y RDF [27] no representan particiones.

La Figura 3.14 presenta la definición de una partición llamada Edad, expresada bajo la nueva notación.

```
<concept>Ser humano
        <Language>Spanish <word>Ser humano, persona</word></Language>
        <subset> Entidad concreta</subset>
        <relation>Partition=Edad
                {0<edad<=1 : bebé;
                1<edad<=10 : niño;
                10<edad<=13 : adolescente;
                13<edad<18 : joven;
                18<=edad<40 : adulto;
                40<=edad<60 : maduro;
                60<=edad : adulto mayor }
        </relation>
</concept>
```

Figura 3.14 Representación de una partición usando la nueva notación de ontologías

La Figura 3.15 muestra la representación gráfica de la misma partición.

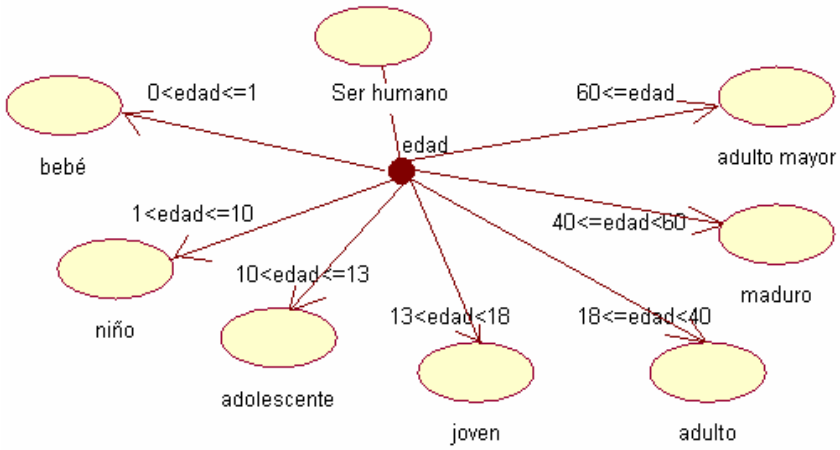


Figura 3.15 representación gráfica de una partición usando la nueva notación de ontologías

3.4.3 Un concepto puede tener múltiples antecesores

Puede tener varios antecesores o padres y no solo uno, como lo requieren muchos de los lenguajes de definición de ontologías. Por ejemplo, el concepto `apple` es un tipo de `fruit` y también un tipo de `food`. En la anotación nueva, un concepto definido en alguna parte de la ontología puede ser *extendido* en otra parte, añadiéndole más relaciones.

Ventajas:

Un concepto con múltiples padres proporciona más detalles en su representación.

Se incrementa la semántica del mismo.

3.4.3.1. Adaptación del *Caso A* al uso de múltiples padres

Las mejoras en el Caso A, se presentan en la Figura 3.16 a) donde C_A casa con C_B pero A_A casa con P_B, a este caso se le llama *Abuelo simétrico* porque es la simetría de A abuelo, y en b) donde el caso es muy útil en la fusión de ontologías usando el algoritmo OM (Ontology Merging) explicado detalladamente en §3.2.1.

El COM original [22] no contempla el uso de los múltiples padres en las ontologías, es decir que tanto C_B como C_A puedan tener más de un antecesor. Por ejemplo: el concepto Chiapas no solo es *parte* del concepto México sino es *miembro* de los Estados de la República Mexicana. Por lo tanto cuenta con dos antecesores: México y Estados de la República Mexicana.

En la Figura 3.16 b) se presenta gráficamente este Caso A. Es de importancia mencionar que en todas las situaciones del Caso A de COM [22] (tanto en el original como en el mejorado) se comparan solo las definiciones (glosas, conjuntos de palabras) de los conceptos involucrados, es decir, las definiciones de los conceptos C_A, C_B, las del concepto papá P_A con las del concepto papá P_B. La línea delgada punteada indica el casamiento y la gruesa, la ruta de C_A y P_A hacia los conceptos C_B y P_B hallados.

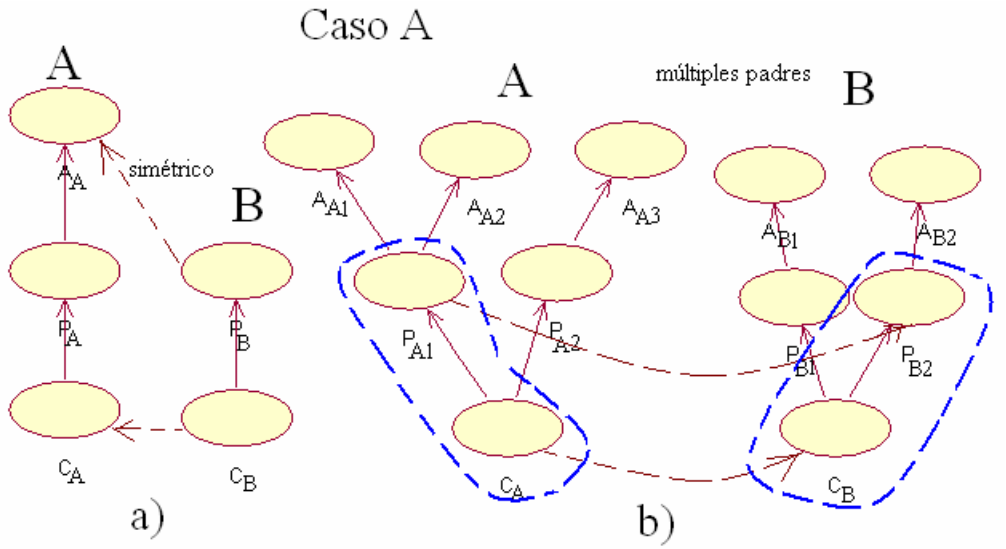


Figura 3.16 Caso A con las mejoras realizadas a COM, donde en a) se presenta el caso Abuelo simétrico y en b) se halla a P_{B2} como el más similar de una lista de padres de C_B

Para cada uno de los casos del COM [22] original (Caso A, Caso B y Caso C) se ha aplicado la búsqueda en múltiples antecesores, un esquema se presenta en la Figura 3.17.

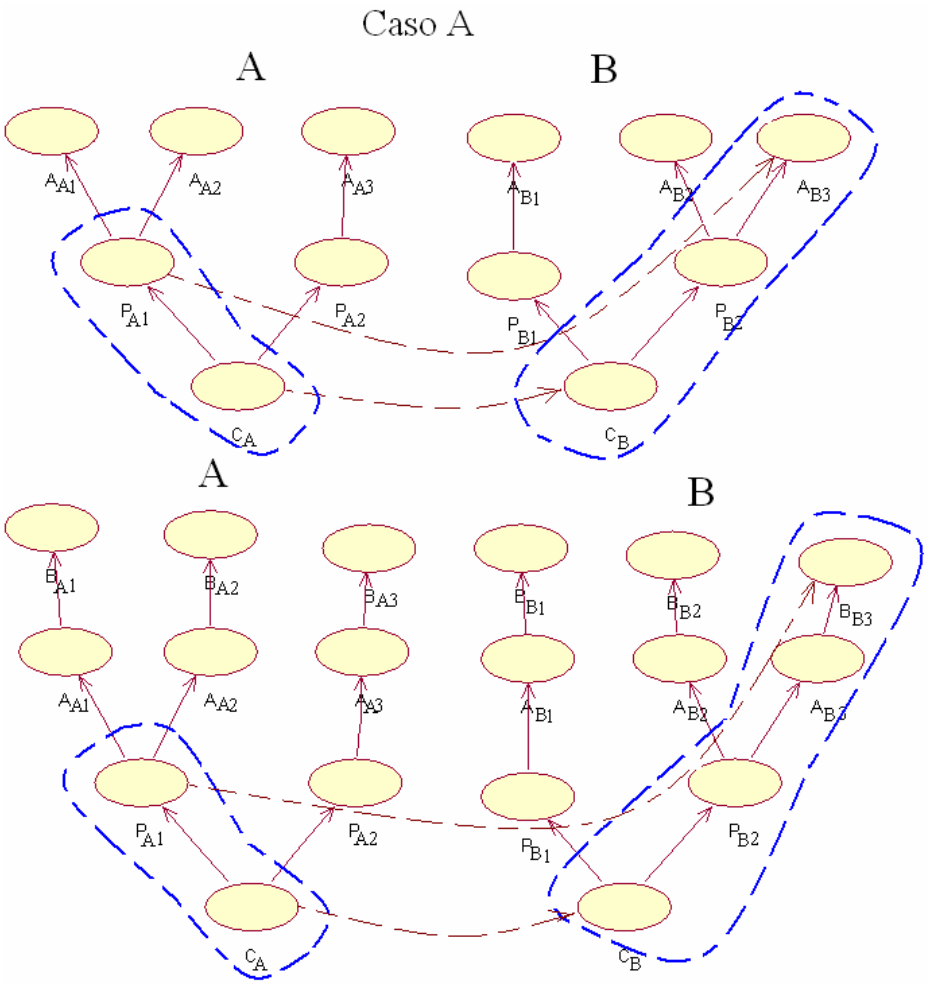


Figura 3.17 Muestra en a) donde P_{A1} casa con A_{B3} uno de los abuelos de C_B y en b) P_{A1} casa con B_{B3} uno de los bisabuelos de C_B

3.4.4 Búsqueda de mejor casamiento de las propiedades

En los Casos B y C de COM [22] y durante la comparación de las relaciones, propiedades o características se realizan los pasos siguientes:

1) Se toman las relaciones del concepto C_A a comparar
2) Se toman las relaciones de los demás conceptos C_B a comparar
3) Se comparan cada una de las relaciones de C_A con cada una de las relaciones de C_{B1}, C_{B2}, …, C_{Bn}
4) Se computa el número de relaciones de C_A que hayan casado con cada uno de C_{B1}, C_{B2}, …, C_{Bn}
5) Se elige el C_B cuyas relaciones hayan casado en su mayoría con las de C_A

Al final, se determina el mejor mapeo de relaciones de un concepto C_A a otro C_B.

Durante la comparación de las propiedades se aplica el algoritmo COM de forma recursiva; es decir, al buscar el nombre de una propiedad de C_A hacia C_B se vuelve a aplicar el algoritmo COM, lo mismo sucede con los valores de la propiedad y también con la búsqueda de los elementos de una partición.

Las mejoras en la búsqueda de propiedades, se enlistan a continuación:

3.4.4.1. Búsqueda de nombre y valor de una propiedad

En la búsqueda del nombre y valor de una propiedad, se verifica los sinónimos.

Si la glosa de C_A contiene una frase como: “vive en” y en la glosa en C_B contiene una frase como: “radica en” entonces C_A y C_B se consideran sinónimos, si y solo si:

1) “vive en” es un concepto en A y “radica en” es parte de su definición.
2) “vive en” es un pre-concepto en A y “radica en” es un concepto en B y el primero se encuentra en la definición del último.
3) Ambos (C_A y C_B) son conceptos con argumentos y los dos (“vive en” y “radica en”) están en ambas definiciones (de C_A y C_B).

El nombre de una relación puede ser un concepto y el valor de una relación puede ser un concepto o una lista de éstos.

Además del uso de sinónimos existe otra forma de identificar la igualdad de dos conceptos:

1) Si el nombre de una relación en A es: “colinda de norte a sur” y el nombre de otra relación en B es: “colinda desde el norte hasta el sur” se identifican como iguales. La razón es que se excluyen los artículos, conectores de la frase comparándose las frases que quedan. En este ejemplo las palabras excluidas en la relación A son: “de”, “a”. Las frases excluidas en la relación B son: “desde”, “el”, “hasta”, “el”. Las palabras comparadas después de ambas exclusiones son, para la relación en A: “colinda norte” y para la relación en B son:”colinda norte” por esa razón se identifican como iguales.
2) Si el nombre de una relación en A es: “abarca de norte a sur con” y el nombre de otra relación en B es: “abarca de sur a norte” se identifican como diferentes. Porque después de excluir las frases en la relación A se tiene: “abarca norte sur” y la relación en B se tiene:”abarca sur norte” por lo que se identifican como diferentes.

3.4.4.2. Adaptación del Caso C a conceptos sin hijos

El nuevo COM [7] devuelve el *cms* de C_A aún cuando estos no tengan hijos ni propiedades. En la Figura 3.18 hay casamiento de C_A con C_B pero no P_A con P_B, se buscan las propiedades e hijos de C_A y C_B si existen, devuelve msg = “Caso C sin propiedades”, *cms* = C_B útil en la fusión de ontologías.

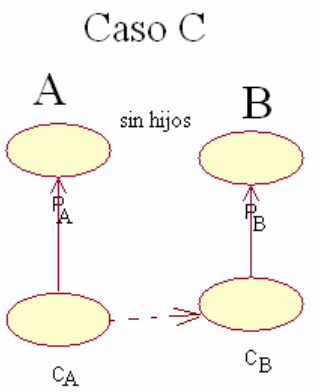


Figura 3.18 Presentación del caso C sin propiedades de COM [7]

3.4.5 Se permiten conceptos sin argumentos en las relaciones o valores

En COM [22] las relaciones y valores deben ser conceptos (nodos con relaciones). La mejora realizada en este contexto es: *Una relación puede ser* pre-concepto (una palabra o una frase). Por ejemplo, es posible que en una ontología haya una relación (`John Smith`, vive en, `New York`), donde `John Smith` y `New York` son conceptos §3.1 (hay un conocimiento adicional en la ontología acerca de ellos) pero "vive en" es un pre-concepto §3.1 (son palabras o frases), no hay un conocimiento adicional a éste. Por supuesto, para otras ontologías, "vive en" podría ser un concepto.

En COM [22], el original solo se permiten conceptos con argumentos en las relaciones, mientras que en el COM [7] mejorado se permiten conceptos con o sin argumentos, es decir palabras o frases que no tienen propiedades.

3.4.6 Búsqueda de la sinonimia entre los conceptos

Los elementos de una relación a copiar pueden ser identificados como sinónimos, sea por sus definiciones o por el caso B de COM [7] por cualesquiera de estas dos maneras, el algoritmo OM deberá enriquecer estos elementos en la ontología final, añadiendo nuevas definiciones a los conceptos involucrados.

La búsqueda de sinónimos entre conceptos se realiza de la siguiente manera:

1. *La identificación de la sinonimia entre dos conceptos* C_A *y* C_B *sin argumentos*. Devuelve "verdadero" si ambos conceptos C_A y C_B están en la definición de otro concepto C_C, si no devuelve "falso".
2. *La identificación de la sinonimia entre dos conceptos* C_A *y* C_B *con argumentos*. Devuelve "verdadero" si la mayoría de las palabras de la definición C_A se encuentran en la definición de C_B, si no devuelve "falso".
3. *La identificación de la sinonimia entre un concepto* C_A *con argumentos y otro* C_B *sin argumentos*. Devuelve "verdadero" si el nombre (etiqueta) de C_B está presente en la definición de C_A, si no devuelve "falso".

4. *La identificación de la sinonimia entre un concepto* C_A *sin argumentos y otro* C_B *con argumentos*. Devuelve “verdadero” si el nombre (etiqueta) de C_A está presente en la definición de C_B, si no devuelve “falso”.

5. *La identificación de la sinonimia entre dos relaciones o propiedades* r_A y r_B. Considera lo siguiente:

Devuelve “verdadero” si la relación r_A es sinónimo de la relación r_B, si no devuelve “falso”.

Devuelve “verdadero” si la etiqueta $C_{nombreA}$ es sinónimo de la etiqueta $C_{nombreB}$, si no devuelve “falso”.

Devuelve “verdadero” si el concepto C_{valorA} es sinónimo de uno de los elementos de la lista de conceptos lis_{valorB} si no devuelve “falso”.

Devuelve “verdadero” si concepto C_{valorB} es sinónimo de uno de los elementos de la lista lis_{valorA} si no devuelve “falso”.

Devuelve “verdadero” si todos los elementos de la lista lis_{valorA} son sinónimos con todos los de la lista lis_{valorB} si no devuelve “falso”.

Devuelve “verdadero” si la partición p_A es sinónimo de la partición p_B si no devuelve “falso”.

Para demostrar la utilidad de la búsqueda de sinónimos, se presenta el siguiente ejemplo:

Se tienen dos conceptos sin argumentos: `Mover` y `Trasladar`, se reconocen como sinónimos, si:

1) ambos: `Mover` y `Trasladar` están en el conjunto de definiciones de un tercer concepto Cambiar,

2) `Trasladar` y `Mover` son dos conceptos con argumentos y la mayoría de las palabras de la definición del concepto Mover están en la definición del concepto Trasladar,

3) `Mover` es un pre-concepto y se encuentra en las palabras de la definición del concepto `Trasladar` y viceversa.

Estas formas de descubrir sinónimos es el primer recurso que usa el algoritmo OM previo a la fusión, el hecho de usar este primer recurso proporciona una mayor precisión en la búsqueda de los datos en el proceso de la unión. El segundo recurso es el uso del caso B de COM [22].

Otro ejemplo de este sustancial método se explica en la sección §4.2.2.

3.5 Unión de conceptos homónimos

OM puede encontrarse conceptos con etiquetas ambiguas, en este caso, identifica la diferencia a través del algoritmo COM mejorado [7] posteriormente le da el mismo tratamiento que cualquier fusión de conceptos diferentes.

Se presenta el caso en que el concepto `bote` en la A puede confundirse con el concepto `bote` en la B (ambos en circulo remarcado) pero las definiciones de cada uno de estos conceptos, así como sus relaciones los hacen diferentes.

La Figura 3.19 presenta la A y B con los elementos de bote.

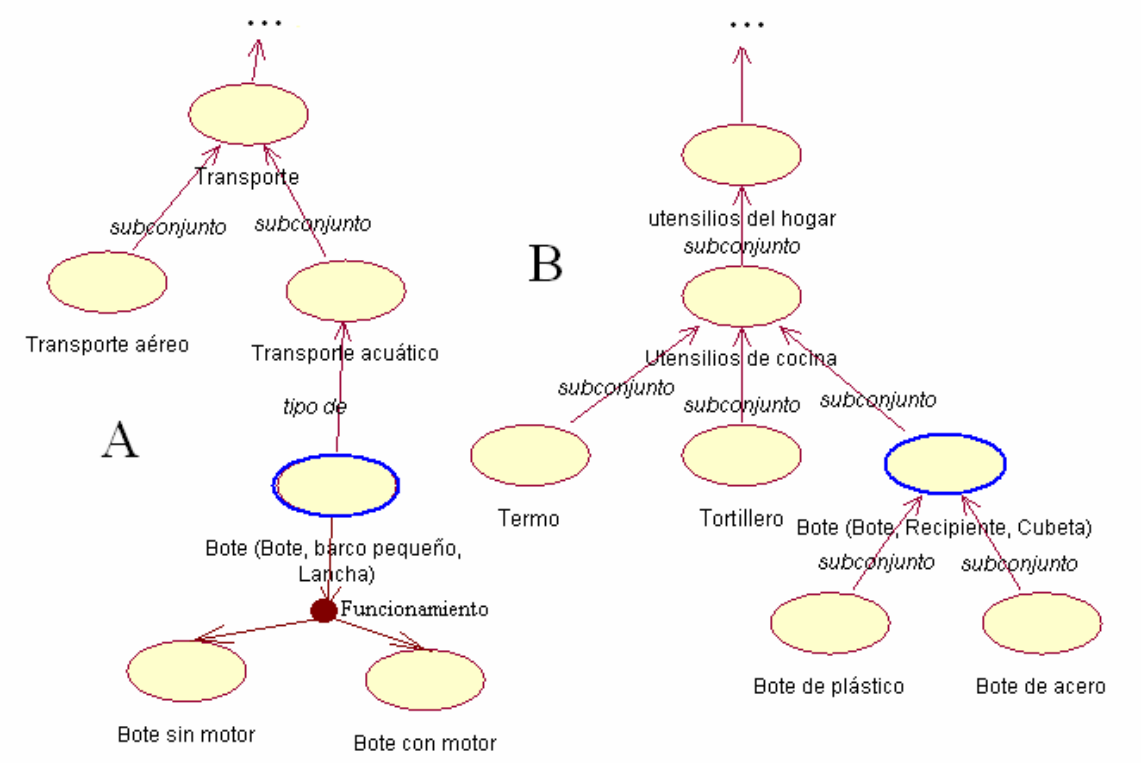


Figura 3.19 Presentación del concepto bote en A y B

La unión se realiza de tal manera que se reconoce al concepto `bote` en B como un nuevo concepto, obteniéndose la C como se muestra en la Figura 3.20.

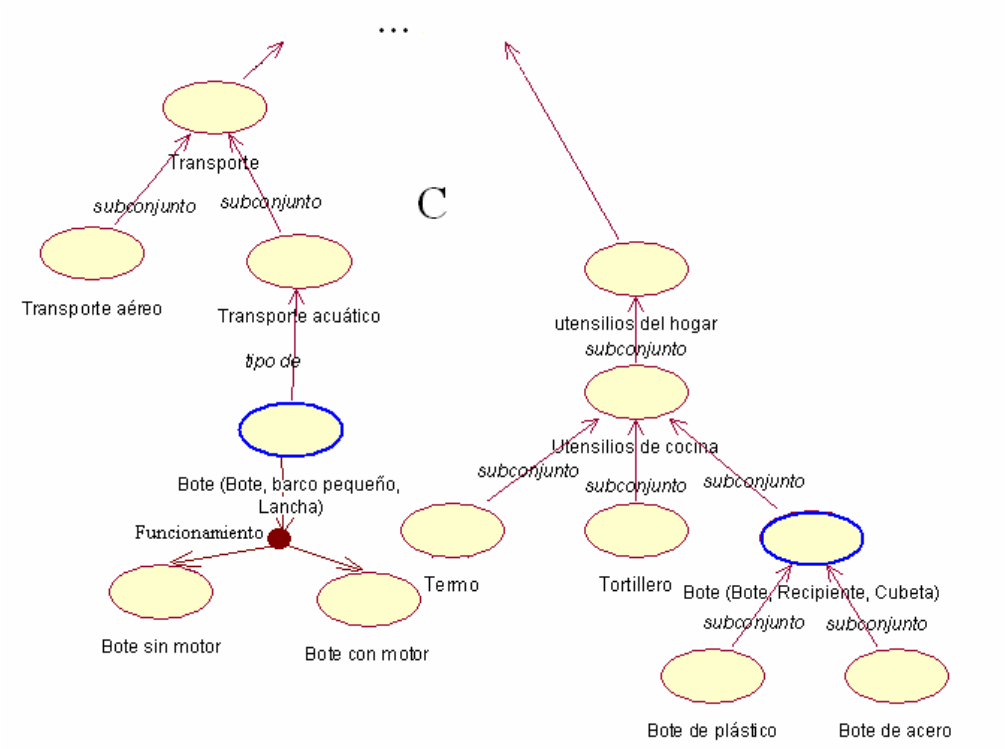


Figura 3.20 Presentación de la C después de la fusión

3.6 Copia de particiones

La fusión de las particiones (§3.2.1) lleva la misma dinámica que las relaciones explícitas (§3.7.2), es decir, si hay una partición p_B en un concepto más similar *cms* de la B y se quiere fusionar con las particiones de un concepto C_C en C, se verifica p_B en las particiones de C_C si no se encuentra, tampoco hay sinónimos (§3.4.6) y al aplicar el algoritmo de la teoría de la confusión [21] no logra hallar una cercanía (§3.8.2) entre p_B y las particiones de C_C, entonces añade la partición como nueva en C_C.

Sin embargo, si son sinónimos, esto lleva a otra serie de pasos interesantes.

3.6.1 Adición de particiones que son sinónimos con rangos y valores diferentes

Si son p_A y p_B dos particiones (§3.4.2) cuyos nombres son sinónimos (véase §3.4.6) se complementan los significados de cada nombre de p_A y p_B, luego se comparan la lista de rangos lis_{RangoC} y la lista de rangos en *cms* lis_{RangoB} si existe al menos un rango en p_B que no esté presente en p_A, se crea una nueva partición. Un ejemplo se muestra en la Figura 3.21.

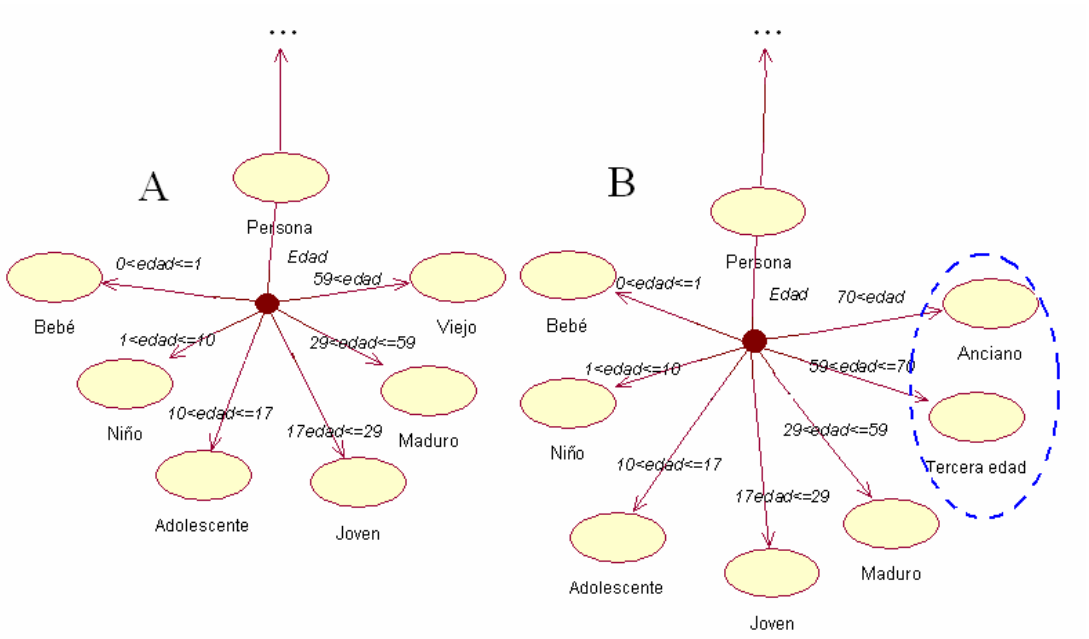


Figura 3.21 Ejemplo de dos particiones p_A y p_B con rangos en p_B que no están en p_A. Los nuevos rangos están indicados en el círculo con líneas discontinuas

OM complementa las definiciones de Edad y Longevidad, al comparar los rangos de p_A y p_B e identificar que hay dos rangos en p_B que no están en p_A, crea una nueva partición con el mismo nombre. En la Figura 3.22 se presenta el resultado.

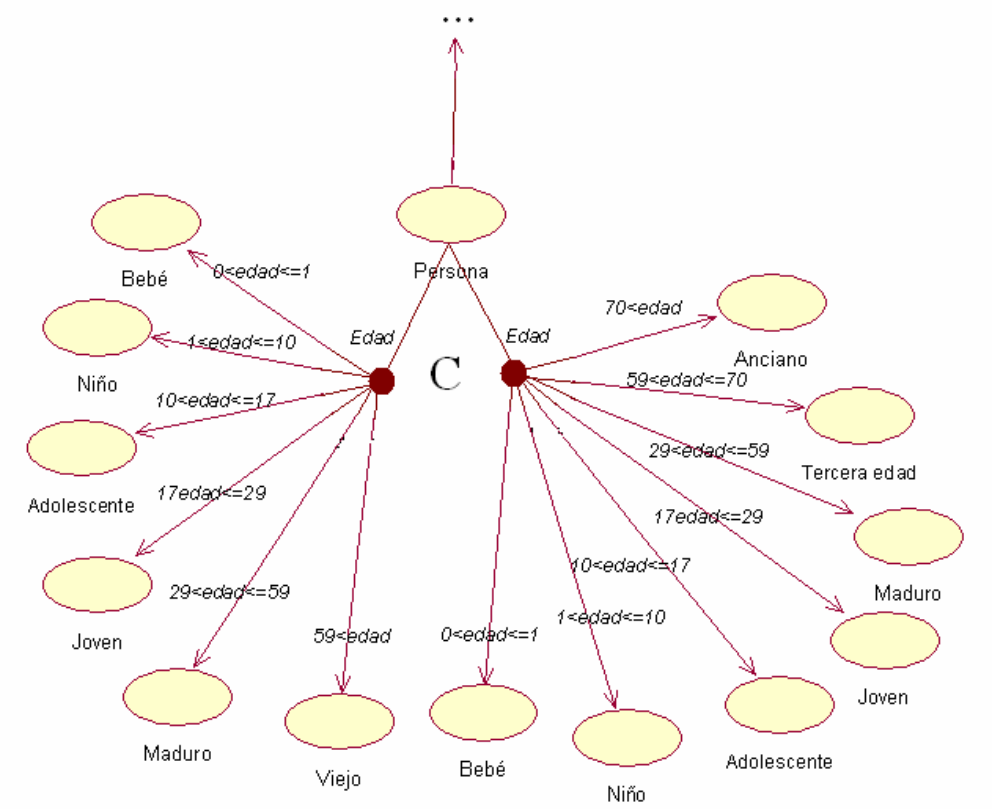


Figura 3.22 Resultado de la unión de p_A y p_B donde se ha creado una nueva partición en p_A con el mismo nombre Edad

3.6.2 Adición de particiones sinónimos con rangos iguales pero diferentes valores

Si los nombres de las particiones son sinónimos, los rangos son iguales y los valores no coinciden, los valores de p_B que no estén en p_A se añadirán a p_C en el rango que corresponda. La Figura 3.23 presenta un ejemplo.

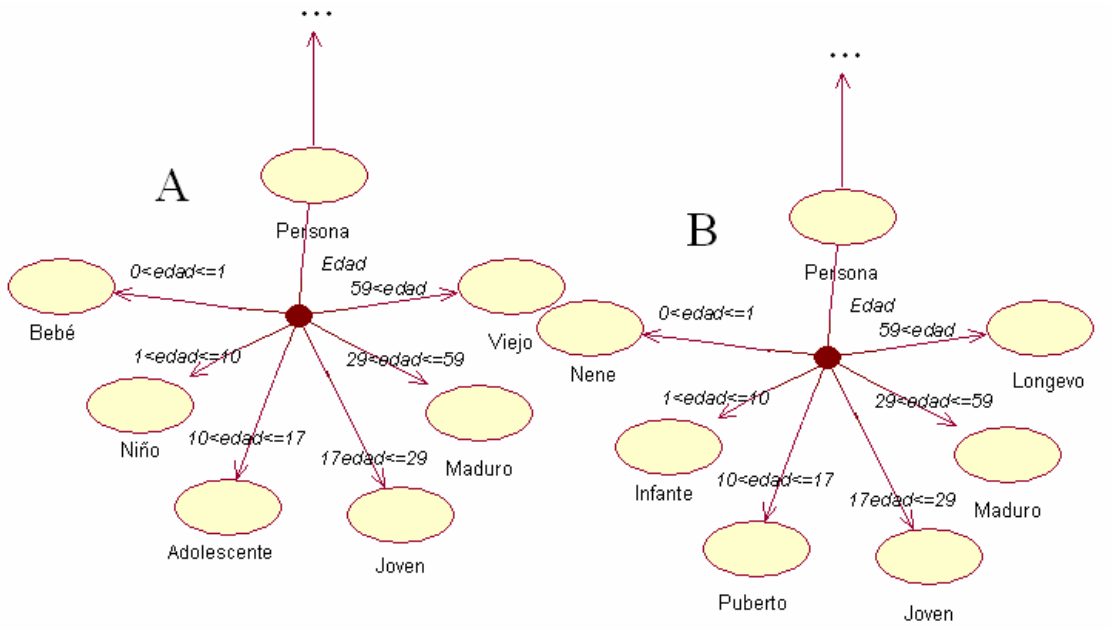


Figura 3.23 Ejemplo donde no coinciden valores de p_A con los de p_B

Los valores de p_B se reconocerían como sinónimos de p_C si comparten el mismo rango. Si estos valores en p_B son conceptos sin argumentos, se adicionarán a las definiciones de los conceptos con argumentos en p_C, luego OM añade los nuevos valores a cada rango en p_C. El resultado se refleja en la Figura 3.24 en la cual se adicionan nuevos valores a la partición.

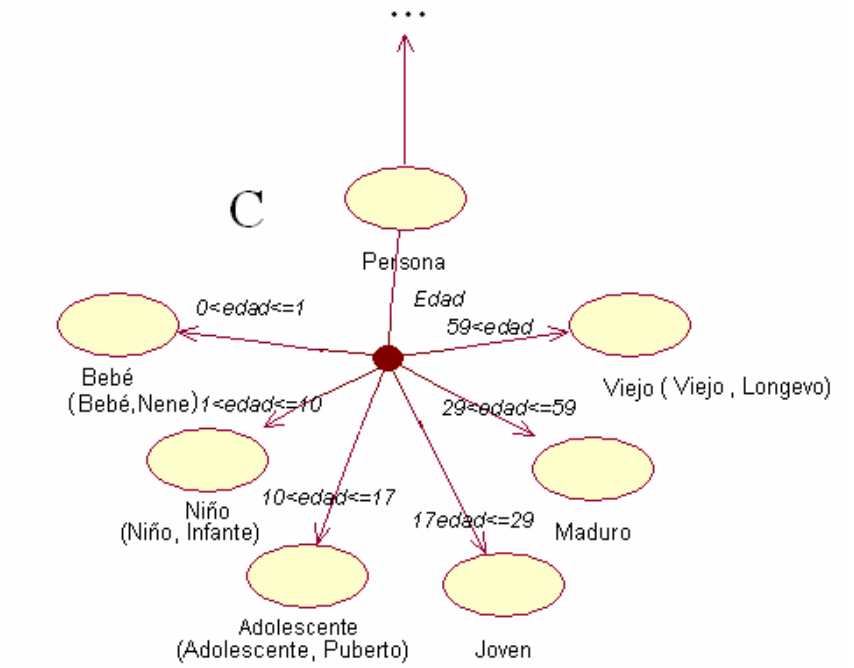


Figura 3.24 Resultado de la fusión de p_A y p_B donde se han añadido nuevos valores identificados como sinónimos en los rangos

3.7 Verificación de las relaciones redundantes

Antes, se explicarán los tipos de relaciones que pueden definirse en una ontología:

3.7.1 Las relaciones implícitas

La notación OM tiene cuatro relaciones implícitas, definidas como sigue:

1. *Subconjunto* (subset): un concepto (conjunto) puede ser un subconjunto de otro concepto (conjunto). Por ejemplo, el conjunto Tortuga es un subconjunto del conjunto Reptil.
   Representado en la notación OM como:
   <concept> Tortuga
   <subset>Reptil</subset>

2. *Tipo de* (type): un concepto (conjunto) es un tipo de otro concepto (conjunto). Por ejemplo, el concepto Pleurodira es un tipo de Tortuga. En WordNet Pleurodira es hipónimo de Tortuga y Tortuga es hiperónimo de Pleurodira.

