Trabajo de Grado
Construcción Colaborativa de Ontologías
CO-Protégé

Guillermo Baldo

Directora: Alicia Díaz

21 de diciembre de 2005

El trabajo de grado es el último peldaño de mi carrera como estudiante de la Licenciatura en Informática, quiero agradecer a muchas personas que me ayudaron y estuvieron a lo largo del camino.

Para empezar a mi familia: a mis padres que me apoyaron todo el tiempo, a mis hermanas Alejandra y Rosana, mis cuñados Ramiro y Sergio, a mi tío Horacio. A mis tres solcitos Lautaro, Fermín y la más reciente incorporación familiar Olivia. Los quiero mucho, gracias.

El cambiar de ciudad para comenzar una carrera implica dejar muchas cosas, varias personas quedaron en Olavarría, mi ciudad natal, a todos gracias. Mariela, por tantas cosas, gracias. Fede por tus viajes de sábados, gracias. Otros siguieron el camino y terminaron en La Plata. Leo y Juan, porque siempre están, muchas gracias. Gaby y Sergio por las primeras épocas y por las de ahora, gracias.

Ya en La Plata conocí a mucha gente. Ale G. (mi concubina), Chipi y Flor que sin ser informáticos saben todo sobre el trabajo, gracias.

En la carrera me encontré a muchas personas, sería muy largo nombrar a todos los que participaron de alguna u otra manera, gracias a todos. Mariana, Mónica, Claudia, Matías, Valeria,....., gracias.

En la cátedra de Funcional conocí a gente muy valiosa para mi carrera y a nivel personal. Fidel, sabés lo que te admiro y lo que me ayudaste, muchas gracias. Germán R. por ayudar a un principiante, gracias. Esteban y Hernán, aprendí tantas cosas, muchas gracias. Andrés, charlamos de muchos temas en los recreos de pasillo y hablamos mucho de ontologías, gracias. Y no me olvido, Fidel gracias por incluirme en la página.

Diego, también nos conocimos en la cátedra. Si no me fallan las cuentas, fueron catorce los finales que rendimos y otras tantas cursadas, muchas gracias.

Las horas de estudio siempre tenían mate y música. Los nervios antes de rendir, que fueron y serán muchos, tuvieron una manera de apaciguarse. Gustavo, Zeta y Charly, gracias.

El trabajo de grado fue inspirado y gestado, luego de rumbear por otros temas, dentro del laboratorio LIFIA. Gracias a todos los que participaron y respondieron a mis inquietudes. Miguel y Emiliano, gracias por los comienzos.

Alicia, *mi Jefa*, hace cinco años fui a hablar con vos para pedirte ser tu becario. Luego, con el paso del tiempo, surgió el trabajo de grado. Un día mencionaste la palabra ontología, y se comenzó a gestar: "Construcción colaborativa de ontologías". Hablamos bastante del trabajo y de sus temas relacionados, largas charlas que derivaban en otros temas, de los más variados. Me ayudaste mucho en todo el trayecto, muchas gracias.

Finalmente hace aproximadamente un año empecé a trabajar, creo que a todos mis compañeros en algún momento les dije "mañana entrego", llegó el día, gracias.

Índice general

Índice de figuras

Capítulo 1

Introducción

La actividad de compartir conocimiento ha ganado un especial interés. Las personas se juntan y comparten experiencias, trabajos, tópicos sobre un dominio particular, acumulan conocimiento y se mantienen relacionados por el valor que encuentran aprendiendo juntos.

El grupo trabaja sobre un dominio específico, que debe ser conceptualizado para lograr una representación formal y consensuada del mismo. Identifican los elementos y tareas relevantes del dominio a modelar, junto con las relaciones que existen entre los mismos. La información obtenida en el proceso, debe ser representada de alguna manera, y es la base principal para la captura de conocimiento. Actualmente se están adoptando las Ontologías [Gru93, Gua98] como forma para capturar conocimiento, pues proveen una representación estándar que facilita la comunicación entre las partes.

Una Ontología es: "una especificación formal, explícita de una conceptualización compartida" [Gru93], se puede entender a las ontologías como el resultado de aplicar sobre un dominio un método para obtener una representación formal de los conceptos que contiene y las relaciones que existen entre los mismos.

El desarrollo de una ontología de manera colaborativa implica un diseño en conjunto, que busca lograr una perspectiva única del universo de discusión de los participantes[1]. La ontología que modelan se denomina ontología de dominio e involucra los conceptos de un dominio particular. La naturaleza del diseño colaborativo lleva a que surjan visiones diferentes por parte de los diseñadores, estas diferencias deben ser superadas para llegar a un consenso. La noción de conflicto es inherente a la idea de diseño colaborativo, los participantes tienen diferentes puntos de vista, los conflictos corresponden a opiniones divergentes sobre un mismo concepto.

[1] El *Universo de Discurso* de una ontología es el conjunto de objetos que están representados en ella sobre los cuales se puede hablar y razonar.

Los diferentes puntos de vista deben poder ser expresados dentro de la misma herramienta, ya que ellos son en sí mismos una fuente de conocimiento. Además los conflictos son el punto inicial para generar discusiones, los participante brindan alternativas para los conflictos según sus puntos de vista, y opinan sobre las mismas para alcanzar el consenso deseado.

Este trabajo de grado plantea el diseño de una ontología en forma colaborativa desde una perspectiva particular, basada en tres pilares:

1. Una metodología para el diseño colaborativo de ontologías, especializada a partir de un proceso de diseño más general. El proceso involucra cuatros pasos: *externalización*, *publicación*, *internalización* y *reacción* [DC04, NT95].

2. Soporte de diferentes puntos de vista, dentro del sistema utilizado para el diseño colaborativo de la ontología.

3. Mecanismos de *awareness*, para asegurar que el grupo esté en conocimiento de los cambios en el diseño.

La metodología define una forma metódica para realizar el diseño colaborativo de ontologías. El punto inicial es la *externalización*, una persona realiza un diseño como una iniciativa individual en un ámbito privado, luego da a conocer su diseño al resto del grupo por medio de la *publicación*. La *internalización* se realiza cuando otro usuario comprende los nuevos elementos de diseño publicados y en consecuencia puede reaccionar. Las reacciones implican una modificación en el diseño compartido, ya sea para extender, modificar o simplemente expresar una posición sobre el diseño anterior.

El nuevo enfoque propone como segundo pilar, tratar las disconformidades propias de la actividad de diseño dentro del sistema. Esto se logra ampliando las bases de representación del conocimiento, lo que en ontologías se denomina arquitectura de meta-clase.

El tercer pilar, el *awareness*, ayuda a mantener la actividad de diseño en "acción". La información de *awareness* es entregada a los participantes, que en todo momento se notifican de los cambios, y pueden accionar según los mismos.

Teniendo en cuenta que las ontologías son una forma de capturar el conocimiento y que las acciones involucradas en la actividad colaborativa son una fuente de conocimiento en si mismas, se define un modelo ontológico que refleje los aspectos de colaboración. La ontología que diseñan en conjunto, incluye la conceptualización del dominio particular de los usuarios, más un conjunto de conceptos para reflejar la actividad de colaboración. Esta convivencia permite representar de forma fácil las relaciones que se arman entre los elementos de la ontología y los de la actividad colaborativa. Por ejemplo al tener representados a los usuarios en la misma ontología de dominio, la relación "fue creado" que se da entre un usuario y un elemento ontológico particular, es fácil de establecer. Si

bien la convivencia de diferentes conceptos es buena, hay que poder identificar fácilmente la fuente de donde proviene los conceptos ontológicos, esto se logra ampliando las bases de representación del conocimiento, de tal manera que el *tipo* de los conceptos modelados sea distinto según su origen.

Para dar soporte al nuevo enfoque en el diseño colaborativo de ontologías se implementó el prototipo CO-Protégé, un conjunto de *plug-ins* que extienden el editor de ontologías Protégé-2000 [PRO05].

CO-Protégé permite la construcción de ontologías en forma colaborativa soportando los tres pilares. Cuando un grupo de personas comienzan con el diseño colaborativo de una ontología utilizando CO-Protégé se encuentran con:

- Un conjunto de usuarios, que participan en el diseño de una ontología de dominio común a todos.
- Un espacio privado por cada usuario, a los que solo "el dueño" tienen acceso; contiene la ontología que permite el diseño individual, llamada *ontología privada*. La edición de las ontologías privadas sigue el método estándar provisto por Protégé-2000.
- Un espacio compartido común a todos los participantes (todos pueden acceder a él) que contiene la ontología que diseñan en conjunto, llamada *ontología compartida*. La edición de esta ontología es indirecta, los usuarios *publican* elementos diseñados previamente en sus ontologías privadas.
- Una nueva arquitectura de meta-clase para la ontología compartida que permite modelar los conceptos colaborativos.
- En el espacio compartido se encuentra además de la ontología del dominio que se está diseñando, ontologías genéricas (capítulo 2 sección 2.3) para modelar elementos propios de la actividad colaborativa: los perfiles de los usuarios, los conflictos, las alternativas, las argumentaciones, las acciones y las notificaciones.
- Funcionalidades tanto para la edición de la ontología privada, como para la comunicación entre una ontología privada y la ontología compartida. Provee también la funcionalidad para la creación y seguimiento de conflictos, la edición de los perfiles, etc.
- Mecanismos para soportar *awareness* que permitan al grupo estar en conocimiento de lo que sucede con el diseño.

El informe está distribuido de la siguiente manera: primero, en el capítulo 2 se hará una introducción a las ontologías, haciendo un análisis general y observando, en particular, las metodologías actuales para su diseño; el capítulo contiene una descripción de la herramienta Protégé-2000, base para la implementación del prototipo CO-Protégé. En el capítulo 3 se analizan las características deseables de una herramienta colaborativa para el diseño de ontologías, a partir del análisis de las características de las herramientas groupware. El diseño colaborativo de ontologías, basado en los tres pilares: la metodología de diseño, soporte de diferentes puntos de vista y mecanismos de *awareness*, se describe en el capítulo 4. La implementación del prototipo CO-Protégé que reúne los conceptos dados en los capítulos anteriores se explica en el capítulo 5. Finalmente en el capítulo 6 se brindan las conclusiones.

Capítulo 2

Ontologías

Desde el comienzo de los años noventa las ontologías se volvieron un tópico común de investigación en el área de Inteligencia Artificial, incluyendo ingeniería del conocimiento, procesamiento del lenguaje natural y representación de conocimiento. La definición de diccionario típica del término ontología las identifica como "la parte de la metafísica que trata del ser en general y de sus propiedades trascendentales, estudiando la naturaleza de la existencia".

En el campo de la informática una ontología es una entidad computacional, y no ha de ser considerada como una entidad natural que se descubre, sino como recurso artificial que se crea [Mah96]. Una ontología ha de pensarse como un entendimiento común y compartido de un dominio, que puede comunicarse entre personas y sistemas computacionales.

El hecho de que las ontologías puedan compartirse y reutilizarse en aplicaciones diferentes explica en parte el gran interés suscitado en los últimos años en la creación e integración de ontologías [SGP98b]. Recientemente la noción de ontología se ha extendido a áreas tales como integración inteligente de información desde orígenes heterogéneos, recuperación de información desde Internet (IR-Information Retrieval), gestión del conocimiento y web semántica.

En la sección 2.1 se explican diferentes definiciones de Ontologías, luego en la sección 2.2 se analizan los distintos componentes que la conforman. La sección 2.3 muestra la clasificación de las ontologías según nivel de generalidad y luego las áreas de aplicación. Cuando se diseñan ontologías se intenta que las mismas cumplan con ciertas características como se ve en la sección 2.5. Las Ontologías son definidas sobre un lenguaje lógico formal, estos lenguajes se explican en la sección 2.6, centrándose en los lenguajes basados en *frames*. En la sección 2.7 se describen las características que deben tener las herramientas para construir y mantener ontologías, luego se describe en detalle la herramienta Protégé-2000 utilizada como base para la implementación del editor de ontologías colaborativo propuesto en este trabajo de grado. En la sección 2.9 se hace un análisis de las diferentes metodología de diseño y finalmente se exponen las conclusiones del capítulo.

2.1. Definición

En filosofía el término ontología se remonta al uso que le dió Aristóteles en su intento de clasificar todo lo existente en el mundo. La palabra Ontología proviene de la conjunción de los términos griegos *ontos* y *logos* que denotan existencia y mundo respectivamente. Aunque en filosofía una ontología es una explicación sistemática de la existencia, en los sistemas basados en conocimiento lo que existe es exactamente lo que se puede representar mediante un formalismo declarativo llamado *Universo de Discurso*. El *Universo de Discurso* de una ontología es el conjunto de objetos que están representados en ella sobre los cuales se puede hablar y razonar.

Dentro de la informática la definición de Ontologías más citada es la de Gruber [Gru93]: "Una ontología es la especificación explícita de una conceptualización", luego fue ligeramente modificada por Borst [Bor97]

"Una ontología es una especificación formal, explícita de una conceptualización compartida".

La definición anterior involucra varios conceptos:

- **Formal:** indica que la especificación debe representarse por medio de un lenguaje de representación formalizado, permitiendo a la ontología ser legible o interpretable tanto por un humano como por un computador.
- **Explícita:** se refiere a que el tipo de concepto usado y la restricciones para su uso son explícitamente definidas, es decir se debe especificar de forma consciente los distintos conceptos que conforman una ontología.
- **Conceptualización:** se refiere a un modelo abstracto de algún fenómeno del mundo del que se identifican los conceptos que son relevantes.
- **Compartida:** refleja la noción de que una ontología captura conocimiento consensual, esto quiere decir que el conocimiento no es privado de un individuo, sino que es aceptado como mínimo por el grupo de personas que deben usarla.

Existen otras perspectivas complementarias a las definiciones dadas, un enfoque define que "una ontología es un conjunto de términos estructurados jerárquicamente que describen un dominio. La ontología sera el "esqueleto" sobre el cual se construye luego la base de conocimiento" [Sow00]. La definición destaca que las ontologías proporcionan la estructura taxonómica de un dominio, que será especializada con los conocimientos específicos necesitados por una aplicación particular.

En conclusión, una ontología es un modelo abstracto de algún fenómeno del mundo sobre el que se identifican los conceptos relevantes, que son representados por medio de un lenguaje de representación formal y que es aceptado como mínimo por el grupo de personas

que pretenden usarla. Se puede entender a las ontologías como el resultado de aplicar sobre un dominio un método para obtener una representación formal de los conceptos que contiene y las relaciones que existen entre los mismos.

2.2. Componentes de una ontología

Una ontología consta de un conjunto no vacío de *conceptos* identificados como entidades relevantes en el dominio a modelar, un conjunto de *relaciones*, un conjunto de atributos que describen los conceptos que pueden ser propios o heredados en una especialización y un conjunto de axiomas que vinculan elementos de la ontología en condiciones que deben cumplirse siempre.

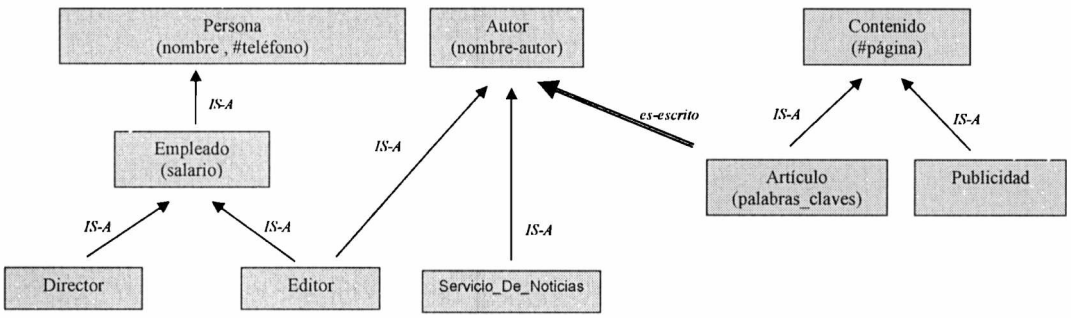


Figura 2.1: Fragmento de una Ontología simple

Cuando se habla de ontologías aparecen los términos: conceptos, relaciones, funciones, instancias y axiomas; a continuación se describen:

- **Conceptos**: un concepto, también denominado clase, puede ser cualquier cosa del dominio a modelar de la cual se pueda aseverar algo, puede ser un objeto físico, la descripción de una tarea, función, acción, etc. Los conceptos tienen asociados un conjunto de atributos que los caracterizan. Por ejemplo en la *figura 2.1* se puede observar el concepto *Persona*, que posee los atributos *nombre* y *número de teléfono*; el concepto *Autor* con el atributo *nombre*; también es equivalente decir que la Ontología contiene las clases *Persona* y *Autor*.

- **Relaciones**: una relación representa la interacción y enlace entre conceptos del dominio; son formalmente definidas como subconjuntos de productos cartesianos de n conjuntos $R = C_1 \times C_2 \times \ldots \times C_n$.

  El ejemplo de la *figura 2.1* muestra la relación *es-escrito* = *Artículo* $\times$ *Autor*, que modela la idea de que un Artículo es escrito por un Autor.

Una de las relaciones más difundidas en la relación *ES-UN* (*IS-A*), por medio de la cual se definen taxonomías (estructuras jerárquicas). En el ejemplo podemos observar varias de estas relaciones, que determinan, entre otras relaciones, "que un *Director* es un *Empleado*". La relación taxonómica *ES-UN* es transitiva (no es reflexiva y es antisimétrica), por lo tanto también se puede afirmar "que un *Director* es una *Persona*"

- **Funciones**: son un tipo concreto de relaciones, donde el enésimo elemento de la relación es único para los $n-1$ elementos anteriores. Formalmente las funciones se definen como $F :: C_1 \times C_2 \times \ldots \times C_{n-1} \rightarrow C_n$

  Por ejemplo se puede definir la función *salario_anual* que dado un *Empleado* retorne el salario que ganaría en un año. La relación *escrito_por* es también un ejemplo de una función.

- **Instancias**: las instancias se usan para representar objetos determinados de un concepto (o clase). Una instancia representa un elemento particular del dominio.

- **Axiomas**: los axiomas son fórmulas, generalmente expresadas en lógicas de primer orden, que se usan para expresar verdades que se cumplen siempre en la realidad modelada y se utilizan para verificar la corrección de la información contenida en la ontología. Dentro de una Ontología se pueden ver dos tipos de axiomas:

  - Axiomas estructurales: establecen condiciones relacionadas a las jerarquías de la ontología, conceptos y atributos definidos. Por ejemplo: "Si A y B están en la jerarquía de C, son subclases directas, entonces A no es subclase de B".

  - Axiomas no estructurales: establecen relaciones entre atributos de un concepto y son específicos de cada dominio. En el ejemplo se puede determinar que "todo *Director* debe tener un *salario* mayor que el de sus *Empleados*".

En base a las definiciones de las ontologías y los componentes descriptos podemos distinguir dos niveles dentro de una misma ontología, el primero dado por el conjunto de las clases y el segundo dado por el conjunto de las instancias. La *figura 2.2* muestra esta división, en la parte superior se encuentran las *clases* (representados por cuadrados) y el último nivel de la jerarquía (las *hojas* - representados por elipses) representa a las instancias.

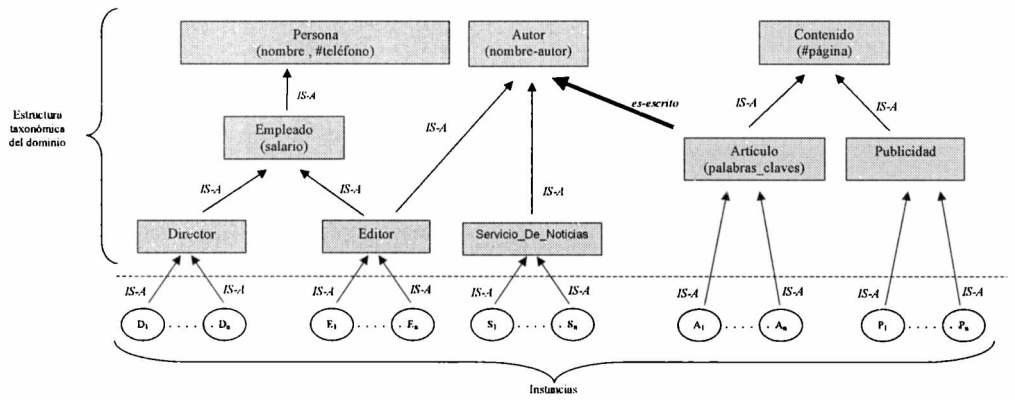


Figura 2.2: Ontología simple, Clases e Instancias

2.3. Clasificación de Ontologías

Existen varias formas de clasificar ontologías [Gua98, HSW97, UG96], la clasificación dada por Steve [SGP98a] es en base al nivel de generalidad que es usado en la descripción del dominio y distingue tres tipos fundamentales de ontologías:

- **Ontologías de dominio:** representan un conocimiento especializado pertinente de un dominio o subdominio como medicina, leyes, aplicaciones militares, administración clínica, protocolos, oncología, etc. Ejemplos de este tipo de ontologías se pueden ver en [Tat96, Kuo93, MGS97, SPKR96]. En el *Apéndice A* se muestra la jerarquía taxonómica (clases e instancias) de una ontología utilizada para representar en dominio de los "vinos" (OntoWiner).