   Representado en la notación OM como:

   <concept> Pleurodira
   <type> Tortuga </type>[60]

3. *Parte de* (part): un concepto (miembro) es parte de otro concepto (miembro). Por ejemplo, Caparacho exterior es parte de Tortuga Pleurodira. En WordNet Tortuga Pleurodira es holónimo de Caparacho exterior. Caparacho exterior es merónimo de Tortuga Pleurodira.

   Representado en la notación OM como:

   <concept> Caparacho exterior
   <part> Tortuga </part>

4. *Existe al menos uno de mis elementos que forma parte de uno de tus elementos* (part*): Por ejemplo, algunos elementos del conjunto País forman parte de uno de los elementos del conjunto Continente.

[60] Existen individuos (instancias) y conjuntos. Entre un individuo y un conjunto puede existir la relación "miembro". Entre dos conjuntos puede existir la relación "subconjunto", que también es "hipónimo", y sus inversas "superconjunto" y "hiperónimo". Entre individuos puede existir la relación "parte de". Desde esta óptica, el conjunto Conejo representa o contiene a todos los conejos (individuos particulares).
Para personas familiarizadas con la Lógica, existen instancias (que serían los individuos) y Tipos (clases o conjuntos). Entre una instancia y un Tipo puede existir la relación "tipo de". Entre dos Tipos puede existir la relación subtipo.
La notación OM permite ambas nomenclaturas.

Representado en la notación OM como:

<concept> País

<part*> Continente </part*>

5. *Miembro de* (member). Un concepto es miembro de otro concepto. Por ejemplo, Continente Americano es miembro de Continente.

Representado en la notación OM como:

<concept> Continente Americano

<member> Continente </member>

3.7.2 Las relaciones explícitas

Aparte de esas cuatro (relaciones implícitas), las otras relaciones son explícitas. Las relaciones explícitas también imponen restricciones semánticas a los conceptos que relacionan. A menudo describen propiedades, características, valores o acciones que identifican al concepto de otros, por ejemplo:

1. `Manzana` color `amarillo`, `verde`, `rojo` Ejemplo en la notación de este trabajo es: <relation> color = amarillo, verde, rojo</relation> (esta relación la tiene el concepto `Manzana`).
2. `Manzana` forma `Globosa`
3. `Gato` bebe `Leche`
4. `Gato` color `amarillo`
5. `Oaxaca` puerto `Puertos del Estado de Oaxaca`
6. `Oaxaca` economía `Economía del estado de Oaxaca`
7. `Tortuga` habita `Zona Intertropical`
8. `Tortuga` puede `Morder`
9. `Juan Pérez` préstamo `José Rodríguez`, `10000`, `10%`, `octubre 2005`

El ejemplo 9 es una relación n-aria, es decir, hay varios elementos (`José Rodríguez`, `10000`, `10%`, `octubre 2005`) que están enlazados a la misma relación préstamo realizado por `Juan Pérez` a `José Rodríguez`. Esta relación se representa en la notación de esta tesis de la manera siguiente:

<relation>préstamo = José Rodríguez,10000, 10%, octubre 2005 </relation>

3.7.3 Detección y corrección de las relaciones redundantes

Al copiar nuevas relaciones implícitas pueden surgir relaciones redundantes en la C o en la B (ver §3.7.3.1 y §3.7.3.2). La redundancia surge cuando existen tres conceptos: x, y, z cuyas relaciones son las siguientes: x es subconjunto de y, y es subconjunto de z y x es subconjunto de z; la redundancia está en: x es subconjunto de z, por lo que el algoritmo OM no añade esta relación en la

ontología final.

La redundancia se evalúa únicamente con las relaciones implícitas del mismo tipo, es decir, si existiese A es subconjunto de B, B es subconjunto de C y A es parte de C; no existiría redundancia en las relaciones.

3.7.3.1. Redundancia en la ontología destino

Dada la ontología A, se quiere copiar una relación implícita de un concepto similar *cms* en B hacia C_C (que se encuentra copiando) en C, la relación implícita se ha buscado en A y se ha encontrado entre los antecesores de C_C en A.

Por ejemplo la relación <subset>Cosa </subset> en el concepto *cms*: Aeropuerto de Oaxaca, que se encuentra en el concepto Aeropuerto en A como lo muestra la Figura 3.25.

OM no añade la relación implícita de *cms* que provoca redundancia sino copia los hijos de *cms* como hijos del concepto C_C en la ontología resultante C, de tal manera que C_C tendrá nuevos hijos.

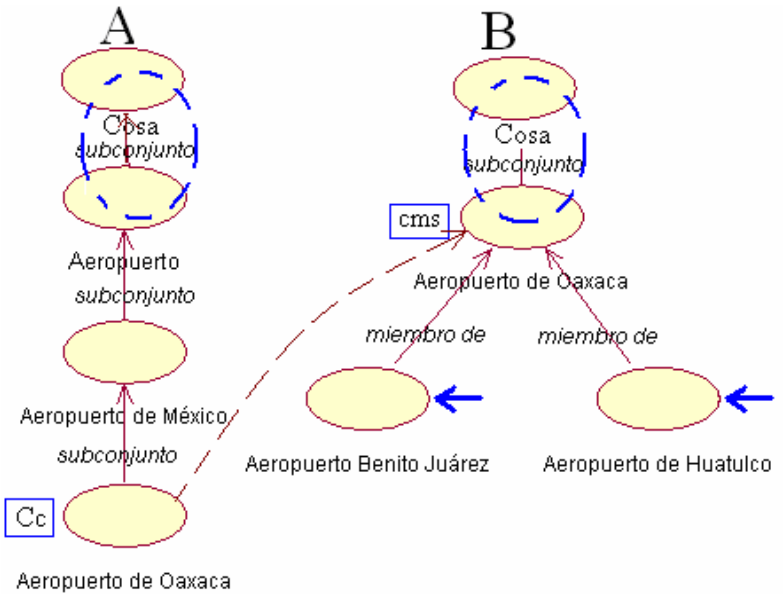


Figura 3.25 presenta la relación implícita en *cms* que se quiere copiar a C_C, pero generan redundancia en las relaciones en A, los rectángulos indican los conceptos involucrados y las elipses las relaciones implícitas, las flechas indican lo hijos de *cms* que se copiarán a C_C

La Figura 3.26 presenta la ontología resultante donde no hay redundancia de relaciones.

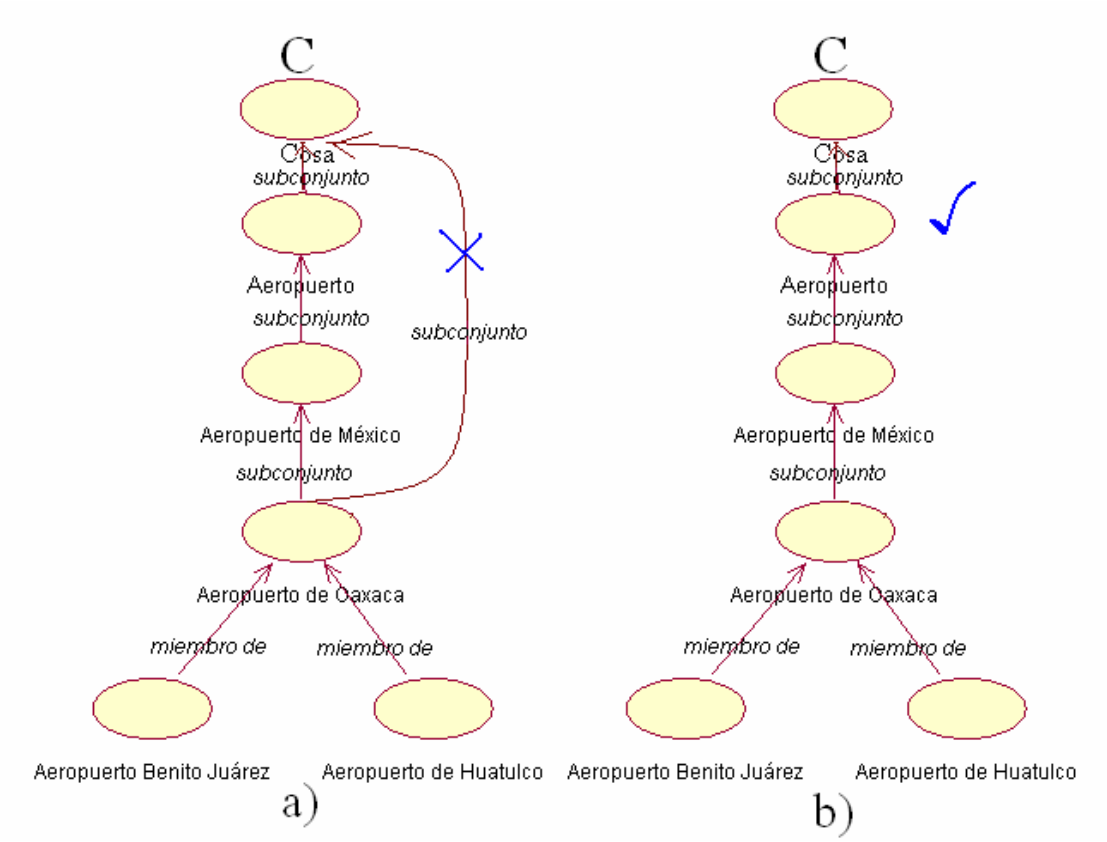


Figura 3.26 La ontología C donde a) tiene relación redundante y en b) no hay redundancia en la ontología final

3.7.3.2. Redundancia en la ontología fuente

La redundancia en la ontología fuente surge cuando se quiere copiar una relación implícita de un concepto más similar *cms* en la B hacia C_C (el concepto que se está copiando) en la A. Se busca la relación implícita de C_C entre los antecesores de *cms* y se encuentra, OM copia todos los antecesores de *cms* en A y elimina la relación implícita en C_C.

Por ejemplo, se tiene en C_C la relación implícita: <subset>Cosa </subset> y se encuentra en la B, véase la Figura 3.27.

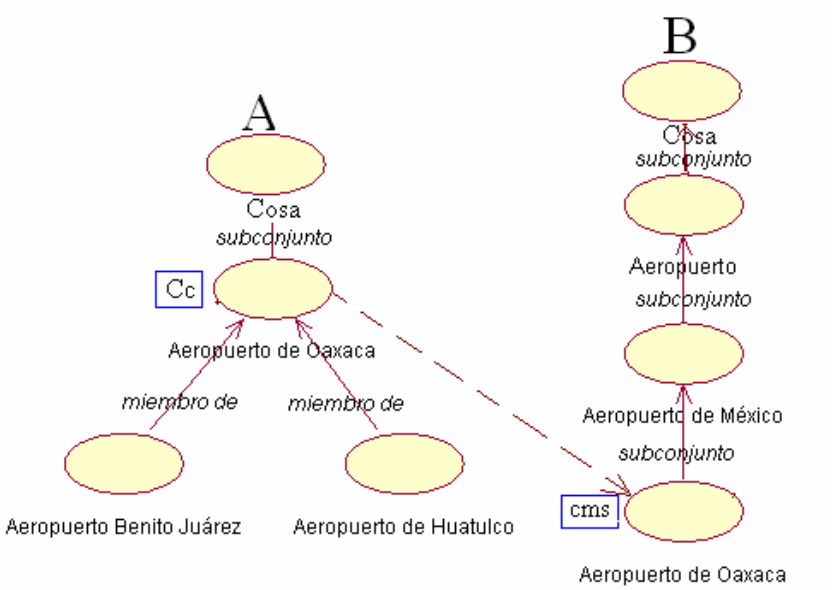


Figura 3.27 Presenta la relación implícita en C_C que se ha encontrado en la B (entre los antecesores de *cms*). Los rectángulos identifican los conceptos que se están copiando, los antecesores de cms se copiarán a la ontología C resultante

El resultado se presenta en la Figura 3.26 donde se puede ver que los antecesores de *cms*, es decir `Aeropuerto de México` y `Aeropuerto` se han añadido como hijos del concepto raíz (`Cosa`) y el concepto C_C (`Aeropuerto de Oaxaca`) se ha añadido como hijo de `Aeropuerto de México`.

Un ejemplo de este relevante método se explica en la sección §4.3.1 que se ilustra claramente con el concepto `Reptil` subconjunto de `Vertebrado` y `Reptil` subconjunto de `Cosa`.

3.8 Uso de jerarquías de conceptos para la solución de contradicciones

Se definen las jerarquías sobre valores cualitativos.

Definición. Una jerarquía *J* es un árbol tal que cada nodo suyo es un valor no numérico o una partición del nodo superior. ♦ También se les llama taxonomías o clasificaciones. Los nodos que son conjuntos (o sea, aquéllos que tienen hijos que forman una partición) a menudo tienen también nombres no numéricos. Ejemplo: véase Figura 2.4. En este trabajo se consideran jerarquías cuyos nodos son conceptos.

Las jerarquías de conceptos [19] se extraen de diccionarios, tesauros, o de documentos de textos. El algoritmo de unión de ontologías usa una jerarquía de conceptos para encontrar el valor mínimo de la confusión [21] entre dos conceptos, y también para resolver las contradicciones (§3.8.1) que detecta durante la copia.

3.8.1 Teoría de la confusión

La confusión, contradicción o inconsistencia surge cuando una relación en C_A que hace incompatible, contradice o hace inconsistente a otra relación en C_B. Por ejemplo la relación: (forma `Tierra`, `redonda`) en A y (forma `Tierra`, `plana`) en B. Nótese que la contradicción surge de *dos relaciones* explícitas (éstas relaciones se detallan en §3.7.2). Es decir, una contradicción surge porque A le da una semántica (expresada en una relación) a un concepto (Tierra, digamos) y B le da otro significado.

Durante la copia se debe conservar la semántica de los elementos que se van a unir. No es posible conservar esta semántica en presencia de contradicciones.[61]

OM trata de formar ontologías C bien formadas (sin contradicciones). Para esto, OM debe:

1. Detectar la contradicción y
2. Resolverla (hallar cuál de las opciones o significados es el correcto).

Esta tesis se limita a detectar la contradicción, y solo resuelve contradicciones en algunos casos, usando la teoría de la confusión [21] que proporciona un modelo formal de la proximidad entre valores simbólicos, que se asemeja al que las personas usan. Por ejemplo: ¿Cuál es la capital de Alemania? Berlín es la respuesta correcta, Frankfurt es un error cercano, Madrid o Bombay es un error medio y Manzana o Polinomio son errores grandes. Hay otros trabajos parecidos a esta teoría de la confusión [21] que llevan otros nombres tales como: Similitud, Distancia, Parecido que tienen propósitos similares pero con definiciones distintas. En este trabajo de tesis se usa la teoría de la confusión y por tanto se hace referencia a ese nombre al aplicarlo en la fusión.

Se formalizarán los conceptos **confusión**, **contradicción** e **inconsistencia**, que surgen cuando dos afirmaciones o hechos [24] sobre el valor que toma una variable monovaluada *v*, no son idénticos. Por ejemplo, un hecho dice que $v = a$ y otro dice que $v = b$, donde los valores *a* y *b* no son necesariamente iguales. Aparentemente hay una contradicción porque *v* es monovaluada.

Por ejemplo, la ontología A afirma que: *Benito Juárez nació en Oaxaca* y la ontología B afirma que: *Benito Juárez nació en México* y *lugar de nacimiento* es una variable monovaluada.

Se dice que hay una **confusión** cuando el error entre *a* y *b* es pequeño o cero y hay **contradicción** cuando ese error no es pequeño. Si existe una contradicción

[61] El algoritmo OM de fusión supone que las ontologías A y B están bien formadas, es decir, no existen contradicciones en A, ni en B. Sin embargo, al tratar de fusionarlas en ellas pueden surgir estas inconsistencias, porque A asegura que la `Tierra` es `plana`, y B que es `redonda`.

entre *a* y *b*, entonces la afirmación *v* = *a* y *v* = *b* es **inconsistente**. La **inconsistencia** se define en general para varios hechos o afirmaciones, no solo para dos. Se puede medir el grado de **inconsistencia** [24] que existe en un conjunto {$v = a_1$, $v = a_2$, … $v = a_k$} de hechos.

Formalmente, para *a*, *b* valores en una jerarquía:

Si $0 \leq \min(\text{conf}(a,b), \text{conf}(b,a)) \leq \varepsilon$ entonces, hay **confusión** entre *a* y *b*, en otro caso hay **contradicción** entre *a* y *b*.

Dos valores *a* y *b* entre los cuales existe un error pequeño se toman como una **confusión** o "error de medición", tal como se esperaría de dos personas que miden la longitud de una mesa y reportan el hecho que *l* = 4.23 m y *l* = 4.26 m, respectivamente. Si tal error es grande (mayor que ε), entonces se dice que entre *a* y *b* existe una **contradicción**, las afirmaciones se contradicen, como sería el caso si las longitudes reportadas fueran: *l* = 4.23 m y *l* = 4.26 m, respectivamente.

¿Cuál es el valor de ε en lo anterior?, ¿Cómo determinar si dos valores diferentes para la misma variable monovaluada *v* son el resultado de una confusión (y por consiguiente "son aproximadamente el mismo valor") o si son contradictorios? El valor de ε es empírico o heurístico, dependiendo del uso o del contexto en que participan estos valores. Típicamente, ε no debe ser mayor a un 10% del valor promedio de *v*, para valores numéricos. Para valores *a* y *b* no numéricos (es decir, una variable simbólica *v*), la teoría de la confusión [21] dice como medir la confusión de usar *a* en vez de *b*, conf(*a*, *b*) y devuelve para *conf* un número entre 0 y 1.

Nótese que para los conceptos **confusión**, **contradicción** e **inconsistencia** solo tienen sentido para variables monovaluadas. Por ejemplo, dos afirmaciones: *Benito Juárez visitó Salina cruz* y *Benito Juárez visitó París* no son inconsistentes cuando se toman simultáneamente, porque *visitó* puede tomar más de un valor (es factible que *Benito Juárez visitó Salina cruz* y *Benito Juárez visitó París*). Pero aún en casos de variables (para OM, se habla de relaciones) multivaluadas, los valores con confusión (o sea, con ε pequeño) se eliminan entre sí y solo prevalece (queda en la ontología resultante) el más específico §3.8.3.4, es decir, {*visitó* = *Delegación Magdalena Contreras*, *Coyoacán*, *Francia*, *Angola*, *Europa* y *visitó* = *Ciudad de México*, *África*, *París*, *Bonn*} quedará reducido en la ontología resultante a: {*visitó* = *Delegación Magdalena Contreras*, *Coyoacán*, *París*, *Angola*, *Bonn* }.

3.8.2 El algoritmo de la teoría de la confusión

Sean *r* y *s* dos valores (dos nodos) en una jerarquía (ver glosario) con altura[62] *h* y sea *r*' cualquier ascendiente de *r*.

[62] La altura de un árbol es el número de aristas que hay en la trayectoria de su raíz a la hoja más distante. Ejemplo: la altura del árbol en la Figura 3.29 es 5.

La función conf(*r*, *s*) que resulta de usar *r* en vez de *s* (el valor deseado) es:
conf(*r*, *r*) = conf(*r*, *r'*) = 0
conf(*r*, *s*) = 1 + conf (*r*, *padre_de*(*s*))

Definición

La función conf(r, s) que resulta de usar *r* en vez de *s* es:

$$conf(r,s) = \frac{conf(r,s)}{h}$$

La función conf(*r*, *s*) es la confusión relativa conf (*r*, *s*) dividida por la altura de la jerarquía.

La teoría de la confusión o la función *conf* ha sido desarrollado para hallar el valor de la confusión de usar un concepto en lugar de otro (hay más detalles de su funcionamiento en [21]).

La función *conf*(*r*, *s*) parte de dos conceptos (valores) *r* y *s* que están en una jerarquía de conceptos J. Se obtiene el valor de usar *r* en lugar de *s* y el valor de usar *s* en lugar de *r*.

Se ubica en la posición del concepto *r* en la jerarquía J trasladándose hasta el concepto *s* de misma jerarquía, sumando solo los niveles descendentes (si los hay), de la ruta hacia *s*, luego se divide esta suma entre la altura del árbol (la jerarquía) a este resultado se le conoce como valor de la confusión *vc*, luego parte de la posición *s* de la jerarquía J trazando una ruta hacia el concepto *r* sumando los niveles descendentes y dividiendo nuevamente esta suma entre la altura del árbol (su *vc*).

Realizadas ambas rutas y teniendo ambos valores de la confusión *vc*, se elige el menor *vc* obtenido de las rutas de *r* hacia *s* y de *s* hacia *r*.

Varios ejemplos de este algoritmo aplicados a casos prácticos se presentan en:

1) la confusión en el nombre de las relaciones explícitas, véase §3.8.3.1,
2) la confusión en el valor de las relaciones explícitas, véase §3.8.3.2 y
3) la confusión en el nombre de las relaciones de tipo partición, véase §3.8.3.3.

3.8.3 Contradicción en la copia de las relaciones de un concepto

Cotidianamente el humano detecta y resuelve un gran número de contradicciones, pero tratar de corregir todas a través de la máquina no es fácil. Este trabajo de tesis está encaminado a detectar un número de contradicciones y resolver algunas usando el algoritmo de la confusión [21]. Introducir el valor correcto en C (es decir, resolver la contradicción) es un problema que rebasa las fronteras de esta tesis. En general, requiere usar más información, inclusive hacer

experimentos y observaciones o mediciones para observar qué ocurre en realidad en el mundo real.

Por lo pronto se citan tres ejemplos en el siguiente apartado, de los cuales se asegura la solución de algunos casos.

3.8.3.1. Aplicación del algoritmo de la confusión en el nombre de las relaciones

El nombre $C_{nombreA}$ de la relación r_A en el concepto C_C de la ontología A puede ser distinto al nombre $C_{nombreB}$ de la relación r_B del concepto *cms* en la ontología B, el algoritmo de la confusión se aplica de la siguiente manera:

Existe una relación r_A (hidrología Oaxaca principal río de Oaxaca) y una relación r_B (río Oaxaca, principal río de Oaxaca), representado en la **Figura 3.28**.

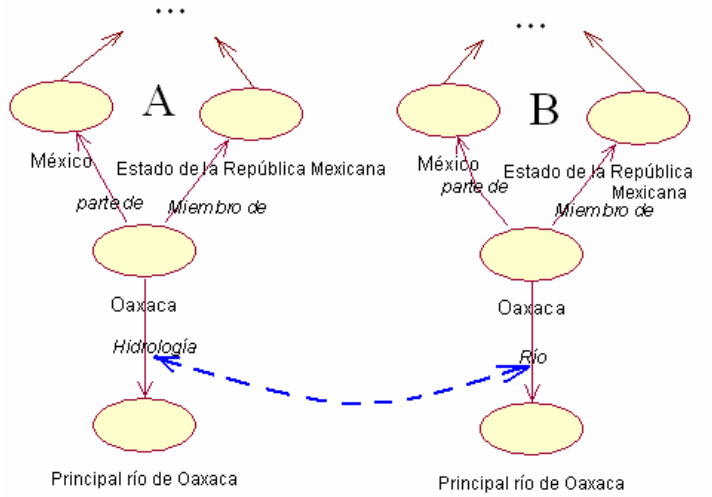


Figura 3.28 Las ontologías A y B cuyas relaciones Hidrología y Río presentan un conflicto

Si se quiere fusionar r_B en r_A se puede observar que hay un conflicto entre ellas, ya que coinciden los valores de r_A y r_B (principal río de Oaxaca) pero tienen nombres distintos (hidrología y río), OM se pregunta: ¿Serán sinónimos? ¿O una relación es más específica que la otra? ¿O serán totalmente distintas y habrá que copiar ambas a C?

OM usa los algoritmos para saber que en estas relaciones (r_A y r_B) una relación es más específica que la otra, para ello: Aplica la función *conf* de la teoría de la confusión [21] *conf*(*hidrología*, *río*) donde *hidrología* es el nombre de un concepto en C y *río* es el nombre de un concepto en B (los que provocan el conflicto).

En la Figura 3.29 a) el valor de la confusión al usar río en lugar de hidrología es: 0; es decir, *conf*(río, hidrología) = 0 porque en la jerarquía se parte de río hacia hidrología y hay un solo nivel que es descendente por lo tanto la suma de niveles descendentes es 0, dividido entre la altura del árbol (se obtiene la altura del árbol o jerarquía eligiendo el número de niveles más profundos del árbol). La altura del

árbol del ejemplo es 2 por lo tanto se divide 0 / 2 el resultado (el *vc*) es 0.

En la misma Figura 3.29 b) se obtiene valor de la confusión al usar hidrología en lugar de río (representados en círculos con línea gruesa), partiendo de hidrología y trazando una ruta (línea gruesa discontinua) hacia río se tiene que el numero de niveles descendentes es 1, dividido entre 5 (la altura del árbol) el resultado es 0/5 (el *vc*).

Finalmente, se tiene que la *conf*(*río*, *hidrología*) = 0/5 y la *conf*(*hidrología*, *río*) = 1/5, por lo tanto OM elige r_B, es decir, *río*.

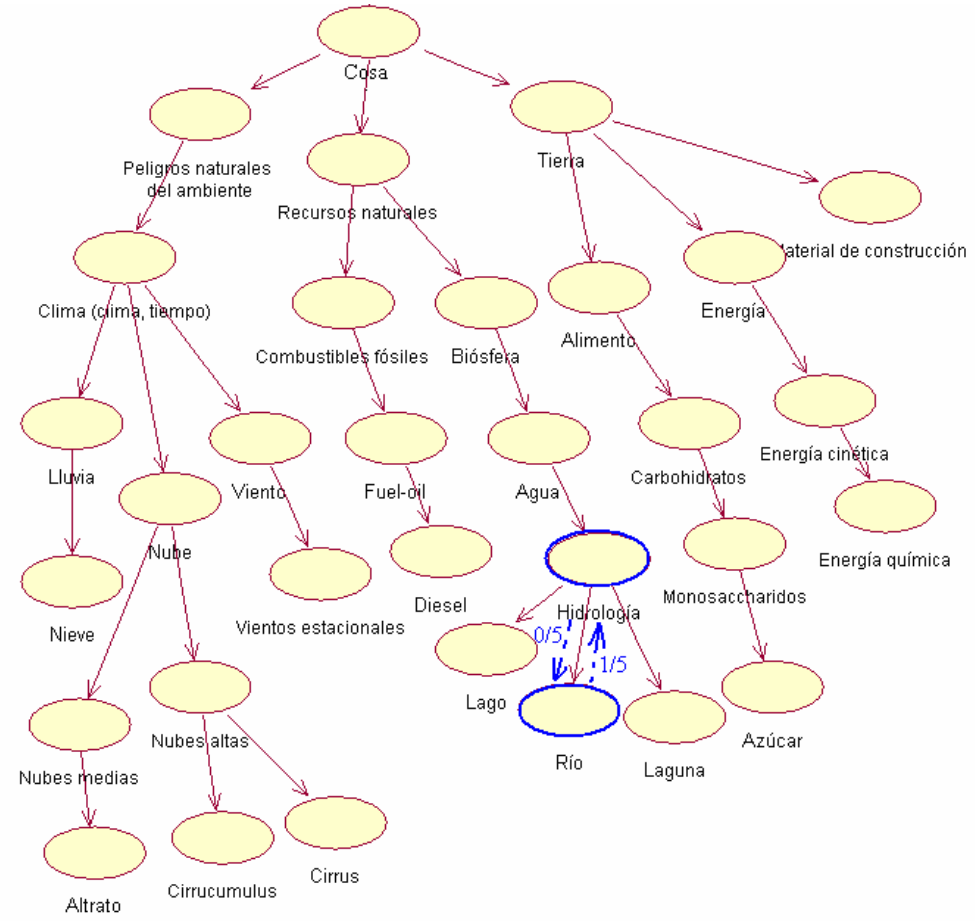


Figura 3.29 Representación de una jerarquía en la cual se usa la teoría de la confusión [21] en el conflicto entre el nombre de las relaciones r_A y r_B

3.8.3.2. Aplicación del algoritmo de la confusión en el valor de las relaciones

Primer ejemplo. *Ambos valores devuelven un valor de la confusión distinto*

Esta contradicción sucede cuando hay una diferencia de contenidos en el valor C_{valorA} de la relación r_A específicamente sucede cuando, tanto las relaciones r_A y r_B como los conceptos $C_{nombreA}$ y $C_{nombreB}$, son semánticamente iguales pero no así

los valores de las relaciones C_{valorA} y C_{valorB} ya que son diferentes y tampoco son sinónimos.

Existe una relación r_A: (Lugar de Nacimiento `Benito Juárez` `San Pablo Guelatao`) y otra relación r_B: (Lugar de Nacimiento `Benito Juárez` `México`)

Aquí la contradicción es que A afirma que el lugar de nacimiento de `Benito Juárez` es `San Pablo Guelatao`, mientras que B afirma que el lugar de nacimiento de `Benito Juárez` es `México`.

OM realiza los siguientes pasos para solucionar el conflicto:

1. Verifica la aridad de la relación. Si la aridad es monovaluada (§3.9) se usa el algoritmo de la teoría de la confusión [21] eligiendo el menor *vc*, si es multivaluada (§3.9) no se considera como una contradicción simplemente se añaden los nuevos valores a la relación r_A. En el ejemplo se supone que la aridad de Lugar de Nacimiento es monovaluada, porque se especifica en la notación que define a la ontología A (además se nace en un solo lugar).
2. Mide el grado de la confusión [21] entre C_{valorA} y C_{valorB}, y entre C_{valorB} y C_{valorA}. La medición usa la jerarquía que contiene los nombres de los valores que confunden (`San Pablo Guelatao` y `México`[63] en el ejemplo). Si el valor mínimo de estas dos confusiones es pequeño, se escoge la `San Pablo Guelatao` que provoca el valor *conf*(San Pablo Guelatao, México) más pequeño, es decir 0 y no el *vc* generado en la *conf* (`México`, `San Pablo Guelatao`) que es mayor, es decir 3/5 (0.6). El valor mínimo es 0. Por consiguiente se escoge `San Pablo Guelatao` como valor a añadirse a Lugar de nacimiento en C.
3. Si el valor mínimo de esas dos confusiones no es pequeño, por ejemplo: A afirma que Lugar de nacimiento es Madrid, B afirma que Lugar de nacimiento es `San Pablo Guelatao`, las confusiones son 5/5 y 3/5, respectivamente, entonces C se queda con el valor de B.[64]

En la
Figura **3.30** se representan los valores de las relaciones en conflicto. La pregunta a responder es: *¿Cuál es el valor de la confusión al usar San Pablo Guelatao en lugar de México*?

A partir de `San Pablo Guelatao` se inicia el recorrido hasta `México`, en este caso la trayectoria hacia el nodo destino es ascendente (no se cuenta el número de nivel ascendente), por lo que el valor de la confusión resulta ser: cero.

[63] Refiérase a México como país y no como Ciudad de México (parte de País).

[64] En caso de confusión, A prevalece. Hay algoritmos más generales que pueden hallar el "promedio" o centro de gravedad de una serie de hechos simbólicos (Benito Juárez nació en Oaxaca), (Benito Juárez nació en Frankfurt), (Benito Juárez nació en la República Mexicana), (Benito Juárez nació en Veracruz), descritos en [24], que se podrían haber usado.

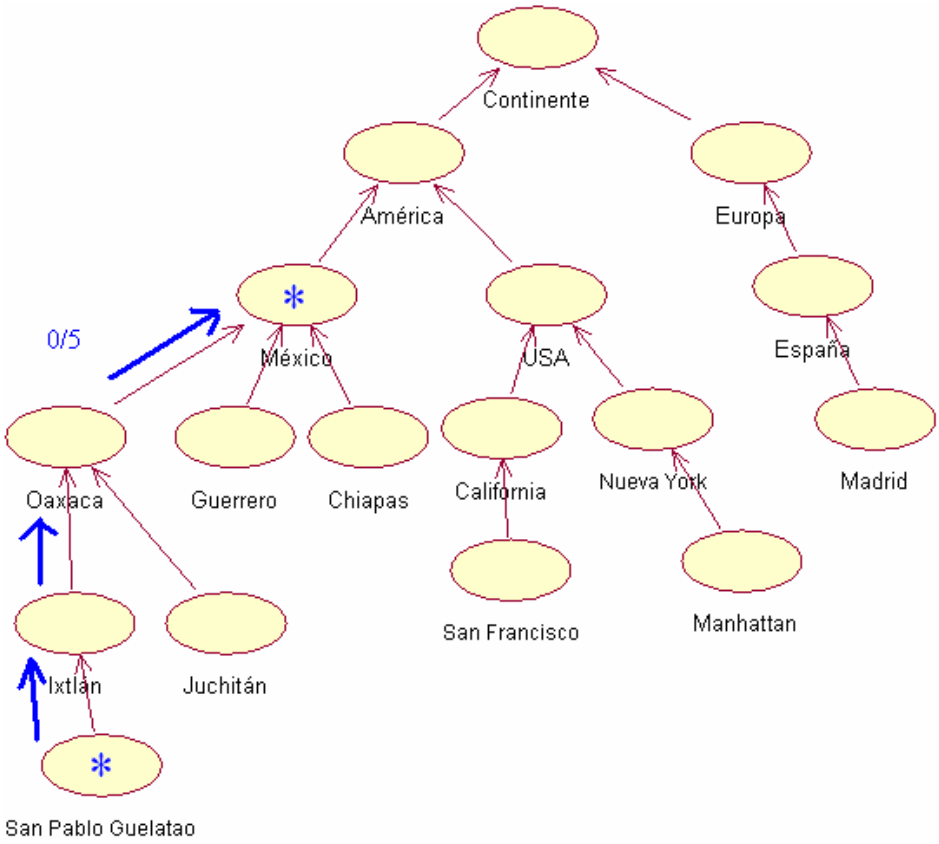


Figura 3.30 El valor de la confusión de usar San Pablo Guelatao en lugar de México

Luego, *¿Cuál es el valor de la confusión al usar México en Lugar de San Pablo Guelatao*?

A partir de `México` se inicia el recorrido hasta `San Pablo Guelatao`, siguiendo una trayectoria descendente, contabilizándose cada vez que desciende un nivel en la jerarquía. Finalmente se tiene el valor de la confusión: 3/5 = 0.6, véase la
Figura **3.31**.

OM elige el valor de la relación: `San Pablo Guelatao` para incluirse en la ontología final.

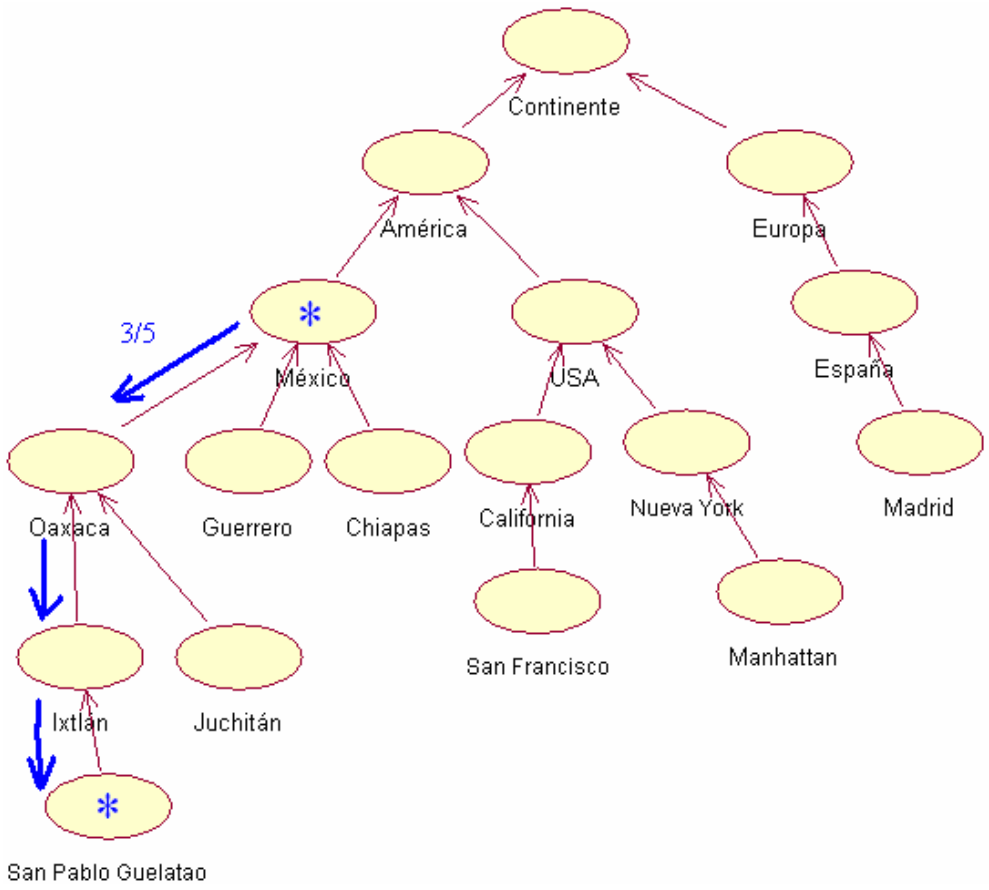


Figura 3.31 El valor de la confusión de usar México en lugar de San Pablo Guelatao

Segundo ejemplo. Las relaciones son sinónimas

Existe otro caso, en la cual el nom bre de la relación r_A es sinónimo del nombre de la relación en r_B. OM identifica la sinonimia entre los nombres y usa la teoría de la confusión [21] entre los valores.

Se tienen dos relaciones sinónimos r_A y r_B, con los nombres $C_{nombreA}$ y $C_{nombreB}$, estos son sinónimos pero no los valores: C_{valorA} y C_{valorB}.

La relación r_A (Lugar de Nacimiento `Benito Juárez México`) y la relación r_B (Lugar de Origen `Benito Juárez San Pablo Guelatao`).

OM verifica si existe sinonimia entre los nombres de las relaciones (véase §3.4.6), (Lugar de Nacimiento y Lugar de Origen) en tal caso; si existe, complementa los significados de ambos nombres, luego usa la teoría de la confusión [21] entre los valores de las propiedades C_{valorA} y C_{valorB} (`México` y `San Pablo Guelatao`) guardándose en la ontología resultante C_{valorC} (el menor valor de la confusión).

Tercer ejemplo. Ambos valores devuelven el mismo valor de la confusión

La contradicción en el valor de la relación se encuentra en §3.8.3.1 donde el concepto lugar de nacimiento tiene dos valores, `Tlacotalpan` y `Ciudad de México` se sabe que en la vida real una persona no puede nacer en dos lugares distintos, aunque es difícil descubrir que lugar de nacimiento = *casa 26 villas Monte Alban, Oaxaca* y lugar de nacimiento = *Oaxaca de Juárez* no es un dato contradictorio. De hecho, no es trabajo de esta tesis, descubrir la verdad ni de resolver todas las contradicciones que se presenten sino darle un tratamiento de manera tal que durante el proceso de la fusión de ontologías, no se requiera la intervención del usuario. Por lo tanto, cuando se presente una contradicción difícil de resolver para OM, decide conservar la relación de A en la C.

Considérese en la A, la relación lugar de nacimiento = `Tlacotalpan` del concepto `Agustín Lara`, otra ontología B contiene el concepto `Agustín Lara` y la relación lugar de nacimiento = `Ciudad de México`, como se observa en la Figura **3.32**.

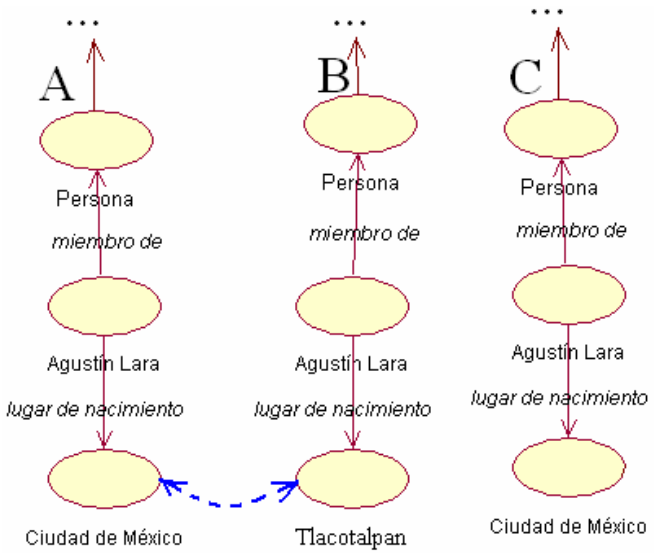


Figura 3.32 Presentación de una contradicción en A y B en el valor de la relación *lugar de nacimiento*

Aplicando el uso de la jerarquía de conceptos se tiene el valor de la confusión de usar `Ciudad de México` en lugar de `Tlacotalpan` con sv = 2 (número de niveles descendientes)/5 (altura del árbol), sv = 0.4, véase la Figura 3.33 donde se indica con flechas gruesas discontinuas. Debido a que ambos valores se encuentran en el mismo nivel de la jerarquía, el valor de la confusión de usar `Tlacotalpan` en lugar de `Ciudad de México` (flechas gruesas continuas) resulta también un valor 0.4. OM se queda con la relación inicial en A, es decir `Tlacotalpan`, la que se presenta en la Figura 3.33.

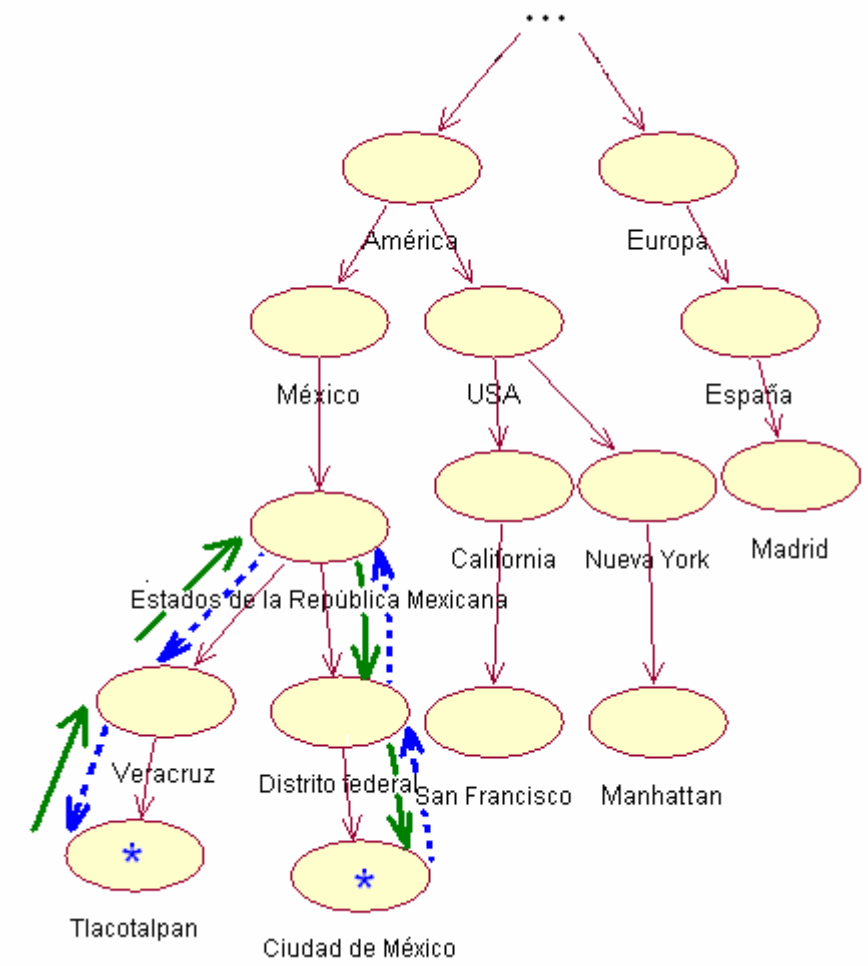


Figura 3.33 El valor de la confusión de usar Ciudad de México en lugar de Tlacotalpan

3.8.3.3. Aplicación del algoritmo de la Confusión en el nombre de las relaciones de tipo partición

En el nombre de las particiones también puede aplicarse el algoritmo de teoría de la confusión [21].

Si la partición en p_A no es sinónimo de la partición en p_B, se aplica el algoritmo de la confusión en los nombres de particiones p_A y p_B también aplica la teoría de la confusión [21] a cada uno de los valores.

Existen particiones que no tienen rango, es decir, se cuenta con los valores y se sabe que existen características que hacen diferente un valor de otro pero no se definen estas características en los rangos, estas particiones “sin rangos” se identifican con un “*” en la posición del rango. Por ejemplo.

La partición p_A Calzado (*: chancla, zapato, bota, tenis, zapatilla) y la partición p_B Indumentaria (*: sandalia, zapato, botín, tenis, calzado informal)

OM verifica la sinonimia entre Calzado e Indumentaria de no serlo, aplica la teoría de la confusión [21] en ellos. La confusión de usar Calzado en lugar de Indumentaria es menor que el de usar Indumentaria en lugar de Calzado. Por lo que se conserva Calzado en p_A, también verifica cada uno de los valores identificando que chancla es sinónimo de sandalia, bota y botín también lo son y es mejor escribir zapatilla en lugar de calzado informal. La partición final queda de la siguiente manera:

p_C Calzado (*: chancla, zapato, bota, tenis, zapatilla). En base de datos corresponden al tipo Enumeration.

3.8.3.4. Depuración de los valores de dos relaciones

Este proceso surge con la finalidad de evitar redundancia en los elementos de una lista, de tal manera que se eliminan los elementos más generales y se priorizan los elementos más específicos. A continuación se explica el proceso.

3.8.3.5. Depuración de las relaciones de un concepto

La depuración consiste en verificar de forma secuencial cada elemento de una lista, por ejemplo: `Istmo`, `Salina Cruz`, `París`, `Francia`, `África`, `Angola`, en la lista no debe haber conceptos antecesores (padres, abuelos, bisabuelos, etc.) y sucesores (hijos, nietos, bisnietos, etc.) de modo que se eliminan los conceptos generales (los antecesores) conservando solo los más especializados (sucesores). Se toma Istmo si es antecesor de alguno de la lista entonces se elimina, se toma el siguiente elemento y así sucesivamente, de tal manera que:

1. `Istmo` es antecesor de `Salina Cruz`, (`Salina Cruz` es miembro del conjunto `Istmo`), por lo tanto se elimina `Istmo`.
2. `Salina Cruz` no tiene sucesores, se conserva.
3. Se compara `París` con el resto de elementos, se conserva
4. `Francia` es antecesor de `París`, se elimina `Francia`.
5. `África` es antecesor de `Angola`, se elimina.

La lista resultante es la siguiente:

`Salina Cruz`, `París`, `Angola`

Este proceso es útil en las relaciones explícitas con más de un valor, se aplica previo a la copia de estas relaciones, por ejemplo en la notación OM el resultado se representa:

<relation>visitó = Salina Cruz, París, Angola </relation>

3.8.3.6. Uso del algoritmo de la Teoría de la confusión en la depuración de los valores de una relación

Consiste tomar dos listas de relaciones r_A y r_B comparando (usando la teoría de

la confusión [21]) cada uno de los valores de la relación r_B con los valores de la relación en r_A. Al final, en este último aparecerán todos sus valores más todos los que están en r_B, sin duplicados ni aquellos que tienen un descendiente (hijo, nieto, bisnieto…) en r_A.

Por ejemplo:

Existe una r_A:

<relation> visitó = Istmo, Francia, Frankfurt, Angola </relation>

Existe una r_B:
<relation> visitó = Salina Cruz, París, Alemania, África </relation>

En el ejemplo se supone que la relación r_B se fusiona a r_A, sobre ésta (r_A) se realiza la re-escritura de nuevos elementos de r_B, es decir r_A se convierte en r_C. Primero se aplica el algoritmo de la confusión a un par de elementos, luego se analiza si uno es antecesor de otro (el lugar del antecesor se re-escribe por el del sucesor).

En este párrafo solo se cita la comparación del primer elemento de r_B con todos los de r_A, en la Tabla 3.1 se presenta el recorrido completo. El proceso es el siguiente:

1. Se obtiene el primer concepto (valor) de r_B: `Salina Cruz`, se aplica el algoritmo de la teoría de la confusión [21] a éste y el primero de r_A: `Istmo`.
    a. El valor de la confusión *vc* de usar `Salina Cruz` en lugar de `Istmo` es: 0.0
    b. El valor de la confusión de usar `Istmo` en lugar de `Salina Cruz` es: 0.14285715,

    ¿Es a < b? La respuesta es *si* y ¿r_A (Istmo) es antecesor de r_B (Salina `Cruz`)? La respuesta es *si* por tanto `Salina Cruz` es un candidato a re-escribirse en Istmo pero no se re-escribe porque falta comparar `Salina Cruz` con los demás elementos de r_A.
2. `Salina Cruz` se compara con `Francia`,
    a. El valor de usar `Salina Cruz` en lugar de `Francia` es: 0.2857143
    b. El valor de usar `Francia` en lugar de `Salina Cruz`: 0.71428573

    ¿Es a < b? La respuesta es *si* y ¿r_A (Francia) es antecesor de r_B (`Salina Cruz`)? La respuesta es *no* por tanto no hay cambios.
3. `Salina Cruz` se compara con `Frankfurt`,
    a. El valor de usar `Salina Cruz` en lugar de `Frankfurt` es: 0.42857143
    b. El valor de usar `Frankfurt` en lugar de `Salina Cruz` es: 0.71428573

    ¿Es a < b? La respuesta es *si* y ¿r_A (`Frankfurt`) es antecesor de r_B

(Salina Cruz)? La respuesta es *no* por tanto no hay cambios.

4. Salina Cruz se compara con Angola,
   a. El valor de usar Salina Cruz en lugar de Angola es: 0.2857143
   b. El valor de usar Angola en lugar de Salina Cruz es: 0.71428573
   ¿Es a < b? La respuesta es *si* y ¿r_A (Angola) es antecesor de r_B (Salina Cruz)? La respuesta es *no* por tanto no hay cambios.

Se ha terminado el recorrido, a continuación:

5. Se re-escribe Salina Cruz en lugar de Istmo en r_A (que cumplió con ambas condiciones en el paso 1).
6. Toma el siguiente valor de r_B y regresa al paso 1) para el resto de los valores en r_B, es decir, París, Alemania y África y se siguen re-escribiendo aquellos que cumplan con las 2 condiciones.

El ejemplo ha generado cuatro iteraciones, éstas se explican brevemente en la Tabla 3.2.

Tabla 3.2 Presenta el recorrido de los conceptos (valores) de r_A y r_B

r_B	r_A	conf(r_B,r_A)	conf(r_A,r_B)	¿conf(r_B,r_A)<conf(r_A,r_B) y r_A es antecesor de r_B?
Salina cruz	Istmo	Salina cruz en lugar de Istmo (0.0)	Istmo en lugar de Salina Cruz (0.14)	si (se cambia Istmo por Salina cruz)
Salina cruz	Francia	Salina cruz en lugar de Francia (0.28)	Francia en lugar de Salina cruz (0.71)	no
Salina cruz	Frankfurt	Salina cruz en lugar de Frankfurt (0.42)	Frankfurt en lugar de Salina cruz (0.71)	no
Salina cruz	Angola	Salina cruz en lugar de Angola (0.28)	Angola en lugar de Salina cruz (0.71)	no
Resultado parcial 1: *Salina Cruz, Francia, Frankfurt, Angola*				
París	Salina cruz	París en lugar de Salina cruz (0.71)	Salina cruz en lugar de París (0.42)	no
París	Francia	París en lugar de Francia (0.0)	Francia en lugar de París (0.14)	si (se cambia Francia por París)
París	Frankfurt	París en lugar de Frankfurt (0.28)	Frankfurt en lugar de París (0.28)	no
París	Angola	París en lugar de Angola (0.28)	Angola en lugar de París (0.42)	no
Resultado parcial 2: *Salina Cruz, París, Frankfurt, Angola*				
Alemania	Salina cruz	Alemania en lugar de Salina cruz (0.71)	Salina cruz en lugar de Alemania (0.28)	no
Alemania	París	Alemania en lugar de París (0.28)	París en lugar de Alemania (0.14)	no
Alemania	Frankfurt	Alemania en lugar de Frankfurt (0.14)	Frankfurt en lugar de Alemania (0.0)	no
Alemania	Angola	Alemania en lugar de Angola (0.28)	Angola en lugar de Alemania (0.28)	no
Resultado parcial 3: *Salina Cruz, París, Frankfurt, Angola*				
África	Salina cruz	África en lugar de Salina cruz (0.71)	Salina cruz en lugar de África (0.14)	no
África	París	África en lugar de París (0.42)	París en lugar de África (0.14)	no
África	Frankfurt	África en lugar de Frankfurt (0.42)	Frankfurt en lugar de África (0.14)	no
África	Angola	África en lugar de Angola (0.14)	Angola en lugar de África (0.0)	no
Resultado final: *Salina Cruz, París, Frankfurt, Angola*				

3.9 Consideración de la aridad en los conceptos

La aridad de un concepto representa la cantidad de valores distintos que puede tomar (el concepto). En la notación OM (§3.4.1), la aridad se representa de la siguiente manera:

<arity> *valor de la aridad* </arity>

Donde:

valor de la aridad es un número que representa la cantidad de valores que puede tomar el concepto.

Las restricciones de aridad en un concepto pueden ser:

Monovaluada. Significa que solo puede tomar un solo valor en el tiempo.
Multivaluada. Significa que puede tomar más de un valor en el tiempo.

Se presenta un ejemplo aplicando la aridad *Monovaluada*.

Para el concepto C_C: `Benito Juárez`, en la ontología A la relación r_A: (mi madre es `Benito Juárez` `Brígida García`).

Nota: La relación r_A es monovaluada, naturalmente toda persona solo tiene una madre y un padre.

Para el concepto *cms*: `Benito Juárez`, en la B la relación r_B: (mi madre es `Benito Juárez Brígida López`).

OM busca B y A si `Margarita Maza` es sinónimo de `Brígida García`, si no aplica la teoría de la confusión [21] de usar `Margarita Maza` en lugar de `Brígida García` y viceversa, si el valor de la confusión *vc* es demasiado distante entre los conceptos conserva `Brígida García` en r_C, véase la Figura **3.34** si es cercano toma el menor (ver §3.8.1).

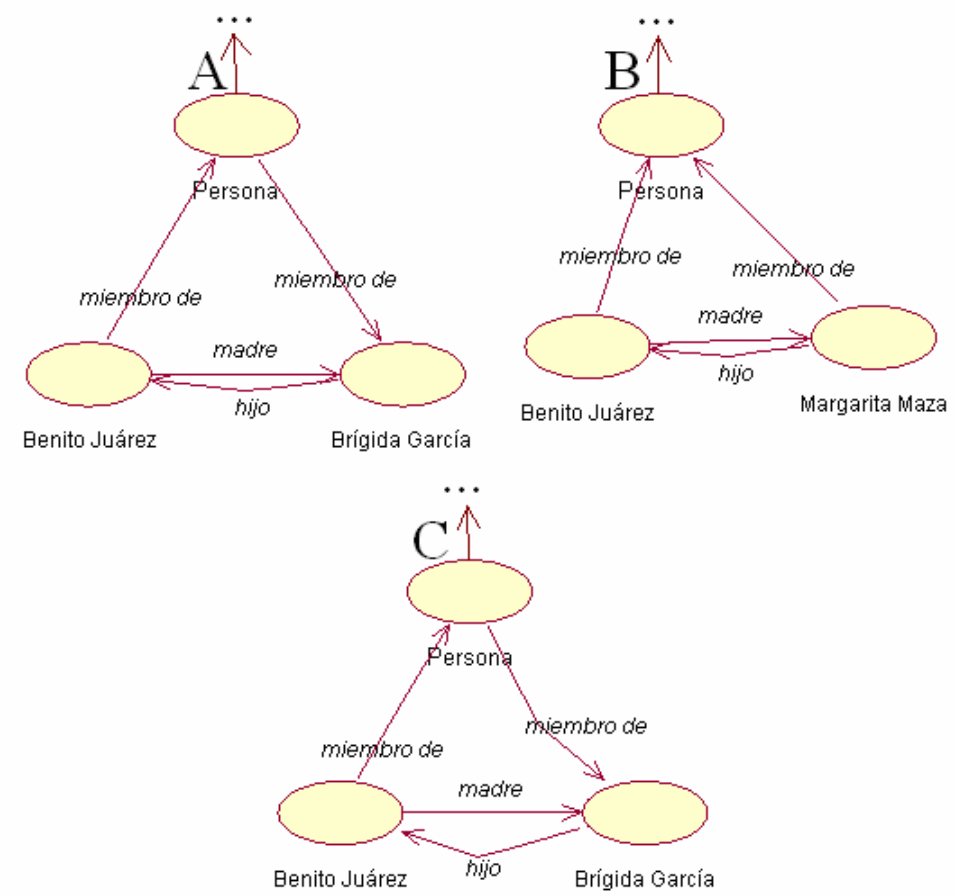


Figura 3.34 Representación de un ejemplo usando la aridad Monovaluada en la relación de un concepto

Se presenta un ejemplo aplicando la aridad *Multivaluada*.

La relación r_A: (Cargo político `Benito Juárez Diputado por Oaxaca`) y la relación r_B: (Cargo político `Benito Juárez Miembro del Congreso`).

Nota: En este ejemplo se considera la relación Cargo político de aridad multivaluada porque podrá tomar valores distintos.

OM identifica los valores que son distintos en la relación (y no son sinónimos), luego usa la teoría de la confusión [21] de usar `Diputado por Oaxaca` en lugar de `Miembro del Congreso` y viceversa, si son distantes (§3.8.1), acepta la nueva relación en r_C.

En la Figura 3.35 se presentan las ontologías de este ejemplo.

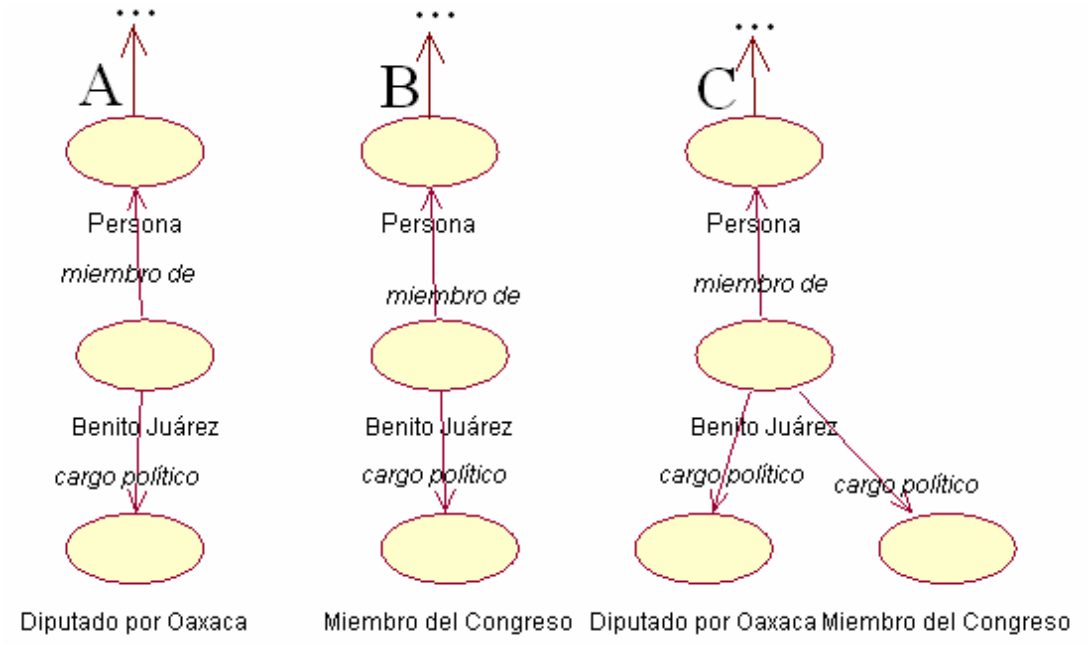


Figura 3.35 Representación de las ontologías B y C del ejemplo usando la aridad Multivaluada en la relación r_C

Las contradicciones expuestas en §3.8.1 son identificadas de acuerdo a la aridad de la relación. De hecho, se pretende que todos los conceptos en la ontología tengan definida una aridad, de lo contrario OM lo considera como multivaluada.

3.10 Organización de subconjunto a partición

Considerando que los conceptos y relaciones forman conjuntos, subconjuntos y particiones (§3.1), algunas veces los subconjuntos SC_A de un conjunto C_A en A, serán los mismos que formen una partición p_B en B. En tal caso, OM copia la partición p_B en C_C.

La organización de un subconjunto a partición se realiza antes de la copia de una partición.

Ejemplo:

En una Ontología A existe un concepto: `Grupo etnolingüístico de Oaxaca` que contiene a cuatro conceptos subconjuntos de éste:

1. Conjunto `zoque`,
2. Conjunto `ixcateco`,
3. Conjunto `huave` y
4. Conjunto `Mixteco`,

En la Figura 3.33 se presenta la ontología en forma gráfica.

En la Ontología B, hay una partición: `etnolingüístico` con los mismos elementos (1, 2, 3 y 4) del conjunto `Grupo etnolingüístico de Oaxaca` en A. En la Figura 3.36 se representan las ontologías.

El algoritmo OM toma el conjunto `Grupo etnolingüístico de Oaxaca` en A y coloca la partición p_B en él, para indicar que todos los subconjuntos (1, 2, 3, y 4) son también particiones, es decir, son mutuamente exclusivos y colectivamente exhaustivos (§3.4.2).

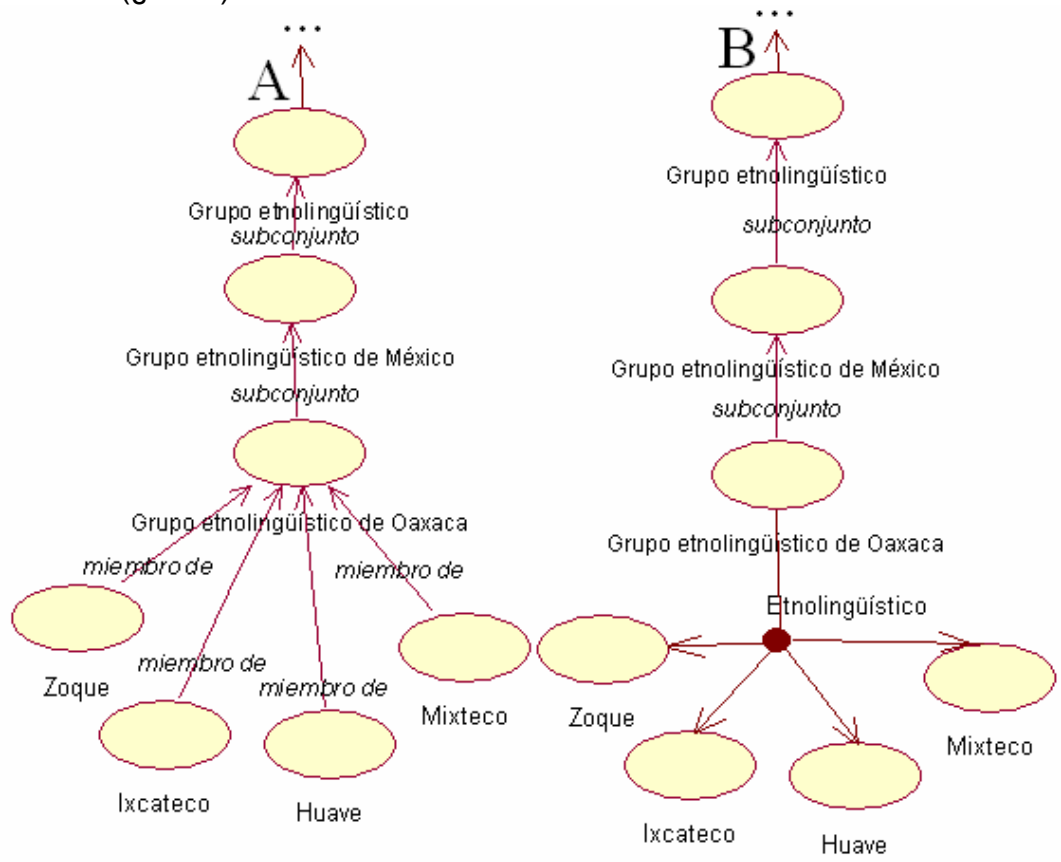


Figura 3.36 Representación de un subconjunto en A y una partición en p_B que se pueden complementar

La Figura 3.37 presenta el resultado en la cual la partición `etnolingüistico` se ha añadido al conjunto de relaciones del concepto `Grupo etnolingüistico de Oaxaca` y cada uno de los valores de la partición apuntan a los conceptos que a su vez son subconjuntos de `Grupo etnolingüistico de Oaxaca`.

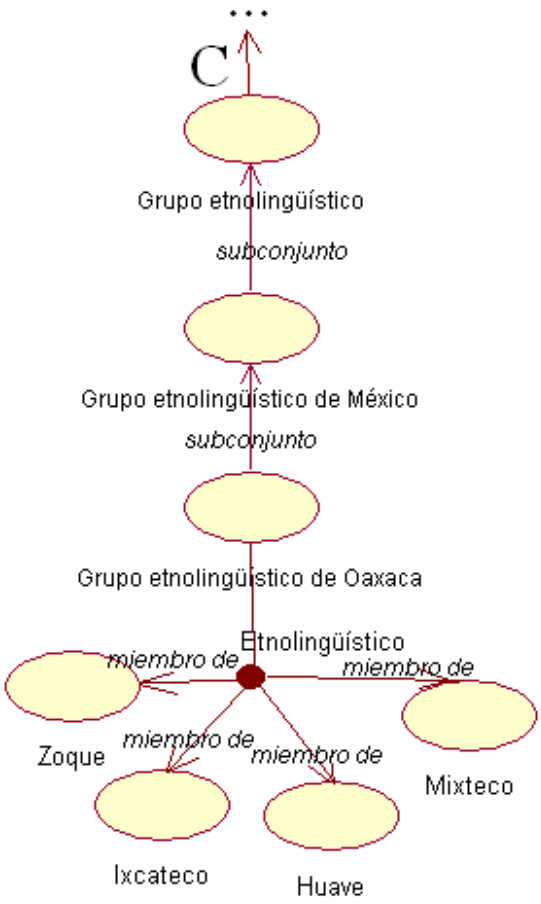


Figura 3.37 El concepto en C_C que ha recibido la partición p_B conserva los subconjuntos ya que cada valor de la partición señala a uno de estos

4. Ejemplos y Resultados

En este capítulo se presentan varios ejemplos reales resueltos con el sistema OM (la implementación del algoritmo OM).