- **Ontologías genéricas:** se usan para representar conceptos generales y fundacionales del conocimiento como las estructuras parte/todo, la cuantificación, los procesos o los tipos de objetos.

- **Ontologías representacionales o Meta-Ontologías**

  Las ontologías representacionales especifican los conceptos que subyacen a los formalismos de representación del conocimiento, por lo que también se las denominan Meta-Ontologías. Pretenden establecer una estructura básica bajo la cual todos los términos en cualquier ontología existente deberían poder relacionarse. No existe una ontología única de este tipo, por ejemplo existen entre otras: la meta-ontología propuesta por Sowa [Sow00], Mikrokosmos Ontology [NR01], The Frame Ontology [Gru93] y la meta-ontología definida en el contexto de la herramienta Protégé-2000, que se describe en la sección 2.8.

2.4. Áreas de aplicación

A partir que la investigación se profundizo en los años 90, las ontologías han ampliado su campo de utilización. Actualmente las ontologías se utilizan en diferentes áreas como:

- **Procesamiento de lenguaje natural**

  Las ontologías se están empleando para construir representaciones independientes de la lengua que puedan servir de punto de encuentro entre dos o más lenguas naturales. Una ontología puede mantener la definición de elementos gramaticales del lenguaje y sus formas de relacionarse, permitiendo por ejemplo el análisis sintáctico de un texto. En este sentido la ontología se considera como el repositorio de conceptos que establecen conexiones entre los símbolos de una lengua.

  Las ontologías creadas para el procesamiento de lenguaje natural, son utilizadas generalmente para la traducción automática de texto, denominando a este proceso *Traducción Automática Basada en el Conocimiento* (KBMT: Knowledge Based Machine Translation [BNM95])

- **Ingeniería del Conocimiento**

  En la Ingeniería del Conocimiento las ontologías se utilizan como un aporte para construir el modelado conceptual [PFS+01], donde se crea un *vocabulario* de la terminología del dominio de la aplicación (los conceptos), se definen relaciones entre dichos términos y restricciones en su uso. Este modelo conceptual explícito es la ontología.

  Además en la Ingeniería del Conocimiento se realiza la construcción de una base de conocimiento, usando la ontología definida en la etapa anterior como un conjunto de esquemas o contenedores de conocimiento, se completa la base de conocimiento con instancias de los conceptos definidos.

- **Interoperabilidad entre sistemas heterogéneos**

  Las ontologías se presentan como una importante solución para lograr una integración inteligente, en particular en el área de bases de datos. Una ontología puede ser un elemento clave asociado a un mediador que integra datos provenientes de fuentes heterogéneas. Con una ontología terminológica se pueden organizar los términos que son usados en interacciones entre sistemas heterogéneos, de manera de reconocer cuándo una aplicación usa un término que es más general o más específico que otro en uso por otra aplicación. Con el formalismo introducido por el uso de la ontología se puede contar con una definición más completa de cómo se relaciona un término de un origen con el de otro y eventualmente usar axiomas definidos que los vinculen por igualdad o que expresen un término exactamente en función del otro, esto permite establecer correspondencias seguras y automáticas entre ellos.

- **Organización de empresas**

  En el área empresarial las ontologías desempeñan, entre otros, el rol de mantener una memoria organizacional colectiva que permita a los distintos niveles de la empresa interoperar en un lenguaje común con reglas únicas. Las Ontologías TOVE (Toronto Virtual Enterprise) desarrolladas en la universidad de Toronto y la Enterprise Ontology desarrollada en la Universidad de Edimburgo son ejemplos de este tipo de ontologías.

- **Indexación**

  Aplicaciones que realizan búsquedas basadas en conceptos (por ejemplo documentos en algún repositorio) utilizan ontologías como índice para localizar los términos; obteniendo un acceso rápido, consensuado y controlado a los recursos de información almacenados. También las ontologías son utilizadas para la indexación de sitios web.

- **Web Semántica**

  La web semántica es un área en gran evolución. La confluencia de la Inteligencia Artificial y las tecnologías web proponen nuevas técnicas y paradigmas para la representación del conocimiento que faciliten localizar, compartir e integrar recursos a través de la WWW [BLHL01]. Estas nuevas técnicas se basan en la introducción de conocimiento semántico explícito que describa y/o estructure la información y los servicios disponibles. La web semántica mantiene los principios originales de la web: descentralización, información compartida, compatibilidad, apertura al crecimiento y uso no previstos de antemano. En este contexto un problema clave es alcanzar un entendimiento entre los diversos actores: usuarios, desarrolladores y programas. La web semántica rescata la noción de ontología del campo de la Inteligencia Artificial, como vehículo para cumplir este objetivo [Gru93].

  En la web semántica la ontología se ve como una taxonomía de conceptos con atributos y relaciones que proporciona un vocabulario consensuado para definir redes semánticas de unidades de información interrelacionadas. Durante los últimos años se han desarrollado diversos lenguajes y estándares para la definición de ontologías, el más recientemente OWL [DCH+04] respaldado y promovido por el consorcio W3C, uno de los principales promotores de la web semántica.

2.5. Características deseables en una ontología

Al diseñar una ontología se deben tomar decisiones de diseño, esas decisiones se basan en que la ontología final debe tener ciertas características.

Según Gruber [Gru93] una ontología debe tener claridad, debe ser completa, coherente, extensible, poseer el mínimo compromiso ontológico. A las características anteriores se le agregaron con el tiempo: principio de diferenciación ontológica, modularidad,

estandarización de nombres, minimización de la distancia semántica entre conceptos hermanos. A continuación se explicarán brevemente cada una de las características antes mencionadas:

- **Claridad**

  Una ontología debe comunicar efectivamente el significado propuesto para los términos definidos en ella. Las definiciones deben entonces ser tan objetivas como sea posible.

- **Completitud**

  Siempre que sea posible debe proporcionarse una definición completa, definir no sólo condiciones necesarias si no también condiciones suficientes. Todas las definiciones deben estar documentadas con lenguaje natural.

- **Coherencia**

  Una ontología debe ser coherente, debe concluir solamente inferencias que sean consistentes con las definiciones que contiene la propia ontología. En definitiva sus axiomas deben ser lógicamente consistentes. La coherencia debe también alcanzar a los conceptos que son definidos informalmente, tales como todo lo que es descripto en lenguaje natural y en ejemplos que se proporcionan para clarificar ideas. Si una sentencia, que puede ser inferida de los axionas de la ontología, contradice la definición de un ejemplo presentado informalmente entonces se debe considerar a la ontología como inconsistente.

- **Extensibilidad**

  Una ontología debe soportar la definición de nuevos términos basándose en el vocabulario existente de manera tal que no requiera la revisión de las definiciones existentes. Esto se logra manteniendo cierto equilibro entre ser lo "suficientemente específico" en la definición como para permitir el uso de la ontología que se construye, pero no demasiado como para que la ontología sea de poca utilidad para otros usos futuros.

- **Mínimo compromiso ontológico**

  Una ontología debe hacer la menor cantidad de aseveraciones posibles acerca del mundo que está siendo modelado, permitiendo de esta forma que las partes que están comprometidas con la ontología (los diferentes agentes que la usarán) tengan libertad de especializar e instanciar la ontología cuando sea necesario.

- **Principio de diferenciación ontológica**

  Las clases en una ontología deben ser disjuntas con el objetivo de obtener una mejor fundamentación estructural que colaborará en posibilitar y favorecer su uso futuro.

- **Modularidad**

  Al definir una ontología se hacen definiciones de diferentes elementos como: clases, relaciones y axiomas, tales definiciones se pueden agrupar en teorías que reúnen los objetos más relacionados de una ontología.

- **Minimización de la distancia semántica entre conceptos hermanos**

  Conceptos similares se deben agrupar y representar como subclases de una clase y deben ser definidos usando las mismas primitivas, mientras conceptos que son menos similares deben estar representados aparte en la jerarquía.

- **Estandarización de nombres**

  Una característica deseable para ayudar en el mantenimiento de una ontología es la estandarización de nombres, definiendo y respetando las reglas para su formación. Una posible estandarización es especificar el nombre de una relación como la concatenación del nombre de la ontología (o el del concepto que es primer elemento de la relación) con el nombre simple de la relación y con el nombre del concepto destino.

2.6. Lenguajes de especificación e implementación

Las ontologías son teorías formales acerca de un universo de discurso y requieren de un lenguaje lógico formal para ser expresadas. En el área de la inteligencia artificial se han desarrollado muchos lenguajes para tal fin, algunos basados en lógica de predicados de primer orden, como KIF [Gin91] y Cycl que proveen poderosas primitivas de modelado (en particular CycL con varios cuantificadores) y la posibilidad de definir fórmulas en términos de otras fórmulas. Hay lenguajes orientados a ser robustos en el razonamiento que provee *lógica descriptiva*, como Loom [Pow91] y Classic [Pow91]. Otros lenguajes son basados en *frames* [KLW90], con más poder expresivo pero menor capacidad de inferencia como Ontolingua, Frame Logic. La herramienta Protégé-2000 utiliza un lenguaje basado en *frames* y extiende la definición de ontologías por medio del lenguaje de axiomas de Protégé (PAL) [pal00].

La elección de un lenguaje de especificación depende del proyecto donde se quiera utilizar la ontología. Si se necesita que el proyecto cuente con mecanismos de inferencia y razonamiento potentes los lenguajes basados en lógica de predicados o en la lógica descriptiva son mas óptimos. En cambio los lenguajes basados en *frames* tiene mayor poder expresivo y son más adecuados cuando las personas que desarrollan la ontología no tienen los conocimientos previos.

Este trabajo se concentra en el lenguaje de especificación definido en el contexto de la herramienta Protégé-2000; un lenguaje propio basado en *frames*, que es extendido por PAL para que su poder de inferencia sea potente. A continuación se describe en forma breve en que consiste un lenguaje basado en *frame*.

2.6.1. Lenguajes basados en *Frames*

Un *frame* es una estructura de datos usada para representar un objeto o una clase o un concepto. Algunos sistemas definen un único tipo de *frame*, otros sin embargo distinguen entre dos o más tipos de *frames* como *frames* de clases (*class frames*) y *frames* de instancias (*instance frame*). Los *frame* típicamente se agrupan formando una jerarquía taxonómica, en la cual cada *frame* es relacionado con un *frame padre*. Si A es el padre de un *frame* B, A representa un concepto más general que B y los *hijos* de B representan un concepto más especifico que B. Una colección de *frame* es una o más jerarquías taxonómicas en una base de conocimiento (KB).

Cada *frame* tiene asociado un conjunto de componentes llamadas *slots*. Los *slots* de un *frame* describen atributos o propiedades que distinguen al *frame*, y pueden describir relaciones binarias entre dos *frames*. Los *slots* contienen un conjunto de características: poseen un nombre, un tipo que determina el valor que representan y un conjunto de restricciones de qué valores pueden ser "almacenados" en ellos, estos componentes son llamados *facets*.

El concepto de *herencia* determina que los *slots* asociados a un *frame* se propague hacia abajo en la jerarquía taxonómica. La herencia es una de las herramientas más complejas y poderosas en una base de conocimiento. Cuando un usuario crea un nuevo *frame*, éste hereda los *slots* definidos por su padre (o de sus padres si el sistema permite múltiple herencia). Como la información de los *slots* y de los *facets* está disponible en tiempo de ejecución (a diferencia de los lenguajes orientados a objetos como C++), el sistema puede chequear las restricciones impuestas y facilitar la creación de instancias por medio de interfaces de usuarios apropiadas.

La *figura 2.3* muestra como funciona el concepto de herencia del ejemplo planteado en la *figura 2.1*. Las clases son representadas por cuadros que tienen un nombre que los identifica y entre paréntesis aparecen escritos los *slots* (atributos o relaciones) asociados a esa clase, donde los *slots* subrayados son lo que se heredan. La relación $IS-A$ determina la jerarquía taxonómica y en consecuencia cómo se heredan los *slots*. En particular observamos la clase *Persona* y sus *slots* asociados *nombre* y *#teléfono*. La clase *Empleado* posee un *slot propio*: *salario*. Además como *Empleado es-una* ($IS-A$) *Persona* hereda los *slots* *nombre* y *#teléfono*. De similar manera la clase *Director* y *Editor* heredan los *slots* de *Empleado* y en particular podemos observar la múltiple herencia en la clase *Editor* que también tiene asociado el *slot*: *nombre – autor*, heredado de la clase *Autor*.

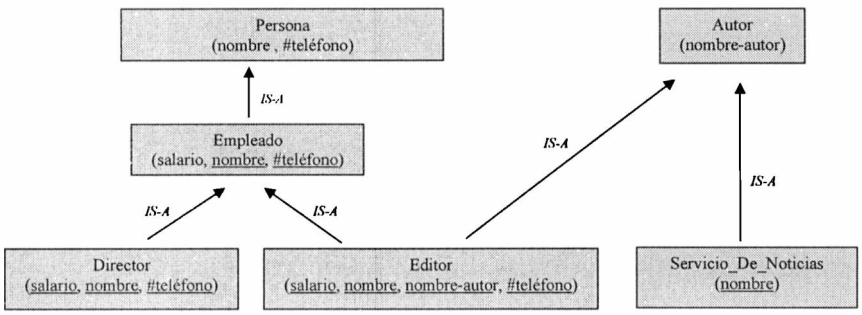


Figura 2.3: Ontología simple, Herencia

Existen problemas asociados a la múltiple herencia, por ejemplo conflictos de nombres en los *slots* que se heredan, cada sistema define como manejarlos.

Los lenguajes que se basan en *frames* proveen un conjunto bastante rico de primitivas, pero imponen fuertes restricciones sintácticas en cómo esas primitivas pueden ser combinadas y cómo pueden ser usadas para definir una clase.

2.7. Herramientas para construir y mantener ontologías

Si bien no existe un consenso de cuales deben ser las características de los editores de ontologías, hay una primera aproximación descripta por Mike Brown y Alan Flett en *"Enterprise-standard ontology environments"* [FB] que determinan que un editor de ontología universal debe contar con las siguientes características :

- **Escalabilidad:** relativa a la facilidad para adquirir a gran escala instancias y relaciones, permitir que múltiples usuarios puedan leer y escribir elementos ontológicos.
- **Adquisición de instancias y axiomas:** la herramienta debe comprobar la consistencia de los tipos, axiomas e inconsistencias. Tender a la adquisición automática de instancias y desarrollar un creador de axiomas gráfico.
- **Visualización:** buscar la claridad de visualización de los diferentes términos, sobre todo en la taxonomía, proporcionar información de contexto cuando sea necesario.
- **Modularización:** controlar las versiones de las ontologías, promover el *merge* e integración, inclusión de entornos visuales.
- **Transparencia del razonamiento:** los motores de inferencia deben ser ajustables, deben tener un depurador de errores y deben tener *tests* de eficiencia.

- **Acceso y características similares a las bases de datos:** el conocimiento debe estar *on-line*, disponible para ser consultado por varios agentes y debe tener las características usuales de gestión de bases de datos.
- **Competencia:** la herramienta deberá realizar *tests* de validación y verificación sobre los términos incluidos en la ontología.

Existen varias herramientas para la construcción de ontologías, en el artículo "*WonderTools?: a comparative study of ontological engineering tools*"[DSW+00], se menciona que "el desafío más importante en el proceso de construcción de ontologías de manera compartida está más en el proceso social de **acordar** el significado de algo", que en la tecnología de representación, por eso las herramientas de software que permiten realizar un trabajo colaborativo entre varios expertos simultáneamente tienen gran relevancia.

2.8. Protégé-2000

Protégé es un ejemplo de herramienta que permite la construcción y mantenimiento de ontologías. Comenzó como una pequeña aplicación diseñada para el dominio de la medicina y evolucionó en una herramienta de propósito más general. Recientemente se ha desarrollado entorno a Protégé una comunidad de usuarios, quienes agregaron mayores capacidades. El crecimiento en el ámbito de la investigación, se ve reflejado en la organización de conferencias[1] y cursos propios de la comunidad Protégé. La evolución de Protégé puede verse en detalle en "*The Evolution of Protégé: An Environment for Knowledge-Based Systems Development*" [GMF+03] y dió lugar a la versión actual: Protégé-2000[2].

Protégé-2000 es una herramienta de software integrada usada para desarrollar sistemas basados en conocimientos[3]. Las aplicaciones desarrolladas con Protégé-2000 son usadas en la resolución de problemas y para la toma de decisiones sobre un dominio particular.

La formalización del conocimiento adoptada por Protégé-2000, se basa en la definición de ontologías, y el lenguaje usado para la representación de esas ontologías es un lenguaje propio basado en *frames*, que utiliza los conceptos estándar: *clases*, *instancias*, *slots* y *facets*. El lenguaje de representación del conocimiento propio de Protégé-2000 dá facilidades para trabajar simultáneamente con clases e instancias. Una instancia singular puede ser usada en el nivel de una definición de clases, y una clase puede ser

[1] La más conferencia más reciente al momento de escribir el trabajo fue: *8th International Protégé Conference - July of 2005 in Madrid, Spain*

[2] Última versión disponible en el momento que se escribir este trabajo, es la versión Protégé-2000 3.2 publicada el 1 de diciembre del 2005

[3] Un sistema basado en conocimiento es un sistema de cómputo que incluye una base conocimiento acerca de un dominio y programas que incluyen reglas para tratar el conocimiento y para solucionar problemas referente al dominio.

almacenada como una instancia. De modo semejante, los *slots*, son elevados al mismo nivel que las clases. El lenguaje permite múltiple herencia, y como cada *frame* posee un nombre único, en particular los *slots* poseen un nombre único, se simplifican los problemas acarreados con la múltiple herencia, ya que si una clase C hereda el *slot* s por más de un camino, como el s referencia al mismo *slot*, se adjunta una sola vez s a C.

El modelo de representación del conocimiento, implementado en Protégé-2000, facilita la conformidad con el protocolo OKBC[4] (Open Knowledge Base Conectivity). Permite ser extendido, actualmente existe una extensión que da soporte a *Web Ontology Language*(OWL).

2.8.1. Lenguaje de representación del conocimiento

El lenguaje para representar el conocimiento en Protégé-2000 está basado en *frames* (sección 2.6.1). Las ontologías definidas dentro de Protégé-2000 consisten en un conjunto de clases, *slots*, *facets*, axiomas e instancias; estos elementos son los *frames* correspondientes al nivel conceptual de la ontología y se detallan a continuación:

- **Clases**: son conceptos del dominio de discusión, en Protégé-2000 forman una jerarquía taxonómica, es decir un árbol con raíz en una clase especial llamada *:THING*, donde la relación entre un nodo A y un nodo hijo B es "A es superclase de B" o también puede verse como la relación "B es subclase de A" (éstas son relaciones inversas entre sí). Protégé-2000 soporta múltiple herencia, es decir una clase puede tener una o más superclases. Una clase es la representación abstracta de un concepto de un dominio particular y puede tener un conjunto de *slots* que representan atributos de la clase o relaciones con otros conceptos. Las clases a su vez son instancias de otras clases llamadas meta-clases. Una meta-clase es una clase donde sus instancias son clases.

  La jerarquía de clases es visualizada en forma de árbol, y seleccionando la clase particular se pueden ver sus superclases. La *figura 2.4* muestra la jerarquía de clases del proyecto *newspaper* (1) y las superclases de la clase *Editor* (2)

- *Slots*: describen las propiedades o atributos de una clase e instancias, un *slot* es un *frame* y es tratado como elemento de primera clase. Los *slots* se definen independientemente de la clase. Cuando un *slot* es insertado a un *frame* en la ontología del usuario, éste describe las propiedades particulares de ese *frame*, y puede contener valores. Los *slots* puede ser insertados a un *frame* de dos formas: como un ***template slot*** o como un ***own slot***.