4.1 Unión dos ontologías A y B de productos

Dadas dos ontologías A y B de productos de la empresa SUMESA con la descripción del sitio: www.comerci.com.mx/ donde se ha tomado varias listas de productos y se ha obtenido las ontologías fuentes, la fusión A ∪ B resultó con 394 conceptos. Esta fusión no presentó casos interesantes porque las ontologías no fueron descritas al nivel de detalle (eran tan solo listas de productos) que se requiere para observar el comportamiento de OM.

Este ejemplo ha dejado la experiencia de que la fusión de ontologías se torna más interesante si se toman los datos de documentos que describen un universo (o una parte) del discurso.

Se presentan en este apartado uno de los casos presentados en la fusión.

4.1.1 Copia de nuevas relaciones cuyos elementos son conceptos sin argumentos

Una relación explícita (§3.7.2) puede ser un concepto con o sin argumentos (§3.1), cuando éste es un pre-concepto OM (§3.2) fusiona diferente las relaciones; comparando únicamente la sinonimia entre los conceptos involucrados.

Supóngase que en B, existe un concepto `Meat` con la relación r_B definida como: `taste = various` que desea fusionarse con `meats` en la A y no está definida la aridad de la relación en A (OM evalúa esta aridad como multivaluada, además todas las relaciones con conceptos sin argumentos son multivaluadas), verifica la presencia de relación r_B en `meats`, luego busca relaciones sinónimas, al no encontrarlas añade la nueva relación en `meats` en C.

En la
Figura **4.1**, se presenta la ontología C.

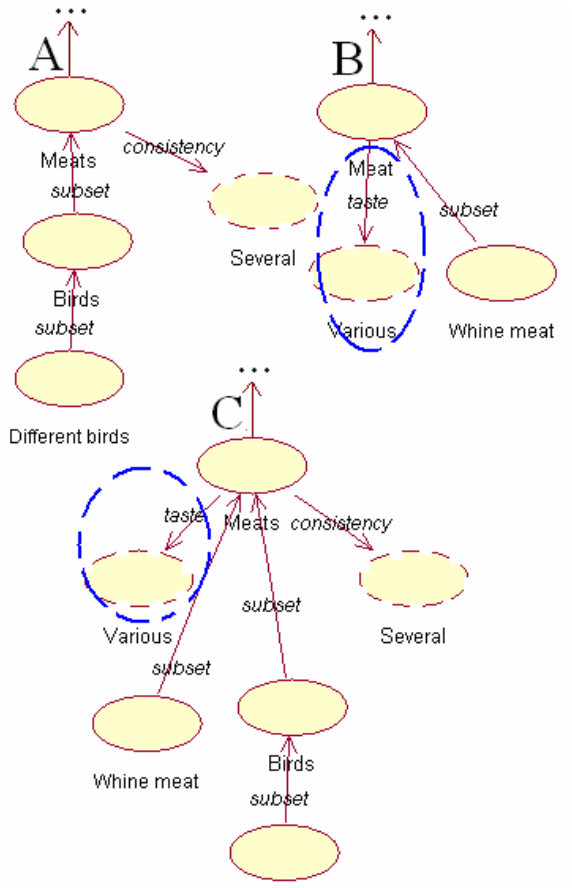


Figura 4.1 Donde las relaciones involucradas se identifican encerrando los óvalos con círculos punteados

4.2 Unión de dos ontologías A y B de localizaciones geográficas.

Se han tomado de Internet tres documentos[65] que describen al estado de Oaxaca, de ellos se han obtenido (manualmente) tres ontologías (una por cada documento Web). La ontología A con 234 conceptos, la B con 117 conceptos y la unión A ∪ B con 309 conceptos.

4.2.1 Adición de relaciones cuyos elementos son conceptos con argumentos y no son sinónimos

Se tiene en la B la relación `Clima = Clima de Oaxaca` del concepto `Oaxaca` se quiere fusionar con las relaciones (solo hay una) `Superficie = Superficie de Oaxaca`, del concepto `Oaxaca` en la A, en ambas relaciones hay conceptos con argumentos (§3.1), OM compara `Clima` de B con `Superficie` de A, no son iguales ni sinónimos, al usar la teoría de la confusión [21] éstos resultan muy distantes (§3.8.1), luego se comparan: `Clima` de

[65] Las referencias a estos documentos son: www.oaxaca-mio.com/atrac_turisticos/infooaxaca.htm y www.elbalero.gob.mx/explora/ html/oaxaca/ geografia.html.

`Oaxaca de B` con `Superficie de Oaxaca de A` tampoco son iguales ni sinónimos y también resultan distantes al usar la teoría de la confusión [21], OM añade la relación `Clima = Clima de Oaxaca` al concepto `Oaxaca` en la C (durante la fusión de relaciones, también se añaden a la C, los conceptos involucrados).

En la Figura 4.2 se presenta una parte de la A antes de la fusión.

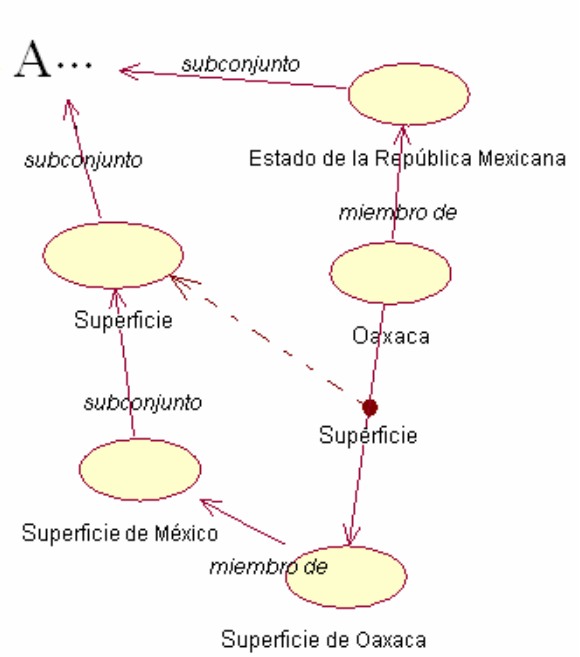


Figura 4.2 Ontología A con el concepto Oaxaca y su relación explícita y sus elementos como conceptos, cada uno de estos apunta (mediante la flecha punteada) a su concepto en la ontología A

En la Figura 4.3 se presenta una parte de la ontología B.

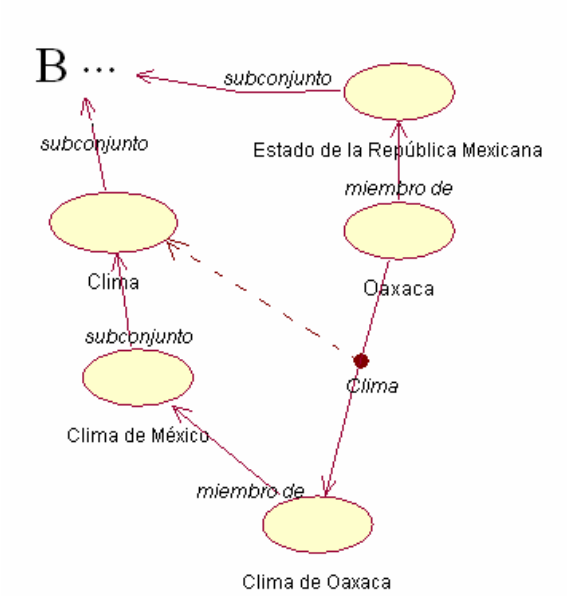


Figura 4.3 Ontología B donde se presenta la relación Clima que es también un concepto, la flecha discontinua apunta al concepto.

En la Figura 4.4 se presenta parte de la Ontología C con las relaciones añadidas, se observa que se han añadido los conceptos de la relación que no estaban presentes en la A.

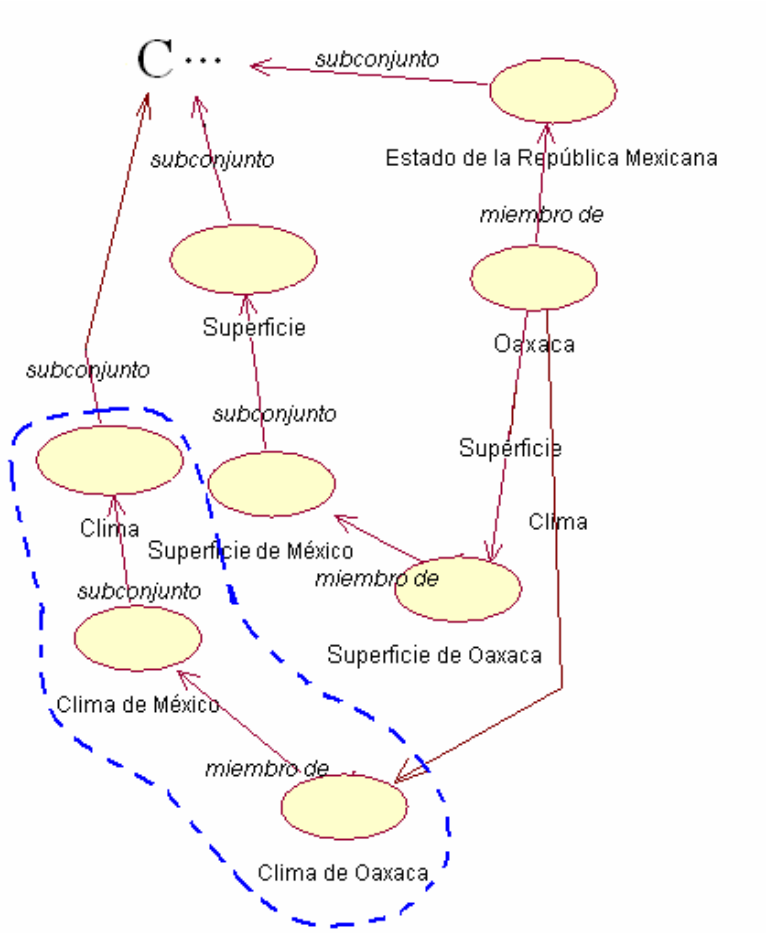


Figura 4.4 Ontología C con la unión de nuevas relaciones en el concepto Oaxaca. La línea discontinua encierra los nuevos elementos añadidos a C

4.2.2 Adición de relaciones cuyos elementos son conceptos con argumentos y son sinónimos

OM identifica los sinónimos (§3.4.6) y no los añade como nuevos conceptos sino compara las definiciones de estos sinónimos para adicionar nuevas frases o palabras de la definición del concepto en B con las de A.

Por ejemplo:

Tomando en cuenta la relación del concepto `Oaxaca` en la A de la Figura 4.2, se pretende copiar la relación `Extensión territorial = Extensión territorial de Oaxaca` de la B cuyos elementos son conceptos y se han identificado como sinónimos de la relación en A, véase la Figura 4.5.

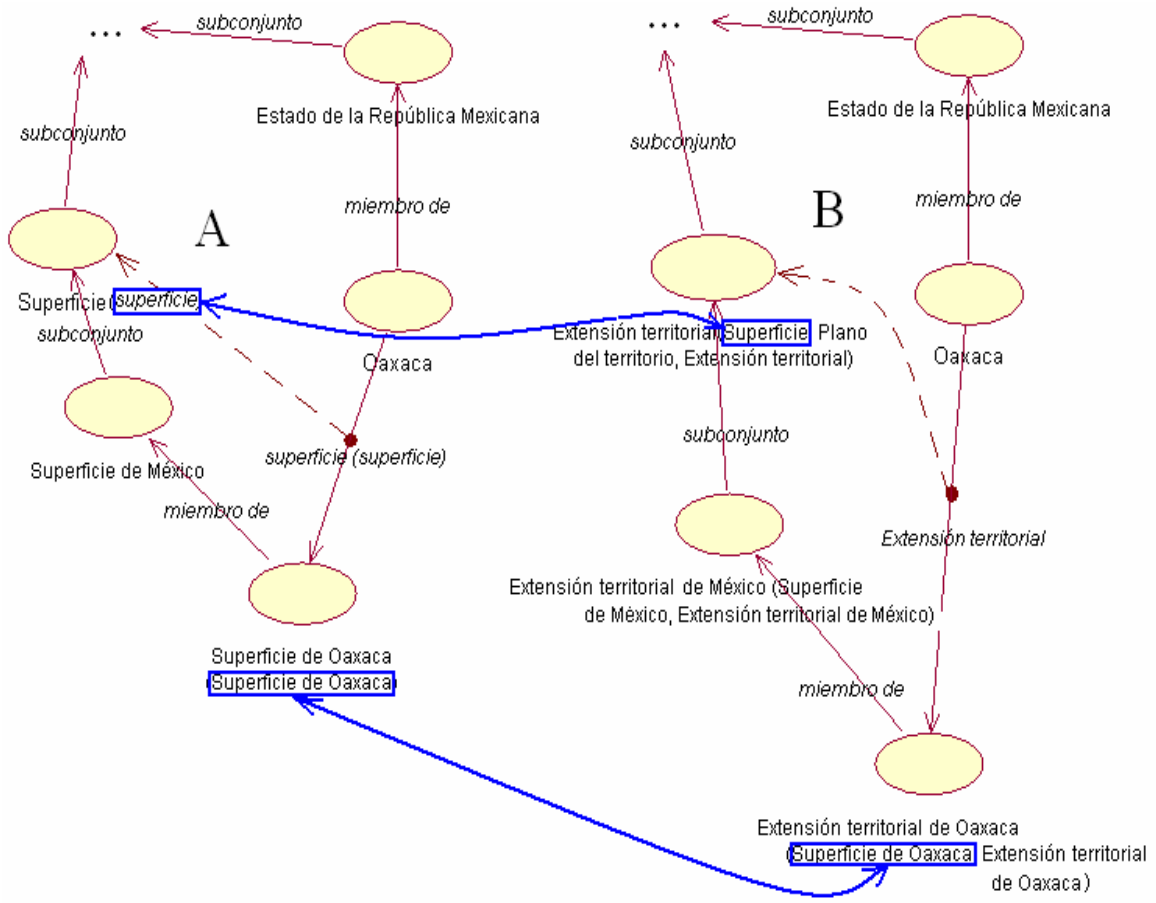


Figura 4.5 Muestra la B donde los rectángulos encierran los conceptos de la relación que indican las definiciones de los conceptos, las flechas con línea gruesa salen de cada elemento de la relación para señalar a su correspondiente concepto sinónimo en la ontología

En la Figura 4.6 se presenta el resultado del algoritmo OM (§3.2). Se observa en la C que no se ha copiado la nueva relación sino se han enriquecido las definiciones de los conceptos identificados como sinónimos.

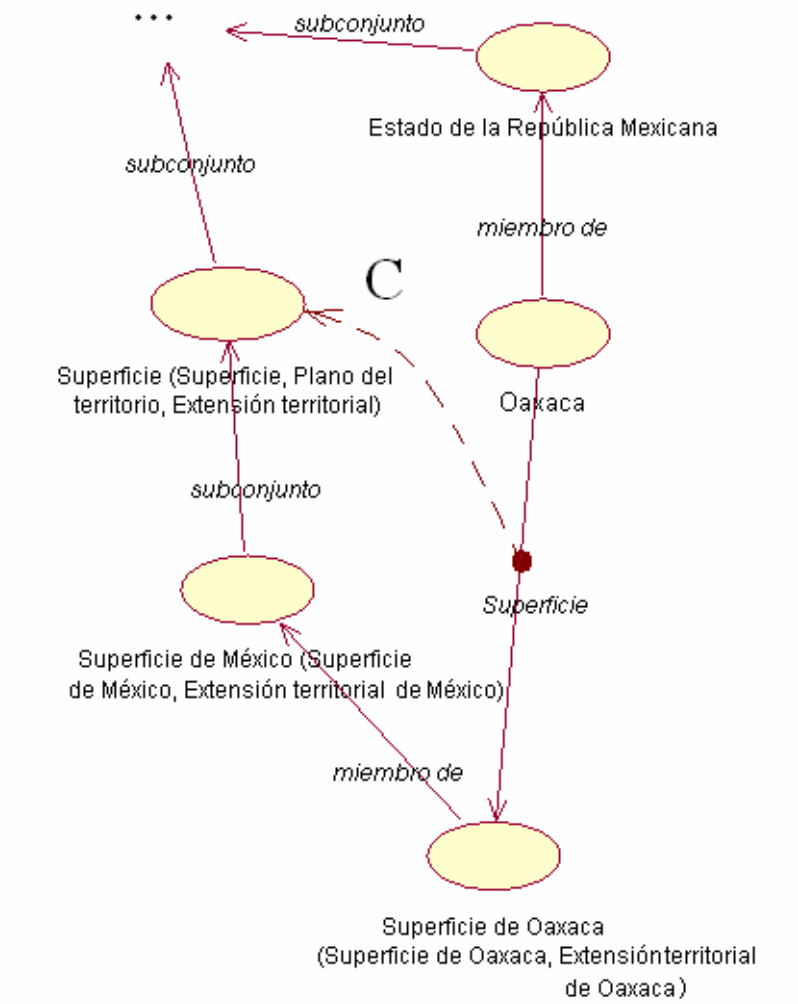


Figura 4.6 Presentación de la C como resultado de la unión, obsérvese en los paréntesis las palabras añadidas la definición de cada concepto identificado como sinónimo

4.2.3 Adición de nuevos conceptos a una relación existente

La relación en A está presente en B pero no el valor de éstas, si la relación en A es un pre-concepto, su aridad es multivaluada (§3.9) entonces, se aceptan los nuevos valores para la relación.

Ejemplo: considerando la relación actividad = `Actividad turística`, `Actividad pesquera` del Concepto `Oaxaca` de la A, representada en la Figura 4.7 se pretende añadir a este concepto las relaciones del concepto `Oaxaca` de la B.

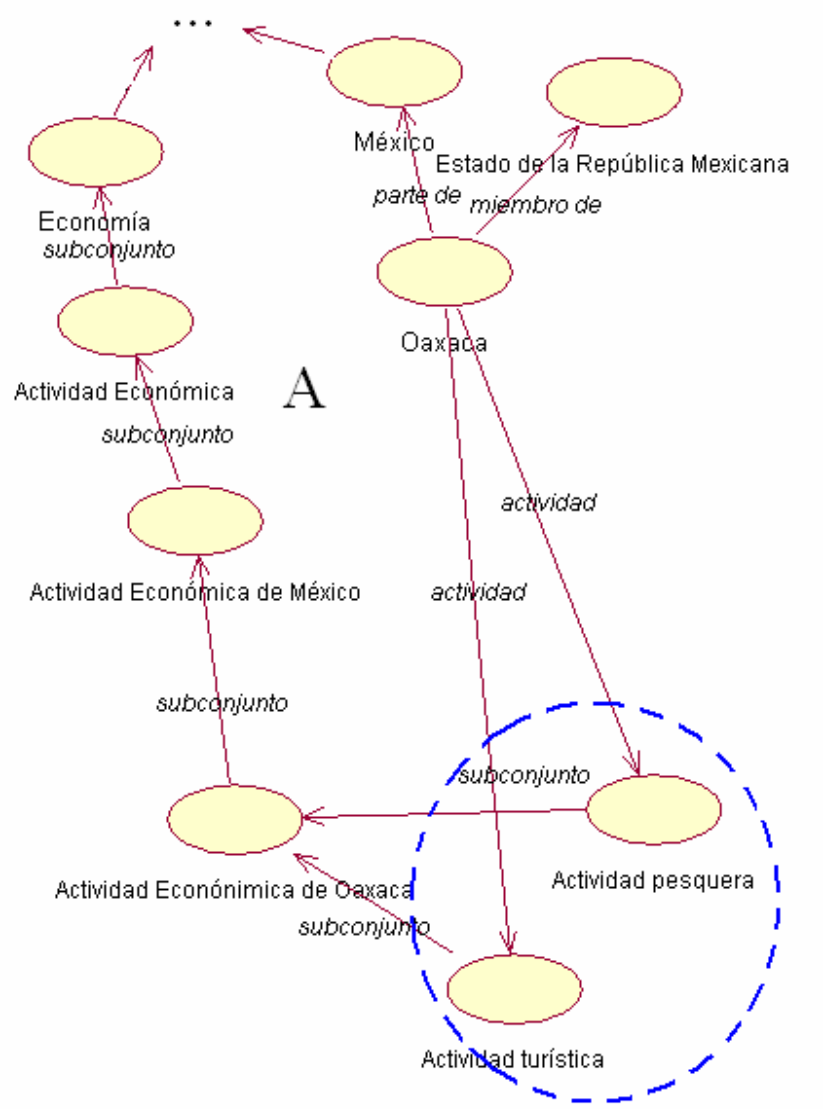


Figura 4.7 Presentación de los valores de la relación que hacen referencia a un concepto. Obsérvese que el nombre de la relación (actividad) es un pre-concepto por lo tanto, se considera de aridad multivaluada

En la B, específicamente en el concepto `Oaxaca` se encuentra la relación, actividad = `Actividad comercial` véase la Figura 4.8.

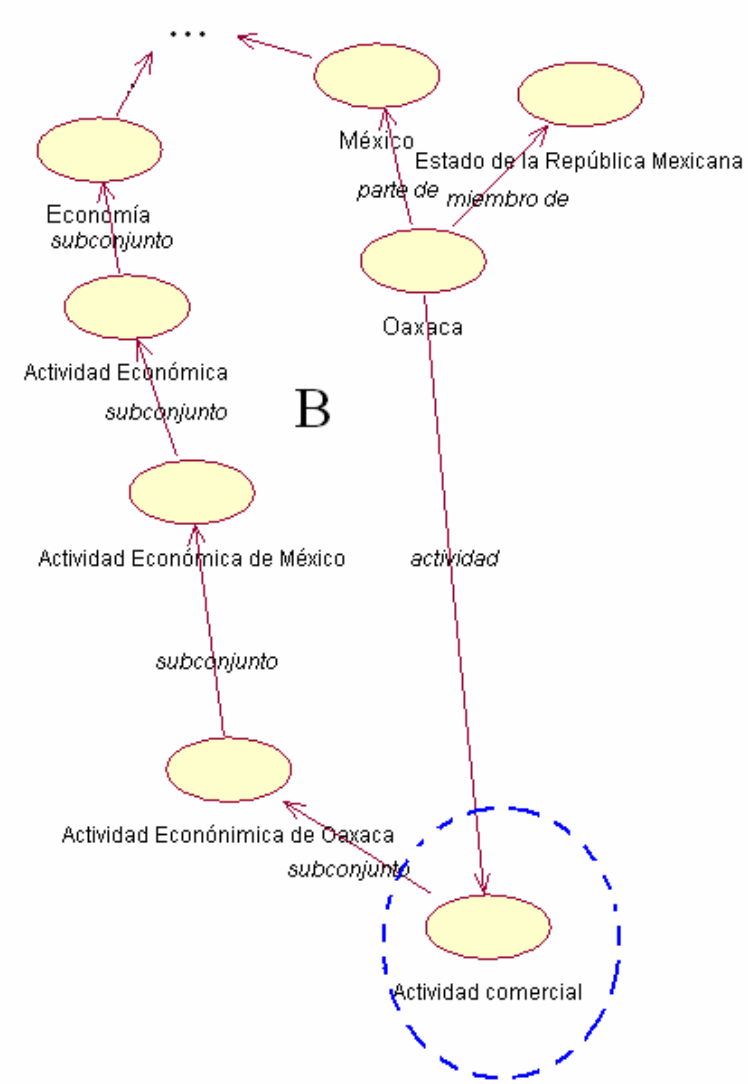


Figura 4.8 Presenta la B con la relación actividad que se quiere añadir a C

Se ha añadido el nuevo valor a la relación en C, véase la Figura 4.9 en la que se presenta la relación actividad con el nuevo valor como un concepto cuyos antecesores ya están en C, por lo que solo se añade el concepto.

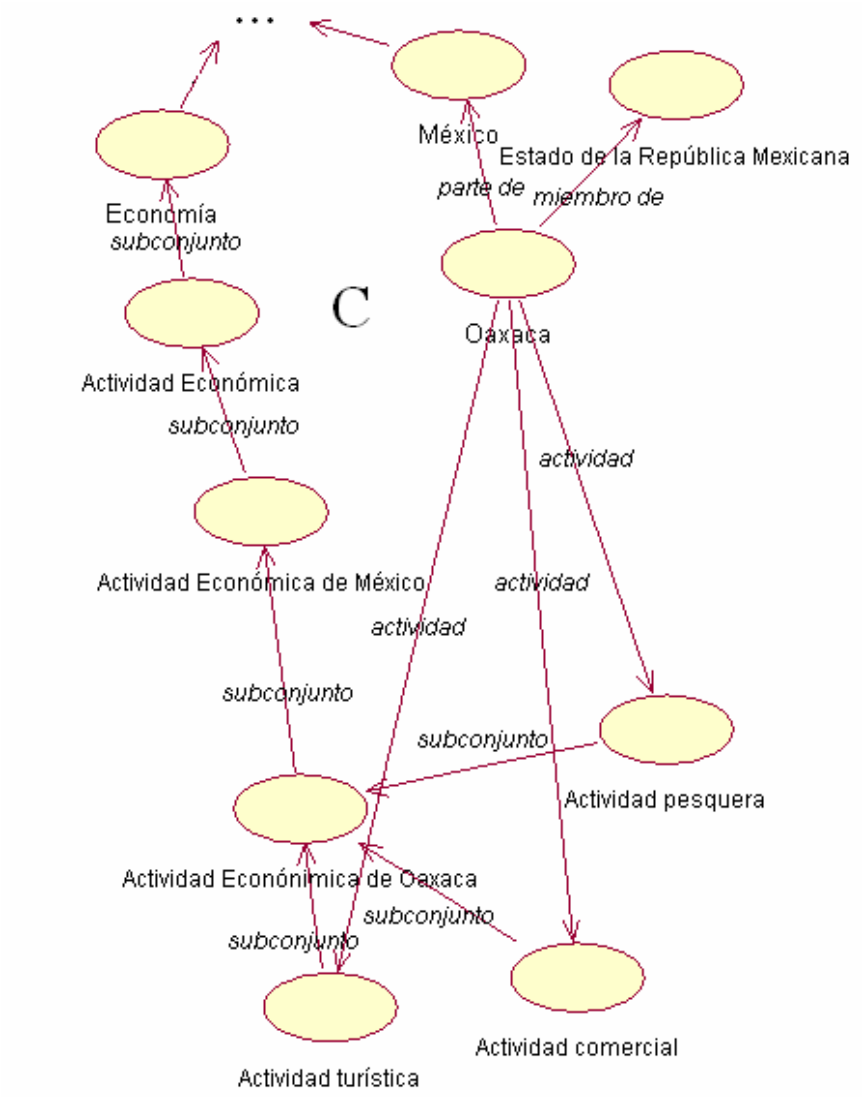


Figura 4.9 Presentación la C como resultado de la fusión

4.2.4 Adición de nuevos conceptos con argumentos en una relación existente

Considerando que la relación en la A es similar a otra relación en la B, se comparan cada uno de sus valores (si tiene varios), si sus valores son distintos se comparan sinónimos, si no hay sinónimos, se halla el grado de confusión entre estos (para dejar los conceptos más específicos y eliminar los más generales §3.8.3.6). Por ejemplo si se usa la teoría de la confusión [21] entre `Oaxaca` y `Santa maría tecomavaca`, el valor más específico es `Santa maría tecomavaca` porque es un subconjunto (municipio) de `Oaxaca`. Si en una lista de municipios de `Oaxaca`, también se encuentra `Oaxaca`, este valor será eliminado de la lista (por redundancia).

El ejemplo se representa en la Figura 4.10, donde los conceptos involucrados se indican en los círculos discontinuos.

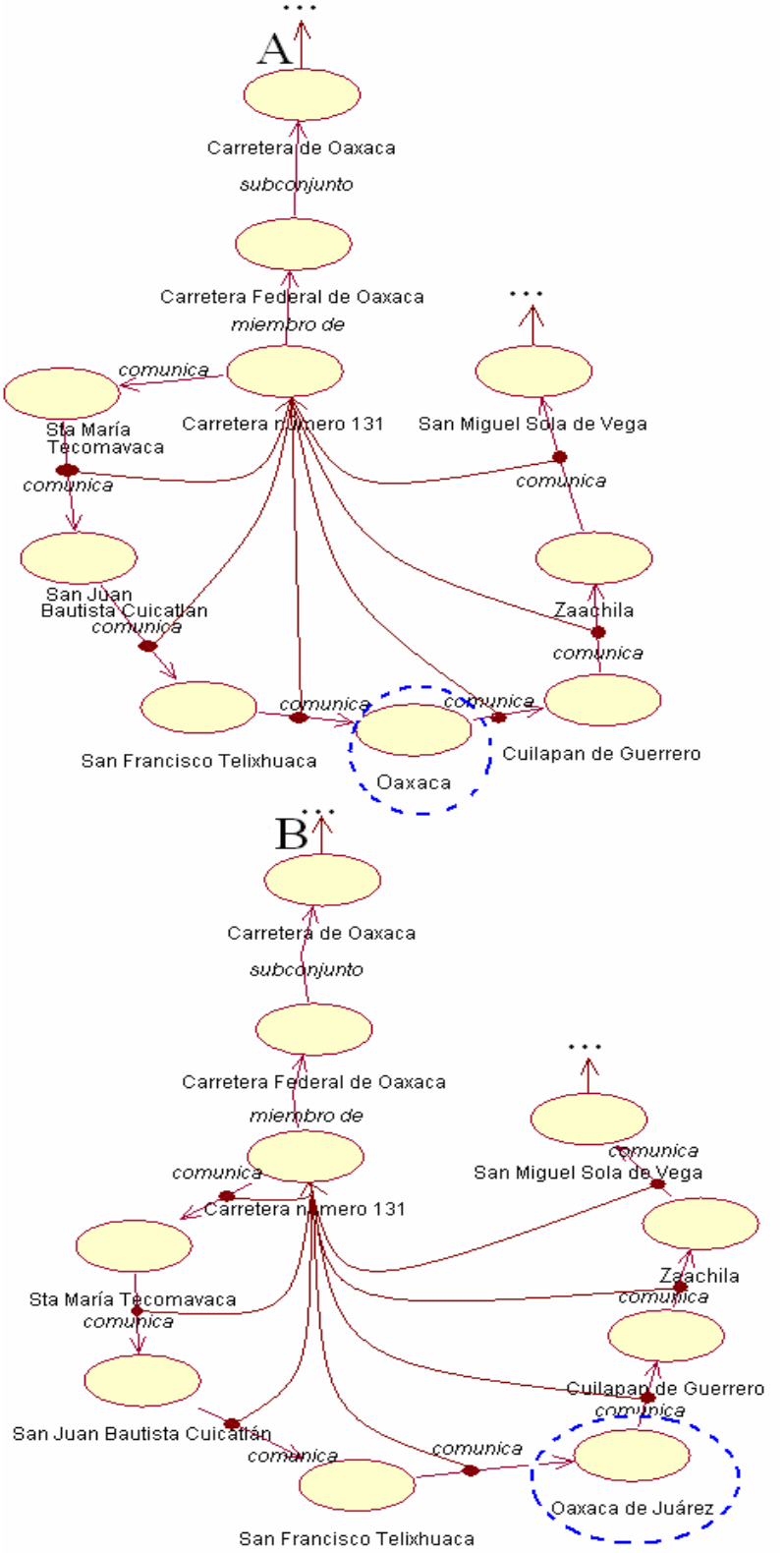


Figura 4.10 Las ontologías A y B con la relación número 131 donde hay un valor: Oaxaca que será eliminado cuando se realice la fusión

En la
Figura **4.11** se presenta la ontología C donde se ha eliminado el valor Oaxaca.