[4]OKBC define una interfaz de programación para aplicaciones que trabajan con sistemas de representación de conocimiento[CFF+98]

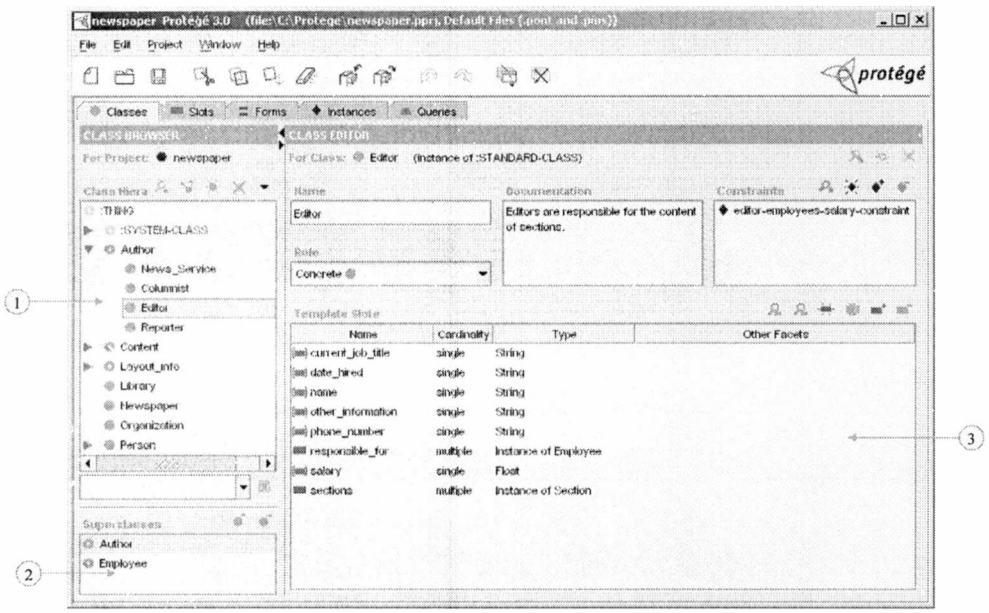


Figura 2.4: Protégé-2000, Jerarquía de clases

- ***template slots*** describen las propiedades de las instancias de una clase y se reproducen para cada una de esas instancias. Además una clase hereda de sus superclases el conjunto de *template slots* y pasan a ser *template slots* propios.
- ***own slots*** describen la propiedad de un *frame* en sí mismo, en vez de propiedades de las instancias de ese *frame*. Los *own slots* no se reproducen ni para las instancias propias, ni son heredados por las subclases del *frame*. Por ejemplo el *slot* : $NAME$ no es reproducido en las instancias de ninguna clase.

Una instancia individual puede adquirir *slots* sólo siendo una instancia de una clase que tiene definido esos *slots* como *template slot*. Una clase puede adquirir *slots* sólo siendo una instancia de una metaclase (sección 2.8.2) que tiene esos *slots* como *own slot*. En la *figura 2.4* se observan los *template slot* (3) de la clase *Editor*

- **Facets**: describen propiedades de los *slots*, son una forma de especificar restricciones sobre los valores que pueden tomar los *slot*. En Protégé-2000 tenemos las siguientes propiedades de los *slots*:
  - *Name*: describe el nombre de un *slot*
  - *Type*: determina el tipo de valores que un *slot* puede soportar. Los tipos disponibles están resumidos en la siguiente tabla:

Tipo	Descripción	Ejemplos
Any	cualquier tipo	
Boolean	valor lógico	True, False
Class	una clase en la KB	Persona
Float	número decimal	1.0, 3.4e10, -0.3e-3
Instance	instancia de una clase en la KB	*instance_00010*
Integer	número entero	1, 2, -4
String	secuencia de caracteres alfanuméricos (puede incluir espacios)	"Juan Perez"
Symbol	secuencia de caracteres alfanuméricos (no puede incluir espacios)	rojo, azul y verde

- *Cardinality*: permite especificar el número de valores permitidos o requeridos para el *slot* combinando las propiedades *required* (el valor es requerido), *at least* (al menos n valores), *multiple* (acepta múltiples valores) y *at most* (como máximo m valores).
- *Minimun*: es opcional y sólo se aplica a los tipos *Integer* y *Float*. Permite especificar un valor mínimo para el *slot* (por ejemplo, un mínimo en cero, quiere decir que las instancias no pueden tener valores negativos).
- *Maximum*: es opcional y sólo se aplica a los tipos *Integer* y *Float*. Permite especificar un número máximo para el valor del *slot*. Conjuntamente, *Minimum* y *Maximum* pueden estar *seteados* para definir un rango de valores.
- *Documentation*: permite dar una descripción textual del *slot*. Completar este campo es opcional, pero recomendado.
- *Inverse Slot*: es opcional y sólo está disponible para los *slots* de tipo *Class* o *Instance*. Le permite crear una relación recíproca entre dos *slots* (por ejemplo las relaciones "superclase de" y "subclase de", son relaciones recíprocas).
- *Template Values* : es opcional y permite especificar el valor o los valores por defecto para un *slot* a nivel de clases. No permite ser modificado por las subclases.
- *Defaults*: permite especificar el valor o los valores por defecto para un *slot* (es opcional).

Cuando el *slot* tiene el tipo *Class*, *Instance* o *Symbol* se le agrega una nueva propiedad: *Allowed Parents* , *Allowed Classes* o *Allowed Values* respectivamente, que permiten restringir los valores que se pueden usar. La *figura 2.5* muestra la pantalla de edición de las propiedades (*facets*) de un *slot* "es-escrito".

Los *slots* que están adheridos a las clases pueden modificar algunos de sus *facets* sin que esto involucre modificar las propiedades del *slot* en otros lugares. Las

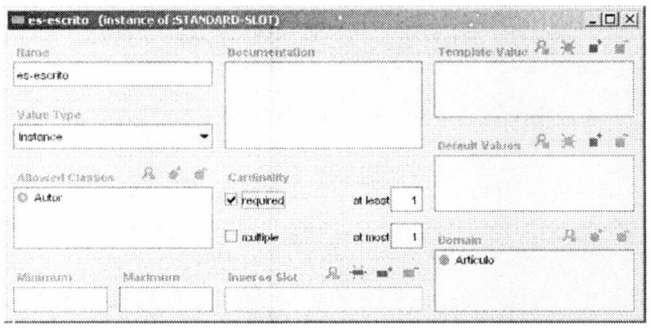


Figura 2.5: Propiedades de los *Slots*

modificaciones son especializaciones del *slot*, por ejemplo si un *slot* tiene como valor *maximun* a 10, la modificación a nivel clase sólo puede modificar *maximun* por un valor menor a 10. La diferenciación es dada por lo que se denomina *Slot Top-Level* y *Slot at a Class* para representar los *slots* definidos en forma independiente y los que son especializados para una clase respectivamente.

- **axiomas** declaran restricciones adicionales. En Protégé-2000 la declaración de axiomas está soportada por medio del lenguaje de axiomas de Protégé (PAL - Protégé Axiom Language). PAL está basado en la lógica de primer orden y las restricciones se modelan por medio de *frames* de propósito especial, de tal manera que puedan formar parte de la base de conocimiento. No se entrara en detalle sobre el PAL, a modo de ejemplo se ve una restricción que determina que "no puede haber editores con menores salarios que los empleados que es responsable".

```
(defrange ?editor :FRAME Editor)
(defrange ?employee :FRAME Employee responsible_for)

(forall ?editor (forall ?employee
    (=> (and
    (responsible_for ?editor ?employee)
    (own-slot-not-null salary ?editor)
    (own-slot-not-null salary ?employee))
(> (salary ?editor) (salary ?employee)))))
```

- **instancias** las instancias en Protégé-2000 son ocurrencias concretas de información acerca de un dominio que forma parte de una base de conocimiento. Las instancias tienen un tipo, que es determinado por la clase a la que pertenecen.

2.8.2. Meta-classes Protégé

Protégé-2000 soporta un tipo de reflexión, en la misma dirección que algunos lenguajes de programación orientados a objetos, excepto que en Protégé-2000 la reflexión está dada en el sistema de representación de conocimiento y basada en el concepto de meta-clase.

Una meta-clase M, es un *template* para definir clases que son instancias M, la meta-clase describe qué propiedades tendrán las clases, por ejemplo un nombre, un conjunto de *template slots*, o determinar si la clase es abstracta[5].

En Protégé-2000 por defecto las clases son instancias de la meta-clase *:STANDARD-CLASS*, y siguiendo el camino ascendente en la jeraquía de clases tenemos que las superclases son: *:CLASS*, *:META-CLASS*, *:SYSTEM-CLASS*, :THING, como se ve en la *figura 2.6*.

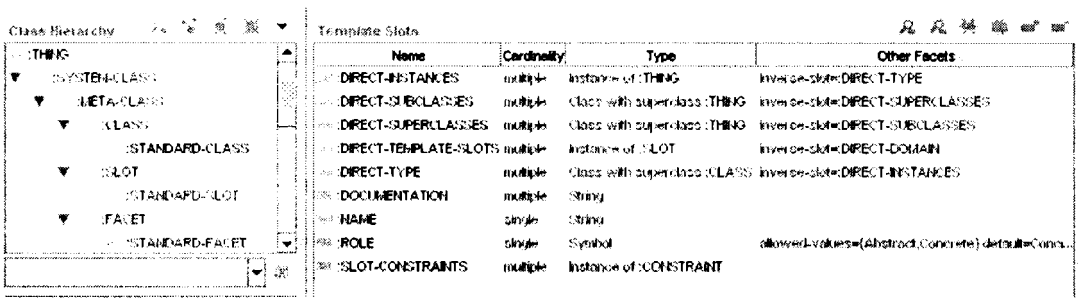


Figura 2.6: Jerarquía de meta-clases

La arquitectura de meta-clases de Protégé-2000 permite tratar a las clases y a las instancias de manera similar, facilitando la adquisición y el proceso de edición de la ontología. Trabajar en la definición de las meta-ontologías o en el nivel de la ontología concreta es similar. Los usuarios pueden extender el meta-modelo subclasificando las meta-clases y definiendo (si es necesario) los *own slots* para esas meta-clases. Este mecanismo es aplicado para definir un nuevo meta-modelo en CO-Protégé dando un marco de soporte a la edición colaborativa de ontologías (sección 5.4)

Protégé-2000 define las clases *:CLASS*, *:SLOT* y *:FACET* para representar de forma abstracta el modelo de un sistema basado en *frames* y para representar el modelo concreto de conocimiento se utilizan tres subclases de las anteriores *:STANDARD-CLASS*, *:STANDARD-SLOT* y *:STANDARD-FACET*.

- ***:STANDARD-CLASS***: define la meta-clase por defectos para las clases. Los *template slots* de esta clase definen los *own slots* estándar para las clases, es decir las propiedades de las clases y son los siguientes:

[5]en Protégé-2000 una clase abstracta es aquella que no puede tener instancias

Nombre del *slot*	Descripción
:DIRECT-INSTANCES	el conjunto de instancias.
:DIRECT-SUBCLASSES	el conjunto de subclases.
:DIRECT-SUPERCLASSES	el conjunto de las superclases (por defecto *:THING*).
:DIRECT-TEMPLATE-SLOTS	el conjunto de los *templates slot*
:DIRECT-TYPE	determina quien es la metaclase (por defecto *:STANDARD-CLASS*).
:DOCUMENTATION	permite a los usuarios agregar.
:NAME	nombre de la clase.
:ROLE	determina si la clase es abstracta o no.
:SLOT-CONSTRAINTS	un conjunto de instancias de *:CONSTRAINTS*.

Usualmente cuando los usuarios quieren definir nuevas meta-clases, subclasifican *:STANDARD-CLASS*.

- ***:STANDARD-SLOT***: define la meta-clase para representar los *slots*. Los *templates slots* de esta clase, representan las propiedades de los *slots* de forma similar que *:STANDARD-CLASS*, agregando *templates slots* para contener los *facets* descriptos en la sección 2.8.1. La *figura 2.7* muestra los *templates slots* de la clase *:STANDARD-SLOT*.

Name	Cardinality	Type	Other Facets
:ASSOCIATED-FACET	single	Instance of :FACET	inverse-slot=:ASSOCIATED-SLOT
:DIRECT-DOMAIN	multiple	Instance of :CLASS	inverse-slot=:DIRECT-TEMPLATE-SLOTS
:DIRECT-SUBSLOTS	multiple	Instance of :SLOT	inverse-slot=:DIRECT-SUPERSLOTS
:DIRECT-SUPERSLOTS	multiple	Instance of :SLOT	inverse-slot=:DIRECT-SUBSLOTS
:DIRECT-TYPE	multiple	Class with superclass :SLOT	inverse-slot=:DIRECT-INSTANCES
:DOCUMENTATION	multiple	String	
:NAME	single	String	
:SLOT-CONSTRAINTS	multiple	Instance of :CONSTRAINT	
:SLOT-DEFAULTS	multiple	Any	
:SLOT-INVERSE	single	Instance of :SLOT	inverse-slot=:SLOT-INVERSE
:SLOT-MAXIMUM-CARDINALITY	single	Integer	default=1
:SLOT-MINIMUM-CARDINALITY	single	Integer	
:SLOT-NUMERIC-MAXIMUM	single	Float	
:SLOT-NUMERIC-MINIMUM	single	Float	
:SLOT-VALUE-TYPE	multiple	Any	default=String
:SLOT-VALUES	multiple	Any	

Figura 2.7: *Template Slots* de la clase *:STANDARD-SLOT*

Los usuarios definen nuevos meta-*slots* subclasificando *:STANDARD-SLOT*. Como última incorporación Protégé-2000 permite armar una taxonomía de *slots* definiendo los *superslots* para un *slot*.

- ***:STANDARD-FACET***: define la clase que representa a los *facets* definidos por el usuario. Es subclase de *:META-CLASS* y no tiene *slots* propios. Los *facets* descriptos anteriormente no son modelados con esta clase.

La arquitectura meta-clases en Protégé-2000 permite a usuarios adaptar y cambiar el modelo de conocimiento del sistema para satisfacer las exigencias de su dominio. Esta es una de las características fundamentales por la cual se decidió usar Protégé-2000 para la construcción del prototipo CO-Protégé. Se puede especializar *:STANDARD-CLASS* para adaptar el meta-modelo que se define en CO-Protégé para incorporar los conceptos relacionados con la edición de colaborativa de ontologías.

2.9. Metodologías de diseño

El diseño de una ontología no es trivial. Se necesita un acuerdo de múltiples personas para adoptar una ontología particular sobre el dominio de interés, este acuerdo se denomina compromiso ontológico. Aún cuando los participantes no tengan las mismas experiencias, teorías, o percepciones sobre el dominio a modelar, es esperable que todos participen en el compromiso ontológico.

En la publicación *A collaborative approach to ontology design* [HJ02] se menciona que "el desarrollo de una ontología debería estar enfocada para asegurar que sus usuarios potenciales encontrarán sus caracterizaciones del dominio, siendo éstas completas, correctas, claras y concisas. En consecuencia el compromiso ontológico debería ser un aspecto integral de la ingeniería ontológica."

La ingeniería de ontologías es la rama de la informática que estudia lo concerniente al diseño, modificación, aplicación y evaluación de las ontologías. Actualmente es una disciplina en crecimiento, y entorno a las metodologías de diseño ontológico vislumbra cinco perfiles para su desarrollo: inspiración, inducción, deducción, síntesis y colaboración. Estas metodologías pueden ser usadas tanto en el diseño inicial de una ontología como en la modificación o ampliación de un diseño preexistente . A continuación se describen los cinco acercamientos y cuáles son las bases que utilizan cada uno para el diseño:

- **Inspiración.** Se basa en puntos de vista individuales, un desarrollador comienza de una premisa sobre por qué debe construir la ontología. Usando la imaginación individual, la creatividad, y un punto de vista personal sobre el dominio de interés, se pone a diseñar una ontología que apunta a resolver las necesidad planteadas.

- **Inducción.** Especifica casos del dominio, la ontología es desarrollada por observación, examinación y por el análisis de casos específicos del dominio de interés; la caracterización resultante es una ontología para un caso específico que luego es aplicada a otros casos en el mismo dominio.

- **Deducción.** Se concentra en la adopción de principios generales y luego los adapta para construir una ontología que apunta a un caso específico, esto implica la filtración y la destilación de las nociones generales, personalizándolas a un subconjunto de dominio particular, funciona de forma inversa que el método inductivo.

- **Síntesis.** Se toma un conjunto de ontologías, cada una provee una parte de la caracterización del dominio. Sintetizar una ontología única del conjunto seleccionado implica la integración sistemática de los conceptos de cada unos, eliminando las caracterizaciones más incompletas a favor de las que se encuentran totalmente desarrolladas, y es necesario unificar terminologías diferentes.

- **Colaboración** Múltiples individuos acercan sus puntos de vistas sobre el dominio. Con un acercamiento de colaboración al diseño de ontología, el desarrollo es un esfuerzo conjunto que refleja experiencias y puntos de vista de las personas que intencionadamente cooperan para producirlo. Esto ayuda a reducir "puntos oscuros" en la ontología y enriquecer su contenido. Como contraposición la coordinación del proceso de diseño puede resultar muy compleja, si demasiadas personas están directamente implicadas. Para ejecutar un acercamiento de colaboración, es necesario establecer y emplear mecanismos para determinar acuerdos generales.

El acercamiento por inspiración puede ser refutado por la carencia de un sostén teórico y puede ser poco práctico. Sin embargo, este acercamiento también puede generar ontologías únicas e innovadoras. Una ontología diseñada vía el acercamiento de inducción puede encajar en un caso específico, pero puede no ser generalizable. El acercamiento deductivo presupone la existencia y la selección de un esquema apropiado de caracterizaciones generales de lo cual extraer una ontología para un caso específico. El acercamiento sintético implícitamente cubre las tres primeras metodologías, el conjunto de las ontologías que intervienen pueden ser creadas con alguna o todas las metodologías anteriores, este acercamiento es interpretativo por naturaleza, y está ligado fuertemente a la habilidad que se tenga en sintetizar (unificar) las diferentes ontologías.

Ninguno de los cuatro primeros acercamientos tiene la facilidad de contener métodos de evaluación propios que determinen la calidad o aceptabilidad de la ontología que dan por resultado. En cambio, el acercamiento de colaboración confía en evaluaciones y correcciones de todos los miembros y tiende a construir un consenso ya que iterativamente va reduciendo las objeciones de los participantes. Este acercamiento confía en la naturaleza de los participantes, el grado de su participación, y habilidades en la supervisión del proceso de colaboración.

El enfoque colaborativo es el más propicio para lograr el compromiso ontológico deseado y es el fundamento de este trabajo. La mayoría de las herramientas de software analizadas para el diseño de ontologías permiten la construcción en forma colaborativa, pero carecen de la posibilidad de manejar en el propio ambiente los diferentes puntos de vista de cada usuario y no soportan procesos de discusión que permitan llegar a un consenso del diseño final. En el capítulo 4 se detallará cómo puede desarrollarse una metodología de trabajo para el diseño de una ontología con un acercamiento netamente colaborativo.

2.10. Conclusión

En este capítulo se mostró la importancia de seguir una metodología para el diseño de ontologías con un acercamiento colaborativo. Con el enfoque colaborativo se mejora el compromiso ontológico de los participantes, aumentando el grado de participación de las personas involucradas. En consecuencia surge la necesidad de tener herramientas de software que den soporte al acercamiento colaborativo.

En la sección 2.8 se explicó el editor de ontologías Protégé-2000, este editor fue elegido para implementar el prototipo CO-Protégé, que da soporte al diseño colaborativo de ontologías permitiendo manejar los diferentes puntos de vista, los conflictos que ellos generan y las discusiones para llegar a un consenso, todo dentro del mismo ambiente.

Capítulo 3

Groupware y Computer-Supported Cooperative Work

Este capítulo se dedica a mostrar las características de los sistemas *Computer Supported Cooperative Work* (CSCW) y qué servicios debe aportar una herramienta groupware para soportarlos. Si bien se abordan los aspectos sobre la colaboración en forma general, se hace incapíe en las características que debe tener una herramienta para la edición colaborativa de ontologías. A tal fin en la sección 3.1 se presentan definiciones y diferencias de los términos Groupware y CSCW. En la sección 3.2 se describen diferentes formas de clasificar a los sistemas groupware, centrándose en la clasificación espacio-tiempo. En la sección 3.3 se verán cuales son los aspectos más importantes que involucran a las herramientas groupware.

3.1. CSCW vs Groupware

Cuando se habla de colaboración, aparecen recurrentemente los términos Groupware y Computer-Supported Cooperative Work (CSCW), que están estrechamente ligados al trabajo entre grupos de personas que colaboran utilizando computadoras. Abordar el tema de trabajo colaborativo implica comprender estos dos términos que generalmente son utilizados de forma indiferente como si fueran sinónimos. Existen diferencias sustanciales entre ellos, mientras que Groupware se refiere a la tecnología diseñada para facilitar el trabajo en grupo, y puede usarse para comunicarse, cooperar, coordinar, solucionar problemas, competir, etc. CSCW en cambio, se refiere al campo de estudio que examina el diseño, la adopción y el uso de groupware. A continuación se detallan estos dos términos.

3.1.1. Computer-Supported Cooperative Work (CSCW)

El término CSCW fue introducido por GRIEF y CASHMAN como: "una vía para describir cómo la tecnología de las computadoras puede ayudar a los usuarios a trabajar juntos en grupos" [Cro94].

CSCW es la disciplina científica que describe cómo desarrollar aplicaciones groupware, teniendo por objetivo el estudio teórico y práctico del comportamiento de las personas cuando trabajan en cooperación y cómo afecta el groupware al comportamiento del grupo. Su objetivo es observar la forma en que las personas interaccionan y colaboran entre sí. Se propone a partir de estas observaciones, presentar líneas de actuación para el desarrollo tecnológico que sirvan para asistir al proceso de comunicación.

El campo de investigación que acapara CSCW es amplio y atrae a personas de diferentes ámbitos, desde el punto de vista de la tecnología atrae a personas interesadas en diseño del software, científicos de sistemas, especialistas en redes, etc. Desde el punto de vista social y organizativo se acercan: psicólogos, antropólogos, gente de comercio, investigadores de comunicaciones, entre otras especialidades.

Aspectos tecnológicos claves del CSCW son el concepto de grupo, la interfaz multiusuario, la coordinación y comunicación dentro del grupo, los espacios de información compartida y el soporte de un entorno abierto heterogéneo que integre aplicaciones preexistentes basadas en un solo usuario.

3.1.2. Groupware

Las aplicaciones groupware suponen "disponer de sistemas basados en computadoras que soportan grupos de usuarios comprometidos en un trabajo común y que proveen una interfaz a un entorno compartido" [EGR91]. Groupware es el hardware y el software que soportan y aumentan el trabajo en grupo. Es un conjunto de productos orientados al trabajo en grupo, es decir ayudan a grupos de personas a trabajar juntas. Otras definiciones de groupware son las siguientes:

- "Proceso de trabajo en grupo que tiende a un objetivo preciso y aplicaciones concebidas para facilitar dicho trabajo" [JL78]
- "Sistema de herramientas lógicas para facilitar la cooperación de las personas en el trabajo" [EL88].
- "Cooperación asistida por computadoras que aumenta el rendimiento de los procesos de comunicación interpersonales" [Col97].
- "Software que ayuda a grupos de personas a comunicarse electrónicamente", [Gol94].

- "Conjunto de métodos, medios y herramientas que permiten a un grupo mejorar la comunicación, la coordinación y la cooperación" [EGR91].

  La definición expresa que el groupware sirve para aumentar la eficacia en tres niveles puntuales:

  - **Comunicación**: proceso de intercambio de información dentro del grupo. Se intenta que sea eficaz; quien envía y quien recibe la información perciban el mismo concepto; y eficiente en cuanto al gasto mínimo de recursos.
  - **Coordinación**: conjunto de mecanismos utilizados para establecer un enlace coherente entre las actividades que desarrolla el grupo.
  - **Cooperación**: participación intencionada y coordinada de los miembros de un grupo.

  Los tres niveles (comunicación, coordinación y cooperación) son conocidos como el modelo de las **3C** del groupware y sin ellos un grupo de personas no puede prosperar en el trabajo conjunto.

3.2. Clasificación Espacio-tiempo

Groupware puede concebirse como ayuda a grupos que trabajan en el mismo lugar (cara a cara) o grupos que están distribuidos en varias locaciones (distribuida). Además la comunicación y colaboración pueden ser realizadas al mismo tiempo (sincrónica) o en tiempos diferentes (asincrónica). Estas consideraciones sugieren cuatro categorías de groupware representadas en una matriz de 2x2 como se muestra en el *cuadro 3.1* y fue realizada por ROBERT JOHANSEN. [JSB+91].

Clasificación espacio/tiempo	Mismo tiempo	Diferente tiempo
Mismo lugar	Interacción cara a cara	Interacción asíncrona
Diferente lugar	Interacción distribuida síncrona	Interacción distribuida asíncrona

Cuadro 3.1: Clasificación espacio-tiempo

Interacción cara a cara: mismo lugar - mismo tiempo

Estos tipos de sistema garantizan la comunicación entre usuarios que se encuentran en el mismo espacio al mismo tiempo, pero que no interactúan directamente, sino a través del medio elegido. Por ejemplo el sistema de cuadro de mandos de un avión (medio de comunicación) que permite la interacción entre piloto y copiloto. Otro ejemplo es la compra-venta de acciones en la bolsa de valores donde el medio elegido para la interacción son las pizarras electrónicas.

Interacción sincrónica distribuida: diferente lugar - mismo tiempo

La interacción entre usuarios en diferente lugar pero al mismo tiempo es uno de los avances tecnológicos que permitieron abrir una variedad de posibilidades, el hecho de que distintas personas en puntos geográficos diferentes puedan colaborar en la solución de un problema al mismo tiempo es uno de los avances más importante dentro del campo colaborativo. Este tipo de sistemas permiten a varios usuarios ubicados en distintas locaciones interactuar al mismo tiempo. Los primeros sistemas que se propusieron fueron los editores distribuidos sincrónicos, el editor GROVE [EGR91] (GRoup Outline Viewing Editor) fue el primero, soporta varios usuarios editando un mismo texto simultáneamente. El problema asociado con estos sistemas es el control de concurrencia, por ejemplo en GROVE dos o más usuarios pueden estar editando la misma línea del documento, es más pueden editar el mismo caracter. Generalmente se asume que cada usuario edita un sitio distinto del archivo y por tanto no hay colisiones, pero no siempre es posible asumir esta característica. Otro aspecto a tener en cuenta es la noción de "presencia", los participantes deben darse cuenta quién o quienes están participando. GROVE incluyen fotos de los participantes que se encuentran trabajando en la edición.

Las investigaciones que surgieron a partir de Grove y otros sistemas similares como rIBIS [RE91] fueron los primeros pasos hacia aplicaciones comerciales, por ejemplo el paquete informático Office de Microsoft permite la edición simultánea de archivos de Microsoft Word y Excel.

Interacción asincrónica: mismo lugar - diferente tiempo

La interacción asincrónica realizada en el mismo lugar pero en diferente tiempo supone la utilización de la misma computadora por parte de distintas personas en tiempos diferentes. Un ejemplo típico es la escritura colaborativa de un texto donde los usuarios se turnan para escribir un documento que siempre se encuentra en la misma computadora. También la utilización de un pizarrón de anuncios es un sistema asincrónico, en este caso sin contar con una computadora. La interacción asincrónica en el mismo lugar es un caso particular de la interacción asincrónica distribuida, en estos casos el control de cambios entre los usuarios es la parte más importante de estos sistemas, de manera que todos los participantes en la actividad colaborativa tengan constancia de los cambios producidos.

Interacción asincrónica distribuida: diferente lugar - diferente tiempo

Este tipo de sistemas presentan menos complicaciones que los anteriores, el más difundido y utilizado es el correo electrónico o e-mail. Su principal problema es que está poco estructurado y además si el número de correos electrónicos recibidos diariamente es elevado (y con abundantes correos "basuras") el sistema se vuelve poco práctico y se hace necesario por lo tanto que haya alguna organización y filtrado de lo recibido.

3.3. Aspectos de un sistema CSCW

Cuando se diseña una herramienta groupware, hay un conjunto de aspectos que se deben tener en cuenta: comunicación entre los miembros del grupo, información compartida, coordinación y control de objetos compartidos, espacio de trabajo compartido y *awareness*. A continuación se explican cada uno de estos aspectos:

- **Comunicación entre los miembros del grupo**

  Los miembros del grupo deben poder comunicarse entre sí. Un miembro debe ser capaz de expresar "información" y que ésta sea receptada por el resto de las personas que están participando.

- **Información compartida**

  El grupo necesita tener un repositorio de información compartida y un conjunto de herramientas que permitan la edición, navegación y extracción de la información, organización y entendimiento común del proceso de trabajo.

- **Coordinación y control de objetos compartidos**

  La coordinación y control de objetos compartidos es importante para evitar inconsistencias y trabajo replicado . Si se produce la modificación de un *objeto* por uno de los miembros del grupo, por ejemplo el cambio de un diagrama, la modificación ha de ser visualizada por el resto de los miembros. Hace falta controlar las actualizaciones simultáneas para no trabajar con información obsoleta o que lleve a inconsistencias, en muchos casos se tiene un control de versiones.

- **Espacio de trabajo compartido**

  Los miembros del grupo necesitan un espacio de trabajo común donde desarrollar sus ideas. En una reunión presencial este papel lo realiza una pizarra. El espacio de trabajo compartido es el punto de encuentro de los participantes, donde trabajan en forma colaborativa.

- **Organización y entendimiento común del proceso de trabajo**

  Las personas que trabajan juntas tienen que saber qué es lo que quieren hacer y cómo. Cómo organizar y hacer que el grupo se entienda es un factor a tener en cuenta para el éxito o el fracaso del trabajo colaborativo. La herramientas tienen que tener una forma clara de cómo llevar a cabo el proceso de trabajo, delineando y ayudando a los usuarios a determinar qué acciones ejecutar.

- **La ayuda a la toma de decisiones**

  Es fundamental para un grupo de personas que trabajan juntas la capacidad que tengan para tomar decisiones. La decisión puede estar relacionada con los objetivos de una tarea común, el método de trabajo que ha de ser adoptado por el grupo, la elección de los miembros del grupo, etc.

- **Awareness**

  Además de la comunicación explícita, como enviar un mensaje o hablarle a alguien, muchas situaciones del trabajo en grupo aprovechan la comunicación implícita, acciones realizadas por algún usuario influye en posibles acciones que tomen otros usuarios. Este aspecto fundamental es comprendido por medio del awareness. Awareness es entendido como *"lo que un usuario necesita comprender de las actividades de los otros participantes, para proveer un contexto a su propia actividad. Este contexto se utiliza para asegurar que las contribuciones individuales son pertinentes para la actividad del grupo como un todo y para evaluar acciones individuales en relación a los objetivos y progresos grupales"*[DB92].

  Las herramientas de awareness pueden ser diseñadas brindando información acerca del ambiente y dejando a un usuario conocer información del resto de los participantes. Un usuario puede determinar a quién informarle de sus actividades, pero además un usuario puede decidir de qué quiere ser notificado.

3.4. Características deseadas de una herramienta groupware para el desarrollo colaborativo de ontologías

Teniendo en cuenta los puntos explicados en este capítulo se determino que la herramienta groupware que permita el diseño colaborativo de ontologías soporte los siguientes aspectos puntuales CSCW:

- En primer lugar en el contexto de la clasificación espacial, la herramienta debe proveer una interacción sincrónica distribuida (diferente lugar - mismo tiempo).
- proveer un espacio de trabajo compartido.
- coordinación y control de objetos compartidos.

- comunicación entre los participantes.
- mecanismos de awareness.

Como se explicó en el capítulo, una de las características que se desea de la herramienta, es el soporte los diferentes puntos de vista, esto introduce la necesidad de tener que consensuar puntos de vista contradictorios, es deseable por lo tanto que la herramienta de algún tipo de mecanismo para soportar la discusión, esto se puede realizar siguiendo los lineamientos generales de algún modelo básico de discusión estructurada, como el planteado por de modelo IBIS [KR70] que se describe en el *Apéndice C.*

En la introducción de este trabajo de grado se mencionó el diseño de la herramienta CO-Protégé, un prototipo de herramienta groupware para diseñar colaborativamente ontologías. El prototipo CO-Protégé cuenta con las características citadas anteriormente; en el resto de éste trabajo se volverán a abordar cada una de ellas explicando como son implementadas en CO-Protégé.

Capítulo 4

Diseño colaborativo de ontologías

CSCW y aplicaciones groupware se han vuelto populares en diversas áreas, particularmente en el diseño de ontologías, donde el interés se centra en las actividades que un grupo de personas realizan para llevar a cabo el diseño final de la ontología de dominio que quieren representar.

El grupo de personas que diseñan una ontología en forma colaborativa, intentan encontrar una perspectiva única del dominio que quieren modelar. La naturaleza del diseño colaborativo implica que surjan visiones diferentes por parte de los diseñadores, estas diferencias deben ser superadas para llegar a la visión común. Este trabajo de grado plantea el diseño de una ontología en forma colaborativa desde una perspectiva particular, basada en tres pilares:

1. **Una metodología para el diseño colaborativo de ontologías.**

   La metodología para el diseño colaborativo de ontología se basa en un proceso de diseño más general que involucra cuatros pasos: *externalización*, *publicación*, *internalización* y *reacción* [DC04]. Las personas utilizan el proceso en forma metódica para realizar el diseño. La externalización es el punto inicial: una persona realiza un diseño como una iniciativa individual en un ámbito privado, luego da a conocer su diseño al resto del grupo por medio de la publicación. La internalización se realiza cuando otro usuario comprende los nuevos elementos de diseño publicados y en consecuencia puede reaccionar. Las reacciones implican una modificación en el diseño compartido ya sea para extender, modificar o simplemente expresar una posición sobre el diseño anterior.

2. **Soporte de diferentes puntos de vista.**

   Como en cualquier actividad de diseño colaborativo surgen diferencias, los usuarios plantean disconformidades según sus propios puntos de vista. Las disconformidades son vistas como conflictos en el diseño y generalmente son tratados fuera del sistema que utilizan para el diseño. El nuevo enfoque propone como segundo pilar tratar a los

diferentes puntos de vista dentro del sistema, para tal fin los participantes del diseño colaborativo pueden generar *conflictos*, definir *alternativas* y dar *argumentaciones*. Estos elementos además permiten generar un modelo de discusión inspirado en el modelo IBIS (sección 4.3.1)

3. **Mecanismos de *awareness*.**

   Asegurar que el grupo tenga conocimiento de lo que está sucediendo, es una forma de mantener la actividad de diseño en "acción". Esto se logra por medio de mecanismos particulares de *awareness*, que permitan notificar al grupo de los cambios en el diseño. Las notificaciones son generadas en base a las acciones que los usuarios realizan.

Para dar soporte al nuevo enfoque en el diseño colaborativo de una ontología se implementó el prototipo CO-Protégé, un conjunto de *plug-ins* que extienden Protégé-2000. A modo de ejemplificar y empezar a ver la herramienta CO-Protégé, se introducirá en este capítulo los aspectos funcionales del prototipo dejando para el próximo capítulo un estudio más detallado.

En la sección 4.1 se plantea, a modo de ayuda para la compresión de los conceptos introducidos, un escenario que servirá como ejemplo a lo largo del capítulo. En la sección 4.2 se describe la metodología utilizada para el diseño, teniendo en cuenta el proceso de diseño general y su especialización a ontologías. Los diferentes puntos de vistas que poseen los usuarios sobre conceptos del dominio, implican que surjan divergencias en el dominio modelado, en la sección 4.3 se explica como son tratadas las divergencias, siempre dentro del sistema que se utiliza para realizar el diseño. En la sección 4.4 se describe el tercer pilar para el desarrollo colaborativo de ontologías, el *awareness*. Finalmente en la sección 4.5 se dan las conclusiones del capítulo.

4.1. Un escenario de Ejemplo

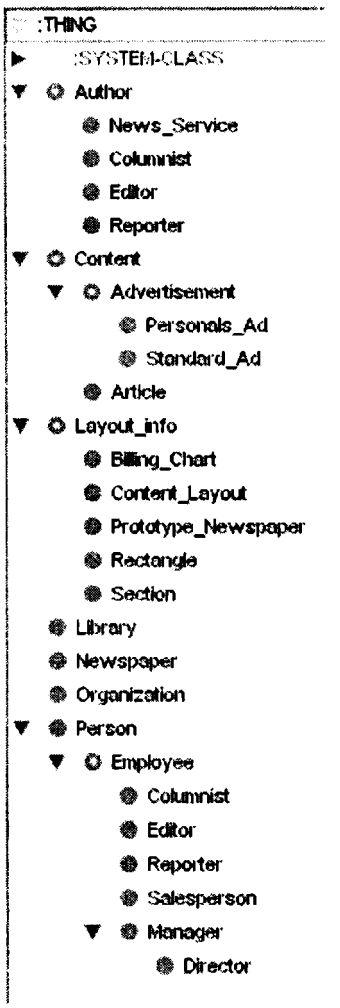
La idea de esta sección es plantear un escenario de ejemplo para el desarrollo de una ontología compartida, que sirva para clarificar varios de los items que se describen en este capítulo. El ejemplo elegido es la ontología *newspaper* provista en Protégé-2000, si bien ya es una ontología diseñada y simple, contiene todos los elementos necesarios para desarrollar un ejemplo. Fue elegida ya que también es utilizada en el manual de usuario de Protégé-2000 permitiendo al lector seguir los aspectos no colaborativos de la edición de ontologías con un hilo de continuidad.

Antes de entrar en la descripción del dominio elegido como ejemplo, necesitamos definir al grupo de personas que lo desarrollarán; tenemos un conjunto de participantes (miembros del proyecto), entre los que se encuentran *Alicia*, *Guille* y *Esteban* que modelan "la publicación" de un periódico. A continuación se da una breve descripción del dominio y se muestra la jerarquía de clase de la ontología *newspaper*:

La ontología *newspaper* incluye:

- La lista de los artículos publicados, indicando cuándo fue publicado, en que sección, quién o quienes fueron los autores, etc.
- Información sobre las secciones estándar del periódico (Deportes, Negocio, etc.)
- Información sobre las publicidades que contiene cada publicación.
- Información de los empleados, conteniendo su sueldo, de quién dependen, etc.
- La lista de los autores, diferenciados según su categoría.

La figura de la derecha muestra la jerarquía de clase de la ontología *newspaper*. La descripción completa de la ontología *newspaper*, que incluye la descripción de las clases y las relaciones (*slots*) entre las mismas, se muestra en el apéndice B.



4.2. Metodología de diseño colaborativo

La metodología utilizada para el diseño colaborativo de ontologías, planteada en este trabajo, se basa en un proceso de edición colaborativa más general. El proceso consta de cuatro pasos: externalización, publicación, internalización y reacción, estos pasos trabajan sobre elementos del modelo conceptual que se utilizan para expresar el conocimiento. El modelo conceptual describe como es la representación de conocimiento y en el caso particular de las ontologías, es proporcionado por diferentes primitivas. Las primitivas son descriptas comúnmente en una meta-ontología (sección 2.3) y en particular nos concentraremos en la representación dada por Protégé-2000, que adopta la política de los lenguajes basados en *frames* (capítulo 2, sección 2.8). A continuación se describen cada uno de los pasos involucrados en el proceso general:

- **externalización:** es el disparador del proceso y se desarrolla en un ámbito individual, un integrante quiere explicitar una idea, concepto u opinión para darlo a conocer al resto del grupo por algún mecanismo definido previamente. Este paso es una actividad privada y ocurre dentro de un espacio individual. Para realizar la externalización un individuo utiliza algún medio de representación, por ejemplo la representación ontológica.

- **publicación:** es el acto de anunciar una nueva contribución, implica poner en el contexto del grupo el conocimiento surgido en la externalización, es la manera en que se comparte conocimiento.

- **internalización:** es un proceso individual que tiene lugar cuando alguien comprende y asume un nuevo conocimiento. Esto significa que el individuo aprende, incorpora el nuevo conocimiento, y como consecuencia puede reaccionar.

- **reacción:** es una contestación a una contribución anterior. La reacción puede ser pasiva o activa, en el primer caso significa el fin del proceso, en cambio si la reacción es activa puede dar como resultado una nueva contribución. La nueva contribución consiste en dar información complementaria, otra versión, o abrir una discusión sobre la contribución original. Como consecuencia de la reacción, pueden existir visiones diferentes sobre parte del conocimiento.

El proceso especializado para ontologías utiliza las primitivas: clases, *slot* e instancias, y son los elementos involucrados el en proceso de diseño colaborativo de ontologías. Un usuario realiza la externalización del conocimiento creando primitivas ontológicas en un ámbito individual. Para dar a conocer el conocimiento externalizado, publica las primitivas ontológicas creadas en el paso anterior. Los usuarios internalizan los conceptos modelados por las primitivas ontológicas publicadas y en base a su interpretación pueden reaccionar. A partir de este punto cuando se haga referencia al *proceso*, se refiere al proceso de diseño instanciado al diseño colaborativo de ontologías.

Una de las características fundamentales del *proceso* es que en cada ciclo se produce un *nueva versión* de la ontología compartida, esta versión es aumentativa porque cada publicación trae nuevos elementos al contexto compartido. Vemos a las publicaciones como una contribución al desarrollo compartido.

Las reacciones son un punto clave, determinan que antes ha ocurrido una internalización, es decir se ha comprendido una contribución. Las reacciones son el motor impulsor de la actividad de discusión ya que pueden provocar nuevas contribuciones. Las contribuciones provocadas por reacciones describen "una secuencia" que reflejan una interacción entre los miembros para iterativamente alcanzar un acuerdo general. Las reacciones pueden ser motivadas por diferentes causas: un usuario quiere ampliar el nuevo concepto vertido

al espacio público. dar otro punto de vista o querer opinar sobre la contribución original. A continuación se describen las causas que pueden provocar una reacción:

- **Complementar una contribución:** en este caso un usuario "acepta" los conceptos agregados por la contribución, está de acuerdo con su modelización y decide complementar contribución agregando nuevos elementos ontológicos, que amplían el modelo anterior.

- **Dar otro punto de vista:** el usuario no está de acuerdo con los conceptos agregados por la contribución y posee una forma propia de modelizar los conceptos. Aquellas reacciones que dan otros puntos de vista introducen divergencias, provocando un conflicto en el entendimiento común. Las divergencias son distintas conceptualizaciones de un mismo concepto y deben ser tratadas en forma especial como se verá en la sección 4.3.