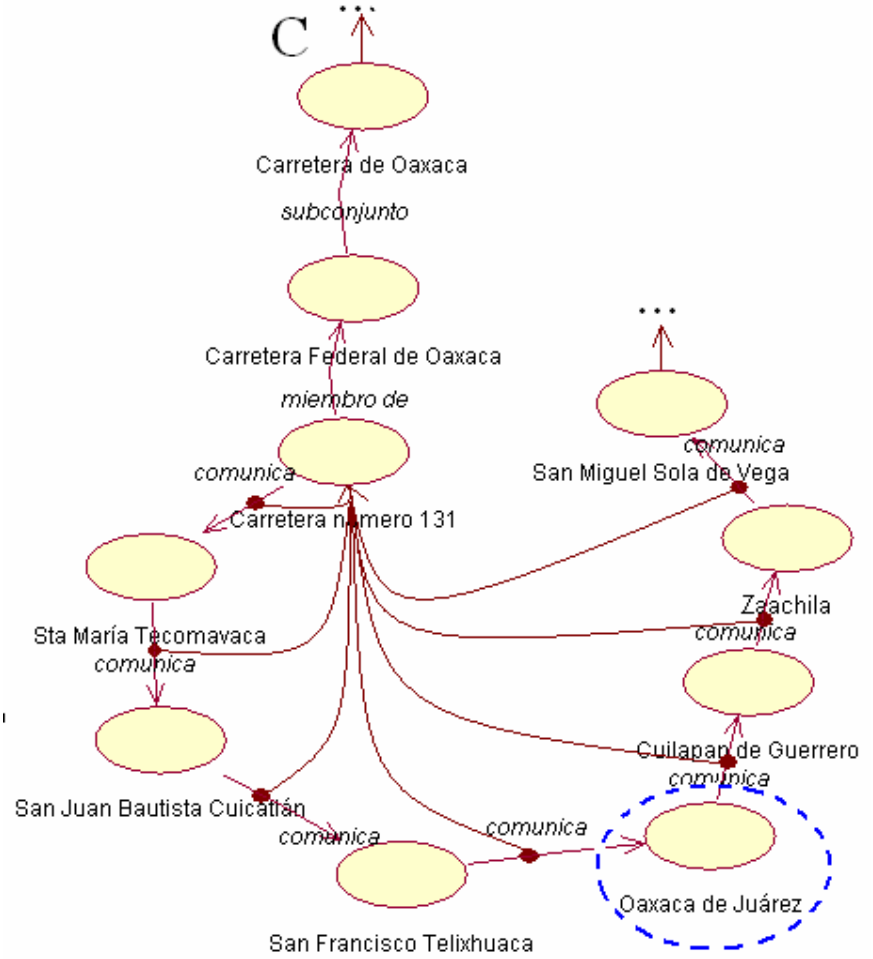


Figura 4.11 Representación de la ontología resultante

4.2.5 Identificación del nombre de una relación

Existen relaciones definidas por un conjunto de palabras, artículos y conectores tales como: "*es un hijo de*", que es lo mismo a: "*hijo*" o bien: "*ubicado de norte a sur*" que es diferente a: "*ubicado de sur a norte*". El algoritmo OM, compara cada una de las palabras que forman el nombre de la propiedad y el orden de estas, eliminando en este proceso los artículos, conectores y demás elementos que complementan este nombre, pero que no son básicos.

Por ejemplo. Sean las relaciones explícitas del concepto: 0axaca de la ontología C (véase la Figura 4.12) y las relaciones explícitas en el concepto más similar *cms*: 0axaca en la B (véase la Figura 4.13).

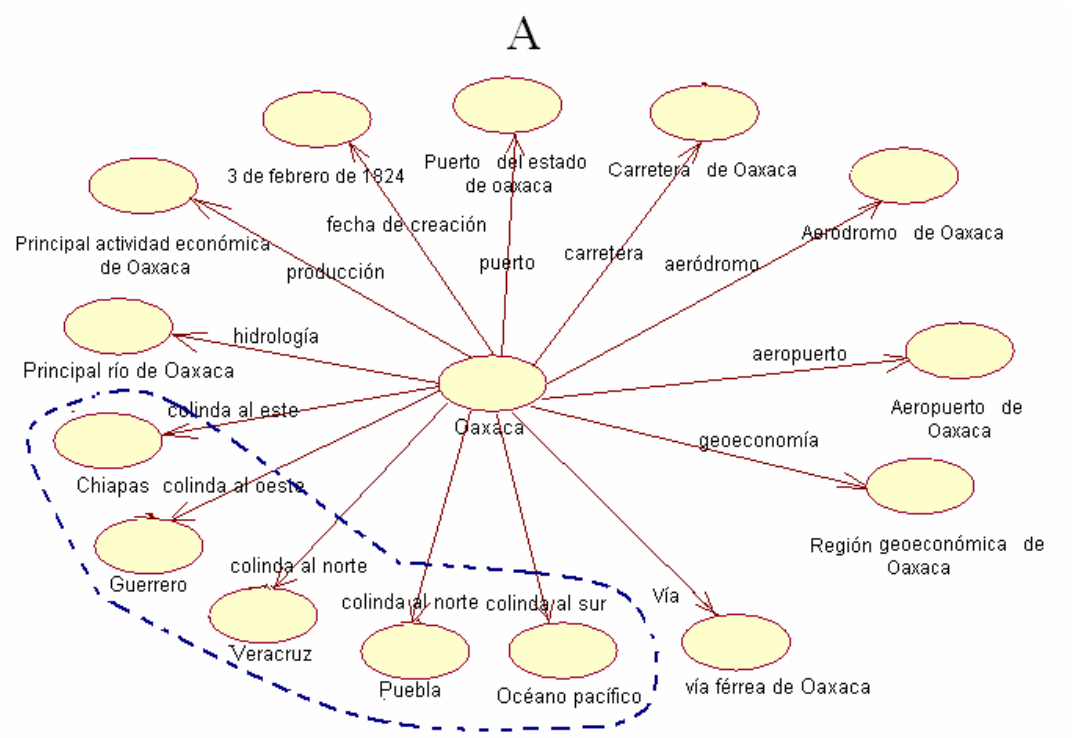


Figura 4.12 Presentación de las relaciones del concepto Oaxaca en la ontología A

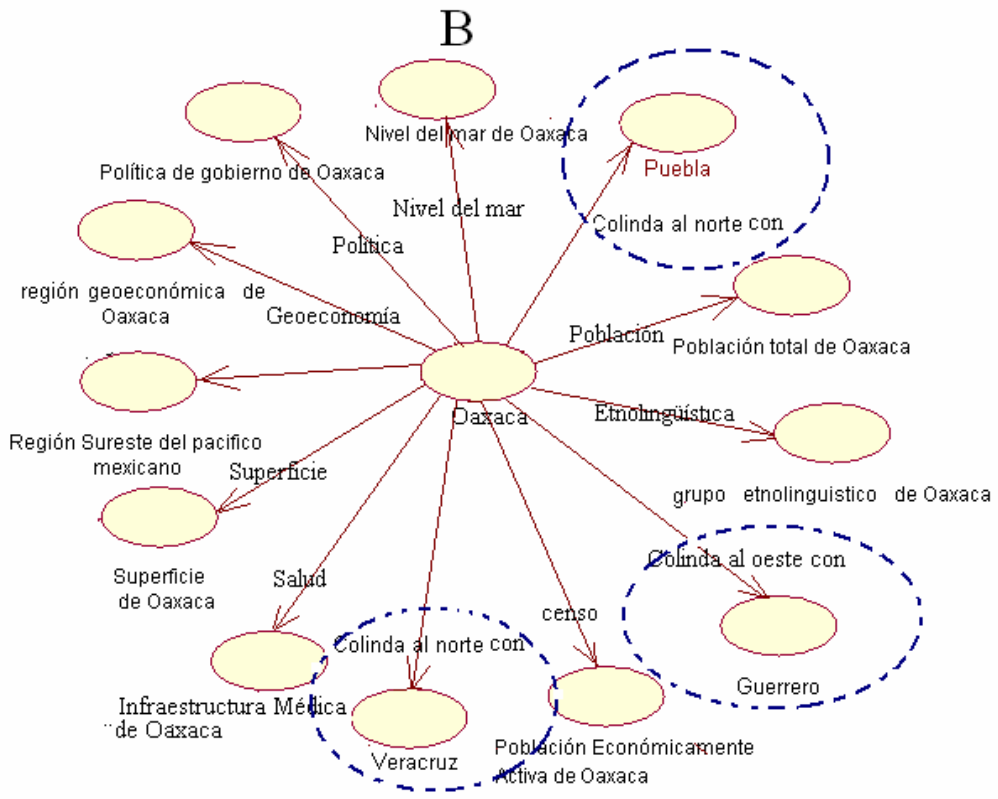


Figura 4.13 Presentación de las relaciones del concepto más similar a Oaxaca en la ontología B

Ambos nombres de propiedad aparentemente son distintos, pero tienen el mismo significado, OM los reconoce como iguales. En el ejemplo, estos nombres son conceptos sin argumentos (palabras o frases), por lo que se identifican como de aridad multivaluada, por lo tanto los valores `Veracruz` en A y `Puebla` en B serán añadidos a C conservándose el nombre de la relación en A. Véase la Figura 4.14.

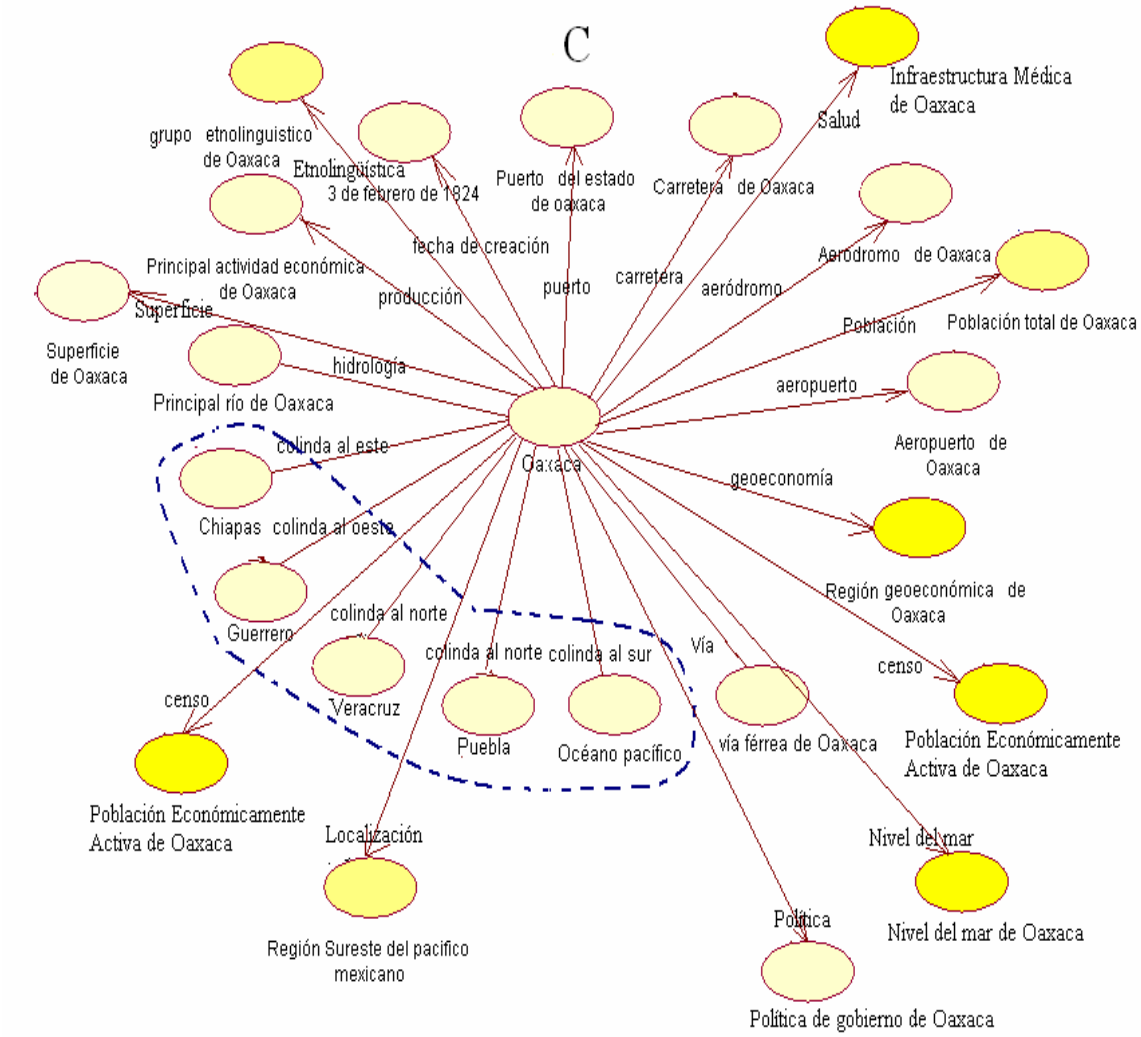


Figura 4.14 Presentación de las relaciones del concepto Oaxaca en la ontología resultante

Si los nombres de las propiedades fuesen conceptos con argumentos, se completarían sus definiciones[66]. Si el nombre de la propiedad en C fuese un pre-concepto y en la B fuese un concepto, el concepto en B se añadiría a la C.

Ocurre lo mismo en la copia de las particiones.

[66] Quiere decir que se adicionan nuevas palabras que forman la definición del concepto con argumento en C con aquellas palabras del concepto con argumento en B.

4.2.6 Relación subconjunto en A y relación partición en B

Se presenta el caso del concepto `Región geoeconómica de Oaxaca` tiene como subconjuntos a los siguientes conceptos: `Región papaloapan`, `Región Istmo`, `Región Mixteca`, `Región Cañada`, `Región Sierra Sur`, `Región Costa`, `Región Valles Centrales`, `Región Sierra Norte`, véase la Figura 4.15.

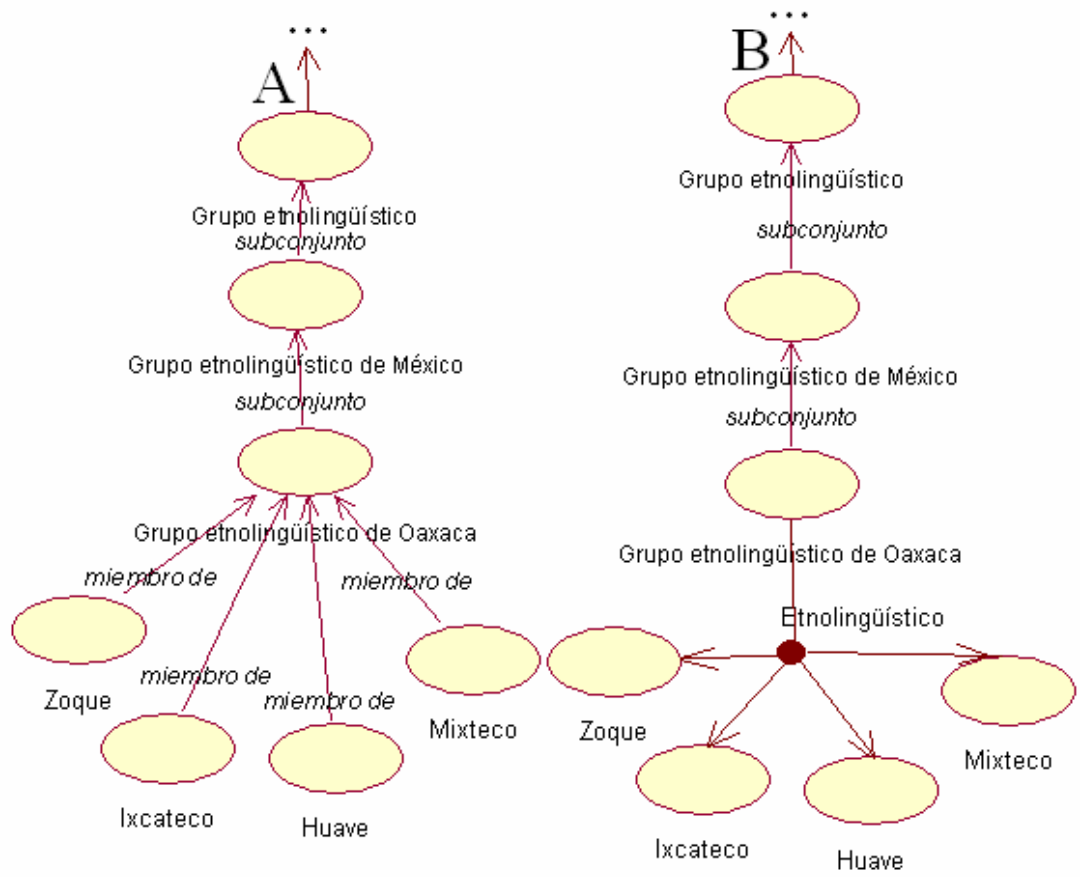


Figura 4.15 Presentación de la Ontología A Oaxaca con los subconjuntos del conjunto Región geoeconómica de Oaxaca. En B, se presenta el mismo concepto pero con distinta estructura, pero se encuentra representado como una partición

OM añade la partición en `Región geoeconómica de Oaxaca` en C, enriqueciendo cada valor de esta partición con los subconjuntos que ya estaban en A, de tal manera que la unión de partición y subconjuntos se presenta en la Figura 4.16 donde se muestra la relación antes y después de la fusión.

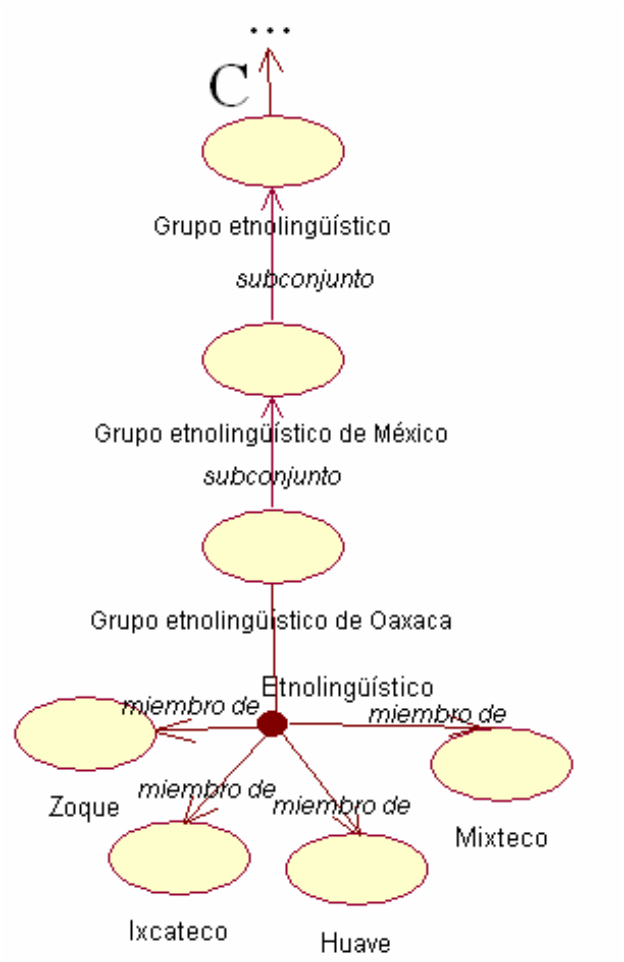


Figura 4.16 Presentación gráfica de la relación Región geoeconómica de Oaxaca

OM aparentemente elimina el subconjunto `Región neoeconómica de Oaxaca` para adicionar la partición y todos sus elementos se vuelvan elementos de la partición, pero el conjunto sigue existiendo, la partición le da el significado: "*no hay otro elemento para este conjunto, estos son todos*", la lógica de las particiones es: cada elemento es mutuamente exclusivo con los otros y juntos son colectivamente exhaustivos.

4.3 Unión de tres ontologías de descripción de animales

Se han tomado de Internet dos documentos que describen las características de las Tortugas[67], se han obtenido (manualmente) dos ontologías (una por cada documento Web). La ontología A con 58 conceptos, la B con 30 conceptos y la unión A ∪ B con 70 conceptos.

[67] Las descripciones que dieron origen a esas ontologías se pueden ver en las direcciones de Internet: www.damisela.com/zoo/rep/tortugas/index.htm y www.foyel.com/cartillas/37/tortugas_en_xtincion.html.

4.3.1 Verificación de las relaciones redundantes

La verificación de las relaciones redundantes se han explicado en §3.7, por lo que se aplicará a un ejemplo particular: El problema de la redundancia surge cuando las relaciones implícitas (§3.7.1) que llevan a los antecesores en B son del mismo tipo que los que lleva a concepto y su antecesor en A.

La
Figura **4.17** se presenta la ontología A cuyo concepto `Reptil` (en el círculo discontinuo) tiene una relación subconjunto con su antecesor `Vertebrado`.

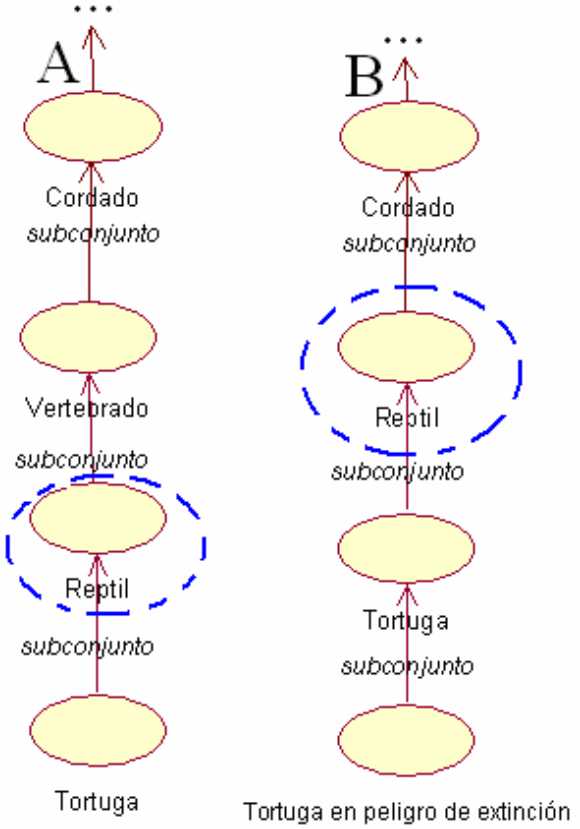


Figura 4.17 Presentación de las relaciones implícitas en la C y la relación implícita entre Reptil y Cosa en B

Antes de añadir una nueva relación en C, OM decide verificar si provoca redundancia. En este caso, `Reptil` es subconjunto de `Cordado` en B y de `Vertebrado` en C, pero `Vertebrado` a su vez es subconjunto de `Cordado` (hay redundancia). OM elimina la relación entre `Reptil` y `Cordado` conservando la relación más específica (`Reptil` subconjunto de `Vertebrado`) Si la relación implícita fuese otra (type por ejemplo) se aceptaría `Reptil` como subconjunto de `Cordado` y `Reptil` como tipo de `Vertebrado` es decir, OM realiza un análisis de las relaciones implícitas entre los conceptos, y no se consideran redundantes aquellas cuyos nombres de relación sean distintos en la trayectoria del concepto a sus antecesores.

La Figura 4.18 muestra la C antes y después de la eliminación de la relación redundante.

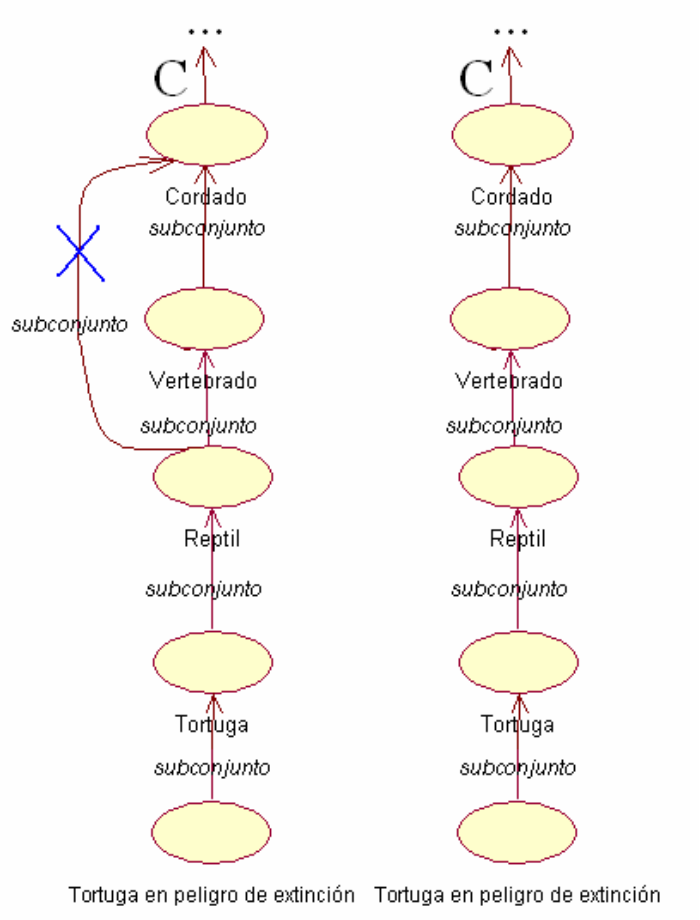


Figura 4.18 Presentación del resultado de la verificación de las relaciones transitivas

4.4 Unión de dos ontologías de descripción de Flores

Se han tomado de Internet dos documentos que describen las características de la Amapola[68], se han obtenido (manualmente) dos ontologías, una por cada documento Web, (se agradece a Paola Nery[69] por apoyar en la construcción de la ontología A y poder tener diversidad en la definición de sus elementos). La ontología A con 34 conceptos, la B con 35 conceptos y la unión A ∪ B con 58 conceptos.

[68] La dirección de Internet donde se pueden consultar los documentos originales es: es.wikipedia.org/wiki/Amapola y www.buscajalisco.com/bj/salud/herbolaria.php?id=1.

[69] Paola Nery, estudiante de doctorado del CIC-IPN, se encuentra desarrollando un analizador de documentos a ontologías.

4.4.1 Adición de las propiedades e hijos de un concepto

Se pretende copiar de B a la ontología A el concepto `Amapola`, que ya se encuentra en A, véase la

Figura **4.19**, donde se presentan las ontologías en la notación OM, los conceptos a fusionar se indican con línea contínua mientras que cada antecesor con línea discontinua.

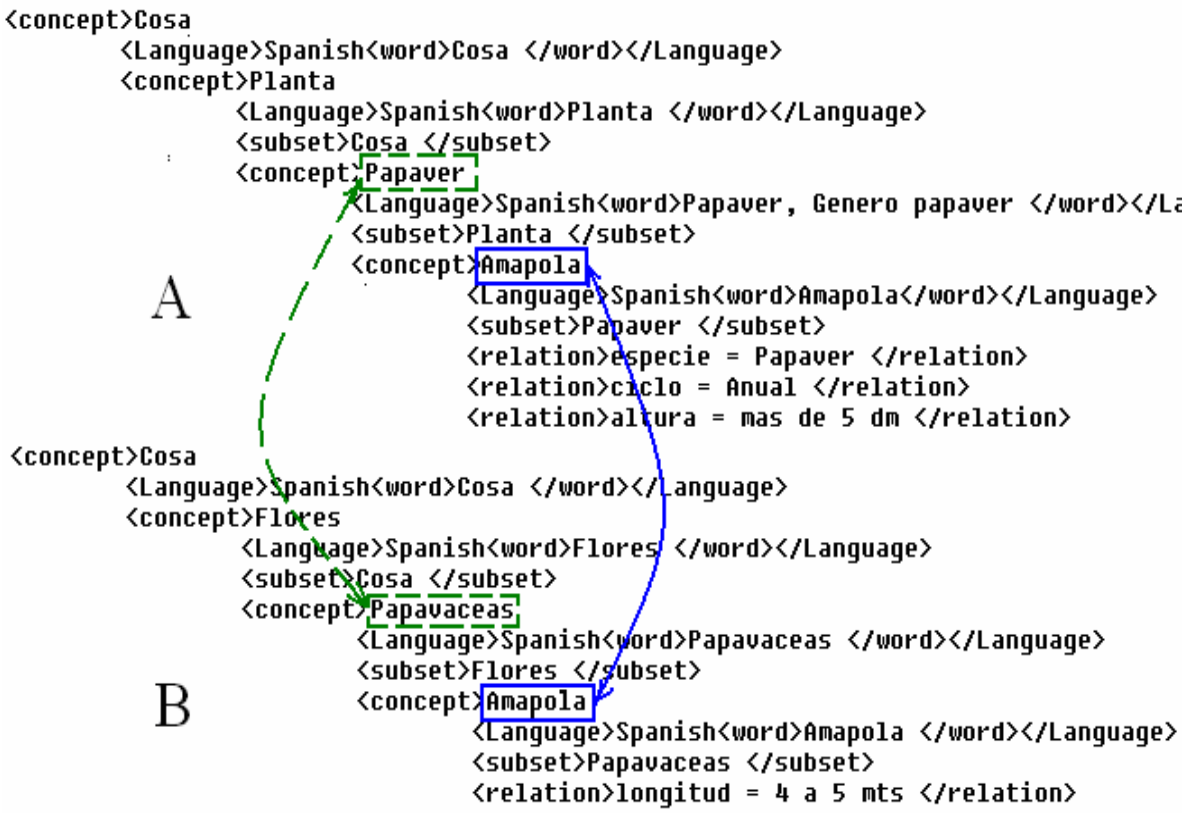


Figura 4.19 se presentan las ontologías A y B con el concepto Amapola cuyos antecesores son distintos

El algoritmo COM [7] no devuelve un concepto más similar en B para `Amapola` en A, debido a que los antecesores no coinciden y la mayoría de las propiedades de los hijos de `Amapola` en la A no coinciden con la mayoría de las propiedades de los hijos de `Amapola` en la B, en la

Figura **4.20** se presentan los hijos de `Amapola` en la A de los cuales solo el concepto `Tallo` en coinciente con `Tallo` en B (indicado en el rectángulo).

A

```
<concept>Amapola
        <Language>Spanish<word>Amapola</word></Language>
        <subset>Papaver </subset>
        <relation>especie = Papaver </relation>
        <relation>ciclo = Anual </relation>
        <relation>altura = mas de 5 dm </relation>
        <concept>Fruto
                <Language>Spanish<word>Fruto </word></Language>
                <part>Amapola </part>
                <type>Cápsula </type>
                <relation>Color = verde pálido </relation>
                <relation>forma = Cónica </relation>
        </concept>
        <concept>Flor
                <Language>Spanish<word>Flor </word></Language>
                <part>Amapola </part>
                <relation>forma = casi esférica, Acampanada</relation>
                <relation>Color = escarlata </relation>
        </concept>
        <concept>Tallo
                <Language>Spanish<word>Tallo </word></Language>
                <part>Amapola </part>
                <relation>ramificación = poco ramificado </relation>
                <relation>forma = Erecta </relation>
                <relation>consta de = Nervadura central </relation>
                <relation>forma = fina </relation>
        </concept>
</concept>
```

Figura 4.20 se presentan los hijos del concepto Amapola en la A con las propiedades de Tallo que tiene una leve coincidencia con Tallo en B

Se presenta la Figura 4.21 con las propiedades de Amapola en la B.

B

```
<concept>Amapola
        <Language>Spanish<word>Amapola </word></Language>
        <subset>Papavaceas </subset>
        <relation>longitud = 4 a 5 mts </relation>
        <concept>Semilla
                <Language>Spanish<word>Semilla </word></Language>
                <part>Amapola </part>
        </concept>
        <concept>Tallo
                <Language>Spanish<word>Tallo </word></Language>
                <part>Amapola </part>
                <relation>Partition = Color {*:Gris, Verde} </relation>
                <relation>tiene = Veta </relation>
                <relation>forma = lisa </relation>
        </concept>
        <concept>Hoja
                <Language>Spanish<word>Hoja </word></Language>
                <part>Amapola </part>
                <relation>tiene = distribución alterna </relation>
        </concept>
        <concept>Ramo
                <Language>Spanish<word>Ramo </word></Language>
                <part>Amapola </part>
                <relation>tiene un = extremo </relation>
        </concept>
</concept>
```

Figura 4.21 se presentan los hijos del concepto Amapola en la B con las propiedades de Tallo que tiene una leve coincidencia con Tallo en A

Al no haber coincidencia entre `Amapola` de A y `Amapola` de B, OM continúa analizando los conceptos que preceden a éste (hijos). Al analizar el concepto `Flor` en A COM [7] devuelve *Caso A* (§3.3.1) hubo coincidencia entre las definiciones de `Flor` (rectángulo) en la B y las definiciones del abuelo: `Amapola` (rectángulo con línea discontinua) en B con el papá: `Amapola` en A, véase la Figura **4.22** y Figura 4.23.

B

```
<concept>Amapola
        <Language>Spanish<word>Amapola </word></Language>
        <subset>Papavaceas </subset>
        <relation>longitud = 4 a 5 mts </relation>
        <concept>Ramo
                <Language>Spanish<word>Ramo </word></Language>
                <part>Amapola </part>
                <relation>tiene un = extremo </relation>
                <concept>Flor
                        <Language>Spanish<word>Flor </word></Language>
                        <part>Ramo </part>
                        <relation>nace del = Extremo </relation>
                        <relation>Partition = color {*:Blanco, Rojo} </relation>
```

Figura 4.22 se presentan el concepto Flor cuyo abuelo es Amapola en la B

El algoritmo OM procede a copiar las propiedades de `Flor`, al copiar las relaciones implícitas se "*percata que*" existe una relación redundante entre el papá de `Flor` en la A y el abuelo de éste en la B, es decir, <part>Amapola </part>, indicado en forma sinuosa, véase la Figura 4.25.

A

```
<concept>Amapola
        <Language>Spanish<word>Amapola</word></Language>
        <subset>Papaver </subset>
        <relation>especie = Papaver </relation>
        <relation>ciclo = Anual </relation>
        <relation>altura = mas de 5 dm </relation>
        <concept>Flor
                <Language>Spanish<word>Flor </word></Language>
                <part>Amapola </part>
                <relation>forma = casi esférica, Acampanada</relation>
                <relation>Color = escarlata </relation>
```

B

```
<concept>Amapola
        <Language>Spanish<word>Amapola </word></Language>
        <subset>Papavaceas </subset>
        <relation>longitud = 4 a 5 mts </relation>
        <concept>Ramo
                <Language>Spanish<word>Ramo </word></Language>
                <part>Amapola </part>
                <relation>tiene un = extremo </relation>
                <concept>Flor
                        <Language>S anish<word>Flor </word></Language>
                        <part>Ramo </part>
                        <relation>nace del = Extremo </relation>
                        <relation>Partition = color {*:Blanco, Rojo} </relation>
```

Figura 4.23 se presentará una relación redundante entre el concepto Flor cuyo abuelo y papá es el mismo en la B, el nombre de la relación ("part") también coinciden

OM decide copiar la ruta más específica que lleva a Flor, elige la ruta en la B, copiando el concepto Ramo (que no se encuentra en la A) y eliminando la relación implícita "part" en Flor (en la A). Al copiar el concepto Ramo en la ontología resultante busca el papá de este (Amapola) en la A entonces COM devuelve Amapola como el más similar, el caso que se ha aplicado en COM [7] es el *Caso C* §3.3.3 habiéndose comparado las propiedades de los hijos de Amapola en la C y los hijos del más similar a este (Amapola) en la B. Se ha integrado el concepto Ramo, ahora los hijos de Amapola en la A con propiedades que coinciden son: Tallo, Ramo y Flor. OM se "*percata*" que no ha copiado las propiedades de Amapola y los copia.

Una vez finalizada la copia de Ramo, continúa con la copia de Flor como hijo de éste. En la

Figura **4.24** (el rectángulo indica los conceptos hijos) se presenta la ontología C después de la fusión, donde cada uno de los hijos de Amapola en la C y en la B se ha acomodado, de acuerdo a su forma más específica, por ejemplo:

1. El concepto Semilla no solo es *parte de* Fruto sino es *parte de* la Tapa del Fruto.
2. El concepto Hoja no solo es *parte de* la Amapola sino es *parte del* Tallo de la Amapola y
3. El concepto Flor no solo es *parte de* la Amapola sino es *parte del* Ramo de la Amapola.

```
<concept>Amapola
        <Language>SPANISH<word>Amapola</word></Language>
        <relation>especie = Papaver</relation>
        <relation>ciclo = Anual</relation>
        <relation>altura = mas de 5 dm</relation>
        <relation>longitud = 4 a 5 mts</relation>
        <subset>Papaver</subset>
        <concept>Fruto
                <Language>SPANISH<word>Fruto</word></Language>
                <relation>Color = Verde pálido</relation>
                <relation>forma = Cónica</relation>
                <part>Amapola</part>
                <type>Cápsula</type>
                <concept>Tapa
                        <Language>SPANISH<word>Tapa, Opérculo</word></Language>
                        <relation>tiene = Grietas</relation>
                        <relation>ubicación = Superior</relation>
                        <part>Fruto</part>
                        <concept>Semilla
                                <Language>SPANISH<word>Semilla</word></Language>
                                <relation>escapan en = Grietas</relation>
                                <part>Tapa</part>
                        </concept>
                </concept>
        </concept>
        <concept>Tallo
                <Language>SPANISH<word>Tallo</word></Language>
                <relation>ramificación = poco ramificado</relation>
                <relation>forma = Erecta, fina, Lisa</relation>
                <relation>consta de = Nervadura central</relation>
                <relation>tiene = Veta</relation>
                <part>Amapola</part>
                <relation>Color{*:Gris, Verde; }</relation>
                <concept>Hoja
                        <Language>SPANISH<word>Hoja</word></Language>
                        <relation>forma = Pinada, Muy dentada</relation>
                        <relation>sin = Peciolo</relation>
                        <relation>ubicación = Alterna</relation>
                        <relation>tiene = distribución alterna</relation>
                        <part>Tallo</part>
                </concept>
        </concept>
        <concept>Ramo
                <Language>SPANISH<word>Ramo</word></Language>
                <relation>tiene un = Extremo</relation>
                <part>Amapola</part>
                <concept>Flor
                        <Language>SPANISH<word>Flor</word></Language>
                        <relation>forma = Casi esférica, Acampanada</relation>
                        <relation>Color = Escarlata</relation>
                        <relation>nace del = Extremo</relation>
                        <part>Ramo</part>
                        <relation>Color{*:Blanco, Rojo; }</relation>
                </concept>
        </concept>
</concept>
```

Figura 4.24 se presenta la ontología resultante con la copia de Flor, Ramo y las propiedades de Amapola

4.4.2 Identificación y solución de relaciones redundantes

La resolución de relaciones redundantes surge en A con el concepto `Semilla` y la relación implícita de nombre <part> Amapola </part> y la ontología B con el concepto `Semilla` y la relación implícita <part> Amapola </part> en los antecesores de `Semilla`, por tanto no se añade la relación en la ontología A sino se conserva la secuencia más específica en B. La representación de este ejemplo se muestra en la Figura 4.25 donde los círculos con línea discontinua indican los conceptos en A y B a copiar en C.

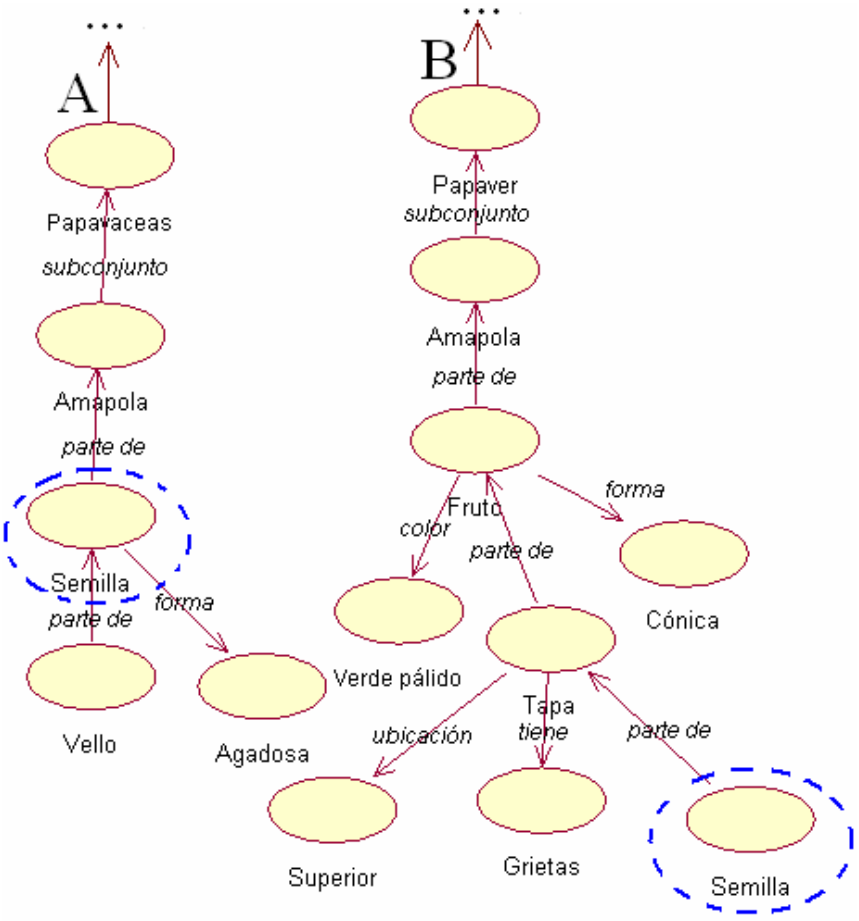


Figura 4.25 Representación de las ontologías fuente

Se presenta la ontología resultante en la cual los conceptos hijos de `Amapola` son ubicados en forma diferente a los de cada una de las ontologías fuente; de acuerdo a la semántica de cada uno de estos conceptos hijos. La representación de la ontología resultante se muestra en la
Figura **4.26**.

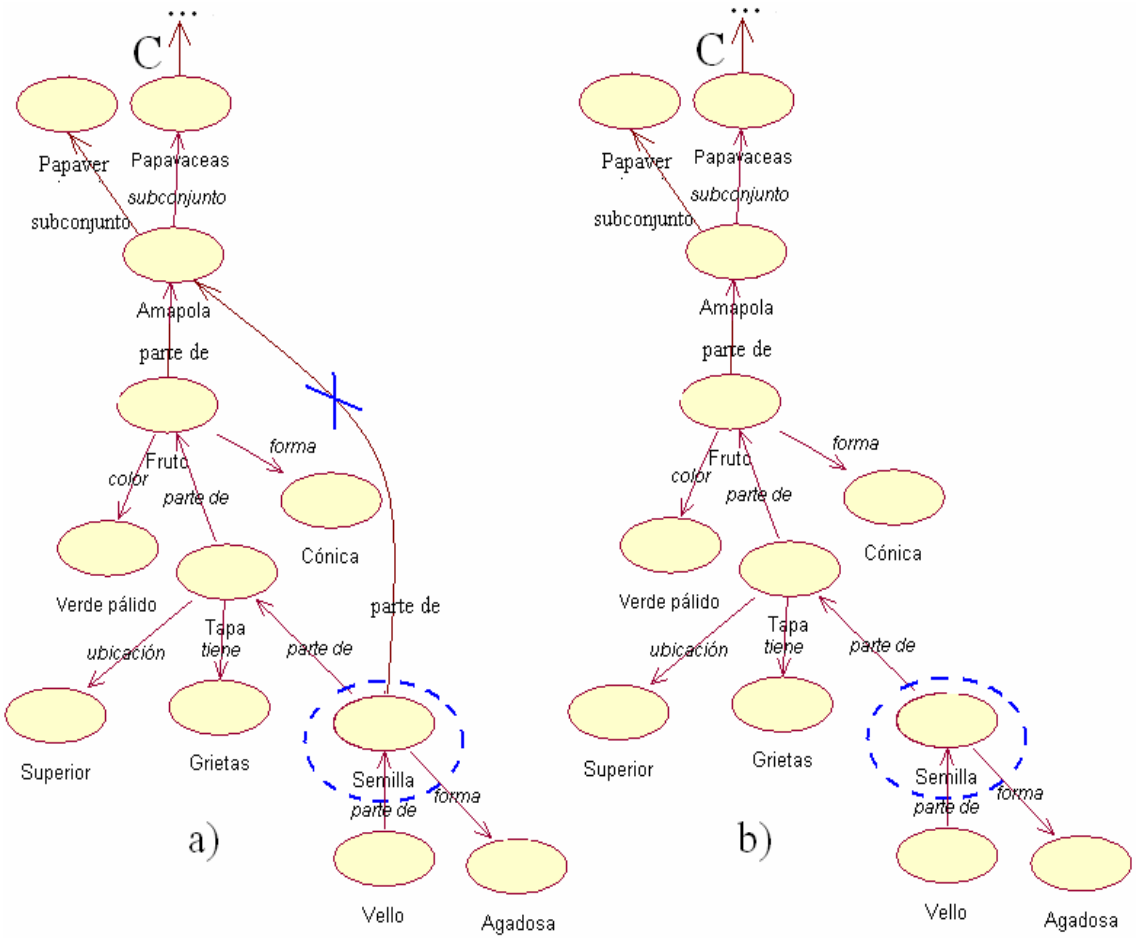


Figura 4.26 Representación de la ontología C donde en a) se presenta la relación redundante entre Semilla y Amapola y en la b) en la cual se ha solucionado la relación redundante

Uno de los casos en los que no provoca relaciones redundantes se cita a continuación:

En la Ontología A el concepto `Semilla` es *parte de* `Fruto`, esto es: <part>Fruto</part> y en la B un concepto más similar: `Semilla` como *parte de* `Girasol` esto es: <part>Girasol</part>. La representación de estas se muestra en la

Figura **4.27**.

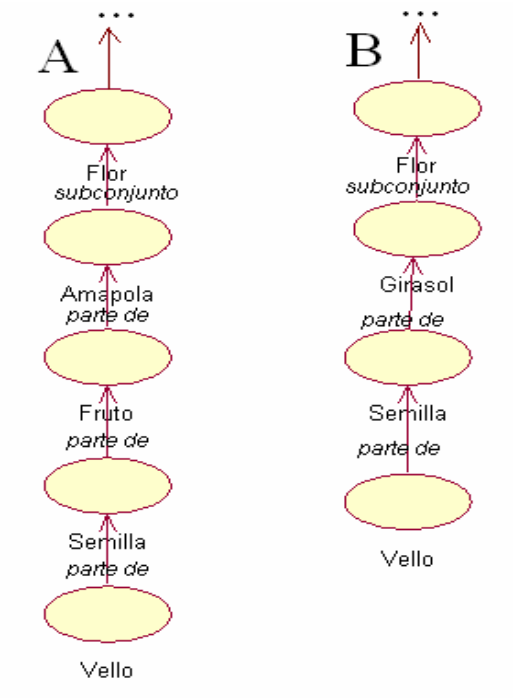


Figura 4.27 Representación de las ontologías fuente

El algoritmo OM considera que el concepto `Semilla` es parte de `Girasol` y también es parte de un `Fruto`, por lo que le asigna los dos antecesores: `Fruto` en A y `Girasol` en B, véase la ontología C resultante en la Figura 4.28.

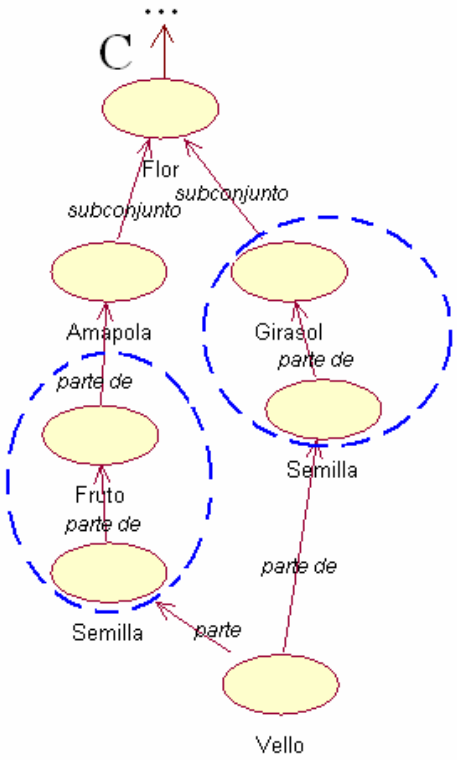


Figura 4.28 Representación de la C después de la fusión

4.4.3 Respuesta a conflicto en las relaciones de un concepto

Existe una lista de frases del que se apoya OM para saber que conceptos no se deben copiar a la C, esta lista se va formando durante la fusión.

Por ejemplo. Una relación del concepto `Hoja` es: sin = `Peciolo`, esto es, `Peciolo` no debe aparecer en la ontología resultante, de lo contrario, estaría contradiciendo la relación del concepto `Hoja`.

En la Figura 4.29 se presenta la ontología A con el concepto `Hoja` y la relación sin `Peciolo` indicado en línea gruesa discontinua, mientras que en la B hay un concepto `Peciolo` indicado en línea gruesa.

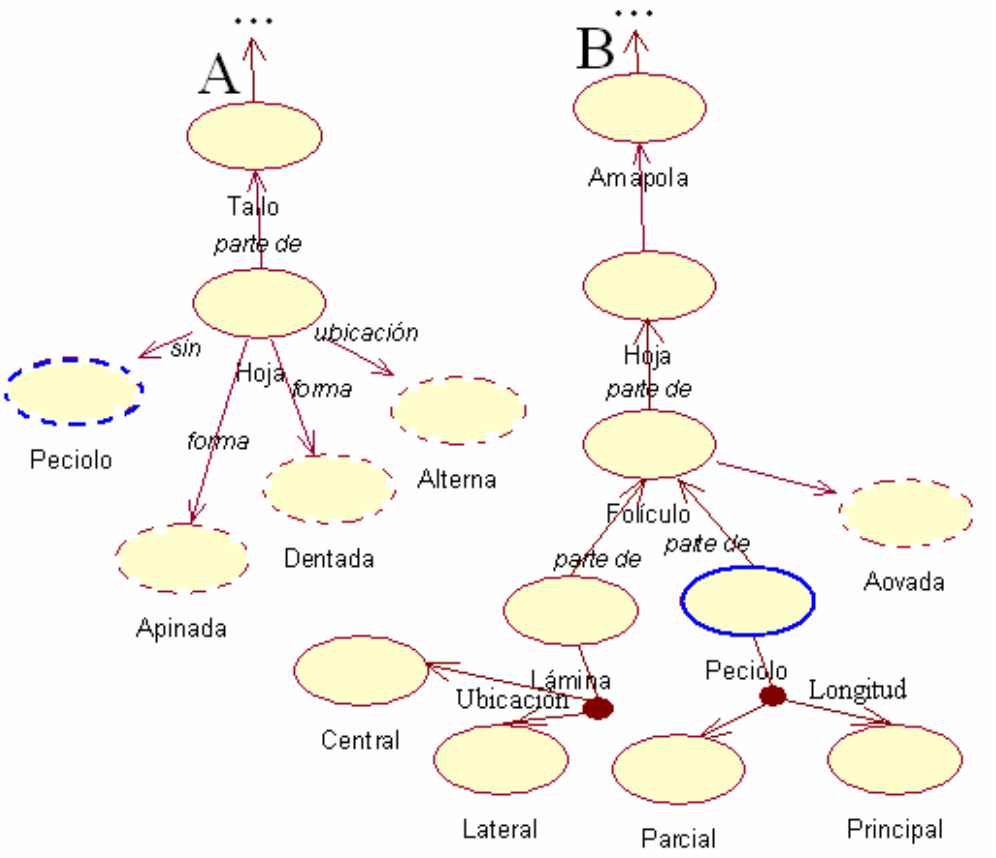


Figura 4.29 Representación de la ontología A con el concepto Hoja y la relación importante en este ejemplo y la B con la presencia del concepto que se niega en A

La lista contiene palabras como: "*excepto*", "*sin*", "*carece de*", "*con ausencia de*" por citar algunas. En la Figura 4.30 se presenta la ontología C después de la fusión.

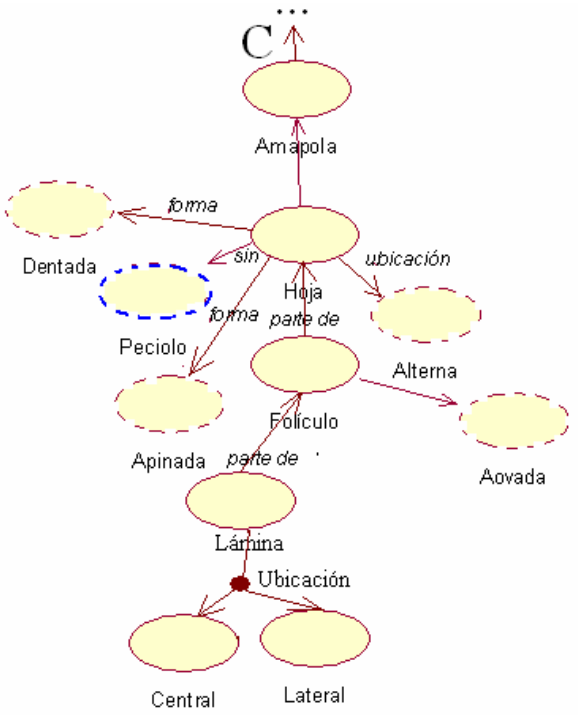


Figura 4.30 Representación de la Ontología resultante en la cual se ha negado la copia del concepto Peciolo

Otro ejemplo: Ahora se tiene el concepto `Peciolo` en A, véase la Figura **3.31** donde hay una relación en Hoja de la B que niega la presencia de `Peciolo`.

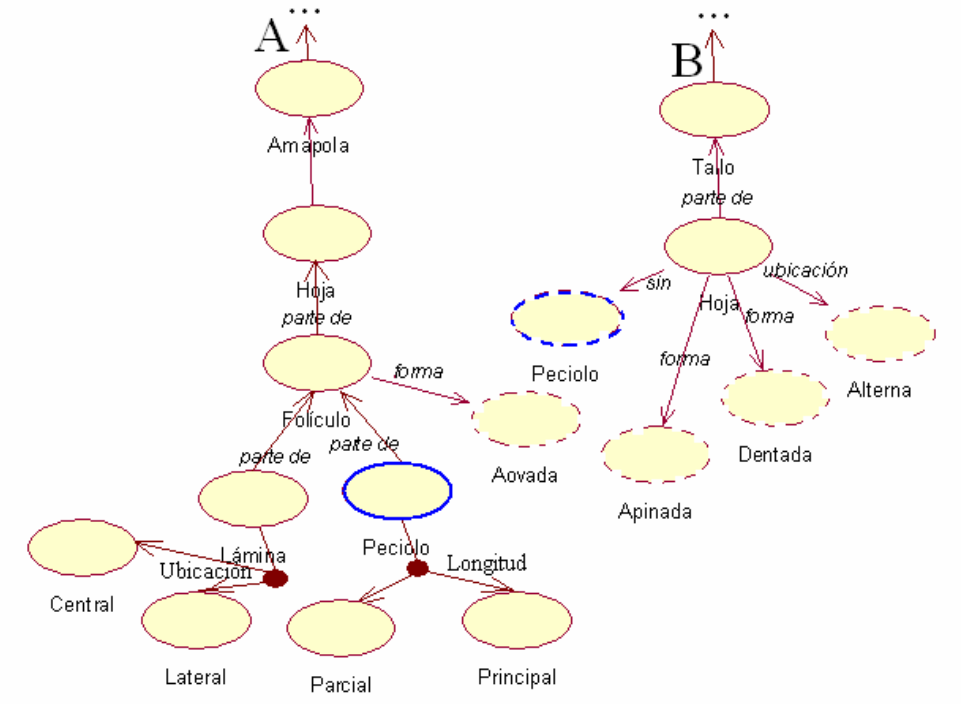


Figura 4.31 Donde se presenta un conflicto en la relación (sin peciolo Tallo) del concepto Tallo, ya

que la relación niega la presencia del concepto Peciolo cuando éste se encuentra definido en la A

En la figura 4.32, OM no copia la relación, el resultado de C después de la fusión se presenta en la Figura 4.32.

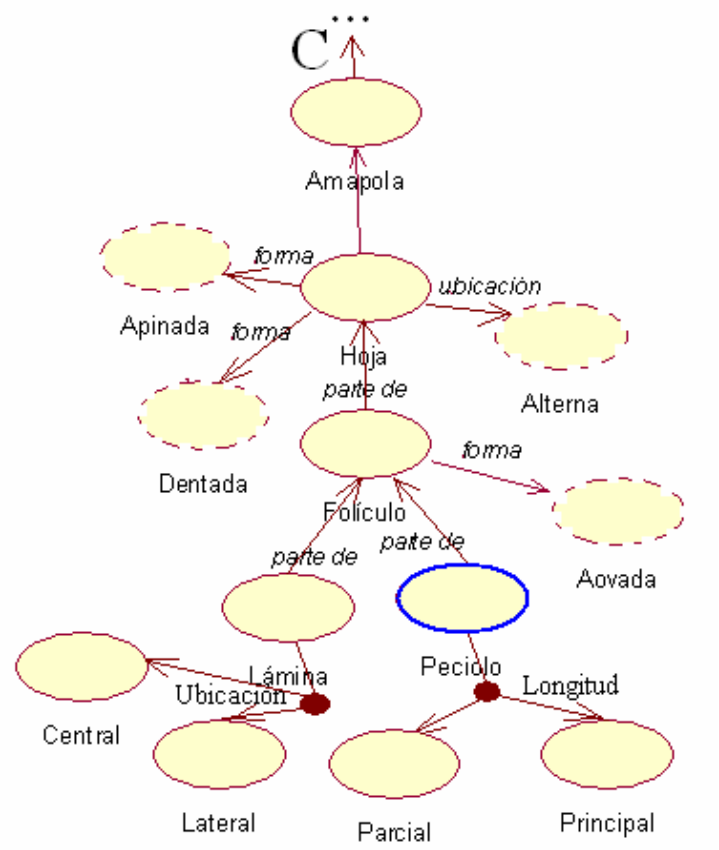


Figura 4.32 Representación de la Ontología C después de la fusión, donde no se ha agregado la relación sin = Peciolo

4.5 Unión de ontologías de descripción de herramientas

Se han tomado de Internet dos documentos[70] de descripción de Martillos, de ellos se han obtenido (a mano) dos ontologías (una por cada documento Web). La ontología A con 24 conceptos, la B con 33 conceptos y la unión A ∪ B con 51.

4.5.1 Varios conceptos de A se relacionan con un concepto en B

En la A, hay una descripción de `martillo`, con dos hijos: `martillo carpintero` y `martillo herrero`. En la B se presenta otra descripción de martillo de manera más general, véase la Figura 4.33. Al alinear B hacia A, OM detecta que COM [7] casa a `martillo` y sus dos descendientes en la A con el `martillo` en la B.

[70] Las referencias a los documentos se citan en: es.wikipedia.org /wiki/Martillo y www.lowes.com/lowes/lkn?action=noNavProcessor&p=spanishBuyGuide/ HammerBG.html&sec=esp.

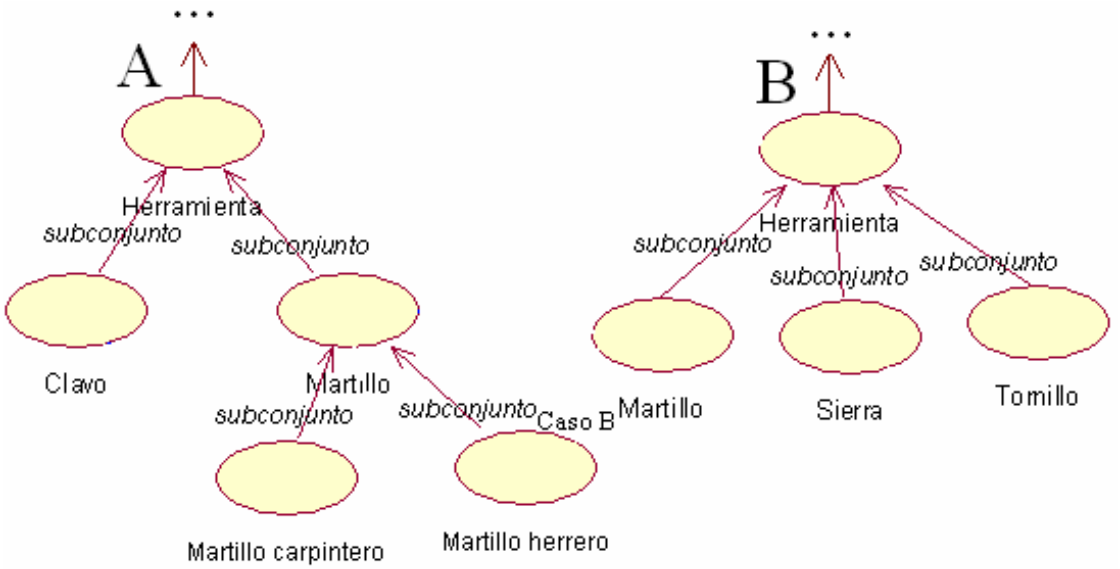


Figura 4.33 Representación de la Ontología A y B con el concepto martillo y sus elementos

Se presenta en la Figura 4.34 todos los conceptos de A que han casado con B (indicado por la flecha discontinua apuntando hacia la derecha) y los de B que han casado con A (indicado por la flecha discontinua apuntando hacia la izquierda), en la que se puede observar que `martillo` de A ha casado con `martillo` de B. Al añadir los hermanos de `martillo` de B en A, OM antes decide buscar cada uno de estos hermanos en A y surge lo siguiente:

1. Para el concepto `tornillo`, COM [7] devuelve *msg*: *Caso B hijos*, *vs*: 1.0 y *cms*: `martillo`, (porque han coincidido los papás) pero OM compara los nombres: `martillo` y `tornillo` al notar que son distintos lo considera como un hijo nuevo de `herramienta` en A y lo copia como un nodo nuevo.
2. Para el concepto `sierra`, sucede lo mismo que en 1.

Luego se siguen buscando más conceptos de A a B:

3. Para el concepto `martillo carpintero` en A COM [7] devuelve *msg*: *caso B: hijos*, *vs*: 1.0 y *cms*: `martillo` en B, OM identifica nuevamente que son conceptos distintos y como ya lo tiene no lo copia.
4. Para el concepto `martillo herrero` sucede lo mismo que en 3.
5. Para el concepto `clavo`, COM [7] devuelve *msg*: *Caso B: hijos*, *vs*: 1.0 y *cms*: `martillo` y como son conceptos distintos este concepto `clavo` se conserva igual que en la ontología A.

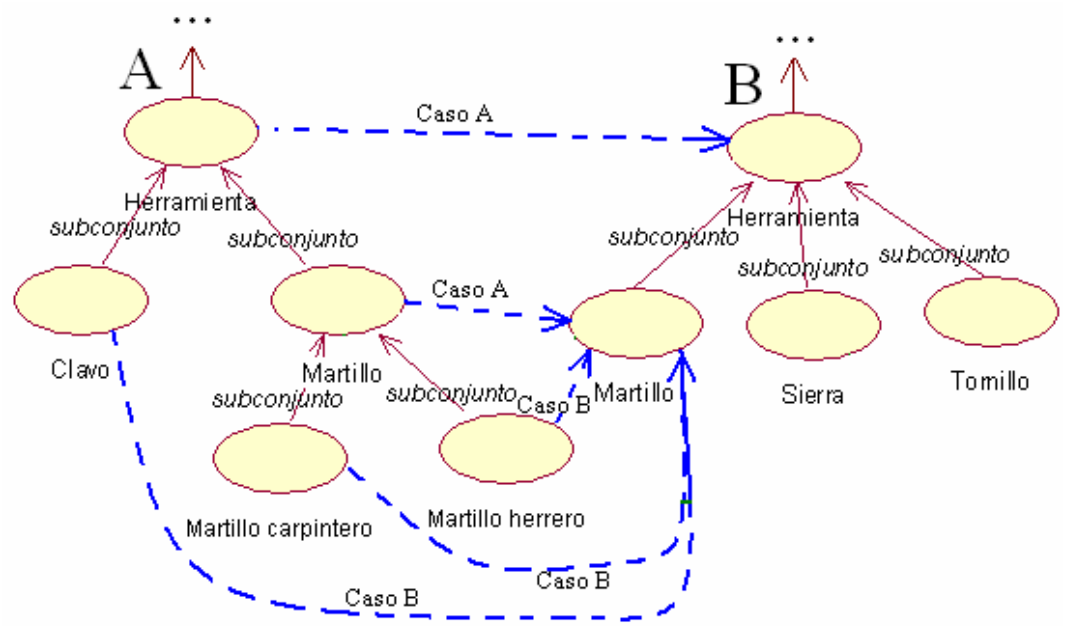


Figura 4.34 Mapeo entre las A y B en la cual se observa que los 4 conceptos de A casan con un concepto en la B

4.5.2 Varios conceptos de B se relacionan con un concepto en A

Ahora se cambian los mapeos, se realiza de B hacia A, véase la Figura **4.35** donde surge lo siguiente:

1. Para los conceptos `Tornillo` y `sierra` de la B que casan (indicado con la línea discontinua) con el concepto `martillo` en la A, OM verifica que son conceptos distintos por lo que decide conservar la misma posición de estos conceptos (`Tornillo` y `sierra`) en la A copiándolos a la C.
2. Para `martillo carpintero` y `martillo herrero` de la A, ambos casan (indicado con la línea discontinua) con `martillo` en la B, OM verifica que son distintos por lo que decide conservar la misma estructura de A pero en C.
3. Para el concepto `clavo` en la A que casa con `martillo` en la B, son conceptos diferentes, por lo tanto, se copia clavo en la C en la misma posición que en A.

Finalmente, esta fusión es simétrica, es decir, ambas fusiones (de B a A y A a B) conservan el mismo resultado.

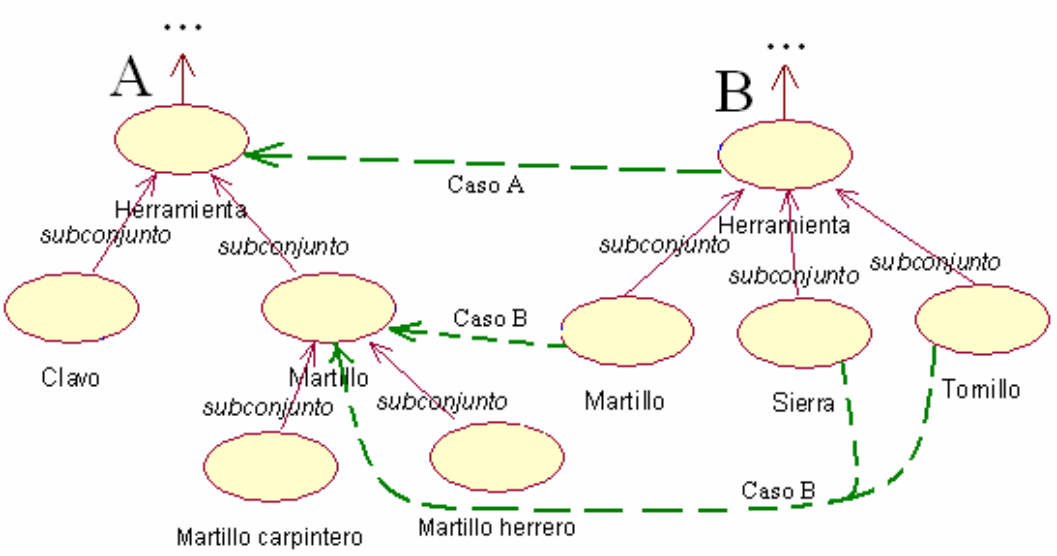


Figura 4.35 Mapeo entre las B y A donde los hijos de herramienta casan con el concepto martillo en A y los hijos de martillo en la B casan con martillo en la A

La ontología resultante se presenta en la Figura **4.36**.