- **Argumentar la contribución original:** son un tipo de reacción particular que surgen cuando un usuario quiere dar una opinión sobre alguna contribución y pasan a tener un valor central cuando se hacen sobre contribuciones que son divergentes, esto se debe a que las argumentaciones pueden ser utilizadas como un elemento de importancia a la hora de tomar decisiones para unificar diferencias.

Las reacciones analizadas anteriormente son reacciones activas y son el punto destacado en el ciclo, pero no se puede obviar el hecho de que una reacción puede ser pasiva, esto quiere decir que no hay una verdadera contribución. Las reacciones pasivas implican la externalización sin la publicación y pueden o no reflejar divergencias, es posible que el individuo exprese en su ámbito privado un desacuerdo para tal vez darlo a conocer más adelante.

Las primitivas ontológicas: clases, *slots* e instancias son los elementos de conocimiento que están involucradas en cada uno de los cuatro pasos del *proceso*. Un usuario realiza la conceptualización del dominio de discusión (o parte de él) creando y componiendo múltiples primitivas ontológicas. Definimos *elemento ontológico* a un conjunto de primitivas ontológicas que juntas modelan alguna porción del dominio de discusión. En la *figura 4.1* se pueden observar entre otras primitivas, las primitivas *Author* y *THING* correspondiente a clases ontológicas y la primitiva *name* que se corresponde a un *slot*. La conceptualización de "autor" es un elemento ontológico y está compuesto por la clase *Author*, su atributo *name* y su superclase *THING*.

En el contexto de este trabajo los participantes llevan a cabo el *proceso* para diseñar una ontología común sobre un dominio de discusión establecido previamente, para tal fin necesitan contar con un espacio individual donde generen su visión particular, es decir un lugar donde externalicen su conocimiento creando una ontología propia y un espacio

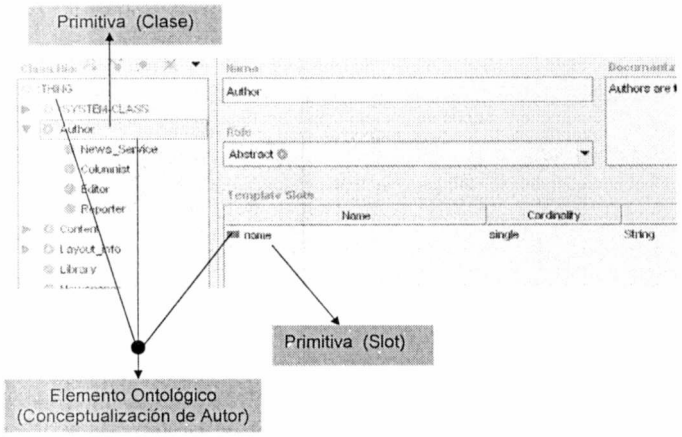


Figura 4.1: Primitivas y Elemento Ontológico

común donde diseñen la visión unificada de la ontología a través de la publicación de sus diseños individuales.

Para observar como funciona el *proceso* la *figura 4.2* muestra un ejemplo abstracto de los pasos involucrados. Para ayudar a la interpretación y ver los conceptos generales, el ejemplo se basa en una jerarquía de clases ontológica, donde las clases poseen nombres genéricos: $A \ldots E$. A continuación se analizan los cuatros escenarios planteados en el ejemplo:

1. En un primer momento los participantes (Pa, Pb, Pc) tienen la misma conceptualización del dominio. Cada una de las ontologías privadas vinculada a cada participante es igual a la ontología compartida.

2. El usuario Pa produce una **externalización** en su espacio privado, agregan a su diseño la clase E y luego lo da a conocer al resto del grupo **publicando** E al espacio compartido.

3. Los participantes Pb y Pc comprenden el nuevo concepto, lo **internalizan**. Pb incorpora E a su espacio privado.

4. Pc **reacciona** definiendo una nueva conceptualización de E (una nueva externalización) que plantea una visión diferente y determina a E como subclase de C y no de B como definió previamente Pa.

   En este caso la reacción pasiva, el participante Pc reacciona ante la publicación del elemento E sin todavía dar a conocer su punto de vista (divergente) al resto del grupo.

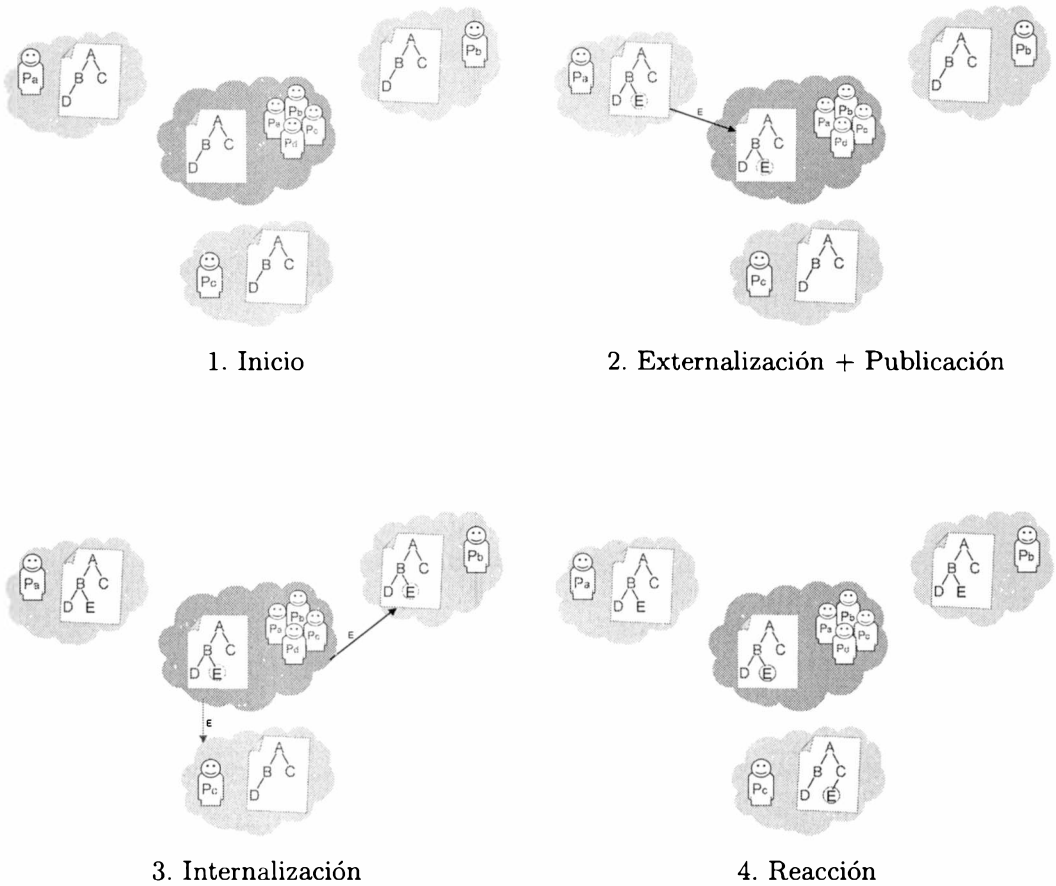


Figura 4.2: Pasos dentro del *proceso*

En resumen, un usuario externaliza su conocimiento generando un elemento ontológico E en su espacio privado, si desea darlo a conocer al resto del grupo, publica E al espacio común. La iternalización y la subsecuente reacción, que pueden seguir a la publicación, es referente a E o algún subconjunto de primitivas incluidas en E (es decir a otro elemento ontológico), por ejemplo si un usuario publica el concepto *Author*, otro usuario puede reaccionar con respecto al *slot name*.

4.2.1. Espacios de trabajos

Continuando con el diseño colaborativo ya se mencionó varias veces la noción de espacios de trabajo publico y espacio de trabajo privado. Es la metodología de trabajo, basada en el *proceso*, quien determina la necesidad de contar con espacios de trabajos distintos: una persona diseña elementos ontológicos en forma privada, es decir agrega elementos ontológicos a una ontología propia; para luego darlos a conocer al resto del grupo, es decir publica elementos ontológicos en una ontología común. Esta forma de interacción delinea la necesidad de tener espacios de trabajos distintos: espacios para el trabajo privado de cada individuo y un espacio para el trabajo compartido, a estos espacios

se los denomina *espacios privados* y *espacio compartido*. Cada uno de los espacios de trabajo contiene una ontología, denominadas: *ontologías privadas* y *ontología compartida* respectivamente. A continuación se muestran la características de cada espacio de trabajo:

- **Espacio privado:** es un ámbito privado, contiene una ontología denominada ontología privada. La ontología contenida en este espacio es diseñada y editada sólo por el dueño del espacio. El espacio privado es donde se produce la externalización, realizando la conceptualización del dominio según el punto de vista del dueño, él edita los elementos ontológicos según su visión.
- **Espacio compartido:** es el lugar de comunicación del grupo, en él se desarrolla la ontología común a los participantes, denominada ontología compartida. La edición de la ontología en este espacio no es directa, los participantes realizan el diseño por medio de la publicación de elementos diseñados en sus espacios privados. El espacio de trabajo compartido es un espacio común que apoya el desarrollo colaborativo de la ontología de dominio del grupo, este desarrollo se lleva a cabo por medio del *proceso*.

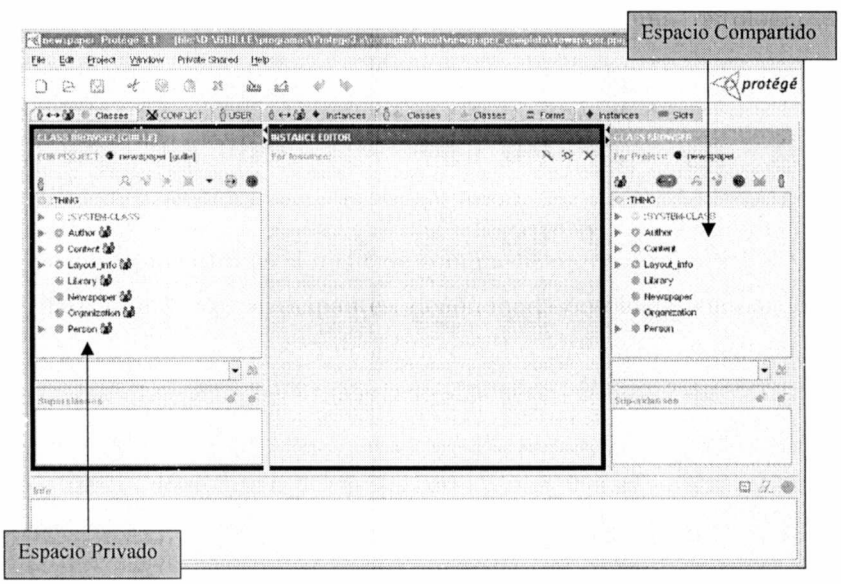


Figura 4.3: Espacios de trabajos en CO-Protégé

CO-Protégé soporta espacios de trabajo privados y un espacio público. Los espacios de trabajo están definidos por la ontología que se edita en ellos. La *figura 4.3* muestra a un usuario (*Guille*) utilizando el sistema. En particular se pueden observar los dos espacios de trabajos, en la parte izquierda se encuentra la ontología privada y en la parte derecha la ontología compartida. El espacio de trabajo público es único y es un espacio de trabajo compartido accesible por cualquier miembro, contiene la ontología de dominio compartida formada por elementos ontológicos publicados por los participantes.

4.2.2. Acciones y comunicación de los diferentes espacios

Si bien tenemos espacios de trabajos bien diferenciados, éstos están fuertemente relacionados: hay muchas versiones ontológicas del dominio de discusión del grupo que coexisten. Las ontologías privadas y la ontología compartida contenidas en los diferentes espacios pueden ser consideradas como versiones diferentes de una misma conceptualización. Un individuo claramente maneja dos versiones de la misma ontología, su versión privada y la versión compartida. En el diseño el usuario debe alternar entre su ontología privada y la ontología compartida permitiendo el intercambio de elementos ontológicos entre ambas. El *proceso* en sí mismo incluye la comunicación entre ambos espacios y la forma de soportarla es por un conjunto de acciones que podemos clasificar entre acciones privadas y acciones públicas.

- **Acciones privadas:** son las acciones necesarias para hacer explícito el conocimiento; es decir acciones para crear y editar elementos ontológicos por medio de la creación y edición de primitivas ontológicas. Las acciones privadas dentro de *proceso* se encuadran en acciones de externalización, se dan en la ontología privada y no son visibles al resto del grupo.

- **Acciones públicas:** permiten realizar contribuciones al diseño de la ontología compartida. Una contribución se realiza por medio de una publicación desde una ontología privada hacia la ontología compartida e implican cambios en la ontología compartida. Dentro del *proceso* las acciones públicas por excelencia, son las acciones de publicación y sobre ellas es de esperar que surjan las reacciones. Otra acción catalogada como pública es la acción que permite trasladar elementos desde el espacio compartido al individual, aunque ésta no es una acción propia del proceso, puede verse como una acción derivada de la internalización de algún elemento publicado previamente.

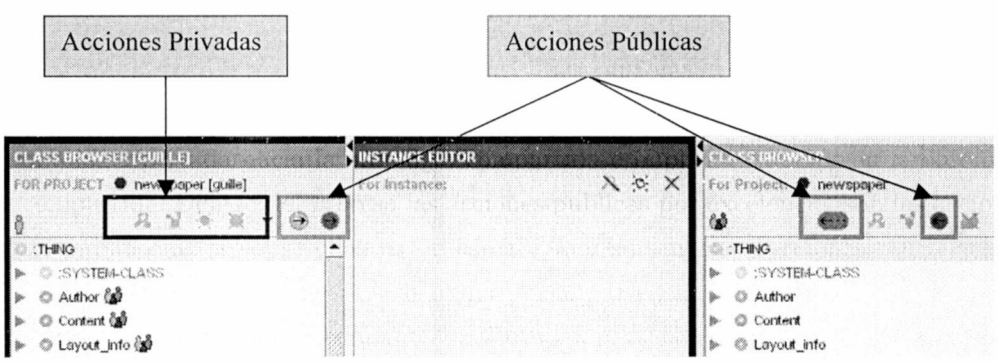


Figura 4.4: Acciones privadas y públicas

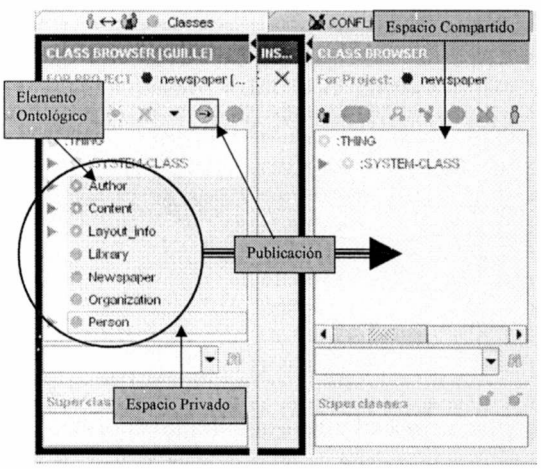
Dentro de CO-Protégé las acciones privadas son las acciones habituales de edición de primitivas ontológicas. Las acciones públicas es una nueva funcionalidad, que permite publicar o actualizar elementos desde la ontología privada a la ontología compartida. También se cuentan con acciones para traer elementos ontológicos desde el espacio público al privado.

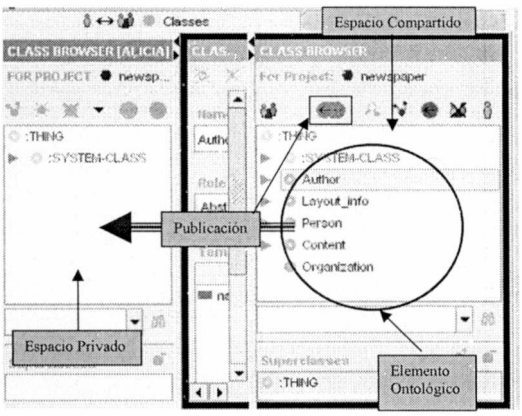
La *figura 4.4* muestra las acciones disponibles en una ventana particular de CO-Protégé. La publicación desde la ontología privada (izquierda de la figura), a la ontología compartida (derecha de la figura) está soportada por dos acciones: *publicación* y *actualización*, donde la primer acción sirve para agregar nuevos elementos y la segunda para actualizar elementos existentes. Las dos acciones pueden ser utilizadas, siempre y cuando no provoquen inconsistencias en la ontología compartida, esta restricción es explicada en la sección 4.2.3.

Comenzando con el diseño

Para ver cómo se diseña una ontología colaborativamente, siguiendo la metodología descripta en esta primera etapa, se analiza el desarrollo colaborativo de la ontología *newspaper* planteada como ejemplo en la primera sección. La edición se realiza utilizando el prototipo CO-Protégé.

La imagen de la derecha muestra el espacio privado de *Guille*, con los elementos que él diseñó usando las acciones privadas. *Guille* quiere dar a conocer al grupo su diseño, entonces publica el diseño utilizando la acción *publicación* (una acción pública). Antes de la publicación, la ontología compartido no contenía elementos ontológicos que modelen el dominio de discusión de los participantes. Luego de que *Guille* publicó toda su conceptualización, la ontología privada y la compartida quedan "idénticas", en lo que respecta a la ontología del dominio.

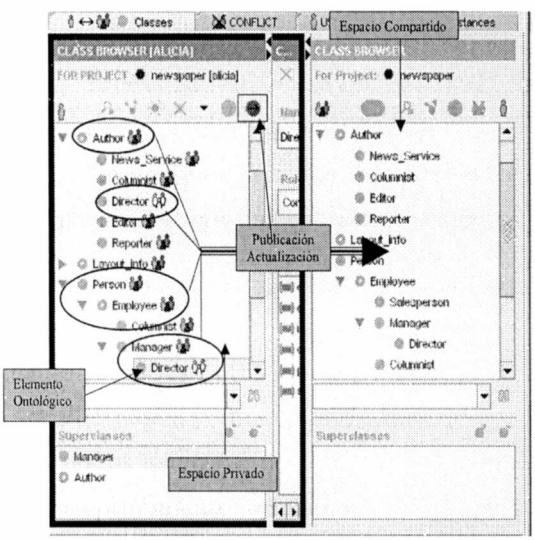




Tanto *Alicia* como *Esteban* comienzan el trabajo dentro del proyecto con su ontología privada sin ningún elemento. Comprenden lo publicado por *Guille* y deciden incorporarlo a sus espacios de trabajos privados. En este caso los dos participantes internalizan el conocimiento y como consecuencia de ello lo incorporan a su ontología. La figura de la izquierda muestra esta situación, *Alicia* incorpora los elementos publicados *Guille* en el paso anterior.

En este punto existen cuatro versiones "idénticas" de la ontología del dominio, las ontologías privadas de *Guille*, *Alicia* y *Esteban* y la ontología compartida. Se dice que las ontologías son "idénticas" con respecto al dominio de los usuarios, por que los elementos ontológicos que se encuentran en ellas, modelan conceptualmente lo mismo.

Alicia modifica la clase *Director* definiendo que un director también puede ser un escritor del periódico; con tal fin agrega como *superclase* de *Director*, la clase *Autor*. Este paso refleja una reacción activa a la publicación *Director*, que desencadenó una nueva externalización publicada por *Alicia*. En la imagen de la derecha se observa la situación, la acción a realizar es una publicación que difiere a las anteriores y se denomina actualización. Conceptualmente la actualización es una publicación pero necesitan un tratamiento distinto como se verá en la siguiente sección.



Como se va vislumbrando, los caminos posibles a seguir son muchos, podemos suponer por ejemplo que *Esteban* realice una nueva internalización en base a los cambios realizados por *Alicia* y modifique nuevamente su ontología privada. Pero ¿que pasa si *Guille* no está de acuerdo con los nuevos cambios?, una alternativa es que no refleje los cambios en su ontología privada. Ahora, *Guille* puede querer dar a conocer al resto del grupo su desacuerdo, por lo tanto es necesario ampliar el modelo para contemplar esta situación y permitir soportar diferentes puntos de vista. Estas características se describen en las próximas secciones.

4.2.3. Contribuciones aumentativas

El desarrollo colaborativo es una consecuencia de mezclar acciones privadas y públicas: externalizar el conocimiento en el espacio privado (acciones privadas) y publicarlo subsecuentemente al espacio compartido (acciones públicas). Mientras la externalización es el acto de hacer la conceptualización de un elemento ontológico; la publicación no es una acción directa y su realización está restringida a que la contribución de elementos ontológicos sea aumentativa.

Publicaciones aumentativas representan el espíritu del conocimiento compartido de la comunidad, donde la gente comparte nuevas ideas, soluciones y aún problemas. En este sentido, con cada contribución "el conocimiento" del grupo siempre aumenta [PP03].

Una contribución ontológica es aumentativa si la integración de la contribución ontológica a la ontología compartida es coherente, es decir, evita la presencia de cualquier desajuste ontológico. Entendiendo un elemento ontológico como una ontología, el problema de publicar un elemento ontológico radica en combinar ambas ontologías, el elemento ontológico (que es en sí mismo una ontología) y la ontología de dominio compartida. La integración de la contribución ontológica y la ontología compartida puede ser hecha combinándolas en una "nueva versión" de la ontología compartida.