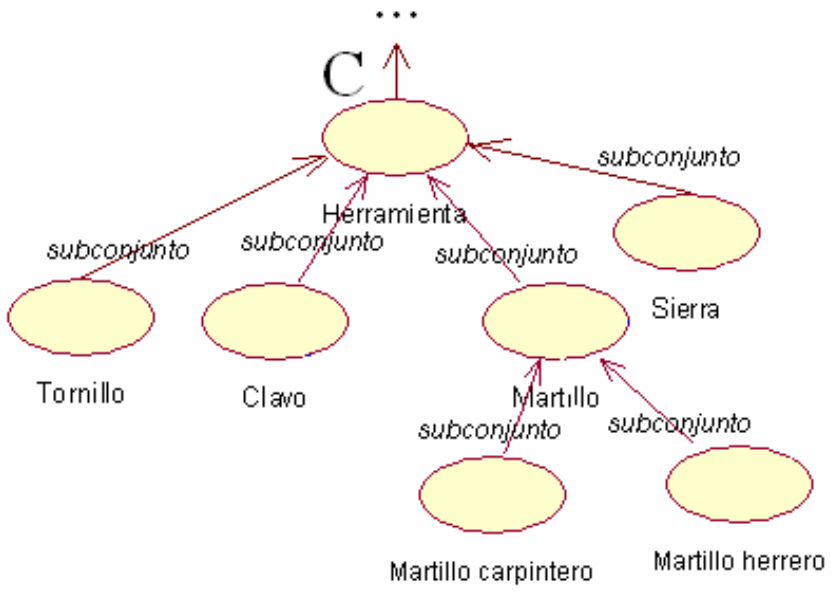


Figura 4.36 Representación de la ontología C simétrica

En este ejemplo se ha apreciado la eficiencia de OM que, pese a que COM [7] devuelve el casamiento de varios conceptos de A hacia B, OM compara sus definiciones y esto determina la diferencia entre los conceptos y decidir copiar las nuevas propiedades o conservar el concepto en la C.

4.6 Unión de ontologías dos ontologías que contienen resúmenes de novelas

Se han tomado de Internet dos documentos[71] con resúmenes de la obra: "*Cien años de soledad*" de Gabriel García Márquez[72], de ellos se han obtenido (manualmente) dos ontologías (una por cada documento Web). La ontología A con 90 conceptos, la B con 126 conceptos y la unión A ∪ B con 141.

4.6.1 Cohesión de las relaciones a la semántica del hecho

En las ontologías se han identificado hechos. Un **hecho** es un evento que tiene un tiempo de inicio y un tiempo de final, las relaciones o propiedades son los hechos en esta prueba, estas relaciones se ha adaptado para obtener coherencia con el mensaje del texto. Se han añadido posfijos, a continuación se describen estos:

El posfijo y: Realiza la conjunción de los dos valores de la relación.

Por ejemplo:

<relation> se casaron y = José Arcadio Buendía, Ursula Iguarán </relation>
Interpretación: *se casaron José Arcadio Buendía y Ursula Iguarán*

El posfijo a: Indica que el primer elemento de la relación ejecuta una acción (la acción es el verbo) al segundo elemento de la relación.

Por ejemplo:

<relation> abandonó a = José Arcadio (hijo), Pilar Ternera </relation>
Interpretación: *José Arcadio (hijo) abandonó a Pilar Ternera*.

O también:

<relation> llegó a = Pietro Crespi, Macondo </relation>
Interpretación: *Pietro Crespi llegó a Macondo*.

El posfijo con: Indica que la acción que se indica en el verbo se ha de realizarse entre el primer y el segundo valor de la relación.

Por ejemplo:

<relation> huyó con = José Arcadio (hijo), una gitana </relation>

---

[71] Los documentos se pueden consultar en: html.rincondelvago.com/cien-anos-de-soledad_gabriel-gar cia-marquez22.html y monografías.com/trabajos10/ciso/ciso.shtml.

[72] Escritor colombiano ha quién fue otorgado el *Premio Novel de Literatura* en 1982, ver: es.wikipedia.org/wiki/Gabriel_García_Márquez.

Interpretación: *José Arcadio (hijo) huyó con una gitana*.

El posfijo de: Indica que la acción en el verbo que realiza el primer valor de la relación, le afecta al segundo.

Por ejemplo:

<relation> se enamoró de = Amaranta, Pietro Crespi </relation>
Interpretación. *Amaranta se enamoró de Pietro Crespi*.

El posfijo por: La acción en el verbo es aplicada al concepto que contiene la relación, el valor de la relación contiene la descripción de la acción. Por ejemplo:

<concept> Macondo
<relation>estaba rodeado por = agua </relation>
Interpretación. *Macondo estaba rodeado por agua*. En este caso *estaba rodeado* indica la acción del verbo.

El posfijo a favor de: Indica que la acción del verbo lo aplica el primer valor de la relación, donde el segundo es favorecido. Por ejemplo:

<relation> buscó la revolución a favor de = Aureliano, los liberales</relation>
Interpretación. *Aureliano buscó la revolución a favor de los liberales*.

El posfijo en contra de: Indica que la acción del verbo lo aplica el primer valor de la relación, donde el segundo es afectado. Por ejemplo:

<relation> comentó en contra de = Prudencio Aguilar, la familia Buendía</relation>
Interpretación. *Prudencio Aguilar comentó en contra de la familia Buendía*.

4.6.2 Complementación de la información

Dada la ontología A con los elementos mostrados en la
Figura **4.37**, en la cual hay una relación fusilado que señala a un personaje `José Arcadio (nieto)` luego en B hay una relación fusilado donde se encuentra definido un nuevo personaje `Los Conservadores`; OM identifica las relaciones como iguales complementándose ésta (la de A) con el nuevo elemento en B.

A

```
<relation>ignoró= José Arcadio(nieto), era su madre, Pilar Ternera </relation>
<relation>envió a con = Pilar Ternera, José Arcadio(nieto), Santa Sofia de la Pi
<relation>nació =Remedios  </relation>
<relation>murió = José Arcadio(nieto), fusilamiento</relation>
<relation>murió = Pietro Crespi, suicidio</relation>
```

B

```
<relation>ignoró = José Arcadio,era su madre, Pilar Ternera </relation>
<relation>nació = Remedios </relation>
<relation>murió =  José Arcadio(nieto), fusilamiento, Los Conservadores</relatio
<relation>llevaron como = Aureliano, Garineldo Márquez, prisioneros </relation>
<relation>sentenciado a= Aureliano, muerte </relation>
<relation>salvó a = José Arcadio (hijo), Aureliano </relation>
```

C

```
<relation>ignoró= José Arcadio(nieto), era su madre, Pilar Ternera </relation>
<relation>envió a con = Pilar Ternera, José Arcadio(nieto), Santa Sofia de la Pi
<relation>nació =Remedios  </relation>
<relation>murió = José Arcadio(nieto), fusilamiento, Los Conservadores</relation>
<relation>murió = Pietro Crespi, suicidio</relation>
<relation>llevaron como = Aureliano, Garineldo Márquez, prisioneros </relation>
<relation>sentenciado a= Aureliano, muerte </relation>
<relation>salvó a = José Arcadio (hijo), Aureliano </relation>
```

Figura 4.37 Representación de las ontologías con datos añadidos

Con este ejemplo se puede apreciar que la fusión de ontologías tiene un papel importante en la búsqueda de nuevos elementos de la información a partir de la adición de nuevos conceptos a las relaciones.

4.6.3 Identificación compleja de los elementos de una relación, donde interviene una secuencia temporal

La estructura que se define en OM para las relaciones de un concepto no determina un tiempo determinado (hoy, mañana, ayer) por lo tanto la sucesión que sigue la ejecución de estas relaciones es secuencial. Por ejemplo, en la Figura 4.38 se presenta una sucesión de hechos.

```
<relation>fue condenado a muerte = Aureliano</relation>
<relation>liberó a = José Arcadio (hijo), Aureliano</relation>
<relation>reunió a = Aureliano, un ejército</relation>
<relation>proclamó la = Aureliano,  guerra del régimen</relation>
```

Figura 4.38 Representación de la secuencia de hechos (relaciones, propiedades) en una ontología

Al realizar la fusión, OM no hace distinción entre los hechos, solo interesa el nombre de la relación, sinónimos, etc. y todo va en función, al reconocimiento de los elementos de la relación y no al tiempo en que se desarrolla éste. Durante la fusión se pueden intercalar nuevos hechos haciendo difícil la interpretación de la secuencia inicial de estos.

4.7 Resultados de los casos reales

Las ontologías fueron obtenidas manualmente de varios documentos [§4.2, §4.3, §4.4, §4.5 y §4.6]. Cada par de ontologías a unir describen el mismo tema, por ejemplo, las dos ontologías de tortugas son el resultado de dos documentos hallados en distintos sitios de Internet y cada uno describe tortugas y así, con los otros casos reales. Las ontologías obtenidas fueron unidas (automáticamente) por OM y la validación de la ontología final se hizo de forma manual, obteniéndose buenos resultados (véase la **Tabla 4.1**).

Tabla 4.1 Funcionamiento de OM en algunos ejemplos reales

Ontologías	Relaciones	% error	Conceptos	% error
Tortuga §4.3 (4 segundos de ejecución)	Las 6 relaciones de B se añadieron y fusionaron correctamente a las 8 de A, dando un total de 10 relaciones en C	0	Los 35 conceptos de B se añadieron y fusionaron correctamente a los 29 de A, dando como resultado 35 conceptos en C	0
Martillo §4.5 (6 segundos)	Las 30 relaciones de B se añadieron y fusionaron correctamente a las 8 de A, dando un total de 36 relaciones en C	0	Los 33 conceptos de B se añadieron y fusionaron correctamente a los 24 de A, dando como resultado 51 conceptos en C	0
Amapola §4.4 (14 segundos)	Las 20 relaciones de B fueron añadidas y fusionadas correctamente a las 21 de A, dando un total de 37 relaciones en C	0	Los 35 conceptos de B se añadieron y fusionaron correctamente a los 34 de A, dando como resultado 58 conceptos en C	0
100 Años de Soledad §4.6 (10 minutos)	Las 283 relaciones de B se añadieron y fusionaron con las 231 de A dando un total de 420, el método manual dio 432 (12 de 432 no fueron copiados)	0.027	Las 126 relaciones de B se añadieron y fusionaron con las 90 de A dando un total de 141, el método manual dio 149 (8 de 149 no fueron copiados)	0.053
Oaxaca §4.2 (5 minutos)	Las 43 relaciones de B fueron añadidas y fusionadas a las 61 de A dando un total de 96 relaciones en C	0	Las 117 relaciones de B se añadieron y fusionaron con las 234 de A dando un total de 309, el método manual dio 310 (faltó 1 de 310 conceptos)	0.003

5. Conclusiones

Este trabajo resuelve el problema de fusionar dos ontologías A y B, construyendo una nueva ontología C que contiene la suma de la información contenida en A y en B sin repeticiones ni contradicciones.

Entre los desarrollos importantes del Algoritmo OM se encuentran algunas aportaciones originales (puntos 3, 4 y 5):

1. Notación OM (§3.4.1), que adiciona semántica a la representación de ontologías.
2. Mejoras al Algoritmo COM (§3.4) para darle más funcionalidad a la comparación de ontologías previa a la fusión.
3. Consideración de la sinonimia (§3.4.6) en el proceso, permitiendo una fusión más acertada de las ontologías.
4. Uso de particiones (§3.4.2) en la representación de ontologías, además del uso acostumbrado de subconjuntos, lo que permite mayor precisión en la clasificación de los conceptos.
5. Aplicación de la teoría de la confusión §3.8.1 como una opción para resolver los problemas de inconsistencias que surgen en la fusión y evitar que el proceso se detenga.

La conclusión principal es que **es posible fusionar dos ontologías de forma automática (sin intervención humana), robusta, sin redundancia, contradicciones y preservando el conocimiento de cada una de las ontologías fuente.**

OM toma en cuenta la semántica de los conceptos de cada ontología, proporcionada en sus descripciones (los nombres, etiquetas o frases temáticas que ellos tienen, denotados por <word> … </word>, véase §3.4.1) y en su contexto o vecindad ontológica (las relaciones en las que cada concepto toma parte).

OM automatiza los trabajos de fusión [11, 13, 28 y 40] que la llevan a cabo con la intervención de un usuario.

Es posible definir un enfoque matemático riguroso (Apéndice A), pero no es apto para representar ni manipular ontologías extensas que describen partes complejas de la realidad, donde abundan las inconsistencias, contradicciones, redundancias, información parcial o incompleta y distintos niveles de detalle. **En cambio, con OM es posible hacer esto.**

La información en documentos de texto resulta adecuada para el procesamiento humano, en tanto que la información en ontologías y en particular

en la notación OM resulta adecuada para el procesamiento en computadora e inteligencia artificial.

5.1 Trabajos futuros

OM es parte del proyecto de un grupo de investigación del CIC-IPN en México, que usa la Inteligencia Artificial para representar y manipular el conocimiento, OM se sustenta en algunos trabajos de este equipo, tales como:

- El algoritmo COM [22], halla los elementos similares de dos ontologías.
- La teoría de la confusión [21], ayuda a resolver algunos problemas de inconsistencia.

5.1.1 Extensiones a OM (OM mejorado)

OM usa ontologías para fusionarlas. Pero no hay muchas ontologías disponibles en la Web, lo que abundan son documentos. Por consiguiente, será interesante hacer un proceso que convierta documentos a ontologías mediante su análisis, a fin de conservar en ésta la información que el documento contiene con la mayor fidelidad posible.

Se agregaría a OM este analizador[69] y un contestador de preguntas [5]. Ambos están en desarrollo. El contestador que va inferir relaciones entre bases de datos autónomas también se puede adaptar a las relaciones de una ontología, o quizá el contestador de preguntas podría ser un subproducto del analizador[69]. Véase Figura 5.1.

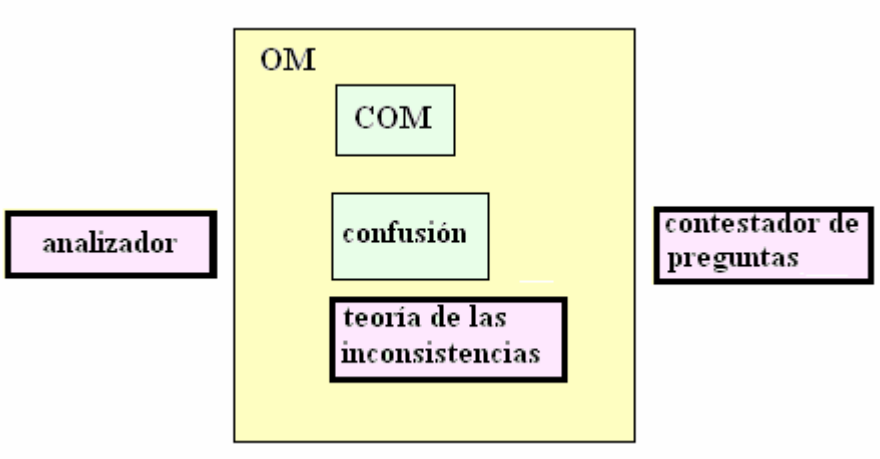


Figura 5.1 Las extensiones a OM indicadas con línea gruesa permitirán conjuntar y recopilar de manera automática, en una ontología consistente y no redundante, el conocimiento que ahora yace disperso en documentos en español que abundan en la red (o están concentrados en Wikipedia).

5.1.2 Analizador de documentos a ontologías

Tesis que está iniciándose[69], este analizador (“parser”) convertirá un documento en español a una ontología que capture la información que tal

documento contiene.

Importancia

Hace accesible el procesamiento por computadora de la información o conocimiento que hasta ahora era solo procesable por personas, por estar guardado en documentos escritos en español u otro lenguaje natural. Se combinará maravillosamente con OM.

5.1.3 Teoría de la inconsistencia

Tesis que está en proceso de desarrollo [24], permite medir o cuantificar el grado de falsedad o contradicción entre dos hechos inconsistentes. Parecido a la lógica difusa. Encuentra el "centro de gravedad" o "promedio" (el valor que minimiza la inconsistencia) de una serie de hechos o afirmaciones contradictorias.

Permite hallar varios centros de gravedad para un conjunto de hechos, cuando así se requiere.

Ajusta un autómata finito a un proceso o evento que transcurre en el tiempo y se describe mediante hechos (observaciones o afirmaciones) cuyas variables no son numéricas, sino simbólicas.[73] Es decir, modela tal proceso mediante el mejor autómata disponible, diciendo cuál autómata (de una lista predeterminada de ellos) es el que mejor describe el proceso.

Importancia

Formaliza la concordancia o acuerdo a que se llega cuando se tienen datos simbólicos inconsistentes, cuando se tienen opiniones o puntos de vista distintos. Permite medir cuál es el grado de "descontento" de cada persona que acepta o adopta un punto de vista "intermedio" o consensuado, qué tanto difiere de su punto de vista particular. Mide el "descontento global" en un grupo de personas que han adoptado un punto de vista consensuado, en vez del punto de vista particular de cada quien.

[73] Una persona obtiene información de un objeto (Juan Pérez) o aspecto de la realidad, mediante mediciones {(altura, Juan Pérez, 1.77m), (peso, Juan Pérez, 67 Kg)} de sus propiedades numéricas y observaciones (también llamadas hechos o afirmaciones) de sus propiedades simbólicas {(religión, Juan Pérez, católica), (deporte, Juan Pérez, beisbolista), (vive en, Juan Pérez, Zacatenco)}. La persona agrega estas observaciones como nodos y relaciones a su ontología (dentro de su cerebro), formando así el *conocimiento* que ella tiene sobre Juan Pérez. Cuando mide o percibe información numérica inconsistente {(altura, Juan Pérez, 1.74m), (altura, Juan Pérez, 1.78m)}, la persona puede hallar el *promedio* (altura, Juan Pérez, 1.76m) y la *varianza.* El promedio viene siendo "la información más cercana a la realidad, dadas las mediciones" y la varianza indica qué tan inconsistentes son estas mediciones. Cuando percibe o recibe información simbólica inconsistente {(vive en, Juan Pérez, Zacatenco), (vive en, Juan Pérez, La Villa), (vive en, Juan Pérez, México)}, la Teoría de la Inconsistencia puede darnos el valor simbólico más viable (¿dónde es más factible que Juan Pérez viva?). Ese consenso (de valores simbólicos) es similar al promedio de valores numéricos, y la inconsistencia (sobre valores simbólicos) es similar a la varianza de valores numéricos. ¡De hecho, la Teoría de la Inconsistencia nos permite promediar manzanas con peras y naranjas!

5.1.4 Contestador de preguntas

Tesis en proceso de desarrollo [5]. Dada una "pregunta compleja" (que requiere más de una base de datos para su solución o respuesta) y un conjunto de bases de datos que contienen la respuesta a esa pregunta, este trabajo hace una "fusión local" (uniendo solo ciertos campos de ciertas tablas) que permite resolver o responder tal pregunta y *halla la respuesta*.

Importancia

Permite explotar (responder preguntas) a un conjunto de bases de datos heterogéneas, que no fueron diseñadas en común (fueron construidos por personas distintas, no tomando en cuenta la existencia una de otras). Permite obtener información derivada de la consideración simultánea de varias bases de datos públicas, disponibles en la Web.

5.1.5 Conocimiento ligado al tiempo y al espacio

OM funciona cuando los conceptos que se fusionan no están definidos en un tiempo, es decir la `Amapola` tiene color `rojo` o `blanco`, pero si se quiere representar el color de la `Amapola` en este momento, el color de la `Amapola` cuando está marchita, el color de la `Amapola` cuando está floreciendo. Estas representaciones se basan en hechos, eventos o instantes de tiempo. El lenguaje OM no representa eventos.

El problema de manejar eventos (hechos) se puede resolver como sigue:

1. Un programa coloca sobre la línea del tiempo los eventos de la ontología A y la ontología B. Aquí surgen ciertas restricciones:
    a. Algunos eventos no dicen cuándo sucedieron, por ejemplo: "*compró un globo verde*",
    b. Otros eventos se indican aproximadamente, por ejemplo: "*compró un globo verde después de ir al cine*" o en forma relativa.
    c. Otros son "siempre", por ejemplo: "*Juan y Pedro son primos hermanos*", "*el cielo es azul*" (estos pueden representarse en la notación OM)

Esto origina que un evento tenga dos fechas (de inicio y fin) que sean conocidas (constantes) o no sean conocidas completamente (variables), por ejemplo: "*compró un globo verde en* ***t***", otro ejemplo: "*fue al cine en el momento* ***u***" donde ***t*** y ***u*** son desconocidos porque pueden indicar tiempo o espacio. Además pueden haber restricciones sobre esas variables (***t*** > 1812, ***t*** > ***u***).

2. Una vez colocados los eventos de A y los de B sobre la línea del tiempo, se puede tratar de ver qué eventos de A son compatibles (en el tiempo) con los de B y pedirle al sistema OM que trate de unirlos; es decir, OM no conoce del tiempo, por lo que el colocador de eventos en la línea del tiempo le indicará los conceptos que puede tratar de unir. Por ejemplo, si A tiene un hecho "*compró*

un globo verde antes de 1815", y B tiene otro: "*compró un globo verde después de 1817*", se refiere a dos eventos distintos como se explica en [8], por lo tanto, no deben unirse.

Considerados los pasos 1 y 2, OM podrá unir ontologías donde exista o transcurra el tiempo, siempre y cuando exista o se diseñe:

1. Un programa que coloque a los eventos de cada ontología en la línea del tiempo (o sea, que tome en cuenta su cronología). Actualmente, exiten trabajos como en [8] que toma en cuenta estos eventos en la línea del tiempo.
2. Un programa que vea cuáles eventos son compatibles (en el tiempo), y entonces OM tratará de hallar aquéllos que además son "compatibles en base a sus propiedades" (y unirlos).

Con estas adaptaciones OM podría complementar a Wikipedia para los que desean respuestas a preguntas. Mientras Wikipedia responde con documentos que requieren procesamiento manual, el OM mejorado *daría él mismo las respuestas.* Con este propósito, una de las primeras cosas que OM mejorado haría es procesar los documentos de Wikipedia para convertirlos en una gigantesca ontología e ir formando una de conocimiento común.

Índice del Anexo

Índice de tablas del Anexo

Anexo A

Se proporciona el modelado matemático de la unión de ontologías, aunque el camino que se ha seguido para la realización del motivo principal de la tesis no ha coincidido con este modelado, ya que se ha concluido que regularmente no existe isomorfismo entre las ontologías en la Web.

También se citan las definiciones básicas necesarias usadas en la unión de ontologías, así como el planteamiento del problema a resolver, la propuesta de solución a este planteamiento, un algoritmo que describe de forma general el proceso de fusión (el algoritmo más especifico se encuentra en §3.2.2) y una comparación del planteamiento formal a la realidad expresada en una tabla con hechos del entorno real y por otro lado el comportamiento del formalismo.

A.1 Introducción

Actualmente no existe una teoría satisfactoria que caracterice formalmente al mapeo y la fusión de ontologías, no obstante ello han habido algunas aproximaciones [25].

En este apartado se citan los conceptos que usados en la unión de ontologías, se plantea el problema y la posible solución, finalmente se realiza una comparación de la teoría con lo que sucede en la realidad.

A.2 Definición

La mayor parte de las definiciones que a continuación se presentan se obtuvieron y adaptaron de [25].

Una *comunidad* es un conjunto finito, no vacío de elementos, los cuales pueden ser instancias y tipos, las instancias y los tipos representan una comunidad, una instancia está clasificada en tipo(s), esta clasificación puede variar entre comunidades.

Dos comunidades se comunican a través de *mensajes* o *instancias*. Por ejemplo: una instancia *x* es clasificada del tipo *a*.

Una instancia normal es aquella que satisface el conjunto de restricciones de su tipo. Una *instancia* que no es *normal*, se llama *instancia anormal*. Por ejemplo: Si *x* es una *instancia normal* clasificada por su *tipo a*, entonces *x* es restringida por las propiedades del *tipo a*.

Los *tipos* definen las cosas de un universo. Por ejemplo, la determinación de una *instancia* de acuerdo a una clasificación.

Las comunidades disjuntas interactúan regularmente usando diferentes *tipos* e *instancias*, las cuales serán clasificadas en base a dichos *tipos*. Cada comunidad tendrá sus propias *restricciones* que describen el comportamiento de sus instancias con respecto a su sistema de *tipos*. La siguiente definición [25] agrupa éstas ideas.

A.2.1 Lógica local

Definición 1

Una *lógica local es una 4-tupla* $L = (I, T, \models, \vdash)$, *donde:*

i) I es un conjunto no vacío de elementos llamados *instancias*.
ii) T es un conjunto no vacío de elementos llamados *tipos*.
iii) $\models$ es una *relación binaria de clasificación* de elementos de I con respecto a los elementos de T.
iv) $\vdash$ es una *relación binaria de consecuencia* entre subconjuntos *de T.*
En la Figura 1 se encuentra la representación de esta definición.

Observaciones:

Existen dos partes de la lógica local que son de particular importancia:

a) La 3-tupla (I, T, $\models$), se llama relación de *clasificación de una lógica local*. Así, $x \models a$, significa que x está clasificada del tipo a donde: $x \in I$ y $a \in T$. En lo sucesivo, el símbolo $\in$ denota la pertenencia de un miembro a un conjunto, por ejemplo $x \in I$.
b) La 2-tupla (T, $\vdash$), se llama *relación binaria de consecuencia de una lógica local*. Esta relación es especificada por el conjunto de consecuencias (Γ, Δ). Donde: $\Gamma, \Delta \subseteq T$. El conjunto de *tipos* Γ es interpretado conjuntivamente y el conjunto Δ es interpretado disjuntivamente; así, si $x \in I$ satisface a (Γ, Δ) mostrando que si x es de cada tipo en Γ entonces x es de algún tipo en Δ. Las consecuencias que pertenecen a esta relación se llaman *restricciones* y es denotado por $\Gamma \vdash \Delta$. Ejemplo, sea $\Gamma \subseteq T$ y $\Delta \subseteq T$, $x \in I$, $a \in \Gamma$ y $b \in \Delta$, si x satisface a cualquiera de las propiedades del *tipo a* entonces x satisface alguna propiedad del *tipo b*.

A.2.2 Infomorfismo

Definición 2. Sea $L=(I, T, \models, \vdash)$ *y* $L'=(I', T', \models', \vdash')$ lógicas locales. Un infomorfismo lógico f: $L \leftrightarrows L'$ es un par de funciones $f=(f^{\wedge}, f_{\vee})$ donde $f^{\wedge}: T \to T'$ *y* $f_{\vee}: I' \to I$, *tales que:*

1. *para* $x \in I'$, $a \in T$, $f_{\vee}(x) \models a$ *si y solo si* $x \models' f^{\wedge}(a)$

2. *para* $\Gamma \subseteq T$, $\Delta \subseteq T$, si $\Gamma \vdash \Delta$, *entonces* $f^{\wedge}[\Gamma] \vdash' f^{\wedge}[\Delta]$ véase la Figura A.1.

Donde $f^{\wedge}[\Delta]$ y $f^{\wedge}[\Gamma]$ denotan el conjunto imagen de Δ *y* Γ *de la función* $f^{\wedge}$ respectivamente.

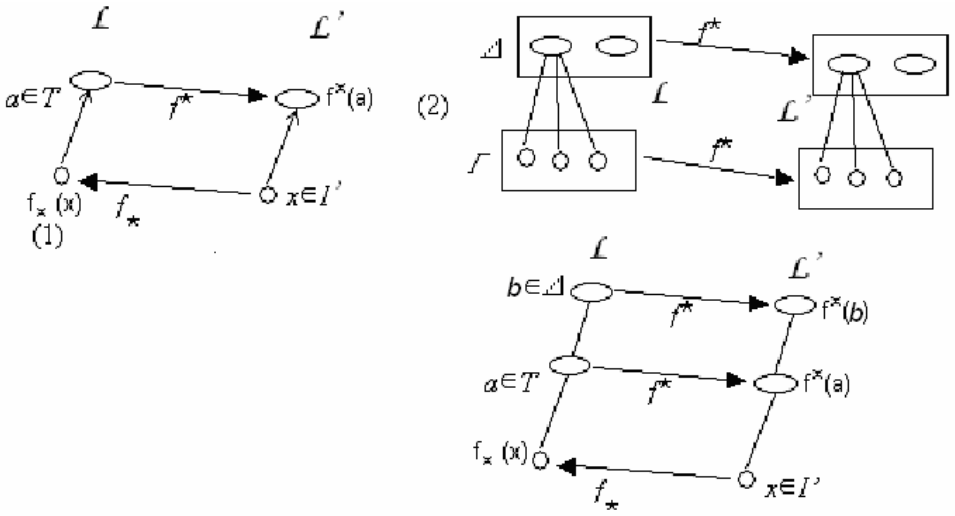


Figura A.1 Representación de las propiedades (1) y (2) de la Definición 2

A.2.3 Ontología

Definición 3. *Una* ontología *es una 6-tupla* $O = (T, R, \leq, \perp, |, \sigma)$ *donde*
1. T es un conjunto finito no vacío de símbolos llamados *Concepto* (C); donde: $C \subseteq T$
2. R es un conjunto finito no vacío de símbolos llamados *Relación (Enlace);*
3. $\leq$ es una relación sobre T tal que es *reflexiva, antisimétrica y transitiva.* Esto es, $\leq$ define un *orden parcial;*
4. $\perp$ es una relación sobre T tal que es *simétrica e irreflexiva*. $\perp$ representa a dos conceptos ***disjuntos***, es decir, cuando no hay instancias $x \in I$ que puedan considerarse en dos conceptos $a, b \in T$.
5. | es una relación sobre T tal que es *simétrica.* El símbolo | representa a la ***cubierta*** de dos conceptos, es decir, cuando todas las instancias $x \in I$ son cubiertas por dos conceptos $a, b \in T$.
6. $\sigma: R \rightarrow T^+$ es una función que asigna a cada símbolo de relación su *aridad;* el operador $^+$ envía un conjunto T al conjunto de tuplas finitas cuyos elementos están en T. (véase Figura **A.2**).

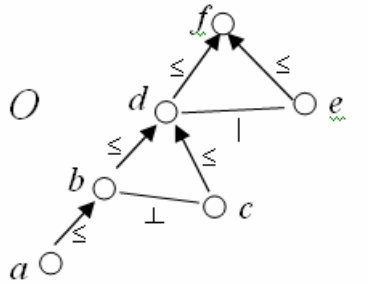


Figura A.2 Una porción de la ontología O muestra los conceptos a, b, c, d, e, f y las relaciones $a \leq b \leq d \leq f$; $c \leq d$; $e \leq f$; es decir, son del tipo a o pertenecen al concepto a lo cual implica pertenecer al concepto b, d y f. Por otro lado, $b \perp c$ indica que una instancia no puede pertenecer simultáneamente a b y c. En cambio, $d \mid e$ significa que cualquier instancia pertenece a d o a e. La aridad de f es 2, mientras que la aridad de b es 1. La ontología O no está poblada, ie. no tiene instancias

A.2.4 Ontología núcleo

Ontología núcleo [25]. Es una ontología que contiene relaciones binarias excepto $\perp$ y |.

A.2.5 Relación binaria de objetos

Definición 4 $\models_C$ Es la relación binaria formada por los objetos de un conjunto **I** y los símbolos de los conceptos en **T**, lo cual determinará una clasificación*:* **C** = (*I, T,* $\models_C$).

A.2.6 Relación binaria de instancias

Definición 5 $\models_R$ Es la relación binaria de instancias válidas en las relaciones representados por los símbolos en R, clasificando las tuplas finitas de objetos de I mediante los símbolos de relación(enlace) en R, lo cual determinará una clasificación **R** = (I, *R,* $\models_R$).

Las clasificaciones deberán definirse de modo tal que se cumpla la propiedad de orden parcial $\leq$, la disjunción $\perp$, la cubierta | y la función de aridad σ

A.2.7 Hiper-relación

Definición 6 Una **hiper-relación** es una conexión entre dos o más nodos de una Ontología. Cada **Hiper-relación** es un conjunto de vértices: E = {u, v, …}. (Las **Hiper-relaciones** son dirigidas o direccionadas).

A.2.8 Hiper-grafo

Definición 7 Un **Hypergrafo** H se puede definir como un par (V, E), donde V es un conjunto finito de vértices y E un conjunto finito de **Hiper-relaciones** entre los vértices.

A.2.9 Ontología poblada

Definición 8. *Una ontología poblada es una 6-tupla* $\tilde{O}$ = (**T**, **R**, $\leq$, $\perp$, |, σ) *tales que* **C** = (*I, T,* $\models_C$) y **R** = (*I, R,* $\models_R$) *son clasificaciones y O* = (*I, R,* $\leq$, $\perp$, |, σ) *es una ontología sin poblar.*

Observacion 1: $\tilde{O}$ y *O* denotan la ontología poblada y la no poblada respectivamente (véase la Figura A.3).

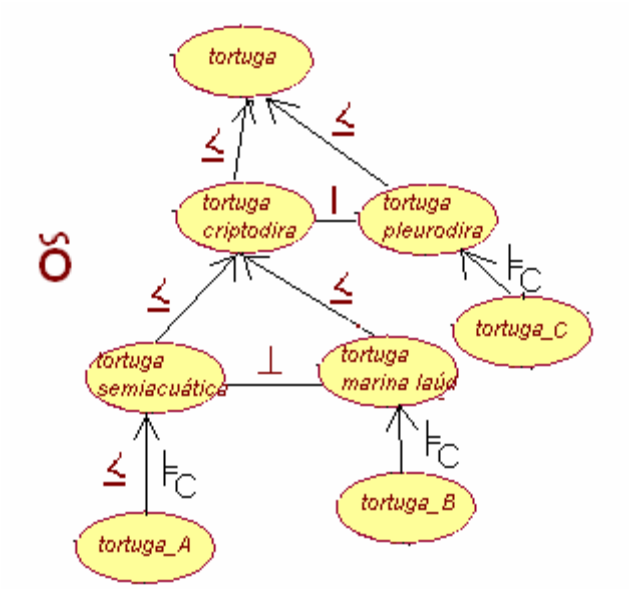


Figura A.3 En la ontología poblada *Õ* se muestran algunas instancias *tortuga_A*, *tortuga_B* y *tortuga_C* y algunos tipos tortuga semiacuática, tortuga marina laúd y tortuga pleurodira. La relación *tortuga_A* $\models_C$ *tortuga semiacuática* quiere decir que *tortuga_A* es del tipo *tortuga semiacuática*; quiere decir que tortuga_A está clasificada como del tipo *tortuga semiacuática*. Se entiende que *tortuga_A* pertenece al conjunto de *tortugas semiacuáticas*, por lo que el concepto *tortuga semiacuática* se considera como el conjunto de todas las instancias de *tortugas semiacuáticas*. *Tortuga_A* es una tortuga particular, instancia o individuo. De la misma forma, *tortuga_B* es una instancia del concepto *tortuga marina laúd,* en tanto que la instancia *tortuga_C* está clasificada como del concepto *tortuga pleurodira*

Observación 2:

Las funciones matemáticas que preservan la estructura que las caracteriza se llaman *homomorfismos* o *morfismos*.

A.2.10 Morfismo entre ontologías

Definición 9. *Sea* $O = (T, R, \leq, \perp, |, \sigma)$ y $O' = (T', R', \leq', \perp', |', \sigma')$ dos ontologías, *un morfismo entre ontologías* es una función $\langle f^*, g^* \rangle : O \rightarrow O'$ *es un par de funciones* $f^* : T \rightarrow T'$ *y* $g^* : R \rightarrow R'$ *tales que,* $\forall\ c,d,e \in C, r \in R,$ *y* $\sigma(r) = \langle c_1, \ldots, c_n \rangle$

1. *Si* $c \leq d$, *entonces* $f^*(c) \leq' f^*(d)$
2. *Si* $c \perp d$, *entonces* $f^*(c) \perp' f^*(d)$
3. *Si* $c | d$, *entonces* $f^*(c) |' f^*(d)$

La Figura presenta la Definición 9.

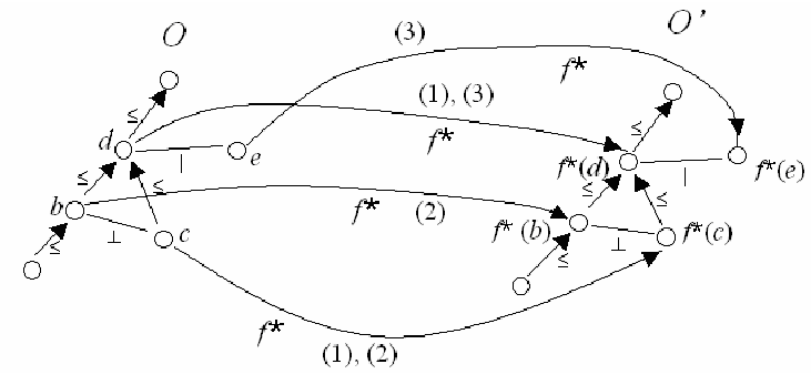


Figura A.4 Morfismo ⟨*f**, *g**⟩ sobre las ontologías *O* y *O'*. Casos (1), (2) y (3) de la Definición 9

Las ontologías que se aplican en esta tesis doctoral son pobladas y estructuralmente son reconocidos como Hipergrafos.

A.2.11 Flujo de información entre ontologías

Una ontología poblada $\tilde{O}$ = (**T**, **R**, ≤, ⊥, |, σ) determina una lógica local *L* = (*I, T,* $\models_C$, ⊢) cuya teoría (*T*, ⊢) viene dada por la relación de secuencia más pequeña (es decir, la relación cerrada que cubre las propiedades de una relación binaria de consecuencia de una lógica local) tal que, $\forall\ c,d \in T$

$c \vdash d$	si y solo si	$c \leq d$
$c,d \vdash$	si y solo si	$c \perp d$
$\vdash c,d$	si y solo si	$c \mid d$

La caracterización de una ontología como una lógica local permite mapear una ontología sin poblar *O* = (*T*, *R*, ≤, ⊥, |, σ) en una ontología poblada $\tilde{O}'$ = (**T'**, **R'**, ≤', ⊥', |', σ') observando el flujo de información. Por lo que se pueblan los tipos de concepto dados en ***T*** y los tipos de relación dados en *R* para obtener las clasificaciones **C** = (*I, T,* $\models_C$) y **R** = (*Z*, *R,* $\models_R$) (nótese que, a diferencia de una ontología poblada, las instancias de *R* no necesitan ser tuplas finitas de instancias de *T*), y más aún se establecen infomorfismos *f* : **C** ⇆ **C'** y *g* : **R** ⇆ **R'**, tales que sus componentes a nivel de tipo *f** y *g** constituyen un morfismo sobre ontologías; porque en este caso se sabe que la ontología poblada $\tilde{O}'$ será una *extensión legitima de O,* en el sentido de que las imágenes de las instancias de $\tilde{O}'$ conforman a *O*.

A.3 Planteamiento del problema a resolver

Planteamiento. Suponiendo dos comunidades cada una con su lógica local *L* y *L*' respectivamente, las que inducen las ontologías pobladas $\tilde{O}$ y $\tilde{O}'$, se trata de hallar el infomorfismo *f*=⟨*f**, f_*⟩ que respete las restricciones que sobre las instancias imponen las teorías de tales lógicas. Equivalentemente, se trata de hallar un "morfismo" ⟨*f**, *g**⟩ entre las ontologías $\tilde{O}$ y $\tilde{O}'$, que preserve la estructura de $\tilde{O}$ y $\tilde{O}'$.

Hallado este morfismo $\langle f^*, g^* \rangle$, se formará una nueva ontología:

$$\tilde{O}'' = \{c \mid c = \text{fus}(o, f^*(o)) \ \forall \ o \in C\}$$

Donde:

fus(o, o') es el resultado de fusionar la instancia $o \in C$ con su imagen $f^*(o) \in f^*(C)$, tomando en cuenta el morfismo $f^*: C \to C'$ y el morfismo $g^*: R \to R'$ (Las clasificaciones de las instancias son $\hat{C}$, $\hat{C}'$; las restricciones son R, R'). Por razones prácticas, fus(o, $f^*(o)$) se apoya en la función extend(o, res($f^*(o)$)), donde res($f^*(o)$) son las restricciones a las que la instancia $f^*(o) \in \tilde{O}'$ está sometida en $\tilde{O}'$.

El resultado de la unión de $\tilde{O}$ y $\tilde{O}'$ *será* la ontología $\tilde{O}''$.

A.4 Propuesta de solución al planteamiento

Desde el punto de vista formal, el problema es soluble, pues dadas dos lógicas locales $L = (I, T, \models, \vdash)$ y $L' = (I', T', \models', \vdash')$, se trata de mapear un conjunto finito de tipos T e instancias I hacia otro conjunto finito I' y T', respectivamente. Si $s_i = \min(|I|, |I'|)$ y $s_j = \min(|T|, |T'|)$, entonces el número de pasos que toma el algoritmo es a lo mas $O(s_i! \, s_j!)$.[1]

Desde el punto de vista práctico, este número es muy grande. Por ejemplo: una ontología de productos vendidos en un supermercado tiene típicamente 5,000 instancias y 500 tipos, por lo que estamos hablando de $O(5000! * 500!)$. Aún cuando se descarte el mapeo entre instancias, el mapeo entre tipos es $O(500!)$.

Terminando con el planteamiento formal (§A.3) del algoritmo, se usarán algoritmos heurísticos y las restricciones adicionales del problema planteado para poder implementar un algoritmo que halle la solución (o una buena solución) en un tiempo razonable.

A.5 Algoritmo general para fusionar dos ontologías A y B

Para fusionar dos ontologías A y B,

1. Primero se obtiene $\langle b, sv_{ab} \rangle = \text{sim}(a, B)$, que resulta, para cada $a \in A$, el elemento $b \in B$ más similar a a, así como el *valor de la similaridad* $sv_{ab} \in [0,1]$ entre a y b.[2]
2. Usando tal función, se procede a hallar el mejor mapeo A $\to$ B, tal que $\forall \ a \in$ A, el elemento $b' \in$ B maximice la suma:

[1] Tomará $s_i!$ pasos ensayar todos los mapeos de cada instancia de I a cada instancia de I'. Al final de estos pasos (o antes), se habrán hallado el infomormismo f^* deseado, o concluido que ninguno existe. Un argumento similar corre para g^*.

[2] La función *sim* puede darse de muchas maneras: i) función COM [2]; b) COM mejorado [9]; c) usando consideraciones sintácticas (por ejemplo, el parecido entre *employee* y *employeeID*); d) semánticas [15].

$$\sum_{a \in A} \text{similarity}(a, b')$$

Donde similarity(a, b') $\in$[0,1] da como resultado la similaridad entre a y b' cuando se toman en cuenta: sv_{ab}, las restricciones de A y B (o equivalentemente, las estructuras de A y B). La instancia $b \in B$ es la mejor imagen para a, y sv_{ab} mide la similaridad entre ambos. En cambio, $b' \in B$ es el mejor mapeo global para a, y sim(a, b') mide la similaridad entre ambos.

3. Halladas las parejas óptimas $\langle a, b' \rangle$ en (2), la fusión de a y b' se define como c=ext (a, b'_{rel}), o sea a extendida por las relaciones de b', cuando sim (a, b') excede cierto umbral τ; y como $c = \lambda$ (la cadena vacía) cuando no lo excede;
4. Hallada cada c en (3) correspondiente a cada pareja (a, b'), la fusión de A y B se define como la ontología $C = \{c \mid c \neq \lambda\} \cup \{\{a\ b'\} \mid c = \lambda\}$. Esta es la solución[3] y la definición de fusión de A con B.

Para formar la c correcta tomando en cuenta las restricciones de a y de b, la función *extend* a menudo tiene que:

i) hacer compromisos para cumplir las restricciones de la relación (paso 3)
ii) insertar en C alguna información adicional (paso 4) por ejemplo:
(1) nuevo elemento (nuevo tipo) para c;
(2) nuevas restricciones para c.

Para completar el algoritmo anterior es necesario:

a) definir la función *sim* (paso 1);
b) definir completamente el mapeo $A \rightarrow B$ (paso 2);
c) definir la función *extend* de (paso 3), para poder hallar c=extend (a, b'_{rel});
d) indicar otras relaciones por añadir a C (paso 4).

A.6 Comparando el planteamiento formal a la realidad

Aunque la solución propuesta está bien definida, es necesario ajustarla a las vicisitudes del mundo real, donde no todas las ontologías son legitimas (es decir, no todas las instancias obedecen a todas las restricciones pertinentes), muchos tipos están pobremente definidos, etc. Este tema lleva la solución formal a las exigencias comunes.

Si se comparan las suposiciones e hipótesis con la situación en el mundo real se observa similitudes y diferencias que aparecen a continuación:

[3] La fórmula dice que la ontología C resultante contiene todos aquellos elementos c que resultaron de parejas (a, b') bastante similares, más aquéllos elementos a y elementos b' que provienen de parejas (a, b') poco similares.

A.6.1 Lo que sucede en una comunidad

Dos instancias se distinguen por su identificador solamente. Tal identificador es único. Lo mismo sucede para los tipos. **Falso en la realidad.** Suceden las siguientes situaciones:

a. Lo más cerca que tienen a un identificador dos personas (por ejemplo) es su nombre propio y puede haber dos personas distintas con el mismo nombre. Cuando se describen tipos, la situación se complica porque se usa la misma palabra (*bote*) para describir:

i. Una vasija pequeña;
ii. El dinero de un premio que no se entrega y se acumula para el siguiente;
iii. Una pequeña embarcación a remo.

Esto se llama *ambigüedad.*

b. Sucede también que dos comunidades distintas (o dentro de una misma comunidad) se dan varios nombres al mismo tipo o concepto. Así, al animal doméstico que se usa para montura se le llama *caballo, corcel, jamelgo, equino, rocín, montura.* A este fenómeno del lenguaje natural se le llama *sinonimia*. Y sucede que muchos sinónimos lo son solo aproximadamente. Por ejemplo, existe una diferencia de edad entre *caballo* y *potro*.

c. Ante la ausencia de un identificador único, los tipos heredan a sus entidades propiedades que los distinguen o diferencian. Como ejemplo, las razas de perros se distinguen por su peso, color, talla, fuerza corporal, etc. Es decir, las propiedades o características de las entidades son importantes para distinguirlas e identificarlas. A este fenómeno se le llama *caracterización:* la descripción de los rasgos de una entidad o ejemplar. También existe la conceptualización a partir de ciertos rasgos (darle un nombre a una descripción compleja). Así, al perro de talla alta, aspecto esbelto, cabeza alargada, hocico largo y pecho estrecho y profundo se le llama *lebrel* en ciertas comunidades, mientras que en otras se usa la descripción completa por no existir para ellas el tipo *lebrel.*

d. Al carecer las propiedades o rasgos (*peso, color, talla...*) de un identificador único, poseen los mismos defectos **a** a **c** arriba señalados para los tipos. Es decir, las comunidades no pueden acordar en los nombres de los tipos (conjuntos, conceptos), y ni siquiera en los nombres de los rasgos que usan para distinguirlos. Por ejemplo, una comunidad distingue a las frutas por su cáscara, a la que otra comunidad le llama cutícula.

Si existe el morfismo (f^*, g^*) *entre dos ontologías* A *y* B*, ¿es posible hallarlo?*. **Casi siempre "no"**, salvo para casos triviales, pequeños.

e. Suponiendo que existiese, el problema es de complejidad *n!*

f. Es muy poco probable que las restricciones de A y las de B sean consistentes, de tal suerte que el morfismo pueda existir. Un algoritmo exhaustivo que buscase el morfismo, hallaría varios posibles "mapeos aproximados", inconsistentes pero "con poco error." Aún suponiendo que A y B sean ontologías legitimas, ¿quién garantiza que lo sean "de la misma manera"?

g. Pudiese existir una solución formal (un morfismo) que "no sea el correcto" o el esperado. Si A habla de guerra, batallas, soldados, enemigo, armas masivas, espías, infiltración, y B habla de enfermedad, microbios, infecciones, antibióticos, anticuerpos, glóbulos blancos, marcadores, podría haber un morfismo entre A y B. A tal "mapeo equivocado pero consistente" se le llama *analogía.*

A.6.2 Comunicación entre dos personas con idiosincrasias distintas

Cuando hay confusión o no está claro un mensaje, el receptor busca fuentes adicionales de aclaración o conocimiento:

Usa el *contexto*, lo dicho recientemente, lo que se sabe del emisor del mensaje;

Emplea diccionarios, tesauros, enciclopedias, etc;
Busca textos similares, frases donde se usen los vocablos confusos;
Busca eliminar la ambigüedad mediante pregunta al emisor, por ejemplo: "¿te refieres a un *bote* que es vasija, o a uno que es embarcación?"

Una vez resuelta la ambigüedad, el receptor puede anotar lo nuevo en su ontología, es decir, puede *aprender.*

A.6.3 Asimilación de una persona a los conceptos nuevos que provienen de otra

¿Es posible calcular com(C, O) según el paso 1 del §A.5?

Efectivamente. Además, tiene sentido hallar en una ontología B el concepto C_B más similar C_A en A.

¿Qué significa la fusión del paso 2 del §A.5, que calcula c= ext(R_A, R_B)?

La comunidad con ontología A puede aprender el concepto C_B que proviene de B, bien porque sea nuevo, bien porque contiene nuevas relaciones o restricciones.

¿Es de la forma anterior como se efectúa el aprendizaje de nuevos conceptos y se extienden los conceptos que ya se tienen?

Efectivamente. De esa forma la ontología *A* va creciendo, de modo consistente. Enriqueciéndose con "lo más que se pueda" de B.

¿Cómo se manejan las inconsistencias aparentes? ¿Y las reales?

En general, no hay suficiente conocimiento en A ni en B para resolver contradicciones entre ellas. El formalismo del §A.5 simplemente rechaza el morfismo como inexistente o imposible. En este documento de tesis se desarrollan métodos y heurísticos que permiten reducir o limitar la inconsistencia, de modo que la ontología C resultante es "aproximadamente consistente" y cuando la diferencia es mucha, se toman soluciones drásticas además, en esta parte del proceso de fusión se usa la Teoría de la Confusión [21] que proveen una solución elegante.

A.6.4 Diferencias entre las ontologías reales

Existen algunas diferencias entre las ontologías en la realidad, que se presentan en la Figura A.5.

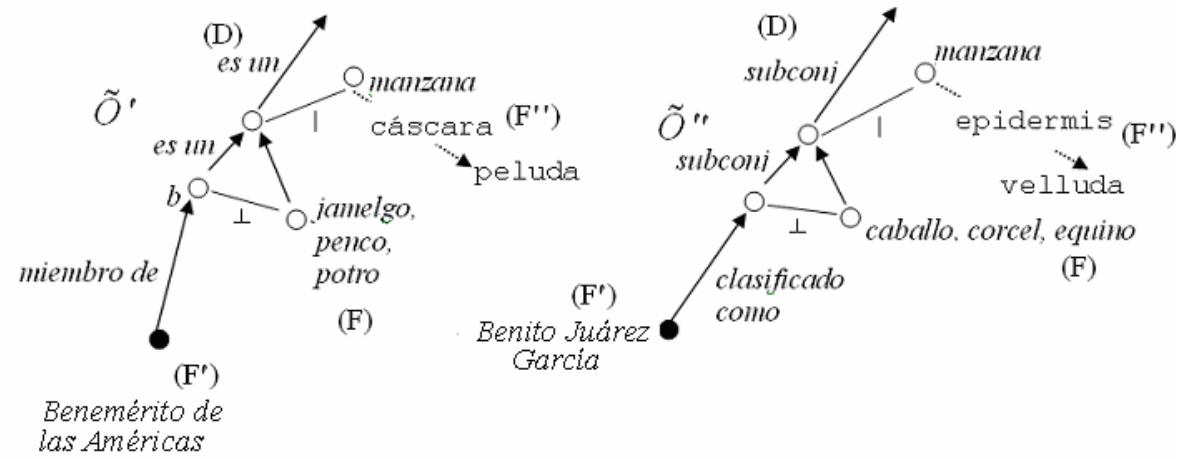


Figura A.5 Algunas diferencias entre las ontologías reales *Õ'* y *Õ''* donde: (D) Diferentes ontologías usan nombres distintos para la misma relación. (F) Ídem para los tipos. (F') Ídem para las instancias. (F'') Los nombres de la misma propiedad (cáscara) son distintos en *Õ'* y *Õ''* y lo mismo sucede con sus valores (velluda). Nótese que las propiedades no son tipos ni instancias; son palabras (cadenas de caracteres) de un lenguaje natural, por eso se escriben en Courier. No pertenecen a la ontología formal, la que carece de propiedades. En la vida real se llaman rasgos, propiedades o características; aquí se señalan con una flecha punteada. Sus valores tampoco pertenecen a la ontología formal, son palabras (o frases temáticas) provenientes de un lenguaje natural, son cadenas de caracteres, números, o fechas

Glosario de términos

Alfabeto.

Es un conjunto finito de símbolos.

Algoritmo.

Es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema. De un modo más formal, un algoritmo es una secuencia finita de operaciones realizables, no ambiguas, cuya ejecución resuelve un problema en un tiempo finito [http://es.wikipedia.org/wiki/Algoritmo].

Anacronismo.

Del griego *ana* "contra" y *chronos* "tiempo" Se refiere a algo que no se corresponde, o parece no corresponderse con la época a la que se referencia, por ejemplo si en una obra de teatro que se desarrollada durante la antigua república romana apareciera un personaje usando una computadora, este objeto sería un anacronismo.

Hay dos tipos de anacronimos: paracronismos y procronismos, el primero consiste en situar hechos del pasado en una época posterior, por ejemplo un carruaje de caballos circulando por una autopista. El segundo consiste en colocar hechos de una época posterior en una anterior, por ejemplo Benito Juárez trayendo consigo un teléfono celular.

Grafo.

Consiste en un conjunto finito de vértices (o nodos) denominados **V** y un conjunto de pares de vértices llamados aristas y que están representados por **A**, denotado por **G** = (**V**, **A**).

Grafo dirigido o **dígrafo**.

Consiste en un conjunto de pares ordenados de vértices de **A**, llamados arcos, denotado como **D** = (**V**, **A**), el arco de **v** a **w** se denota como $\mathbf{v} \rightarrow \mathbf{w}$.

Hiperonimia.

Un término es hiperónimo de otro si el significado del primero incluye al segundo. En el ejemplo de Hiponimia, "animal" se dice que es el hiperónimo o supraordinado de "conejo". El hiperónimo nombra un número mayor de individuos. Por ejemplo, "mueble" nombra más individuos que "silla" [http://www2.udec.cl/~prodocli/semantica1/hiponimia.htm].

Hiponimia.

Esta es la relación de inclusión de significado entre los términos. El vocabulario de una lengua está organizado en relaciones jerárquicas. Se dice que un término es hipónimo de otro si el significado del segundo está incluido (implicado) en el significado del primero. Por ejemplo, la semántica de "animal" está incluido en el de "conejo" por lo tanto, "conejo es hipónimo" de "animal". El hipónimo posee un mayor número de características, es más rico en atributos un término "computadora", que otro como "aparato" [http://www2.udec.cl/~prodocli/semantica1/hiponimia.htm].

Holonimia.

Es una noción semántica que representa el todo de un objeto. Por ejemplo, “bicicleta” es un homónimo de sus partes; esto es, “pedal”, “manubrio”, etc. [es.wikipedia.org/wiki/Holonimia]

Lenguaje formal.

Es un conjunto de cadenas o símbolos tomados de algún alfabeto. Los conjuntos **P** y **S** anteriores se consideran subconjuntos de un lenguaje formal natural.

Jerarquía.

Es el orden de los elementos de una serie según su valor. De igual modo, es la disposición de personas, animales o cosas, en orden ascendente o descendente, según criterios de clase, poder, oficio, categoría, autoridad o cualquier otro asunto que conduzca a un sistema de clasificación [es.wikipedia.org/wiki/Jerarquía].

Mapeo.

En matemática un **mapeo** (o una flecha) es una relación entre conjuntos y es sinónimo de función (porque satisface casi las mismas condiciones), pero más general, pues no necesariamente el dominio o el codominio de la relación son conjuntos numéricos [es.wikipedia.org/wiki/Mapeo].

Meronimia.

Es una noción semántica que representa a una parte de un objeto. Por ejemplo, “sillín”, “pedal”, “aro” y “manubrio” son merónimos de “bicicleta”.

Metadatos.

Del griego *μετα* meta (después de) y latín *datum* (lo que se da), (dato), literalmente (sobre datos), son datos que describen otros datos. En general, un grupo de metadatos refiere a un grupo de datos, llamado *recurso* [http://es.wikipedia.org/wiki/Metadatos].

Nodo.

Sea **V** el conjunto de vértices de una red semántica, a cada **v** que pertenece a **V** se le llama nodo. Dado un conjunto no vacío $\mathbf{V_1}$, se dice que cada uno de los elementos $\mathbf{v_1} \in \mathbf{V_1}$ tienen relaciones con otros nodos y *palabras*. Adquieren su significado a partir de su posición en la red semántica, de las palabras que los representan y de las relaciones que de ellos emanan. Estos nodos también son conocidos como conceptos.

Ortogonal.

Se refiere a las dimensiones independientes de una cantidad medida. Por ejemplo, en un mapa, es posible localizar un punto por su longitud y su latitud. Esas dos mediciones son independientes y se necesitan a ambos para localizar el punto. Se dice que ellos son ortogonales [http://www.dliengineering.com/vibman-spanish/ortogonal1.htm].

Red semántica.

Es un grafo dirigido, donde cada vértice y cada arco tienen una cadena que denota el nombre del dato y el tipo de enlace respectivamente. Donde cada vértice puede estar conectado (hacia arriba) con uno o mas vértices y hacia abajo también.

Símbolo.

Es una entidad abstracta. Las letras y los dígitos son ejemplos de símbolos usados con frecuencia.

Web.

La World Wide Web (*Telaraña mundial*), es un sistema de hipertexto que funciona sobre Internet. Nació alrededor de 1989 a partir de un proyecto CERN (Organisation Eiropéenne pour la Recherche Nucléaire, Consejo Europeo para la Investigación Nuclear), en el que Tim Berners-Lee (TimBL) físico inglés egresado de la Universidad de Oxford; construyó el prototipo que dio lugar al núcleo de lo que hoy es la Web. La intensión original era hacer más fácil el compartir textos de investigación entre científicos y permitir al lector revisar las referencias de un artículo mientras lo fuera leyendo. Un sistema de hipertexto (hipervínculos o referencias cruzadas que parten de un texto a otro) enlazaría todos los documentos entre sí para que el lector pudiera revisar las referencias de un artículo mientras lo fuera leyendo [http://es.wikipedia.org/wiki/Web].

Referencias

1. Atserias, J., Padró, L., and Rigau, G. *Integrating multiple knowledge sources for robust semantic parsing*. In proceesing of International Conference on Recent Advances in Natural Language Processing (RANLP'01), Tzigov Chark, Bulgaria, 2001
2. Banerjee, S., Pedersen, T. An *Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet*. In Proceeding Of The Fourth International Conference On Computational Linguistics and Intelligent Text Processing (CICLING-02), México City. 2002.
3. Bechnofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P. F., Andrea, L. *OWL Web Ontology Language Reference*. W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/.
4. Black, P. *Dictionary of Algorithms and Data Structures*, Ed., National Institute of Standards and Technology NIST www.nist.gov/dads/HTML/hyperedge.html. 2006
5. Botello, B. *Infiriendo Relaciones Entre Bases de Datos Autónomas*. CIC-IPN Tesis Doctoral en proceso de desarrollo. [2006].
6. Connolly, C., Harmelen, F., Horrocks, I., McGuinnes, D. L., Patel-Schneider, P. F. Andrea Stein L. *DAML+OIL Reference description*. March 2001. W3C Note 18 December 2001. http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218.
7. Cuevas, A., and Guzmán, A. *Improving the Search for the Most Similar Concept in other Ontology*. In proc. XVIII Congreso Nacional y IV Congreso Internacional de Informática y Computación. Torreón Coah. México. Octubre 2005.
8. Cunningham, H., Declerck, T., Kuper, J., Reidsma, D., and Saggion, H. *Intelligent Multimedia Indexing and Retrieval through Multi-source Information Extraction and Merging* Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI), pages 409-414, February 2003.
9. Doan, A., Madhavan, J., Domingos, P., and Halevy, A,. *Learning to Map between Ontologies on the Semantic Web*. WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA. ACM 1-58113-449-5/02/0005.
10. (16)Domingue, J., Motta, E., and Corcho Garcia, O., *Knowledge Modelling in webOnto and OCML A User Guide*., kmi.open.ac.uk/projects/webonto/user_guide.2.4.pdf Fisher, G., and Ostwald, J. "Knowledge Management: Problems, Promises, Realities, and Challenges", IEEE Intelligent Systems, 16-1 (60-72). 2001
11. Dou, D., McDermott, D., and Qi, P. *Ontology Translation by Ontology Merging and Automated Reasoning*. In Proc. EKAW Workshop on Ontologies for Multi-Agent Systems. 2002.
12. Fellbaum, C. *WordNet An Electronic Lexical Database*. Library of Congress Cataloging in Publication Data.1999.
13. Fridman, N., and Musen, M. *PROMPT: Algoritm and Tool for Automated Ontology Merging and Alignment*. In Proc. Seventeenth National Conference on Artificial Intelligence. pp 450-455, Austin, TX, USA, 2000.
14. Fridman, N., Musen, M., Mejino, J., Rosse, C. *Pushing the Envelope: Challenges in a Frame-Based Representation of Human Anatomy*, Data & Knowledge Engineering, v.48 n.3, p.335-359. 2004.
15. Fridman, N., and Musen, M. *SMART: Automated Support Ontology Merging and Alignment*. In Proceedings of the Twelfth Banff Workshop on Knowledge Acquisition, Modeling, and Management, Banff, Alberta. 1999.
16. Ganter, B., Stumme, G., and Wille, R. *Formal concept analysis: Foundations and applications.* 1st edition. Ed. New York, NY: Springer, 2005.
17. Genesereth, M. *Knowledge Interchange Format. Draft proposed American National Standard [dp ANS]* NCITS.T2/98-004. http://logic.stanford.edu/kif/dpans.html. 2004
18. Gruber, T. *Toward principles for the design of ontologies used knowledge sharing*. Originally in N. Guarino & r. Poli, (Eds.), International Workshop onf Formal Ontology, Padova, Italy. 1993.
19. (1)Guzmán, A. *Hallando los temas principales de un artículo en español*. Soluciones Avanzadas 5, 47 [15 Jul. 97], págs. 58-63 y 5, 49 [15 Sept. 97], pp. 66-71.

20.Guzmán, A., and De Gyves, V. *A distributed digital text accessing and acquisition system*. BiblioDigital. SoftwarePro International. Lecture Notes in Computer Science 3061, 274-283. (Springer Verlag 2004), ISSN 0302-9743
21.Guzmán, A., and Levachkine, S. [2004] *Hierarchies Measuring Qualitative Variables*. Lecture Notes in Computer Science LNCS 2945 [Computational Linguistics and Intelligent Text Processing], Springer-Verlag. 262-274. ISSN 0372-9743
22.(2)Guzmán, A., and Olivares, J. [2004] *Finding the Most Similar Concepts in two Different Ontologies*. Lecture Notes in Artificial Intelligence LNAI 2972, Springer-Verlag. 129-138. ISSN 0302-9743
23.Hummel, R., and Zucker, S. *On the foundations of relaxation labeling processes*. IEEE Trans. Patt. Anal. Machtne Intell. 5, (May), pp. 267-287. 1983.
24.Jiménez, A. *Caracterización y Ponderación de Propiedades Lógicas sobre Valores Cualitativos Organizados en Jerarquías*. CIC-IPN. Tesis en proceso de elaboración. 2006.
25.(33)Kalfoglou, Y., and Schorlemmer, M. *Information-Flow-based Ontology Mapping*. Proceedings of the 1st International Conference on Ontologies, Databases and Applicatio of Semantics (ODBASE'02), Irvine, CA, USA. 2002.
26.Kotis, K., and Vouros, G., Stergiou, K. *Towards Automatic of Domain Ontologies: The HCONE-merge approach*. Elsevier's Journal of Web Semantic (JWS), vol. 4:1, pp 60-79.. Available on line ant (ScienceDirect): http://authors.elsevier.com/sd/article/S1570826805000259. 2006.
27.Manola, F., Miller, E. *RDF Primer. W3C Recommendation*. 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
28.McGuinness, D., Fikes, R., Rice, J., and Wilder, S. *The Chimaera Ontology Environment Knowledge*. In Proceedings of the Eighth International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues (ICCS 2000). Darmstadt, Germany. 2000.
29.Michael, K., Welty, C., McGuinnes, D. *OWL Web Ontology Language Guide, W3C Recommendation*. www.w3.org/TR/2004/REC-owl-guide-20040210/. 2004
30.Olivares, J. An *Interaction Model among Purposeful Agents, Mixed Ontologies and Unexpected Events*. Ph. D. Thesis, CIC-IPN. México. www.jesusolivares.com/interaction/publica. 2002.
31.Pérez, A., and Suárez M. *Evaluation of RDF[S] and DAML+OIL Import/Export Services within Ontology Platforms*. 3rd Mexican International Conference on Artificial Intelligence (MICAI2004) Mexico pp, 109-118. 2004.
32.(32)Reed, S. L., and Lenat, D. *Mapping Ontologies into Cyc*. In proceeding of AAAI Workshop on Ontologies and the Semantic Web, Edmonton, Canada. 2002.
33.Silva, N., and Rocha, J. *Merging ontologies using a Bottom-up lexical structural approach*. In Proceedings of The Seventh International Society for Knowledge Organization Conference (7th ISKO), Granada, Spain, July 2002.
34.Stumme, G., Maedche, A. *Ontology Merging for Federated ontologies on the semantic web*. In: E. Franconi, K. Barker, D. Calvanese (Eds.): Proc. Intl. Workshop on Foundations of Models for Information Integration (FMII'01), Viterbo, Italy, 2001. lNAI, Springer 2002 (in press).
35.Voutilainen, L., and Padró, L. *Developing a hybrid NP parser*. In Proceeding The 5th Conference on Applied Natural Language Processing, ANLP, pages 80-87, Washingtong DC, 1997.
36.Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S., *Ontology-Based Integration of Information A Survey of Existing Approaches*. In Proc. IJCAI—01 Workshop: Ontologies and Information Sharing. OIS-2001.

Referencias de Internet.

37.AKT: plainmoor.open.ac.uk/ocml/domains/aktive-portal-ontology/techs.html
38.Chimaera http://www-ksl-svc.stanford.edu:5915/doc/chimaera/chimaera-docs.html
39.DAML+OIL http://www.w3.org/TR/daml+oil-reference
40.Loom http://www.isi.edu/isd/LOOM/LOOM-HOME.html
41.OIL http://www.ontoknowledge.org/oil/
42.OntoMerge http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html
43 Protege http://protege.stanford.edu/whatis.html
44.WEB Onto http://137.108.64.26:3000/webonto?ontology=AKTIVE-PORTAL-ONTOLOGY&name=TECHNOLOGY&type=CLASS
45.www.isi.edu/natural-language/projects/ONTOLOGIES.html