Combinar implica que la contribución ontología sea aumentativa: es decir que amplié a la ontología compartida y no provoque desajustes ontológicos. Por ejemplo, si la contribución agrega un *slot* que hace referencia a una clase que no existe en la ontología compartida, provoca un inconsistencias en el modelo.

Para hacer efectiva una publicación, es necesario realizar el chequeo que determinen que los elementos ontológicos a publicar, son una contribución aumentativa. Para llevar a cabo el chequeo se adoptó un acercamiento heurístico, que garantiza que la combinación de dos ontologías es aumentativa. El acercamiento se basa en un conjunto de reglas.

Antes de describir las reglas que determinan si una publicación es aumentativa o no, necesitamos introducir la noción de existencia e igualdad de una primitiva p, con respecto a otra contenida en una ontología destino o_d. Todas las definiciones se basan en el modelo de representación del conocimiento del lenguaje basado en *frames* que provee Protégé-2000 y sus características fueron explicadas en el capítulo 2 sección 2.8.1.

Existencia de una primitiva ontológica en una ontología

Teniendo en cuenta que, por definición, el nombre de *frame* es único, para verificar que una clase o un *slot* existe en o_d, basta verificar que exista un *frame* en o_d con igual nombre e igual tipo al que se está verificando. Por ejemplo si se quiere verificar la existencia de la clase *Author* en o_d, alcanza con encontrar un *frame* con igual nombre y que él mismo sea una clase. Ahora si hay un *frame* con el nombre *Author* y es un *slot*, entonces la clase *Author* no existe en o_d.

La semántica del nombre de una clases o de un *slots*, permite utilizar el atributo *nombre* para verificar existencia. No sucede lo mismo con los *frames* que son instancias, ya que el *nombre* no es un valor que pueda ser analizado, pues no tiene más significado que un mero representador interno y difiere de una ontología a otra. Por lo tanto la noción de existencia sin que denote igualdad se refiere solo a clases y a *slots*.

Se define que p **existe** en o_d si y sólo si

$$\begin{array}{l} \exists\, p_d \in o_d,\ name(p) = name(p_d) \wedge typeFrame(p) = typeFrame(p_d) \\ \wedge \\ typeFrame(p_o) = Class \vee typeFrame(p_o) = Slot \end{array}$$

Donde la función *name*, retorna el nombre que identifica al *frame* y la función *typeFrame* retorna el tipo de un *frame* que puede ser: Instancia, Clase o *Slot*.

En las próximas secciones se utilizan varias funciones: la función # retorna la cantidad de elementos del conjunto pasado como parámetro, mas un conjunto de funciones que se asocian con las definiciones de las propiedades de las clases, *slots* o instancias. Por ejemplo la función *AllowedParents* retorna el conjunto de clases definidas en el *facet Allowed Parents*, de la clase pasada como parámetro.

Igualdad de una primitiva ontológica en una ontología

Una primitiva p_o que se encuentra en o_o **es igual a** una primitiva p_d que existe en o_d si y sólo si:

- $typeFrame(p_o) = typeFrame(p_d) = Slot$, entonces se tiene que cumplir que:

$$\begin{array}{l} Type(p_o) = Type(p_d) \\ \wedge\ Cardinality(p_o) = Cardinality(p_i) \\ \wedge\ \text{si el } Type(p_o) = Type(p_i) = Class \Rightarrow \\ \quad \forall c \in AllowedParents(p_o),\ \exists\, c' \in AllowedParents(o_d),\ name(c) = name(c') \\ \quad \wedge\ \#(AllowedParents(p_o)) = \#(AllowedParents(o_d)) \\ \quad \wedge\ \forall c \in Default(p_o),\ \exists\, c' \in Default(o_d),\ name(c) = name(c') \\ \quad \wedge\ \#(Default(p_o)) = \#(Default(o_d)) \\ \quad \wedge\ name(InverseSlot(p_o)) = name(InverseSlot(p_d)) \\ \wedge\ \text{si el } Type(p_o) = Type(p_i) = Slot, \Rightarrow \\ \quad \forall c \in AllowedClasses(p_o),\ \exists\, c' \in AllowedClasses(o_d),\ name(c) = name(c') \\ \quad \wedge\ \#(AllowedClasses(p_o)) = \#(AllowedClasses(o_d)) \\ \quad \wedge\ \forall c \in Default(p_o),\ \exists\, c' \in Default(o_d),\ name(c) = name(c') \\ \quad \wedge\ \#(Default(p_o)) = \#(Default(o_d)) \\ \quad \wedge\ name(InverseSlot(p_o)) = name(InverseSlot(p_d)) \\ \wedge\ \text{si el } Type(p_o) = Type(p_i) = Symbol, \Rightarrow \\ \quad \forall c \in AllowedValues(p_o),\ \exists\, c' \in AllowedValues(o_d),\ name(c) = name(c') \\ \quad \wedge\ \#(AllowedValues(p_o)) = \#(AllowedValues(o_d)) \\ \quad \wedge\ \forall c \in Default(p_o),\ \exists\, c' \in Default(o_d),\ name(c) = name(c') \\ \quad \wedge\ \#(Default(p_o)) = \#(Default(o_d)) \end{array}$$

- $typeFrame(p_o) = typeFrame(p_i)$ *Class*, entonces se tiene que cumplir que:

$$
\begin{array}{ll}
 & Role(p_o) = Role(p_i) \\
\wedge & \forall c \in DirectSuperClasses(p_o),\ \exists c' \in DirectSuperClasses(o_d),\ name(c) = name(c') \\
\wedge & \#(DirectSuperClasses(p_o)) = \#(DirectSuperClasses(o_d)) \\
\wedge & \forall s \in TemplateSlot(p_o),\ \exists s' \in TemplateSlot(o_d),\ name(s) \textbf{ es igual a } name(s') \\
\wedge & \#(TemplateSlot(p_o)) = \#(TemplateSlot(o_d))
\end{array}
$$

- $typeFrame(p_o) = typeFrame(p_i) = Instance$, entonces se tiene que cumplir que:

$$
\begin{array}{ll}
 & \forall c \in DirectType(p_o)\ \exists c' \in DirectType(o_d),\ s \textbf{ es igual a } s' \\
\wedge & \#(DirectType(p_o)) = \#(DirectType(o_d)) \\
\wedge & \forall s \in DirectSlots(p_o),\ \exists s' \in DirectSlots(o_d),\ value(s) \textbf{ es igual a } value(s') \\
\wedge & \#(DirectSlots(p_o)) = \#(DirectSlots(o_d))
\end{array}
$$

La igualdad definida propone comparar los *slots* por sus *facets* y, en el caso de que ellos se refieran a otros *frames*, solo se necesita determinar que estos últimos existan y no que sean iguales. Esto se debe a que si comparamos el contenido de los valores de los *facets* por igualdad, se obligaría a recorrer, en la mayoría de los casos, toda la ontología destino.

Las clases se comparan por sus atributos, los *slots* adheridos y por la existencia de sus superclases directas. Para las superclases solo se pide que existan, no que sean iguales. Se comparan todos los *slots*, los propios y los heredados. De esta manera no importa de donde la clase hereda los *slots*, si no sólo que los contenga, asegurando también la preservación de la relación *IS_A*.

Una instancia se compara por sus tipos, es decir las clases a las que pertenece y por los valores de los *slot*. Las instancias son un caso especial ya que pueden ser iguales a una o a varias instancias en o_d.

Reglas para la comprobación de contribuciones aumentativas

Ya definidas las relaciones de existencia e igualdad, se puede definir reglas que garanticen que la publicación de un conjunto de primitivas ontológicas, sea una contribución aumentativa.

En el contexto del trabajo, la contribución es un subconjunto de primitivas ontológicas, que provienen de una ontología privada y que quieren ser agregadas a la ontología compartida. Hay que tener en cuenta que los elementos a publicar, pueden tener referencia a otros elementos que no forman parte de la publicación. Por ejemplo se puede publicar una clase sin que sus superclases participen de la operación, esto se debe a que las superclases ya existen en la ontología compartida.

Denominamos o_o al conjunto de primitivas ontológicas a publicar y o_d a la ontología destino. Se garantiza que la publicación de o_o es **una contribución aumentativa para** o_d si soló si $\forall p_o \in o_o$, p_o **es una publicación aumentativa para** o_d.

p_o **es una publicación aumentativa para** o_d si soló si:

- $typeFrame(p_o) = Slot$, entonces se tiene que cumplir que:

$\neg\exists p_d \in frames(o_d), name(p_o) = name(p_d)$
$\wedge$ $Cardinality(p_o) = Cardinality(p_d)$
$\wedge$ si el $Type(p_o) = Class \Rightarrow$

$\forall c \in AllowedParents(p_o)$,

$\vee$ $(\exists c' \in frames(o_d)$, c **es igual a** c'
$(c \in o_o \wedge c$ **es una publicación aumentativa para** $o_d))$

$\wedge$ $\forall c \in Default(p_o)$,

$\vee$ $(\exists c' \in frames(o_d)$, c **es igual a** c'
$(c \in o_o \wedge c$ **es una publicación aumentativa para** $o_d))$

$\wedge$ $(\exists s \in frames(o_d)$, $InverseSlot(p_o)$ **es igual a** s

$\vee$ $(s \in o_o \wedge s$ **es una publicación aumentativa para** $o_d))$

$\wedge$ si el $Type(p_o) = Slot \Rightarrow$

$\forall c \in AllowedClasses(p_o)$,

$\vee$ $(\exists c' \in frames(o_d)$, c **es igual a** c'
$(c \in o_o \wedge c$ **es una publicación aumentativa para** $o_d))$

$\wedge$ $\forall c \in Default(p_o)$,

$\vee$ $(\exists c' \in frames(o_d)$, c **es igual a** c'
$(c \in o_o \wedge c$ **es una publicación aumentativa para** $o_d))$

$\wedge$ $(\exists s \in frames(o_d)$, $InverseSlot(p_o)$ **es igual a** s

$\vee$ $(s \in o_o \wedge s$ **es una publicación aumentativa para** $o_d))$

- $typeFrame(p_o) = Class$, entonces se tiene que cumplir que:

$\neg\exists p_d \in frames(o_d), name(p_o) = name(o_d)$
$\wedge$ $\forall c \in DirectSuperClasses(p_o)$

$\vee$ $(\exists c' \in frames(o_d)$, c **es igual a** c'
$(s \in o_o \wedge s$ **es una publicación aumentativa para** $o_d))$

$\wedge$ $\forall s \in TemplateSlots(p_o)$

$\vee$ $(\exists s' \in frames(o_d)$, s **es igual a** s'
$(s \in o_o \wedge s$ **es una publicación aumentativa para** $o_d))$

- $typeFrame(p_o) = Instance$, entonces se tiene que cumplir que:

$\forall c \in DirectType(p_o)$

$\vee$ $(\exists c' \in frames(o_d)$, c es igual c'
$(s \in o_o \wedge s$ **es una publicación aumentativa para** $o_d))$

$\wedge$ $\forall s \in DirectSlots(p_o)$ y sea $v = value(s, p_o)$

si $type(v) = Class \vee type(v) = Slot \vee type(v) = Instace \Rightarrow$

$\vee$ $(\exists v' \in frames(o_d)$, v **es igual a** v'
$(v \in o_o \wedge v$ **es una publicación aumentativa para** $o_d))$

La definición implica, que un *slots* **es una publicación aumentativa para** o_d, si **no existe** en o_d y para cada una de sus propiedades asociadas o bien son **iguales** en o_o, o bien participan de la publicación y se garantiza que son **una publicación aumentativa para** o_d. Una clase **es una publicación aumentativa para** o_d, si **no existe** en o_d y sus propiedades asociadas: superclases y *slots*, o bien son **iguales** en o_o, o bien cada una participan de la publicación y se garantiza que son **una publicación aumentativa para** o_o. Finalmente, una instancia **es una publicación aumentativa para** o_d, si las clases a la que pertenecen o bien son iguales en o_d, o bien cada una participan de la publicación y se garantiza que son **una publicación aumentativa para** o_d, lo mismo se debe verificar para los valores de los *slots* asociados que tengan tipo *Class*, *Slot* o *Instance*.

Si para cada una de las primitivas que componen o_o, se puede asegurar que garantizan **una publicación aumentativa para** o_d, entonces podemos afirmar que o_o es una **contribución aumentativa** para o_d.

Referencias cruzadas

Como pasa usualmente la contribución puede contener referencias cruzadas. En la *figura 4.5* se muestran tres ejemplos de referencias cruzadas, el primero muestra la clase A que tiene un *slot* asociado que referencia a la misma clase A, los otros dos ejemplos muestran una relación cruzada que involucra más de una clase.

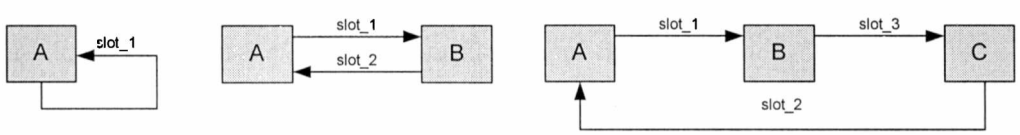


Figura 4.5: Referencias cruzadas

Cuando o_o contiene referencias cruzadas las reglas descriptas anteriormente no son aplicables. Si consideramos el segundo ejemplo de la *figura 4.5* y lo tomamos como o_d y, además asumimos que en o_d no existen ni las clases, ni los *slots* involucrados. Cuando se quiere verificar que la publicación de la clase A **es una publicación aumentativa para** o_d, se debe verificar el $slot_1$ **es una publicación aumentativa para** o_d y éste determina verificar la clase B **es una publicación aumentativa para** o_d y por consiguiente que el $slot_2$ **es una publicación aumentativa para** o_d. La verificación del $slot_2$ necesita verificar nuevamente que clase A **es una publicación aumentativa para** o_d, entrando en un ciclo infinito.

Los casos en los que o_o contienen referencias cruzadas pueden ser analizados en tres etapas, como se describen a continuación:

1. Verificar que las clases y los *slots* puedan ser publicados, solo teniendo en cuenta sus nombres.
2. Crear una nueva ontología destino o'_d, que contenga a o_d completo, a las clases y a los *slots* de o_o, pero sin sus propiedades asociadas.
3. Utilizar las reglas para las propiedades asociadas de las clases y de los *slots* pertenecientes a o_o contra o'_d.

Si se cumplen cada una de las tres etapas entonces se puede garantizar que o_o es **una contribución aumentativa para** o_d.

Cuando un usuario quiere realizar una contribución a la ontología compartida, es decir publicar elementos que contiene en su ontología privada; primero se debe realizar la comprobación de que los elementos ontológicos que componen la publicación son **una contribución aumentativa para** la ontología compartida. Las contribuciones que pasan esta comprobación pueden ser integradas a la ontología compartida sin problemas. Al contrario, contribuciones no aumentativas son rechazadas y son en esencia una fuente potencial de divergencias. En el capítulo 5 sección 5.6.1 se explica cómo se realizan los chequeos en CO-Protégé y cómo son visualizados los resultados de los mismos.

4.2.4. Contribuciones no aumentativas

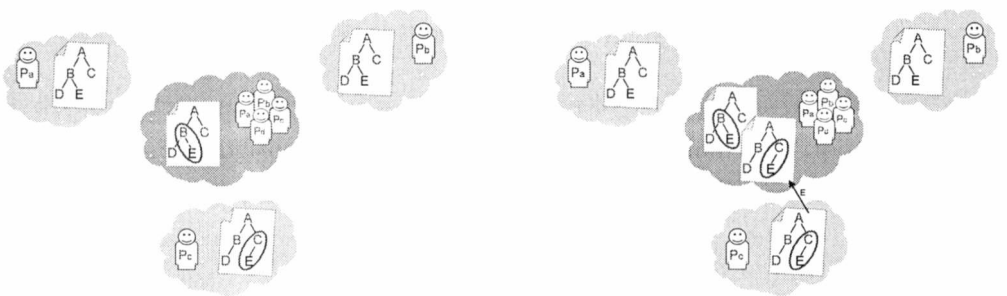
Una contribución no aumentativa denota una divergencia. Cuando el chequeo de una contribución plantea esta situación, el usuario que quiso realizar la contribución. puede reaccionar de las tres siguientes maneras para superar la contradicción:

1. Modificar su ontología privada según los requerimientos que impone el espacio compartido, y así la divergencia privada desaparece, luego puede proceder normalmente a la publicación.
2. Conservar la divergencia entre las versiones privadas y compartidas, de modo que la *divergencia sea privada* y no provoque conflicto alguno en la ontología compartida, pudiendo en caso de ser posible publicar sólo la porción que no genera conflicto.
3. El usuario desea dar a conocer de todos modos la divergencia, pero como la publicación debe ser aumentativa, se impide su realización, en consecuencia se debe optar por un mecanismo alternativo. Primero debe informar al resto del grupo el elemento ontológico que provoca la divergencia y luego realizar la publicación como una alternativa. Este es el eje central para soportar las divergencias y se explica en la próxima sección.

4.3. Soportando divergencias

Como consecuencia de que cada usuario tiene acceso a su espacio de trabajo privado y al público; las divergencias en del diseño de ontologías pueden existir en forma privada y/o publica, como se explica a continuación:

- **divergencias privadas:** son divergencias en la ontología privada, el usuario tiene elementos ontológicos en conflicto con la ontología común. Este tipo de divergencias son las más simples y se encuentran en el contexto personal, no influyendo al resto de los participantes. La versión privada "está en conflicto" (es divergente), si comparándola con la versión compartida hay al menos un elemento E que está en ambas versiones pero difiere (no es igual) y no sería una contribución aumentativa si quisiera ser publicado.
- **divergencias públicas:** una divergencia pública es una divergencia en el nivel de la ontología compartida y ocurre cuando una publicación que no es aumentativa y quiere realizarse de todas maneras.



1-Divergencia privada 2-Divergencia pública

Figura 4.6: Divergencias

La *figura 4.6* muestra los dos tipos de divergencias que pueden ocurrir, en la izquierda se muestra la situación de una divergencia privada: el usuario Pc tiene una conceptualización distinta en su ontología privada a la que se encuentra en la ontología compartida. La parte derecha de la figura muestra el caso en que el usuario quiere publicar su punto de vista y por lo tanto se debe permitir que en el espacio público se encuentren las dos conceptualizaciones.

Cualquier contribución que no es aumentativa, implica que el elemento ontológico involucrado en la publicación producirá conflictos si fuera agregado a la ontología compartida, esto denota la existencia de conceptualización diferente y el usuario debe ser capaz de plantear esta situación al resto del grupo. En el ejemplo el usuario Pc quiere publicar su visión de E pero se ve imposibilitado por que la publicación no es aumentativa.

Para tener diferentes puntos de vista y que estos sean públicos surge la noción de conflicto y alternativas. A continuación se describen los dos conceptos:

- **Conflicto:** se utilizan para modelar la porción del espacio compartido que es divergente, un usuario define explícitamente cuales son los elementos conflictivos según su visión, de esta manera da a conocer al resto del grupo su desacuerdo con la conceptualización.
- **Alternativa:** una alternativa, es otra conceptualización de los elementos que conforman el conflicto.

Teniendo estos dos elementos, un usuario que quiere dar a conocer de todos modos la divergencia, crea un conflicto marcando de forma explícita cuales son los elementos que los componen y después publica su visión como alternativa al conflicto generado previamente. La publicación es aumentativa por que los elementos no forman parte directa de la ontología compartida que modela el dominio de discusión, están en el espacio público como alternativas para que puedan ser observadas por el resto de los usuarios.

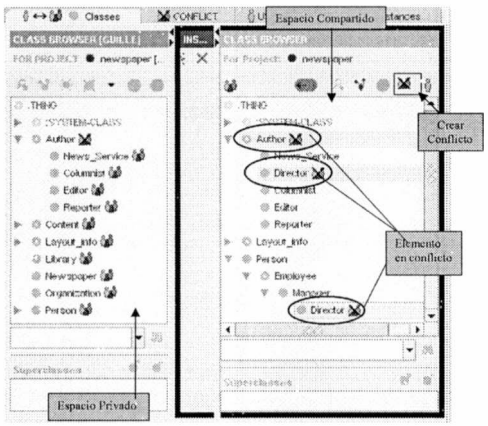
Los conflictos "alertan" a los usuarios de que hay discrepancias, y a decidir cuál es la conceptualización "correcta" para llegar al entendimiento común deseado. Son el punto inicial de una discusión. Las alternativas son posibles soluciones a un conflicto, determinan otros punto de vista e intervienen como elementos de discusión.

Para permitir que los usuarios participen activamente en una discusión, surgen las argumentaciones:

- **Argumentación:** declara una opinión o un juicio que apoya o se opone a una alternativa. El grupo de argumentaciones permite tener fundamentos para la negociación que se realiza sobre una contribución contradictoria.

Explicitando divergencias en el diseño

Retomemos el ejemplo del diseño colaborativo de la ontología *newspaper*. *Guille* quería expresar su desacuerdo con el último cambio realizado por *Alicia*, ya que asume que un *Director* no es un *Autor*



Guille selecciona los elementos en el espacio compartido y crea explícitamente el conflicto. Esto permite que el resto de los participantes se enteren del desacuerdo, esta situación se muestra en la figura de la derecha. Luego *Guille* reacciona y en su espacio privado define que en realidad un *Director* puede ser un *Autor* pero no en forma directa si no por medio de la creación de un alias como un *Columnista*. Para tal fin crea la relación *alias* y la publica como alternativa al conflicto que él generó.

Tanto *Alicia* como *Esteban* observan el conflicto y deciden generar *Argumentaciones*. *Esteban* demuestra su acuerdo y crea una argumentación positiva, *Alicia* en cambio crea una argumentación negativa justificando la conceptualización inicial. La *figura 4.7* muestra la ventana de CO-Protégé que permite observar y editar los conflictos, alternativas y argumentaciones. Esta ventana será explicada en detalle en el capítulo 5 sección 5.6.2.

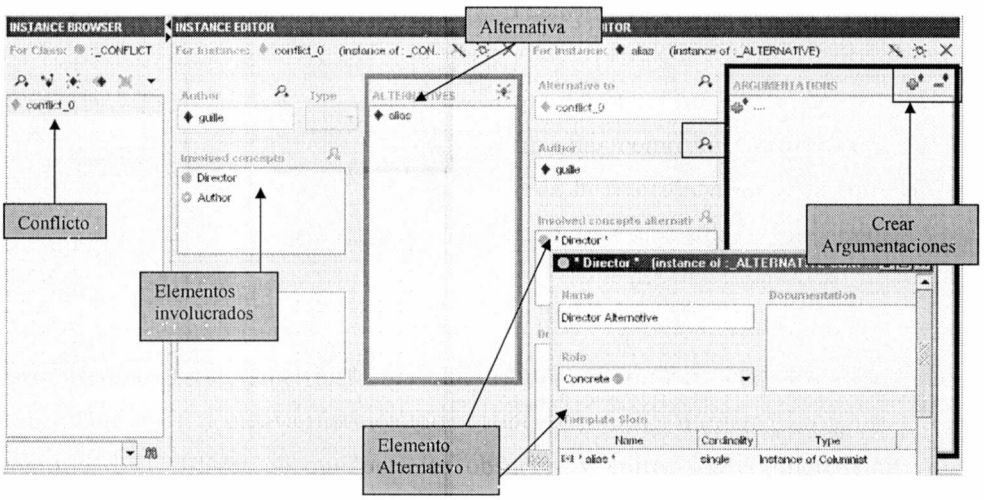


Figura 4.7: Visualización de los conflictos, alternativas y argumentaciones en CO-Protégé

4.3.1. Extensión del modelo de conocimiento

En la sección anterior se comprendió la necesidad de introducir conflictos, alternativas para poder incluir diferentes puntos de vistas. Los nuevos conceptos extiende el modelo de conocimiento y permiten encapsular elementos ontológicos inconsistentes como elementos consistentes.

Se define un nuevo modelo ontológico para la representación del conocimiento, que permite incluir dentro de la ontología compartida elementos ontológicos, que con el modelo de representación clásico serían inconsistentes. El nuevo modelo de conocimiento plantea dos tipos diferentes de *frames*:

1. *frames* **de dominio:** que modelan los elementos ontológicos del dominio de los participantes, siguiendo el modelo de representación clásico planteado en el capítulo 2 sección 2.8.2.

2. *frames* **alternativos:** que modela los elementos alternativos, permitiendo que éstos convivan en la misma ontología que los conceptos modelados como *frames* de dominio, sin que causen incompatibilidades.

La separación de la representación de los elementos ontológicos en *frames* de dominio y *frames* alternativos permite representar en la misma ontología, elementos que podrían introducir incompatibilidades. Ya se mencionó que no podemos tener dos clases con igual nombre en la misma ontología. Si por ejemplo *Esteban* quiere publicar la clase *Content*, que ya existe en la ontología compartida, con el modelo de representación tradicional no podría hacerlo.

Los *frames* alternativos permiten que *Esteban* pueda dar a conocer de todas maneras su visión de la clase *Content*. Como la contribución contradice un elemento existente, es necesario que el resto se entere de la discrepancia, por lo tanto la clase *Content* de la ontología compartida debe "marcarse" en conflicto, finalmente *Esteban* publica la clase *Content*, de su espacio privado, como una alternativa al conflicto creado, es decir la clase es modelada como un *frame* alternativo.

Es importante tener en cuenta que los *frames* alternativos, se utilizan para modelar alternativas a conflictos; son más que un recurso para permitir tener en una misma ontología elementos incompatibles. Por ejemplo si un usuario quiere publicar la clase con nombre A, como otra visión de la clase con nombre B, debe hacerlo como alternativa por más que A pase el chequeo de contribución aumentativa.

Los *frame* alternativos se pueden ver como "ramas" alternativas del diseño de la ontología compartida. Representan diferentes conceptualizaciones, y ponen en evidencia la existencia de conflictos. La necesidad de generar una visión única y cómo las discusiones son útiles para superar los conflictos se vio en el capítulo 3 sección 3.4.

El modelo que generan los conflictos, las alternativas y las argumentaciones junto con los *frame* alternativos, dan oportunidad a los participantes de mantener una discusión cuando ocurren divergencias, permitiendo explorar y analizar la estructura que se arma a partir de ellos. La discusión se ve como una secuencia de las contribuciones de conocimiento relacionadas con un elemento ontológico particular, el elemento en discusión. El usuario manifiesta la divergencia pública de forma explícita, marcando el elemento ontológico como un elemento en conflicto. Este es el elemento disparador de la discusión, a partir del cual el usuario puede publicar su conceptualización (que se encuentra en su espacio privado) como una alternativa al conflicto, otros usuarios pueden publicar otras alternativas. Completando el modelo están las argumentaciones; que se separan en argumentaciones positivas o negativas.

Haciendo una correlación con el modelo IBIS, podemos ver a los conflictos como las preguntas, a las alternativas como respuestas y a las argumentaciones como las posturas que los participantes toman sobre las respuestas.

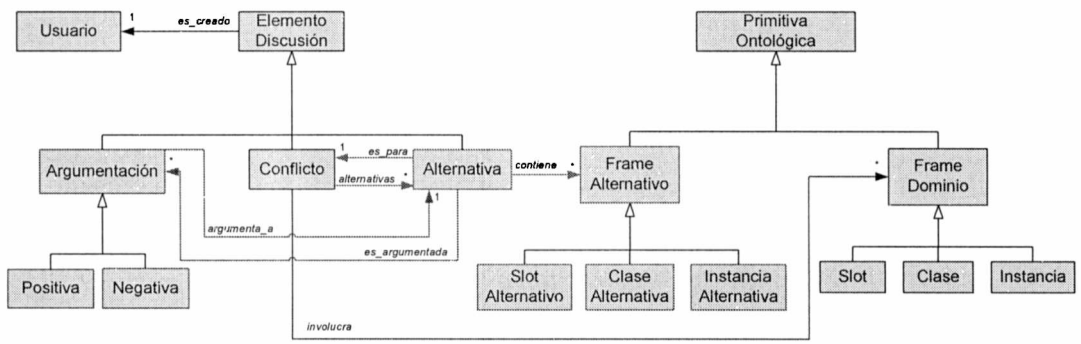


Figura 4.8: Extensión del modelo de conocimiento

En la *figura 4.8* se puede observar en forma esquemática el nuevo modelo de representación del conocimiento. Contiene dos tipos de *frames*, modelados por las *meta-clases* frames *Alternativo* y *frames Dominio.* Estas dos clases son especializadas por diferentes *meta-clases* para modelar los diferentes tipos de *frames* (Clases, *Slots*e Instancias)

La clase *ElementoDeDiscusión* modela en forma abstracta, los conceptos que participan en una discusión: *Conflictos*, *Alternativas* y *Argumentaciones*. Y los relaciona con el usuario particular que los crea.

La clase *Conflicto* modela al conjunto de elementos ontológicos que generan la discrepancia, relaciona a un conjunto de (*frames* de dominio) como una unidad. Cada instancia de la clase *Conflicto* es el punto iniciales de una discusión.

La clase *Alternativa* modela al conjunto de elementos ontológicos que denotan una visión diferente de un conflicto, relaciona a un conjunto de *frames* alternativos como una unidad. Los *frames* alternativos involucrados modelan una conceptualización distinta a las de un conflicto. Un alternativa se refiere solo a un conflicto, en cambio un conflicto puede tener muchas alternativas.

La clase *Argumentación* modela en forma abstracta las posiciones de los usuarios sobre una alternativa y son especializadas por las clases *Positiva* y *Negativa*, que denotan el sentido de la posición de un usuario: a favor o en contra respectivamente. Una argumentación se relaciona con una única alternativa y una alternativa puede tener varias argumentaciones.

4.4. El rol del *awareness*

El tercer pilar para el diseño colaborativo de ontologías es el *awareness*, un componente relevante de cualquier herramienta groupware. El *awareness* sirve para mantener a los usuarios actualizados sobre la actividad de colaboración que se está desarrollando. En el capítulo 3 sección 3.3 se definió al *awareness* como "un entendimiento de las actividades de los otros, que proporciona un contexto para la propia actividad individual. Este contexto es usado para asegurar que contribuciones individuales son relevantes a la actividad del grupo" [DB92].

La naturaleza del diseño colaborativo hace que el *awareness* tome un papel fundamental, gracias a él las personas pasan a ser conscientes de la actividad de diseño, impulsando por esta vía un estímulo básico para que se mantener al grupo "en actividad".

En la metodología de diseño colaborativo propuesta, el *awareness* es necesario para que un usuario esté informado de los cambios que ocurren en el desarrollo del diseño y juega un papel crítico en lo que se refiere al diseño participativo. Ayuda a la iternalización que es el punto para provocar reacciones, cuando la gente reacciona, nuevos elementos pueden ser generados ya sean aumentativos o contradictorios. Estos nuevos elementos pueden surgir gracias al estímulo recibido por la información de *awareness* entregada. Por lo tanto, el *awareness* trabaja como un motor impulsor del *proceso*.

Otra de las características del *awareness*, es la de mantener en un "punto alto" la evolución de una discusión sobre elementos ontológicos contradictorios. Gracias a la información entregada cuando se materializa un conflicto, los usuarios se enteran de las divergencias, esto refuerza la interacción entre las personas, que complementan los conflictos con nuevas alternativas y argumentaciones. La actividad de discusión genera nueva información, que ayuda al grupo a iterativamente, llegar al consenso deseado. El *awareness* mejora el compromiso ontológico, además de ayudar a la internalización, que puede implicar que un usuario traiga elementos a su espacio privado, es útil para promover

a las personas a exteriorizar elementos que se encuentren en su espacio privado y todavía no han publicado.

Por la metodología de diseño utilizada podemos diferenciar dos tipos *awareness*: *awareness* **de actividad** y *awareness* **de sincronización**. El primero involucra directamente las acciones de otros usuarios y ayuda a mantener "activo" el espíritu de colaboración. El segundo surge como consecuencia de tener dos versiones simultaneas de un mismo diseño y sirve para observar (comparar) el estado de un diseño con respecto al otro.

4.4.1. *Awareness* de Actividad

La información que se desprende de las acciones realizadas en el ámbito público, es información de interés para el grupo [BLD03]. Reflejan la actividad del grupo y por consiguiente denotan que el diseño ha evolucionado. Las acciones públicas son realizadas por los usuarios sobre elementos ontológicos del dominio. Son las acciones las que indican los cambios y por lo tanto son ellas la base para dar información, al resto del grupo, de las actividades que ocurre en el diseño.

Para modelar conceptualmente el *awareness* de actividad surgen la necesidad de modelar las acciones, y en particular las acciones que le interesan al grupo son las acciones públicas, por lo tanto deben formar parte del espacio compartido. Ya que las acciones son en sí mismas conocimiento y se relacionan con elementos ontológicos compartidos, se decidió tener un modelo ontológico para representarlas. El modelo ontológico de las acciones captura el conocimiento de las mismas, las relaciones con los elementos ontológicos compartidos es directa ya que "hablan el mismo idioma".

Un usuario se entera de los cambios por medio de *notificaciones*, una notificación refleja que una acción pública ha ocurrido, ya sea la publicación de nuevos elementos o la creación de elementos para la discusión: conflictos, alternativas o argumentaciones.

La *figura 4.9* muestra el modelo conceptual de las acciones y notificaciones, en él se observan las acciones modeladas por la clase abstracta *Acción*. Las especialización de las acciones está dada por las clases *Privada* y *Pública* que modelan acciones privadas y públicas respectivamente.

La clase *Externalizar* modela las acciones que se realizan en un ámbito privado y se pueden dividir en las usuales para la edición de una ontología. Si bien este tipo de acciones no son vista por los otros participantes y no forman parte de la ontología compartida, pertenecen al modelo conceptual de las acciones.

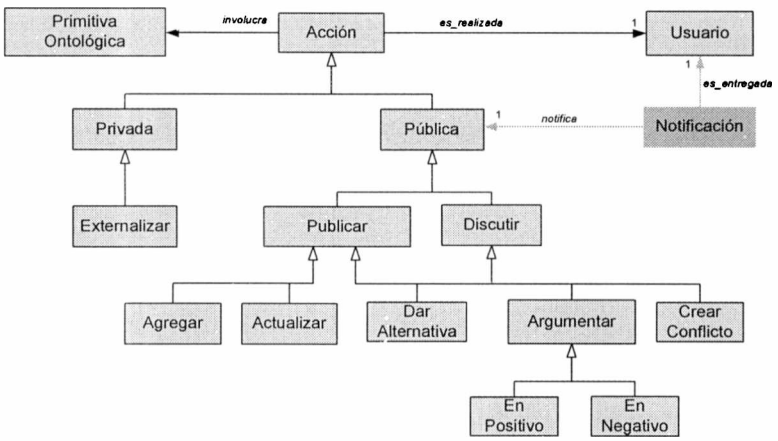


Figura 4.9: Modelo conceptual de las acciones

Las acciones públicas se subclasifican en acciones de publicación (clase *Publicar*) y en acciones pertinentes a la discusión (clase *Discutir*). Las clases *Agregar* y *Actualizar* reflejan las acciones que un usuario realiza cuando publica nuevos elementos ontológicos de su ontología privada a la compartida o actualiza elementos ya existentes.

Las clases *Argumentar*, *DarAlternativa*, *CrearConflicto*, son acciones que agregan información al espacio compartido y por lo tanto son acciones de publicación y, además son elementos de discusión. Estas tres clases se modelan por múltiple herencia siendo subclases de *Discutir* y *Publicar*. La noción de dar una opinión es reflejada por la clase *Argumentar* y la opinión puede ser positiva (clase *EnPositivo*) o negativa (clase *EnNegativo*).

Por último la clase *Notificación*, modela la información que le llegara a un usuario a partir de una acción realizada por otro usuario. Una notificación se relaciona con una acción pública (*notifica*) y con un usuario particular (*es_entregada*)

Las notificaciones deben ser entregadas a los usuarios en el instante que ocurren. Si el usuario está dentro del sistema, cada vez que otro usuario realiza una acción, la misma debe notificarse inmediatamente. Si el usuario no se encuentra dentro del sistema, ni bien ingresa al mismo debe ser notificado de todas las acciones que se realizaron. Puede haber mecanismos auxiliares para "entregar" las notificaciones, por ejemplo pueden ser enviadas por e-mail.

Por otra parte, la información de *awareness* entregada tiene que ser asignada a las necesidades de usuario, cada usuario debe poder seleccionar que información de *awareness* quiere recibir. Por ejemplo un usuario puede querer enterarse sólo de las acciones que realiza otro usuario, o sólo de las acciones sobre un elemento que publicó. El sistema debe proveer un mecanismo para que los participantes elijan sobre que quieren ser informados, la elección se puede mantener dentro del *perfil de un usuario* como se explica a continuación.

Perfil de un usuario

Las acciones que realiza un usuario, conjuntamente con su información personal arman lo que se denomina "el perfil" del usuario. La característica de la información que se ve reflejada en cada perfil puede dividirse en dos: el "perfil social" y el "perfil groupware", el primero reúne los datos personales del usuario. El segundo la información concerniente al trabajo realizado y a inquietudes del propio usuario con respecto a los elementos del dominio, como pueden ser intereses sobre elementos particulares.

Los usuarios editan sus perfiles, el social y el groupware. Pero las acciones que realizan sirven para completar parte del perfil groupware de forma automática. *Alicia* completa sus datos personales y cuando realiza una acción publica, por ejemplo: actualiza la clase *Author*, es el mismo sistema quien va completando su perfil groupware.

Las acciones también sirven para determinar relaciones entre los usuarios y los elementos del dominio. El sistema podría funcionar como una ayuda para indicarle a un usuario cuales son los intereses que posee, basándose por ejemplo en las acciones que realizó. De similar manera las acciones pueden influenciar para determinar que información de *awareness* desea recibir.

En el capítulo 5 sección 5.4.2 se describe en detalle cómo un usuario y el sistema manipulan los perfiles. Y como el usuario puede recibir información de *awareness* de acuerdo a sus necesidades.

4.4.2. Awareness Sincronización

El *awareness* de sincronización surge de tener dos versiones de trabajos simultáneas, y proporciona al usuario información de cómo se encuentran los elementos ontológicos de su espacio privado con respecto al compartido. Esta información es útil para que los usuarios puedan observar en forma simple y rápida el estado de su espacio de trabajo con respecto al espacio común. La siguiente tabla muestra los cinco posibles estados de un elemento que se encuentra en un espacio privado con respecto al espacio compartido, y la noción de igualdad mencionada es la definida en la sección 4.2.3.

El *awareness* de sincronización es un desprendimiento complementario del *awareness* de actividad, también funciona como disparador para mantener el *proceso* activo. Un usuario puede observar que sus elementos de diseño difieren y tomar acciones al respecto, así mismo esta informado constantemente de los elementos que el posee y se encuentran actualmente en conflicto.

Icono	Significado
	El elemento del espacio privado no existe en el espacio compartido
	El elemento del espacio privado existe en el espacio compartido pero no es igual
	El elemento del espacio privado existe en el espacio compartido y es igual
	El elemento del espacio privado existe en el espacio compartido pero no es igual y está en conflicto
	El elemento del espacio privado existe en el espacio compartido, es igual y está en conflicto

Existen diferentes fuentes que modifican la visualización de los elementos de una ontología privada con respecto a la ontología compartida:

- Un usuario edita su ontología privada: en este caso a medida que el usuario modifica o crea elementos en su ontología privada, la visualización va cambiando. Por ejemplo si *Esteban* crea la clase *Person* que ya existe en la ontología compartida, la visualización en su espacio compartido sera: *Person*  (El concepto se encuentra en la ontología compartida, pero no es igual ya que *Esteban* todavía no le agrego ningún *slot*). En este punto el usuario se da cuenta que el concepto ya existe, puede verificar si modela lo que el desea, en caso de ser afirmativo puede traerlo a su espacio de trabajo, en caso de no concordar abre una fuente potencial de conflicto.

- Un usuario modifica la ontología compartida: en este caso la visualización de los elementos de la ontología privada (si es que se encuentran) reflejan los cambios. El usuario que publicó los elementos ontológicos, no le da nayor importancia a la nueva visualización, ya que los elementos publicados desde su ontología privada quedaran iguales () que en la ontología compartida. Si en cambio otro usuario posee esos elementos, verá que cambiaron en la ontología compartida y actuará en consecuencia.

- Un usuario crea un conflicto: la visualización de los elementos de la ontología privada reflejan el conflicto y los elementos se visualizan de forma distinta (). La nueva visualización impulsa a los usuarios a entender que es lo que sucedió y provocar discusiones.

Las fuentes anteriores pueden tener variantes. Lo destacable es ver como lo usuarios de forma simple se pueden dar cuenta de que "algo ocurrió". A partir de este estimulo, pueden identificar las causas que provocaron la modificación en la visualización y seguir acciones particulares a partir de ello.

4.5. Conclusiones

En el capítulo se planteó un nuevo enfoque para diseñar una ontología colaborativamente. Como aspecto fundamental, se definió una forma de soportar de los diferentes puntos de vista dentro del mismo espacio de diseño. La extensión del modelo de representación del conocimiento, permite introducir dentro de una misma ontología, la ontología compartida, elementos que serian inconsistentes con la representación usual.

De similar manera, la ontología compartida posee definidos elementos ontológicos para representar conceptos de colaboración: usuarios, acciones, conflictos, notificaciones, etc. Los nuevos conceptos incluidos en la ontología compartida, dan un soporte para mantener discusiones y son la base para proporcionar *awareness*.

El enfoque que se le dio a las diferentes características que debe tener un editor colaborativo, determinó la creación de ontologías genéricas, que junto con la extensión del modelo de representación de conocimiento, generaron el *meta-modelo* para el desarrollo colaborativo de ontologías. Las opción de tener todos los conceptos modelados dentro de la ontología compartida y no en el lenguaje de implementación de la herramienta, permite que el modelo planteado sea independiente de la herramienta que se utilice (por ejemplo Protégé-2000), y que se pueda definir o ampliar cualquier editor de ontologías basado en *frame*.

Capítulo 5

Co-Protégé

CO-Protégé es un paquete de *plug-ins* para extender Protégé-2000, que respetando los lineamientos generales de Protégé-2000, permite la construcción colaborativa de ontologías soportando el modelo de trabajo planteado en la sección anterior. Se desarrolló teniendo como premisa el soporte de los tres pilares; en CO-Protégé los usuarios diseñan la ontología del dominio que desean modelar siguiendo el *proceso* de diseño colaborativo, plantean diferentes puntos de vista dentro del sistema. Y para fortalecer la actividad colaborativa, CO-Protégé brinda mecanismos de *awareness* ya sea de conocimiento como de sincronización.

El ámbito de trabajo de cada usuario en CO-Protégé es determinado por la edición de dos ontologías: una propia y una común denominadas *ontología privada* y *ontología compartida* respectivamente. La creación y edición de una ontología colaborativamente soportada por CO-Protégé, difiere sensiblemente del método estándar usado en Protégé-2000. Los usuarios no editan en forma directa la ontología compartida, si no que trabajan en una ontología propia (ontología privada), para luego volcar elementos diseñados en la ontología compartida.

Los elementos de la ontología compartida, denominados *frames* colaborativos, extienden a los estándar para poder capturar información propia del diseño colaborativo. Por cada *frame* colaborativo se tiene: la fecha de creación, quién es el autor, si está en conflicto, etc. Para poder modelar el concepto de *frame* colaborativo se definió una nueva arquitectura de meta-clase.

En la sección anterior, se planteo que dentro del espacio compartido se deben encontrar un conjunto de conceptos para apoyar el diseño propuesto: los perfiles de los usuarios, las nociones de conflicto, alternativas, argumentaciones, las acciones realizadas, notificaciones, etc. En CO-Protégé se tomó la decisión de modelar todos los conceptos de forma ontológica y que los mismos convivan en el espacio compartido dentro de la ontología compartida. Esta decisión impacta en el hecho de que no existe una separación en la forma de *modelar el conocimiento*. El modelo de representación del conocimiento del dominio de los

participantes y el propio de la actividad colaborativa es el mismo, un modelo ontológico. Se permite de esta manera que todos los conceptos puedan interactuar en forma directa ya que "hablan" el mismo idioma. Además los conceptos de la actividad colaborativa poseen las características de la representación ontológica.

Los conceptos de la actividad de diseño son modelados para que no causen interferencias y puedan ser diferenciados fácilmente de los conceptos propios del dominio de los participantes. Esto se logra desarrollando con un modelo ontológico que contempla: nombres sintácticos particulares, diferencias en las ramas jerárquicas que se conforman y meta-clases distintas.

La nueva arquitectura de *meta-clase* que modela los *frames* colaborativos, junto con las ontologías genéricas para modelar la actividad de diseño, definen el *Meta-modelo* colaborativo de ontologías y está presente en cualquier diseño de una ontología en forma colaborativa dentro de CO-Protégé.

Cuando el grupo de personas comienzan con la actividad de desarrollar una ontología colaborativa utilizando el prototipo CO-Protégé, lo primero que se realizan es la creación de un proyecto, para luego comenzar con la actividad propiamente dicha. Un proyecto define la forma de acceder al diseño de la ontología y en CO-Protégé todo proyecto cuenta con las siguientes características:

- Un conjunto de usuarios, quienes participan en el diseño de una ontología de dominio común.
- Un espacio privado por cada usuario, a los que soló "el dueño" tiene acceso; contiene la ontología que permite el diseño individual. La edición sigue el método estándar provisto por Protégé-2000.
- Un espacio compartido común a todos los participantes (todos pueden acceder a él). Contiene la ontología que diseñan en conjunto. La edición de esta ontología es indirecta, los usuarios publican elementos diseñados previamente en sus ontologías privadas.
- Una nueva arquitectura de meta-clase para la ontología compartida que permite modelar "*frames* colaborativos".
- En el espacio compartido se encuentra además de la ontología del dominio que se está diseñando, ontologías genéricas (capítulo 2 sección 2.3) que modelan entre otros conceptos: los perfiles de los usuarios, los conflictos, las alternativas, las argumentaciones, las acciones y las notificaciones.
- Un conjunto de *tabs* para acceder a las funcionalidades, tanto para la edición de la ontología privada como para la comunicación entre un espacio privado y el compartido y, para la creación y seguimiento de conflictos, la edición de los perfiles, etc.

- Mecanismos para soportar el *awareness* de conocimiento y *awareness* de sincronización que permitan al grupo estar en conocimiento de lo que sucede con el diseño.

Antes de empezar a describir los detalles de implementación de CO-Protégé, se retoma en la sección 5.1 el análisis de Protégé-2000 desde el punto de vista implementativo. Luego en la sección 5.2 se describe la arquitectura propuesta para implementar CO-Protégé. En la sección 5.3 se explica cómo está compuesto un proyecto en CO-Protégé y cómo se accede al mismo. Para representar la noción de *frames* colaborativos se diseñó un nuevo *meta-modelo* (la arquitectura de meta-clase y los conceptos de edición colaborativa) que caracteriza a la ontología compartida y se explica en la sección 5.4. En la sección 5.5 se explican los diferentes espacios de trabajo y la comunicación ente ellos. La sección 5.7 describe los diferentes mecanismos de *awareness* que provee CO-Protégé. Finalmente se realiza la comparación de CO-Protégé con otros editores y se detallan las herramientas utilizadas para la creación del prototipo.

5.1. Aspectos de implementación de Protégé-2000

Protégé-2000 es un ambiente extensible, independiente de la plataforma para crear y editar ontologías y bases de conocimiento (capítulo 2 - sección 2.8). La edición se realiza manipulando una KB[1] que es la que contiene la ontología; la forma de visualizar a la KB es por medio de un conjunto de *tabs*.

Un proyecto en Protégé-2000 define como están distribuidos físicamente los *elementos* de una ontología particular, alcanza tanto al modelo conceptual como al modelo concreto, es decir todas las clases, los *slots* y las instancias del dominio particular son incluidos en un proyecto. El proyecto es almacenado en un archivo con extensión *pprj* (*Protégé project*), este archivo contiene la información específica del proyecto. La ontología puede ser almacenada en diferentes formatos, por defecto se utiliza *Standard Text Files*, que define dos archivos: el primero con extensión *pont* (*Protégé ontology*) que contiene las clases y los *slots*; el segundo con extensión *pins* (*Protégé instances*) almacena las instancias.

El archivo *pprj* contiene la información referente al formato en el que se almacena la ontología y, en el caso de *Standard Text Files*, contiene las direcciones y los nombres de los archivos involucrados. Los usuarios pueden modificar la visualización de los *frame* de un proyecto, esta especialización también es almacenada en el archivo *pprj*.

Existen otros formatos para almacenar una ontología [?]: JDBC Database, Resource Description Framework (RDF) y el más reciente Web Ontology Language (OWL). Independientemente de cual se elija el usuario interactúa directamente con el proyecto, es decir con el archivo de extensión pprj.

[1] la KB (Knowledge base / base de conocimiento) en un conjunto de clases e instancias

5.2. Arquitectura propuesta

Para la implementación del prototipo CO-Protégé se decidió contemplar la arquitectura original de Protégé-2000 y extenderla por medio de diferentes *plug-ins* que pueden separarse en dos conjuntos:

- *plug-ins* que definen la API[2] de CO-Protégé: permiten tener un nuevo modelo de conocimiento brindando la funcionalidad para acceder a él.
- *plug-ins* que definen la GUI[3] de CO-Protégé: permiten a los usuarios acceder a la funcionalidad de la API de CO-Protégé por medio de diferentes *tabs*.

Finalmente el almacenamiento y acceso "físico" a las ontologías, es por medio de la tecnología cliente/servidor; el server CO-Protégé es el que brinda el acceso remoto a la información pedida desde máquinas clientes. La *figura 5.1* muestra el esquema de la arquitectura CO-Protégé, la linea punteada en la parte superior marca la extensión de la arquitectura realizada para este trabajo de grado.

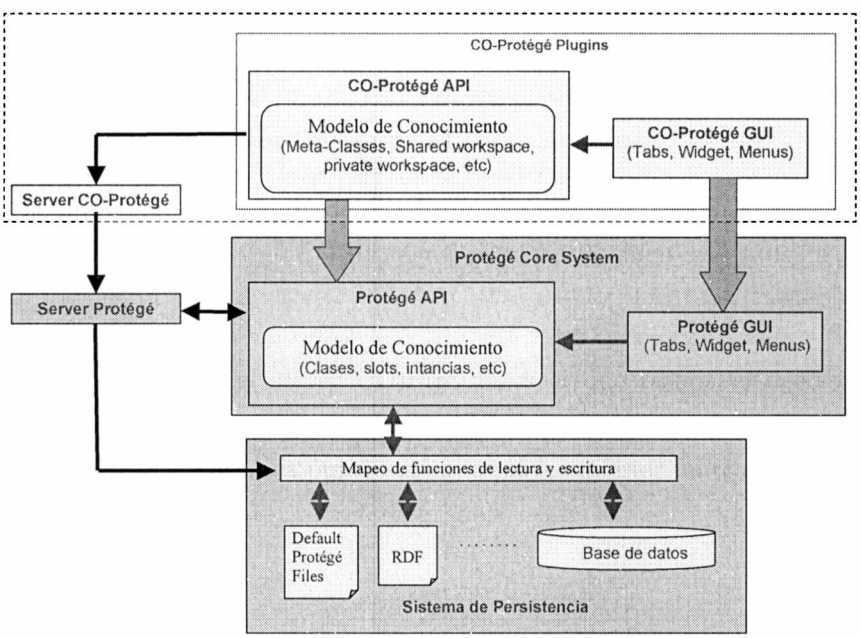


Figura 5.1: Arquitectura de CO-Protégé

La arquitectura propuesta permite, entre otras cosas, utilizar los diferentes *plug-ins* provistos para Protégé-2000 dentro de CO-Protégé, modificar fácilmente el formato de almacenamiento y escribir nuevos *plug-ins* propios del modelo CO-Protégé.

[2]Application Program Interface: conjunto de convenciones que definen cómo debe invocarse una determinada función de un programa desde una aplicación

[3]Graphical User Interface: sistema de interacción programa-usuario a través de "elementos" gráficos

Se mencionó que CO-Protégé es un prototipo, su implementación no es completa. La implementación se restringió al manejo de *clases* y *slots*, dejando el tratamiento de las instancias para una implementación posterior. Siguiendo la arquitectura planteada se pueden ir agregando módulos para contemplar los detalles faltantes.

5.3. Proyecto en CO-Protégé

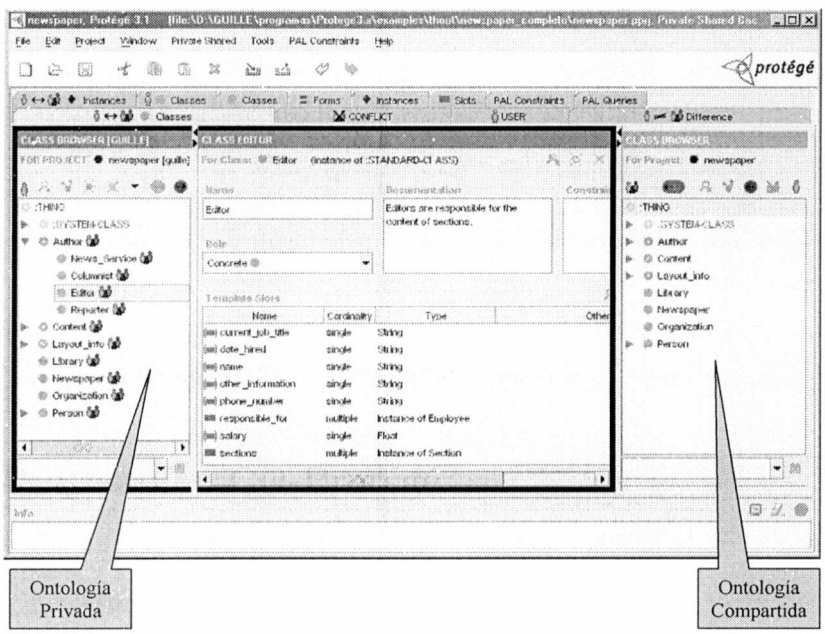


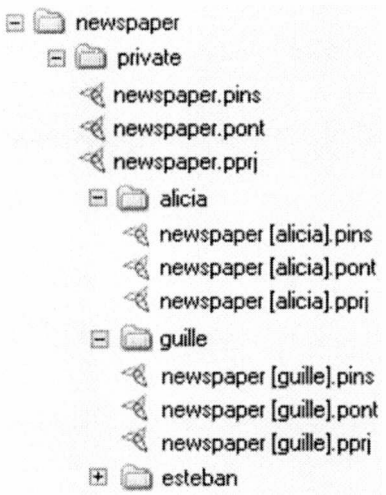
Figura 5.2: *Tab Private_Shared Class*

Los proyectos en CO-Protégé, al igual que en Protégé-2000 definen la forma de acceso que tienen los usuarios para diseñar ontologías. Cada proyecto contiene la información necesaria para alcanzar las diferentes ontologías. El proyecto es almacenado en un archivo con extensión *pprj* (*Protégé project*) y contiene la información específica del proyecto.

Las ontologías son almacenadas en el formato estándar *Standard Text Files*. Si bien los formatos ya están preestablecidos, CO-Protégé puede extenderse fácilmente para permitir utilizar los otros formatos provistos por Protégé-2000.

La visualización de los *frames* de un proyecto se modificó para clarificar los conceptos, la especialización de la visualización también es almacenada en el archivo *pprj*. Por ejemplo una instancia de la clase *User* se visualiza por el valor del atributo (*slot*) *name*.

Si bien no es un requisito, la estructura de archivos que se genera por cada proyecto dentro de CO-Protégé forma un árbol. La carpeta raíz contiene los archivos correspondientes a ontología compartida, hay una subcarpeta por cada usuario conteniendo los archivos de su ontología privada. La estructura facilita la separación de la información permitiendo automatizar el ingreso de nuevos miembros a un proyecto. La imagen de la derecha muestra la estructura de archivos creada para el proyecto *newspaper*.



5.3.1. Acceso a un proyecto

La forma de acceder a la edición de las ontologías (privada y compartida) de un proyecto en CO-Protégé, es usando la tecnología cliente/servidor. Se cuenta con un sistema de permiso básico, provisto por la versión estándar de Protégé-2000, que permite configurar que usuarios pueden editar las diferentes ontologías. Es sistema de permisos es similar a los permisos del sistema operativo UNIX, para manipular los archivos.

La configuración de los permisos de acceso se implementa a través de la ontología *Metaproject*, donde se registran los diferentes usuarios, y por cada proyecto, la ontología compartida y todas las ontologías privadas. Luego se completan los diferentes permisos de accesos.

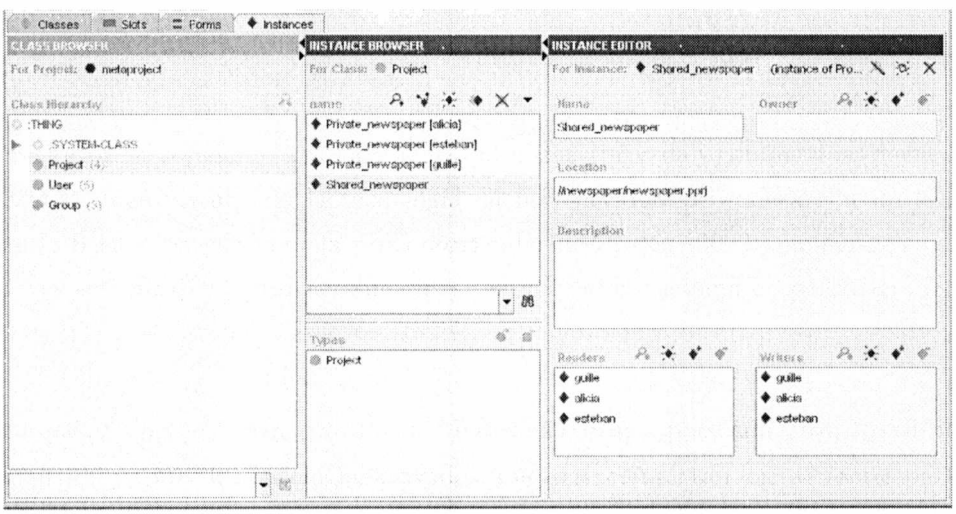


Figura 5.3: MetaProject

La *figura 5.3* muestra la ontología *Metaproject*, donde se pueden observar la clase *Project* que representa a los proyectos. Por cada ontología privada y por cada ontología compartida se crea una instancia de clase *Project*. La clase *User* representa a los usuarios y contiene información sobre el nombre y el *password* para loguerase al sistema. Entre los proyectos y los usuarios existen dos relaciones: una que determina que usuarios tienen permiso para leer (*Readers*) y quienes permiso para escribir (*Writers*). En el ejemplo se ven cuatro instancias que hacen referencia a ontologías del proyecto *newspaper*, una para cada una de las ontologías de los tres usuarios (*Alicia*, *Guille* y *Esteban*) y la cuarta a la ontología compartida. También se observan los permisos de la ontología compartida, tanto en los de escritura como en las de lectura se encuentran los tres usuarios. Para el caso de la ontologías privadas por ejemplo la de *Esteban*, soló él tiene los permisos para la edición y visualización.

Los usuarios ingresan al sistema desde una máquina cliente, se comunican con la máquina servidor, donde se encuentra los proyectos activos. Eligen el proyecto al que desean ingresar (deben tener los permisos previamente definidos) siguiendo el mecanismo habitual de Protégé-2000. Cuando un usuario se *loguea* desde su máquina (cliente) al sistema, pasa a trabajar con dos ontologías simultáneamente, su ontología privada y la ontología compartida, ambas se encuentran en el servidor.

5.4. Meta-Modelo en CO-Protégé

El Meta-Modelo está compuesto por la arquitectura de meta-clase que define cómo se representan los elementos de conocimiento y por diferentes ontologías genéricas que participan en cualquier proyecto.

Los elementos del dominio en ontología compartida, además del concepto que modelan, involucran información sobre el trabajo colaborativo: *quién es el creador de la clase*, *cuándo fue creada la clase*, etc. Las diferencias entre la representación de una clase en la ontología privada y su correspondiente clase en la ontología compartida), se extiende a los diferentes *frames* y define una nueva arquitectura de *meta-clase*.

Para reflejar los diferentes aspectos de interés en un trabajo colaborativo, se necesitan tener modelados a los usuarios, los conflictos, las alternativas, etc. Los nuevos conceptos son modelados también en forma ontológica. Se establece que toda ontología compartida, contiene un conjunto de elementos ontológico que están presentes independientemente del dominio de discusión. Todos los elementos que CO-Protégé incorpora a la ontología compartida poseen un prefijo "*:_*" antes de su nombre (similar a Protégé-2000 que utiliza "*:*") para que los usuarios identifiquen rápidamente los elementos propios del dominio de discusión, de los genéricos. La diferenciación no es solo visual, si no que los *tipos* de los *frames* también son distintos.

5.4.1. Arquitectura de Meta-clases de la Ontología Compartida

Una clase compartida es definida por un nuevo *template* de clase (*Meta-Clase*) que permite agregar los atributos y las relaciones necesarias para controlar la colaboración. Se definió la *meta-clase* *:_STANDARD-SHARED-CLASS* y nuevas relaciones (*slots*) para implementar una clase compartida. A continuación se explican las nuevas *meta-clases*:

- *:_STANDARD-SHARED-CLASS* tipifica a las clases en la ontología compartida, es subclase de *:STANDARD-CLASS* heredando sus *slots* y agregando propios entre los que se destacan:
  - *:_PUBLISH-BY* : relación con el usuario que publicó la clase (quien incorporó el concepto).
  - *:_UPDATE-BY* : relación con el conjunto de usuarios que actualizaron la clase (quienes la extendieron, agregando nuevas relaciones o extendiendo las existentes)
  - *:_ACTIONS* : relación con las acciones que involucran a la clase.
  - *:_IN-CONFLICT* : se utiliza para determinar si la clase pertenece o no a un conflicto, en caso de participar en un conflicto se sabe en cual. Esta relación es utilizada para no permitir su actualización en caso de que efectivamente participe de un conflicto. También determina que la clase pertenece a un solo conflicto.
- *:_STANDARD-SHARED-SLOT* tipifica los *slots* de la ontología compartida, subclase de *:STANDARD-SLOT* y sigue los mismos lineamientos que *:_STANDARD-SHARED-CLASS*.

La nueva arquitectura de meta-clase es una extensión de la estándar, permitiendo la representación de elementos ontológicos de las dos formas (la estándar y la colaborativa). Las diferencia permite identificar los elementos propios de la actividad colaborativa de los que los usuarios modelan, todos los elementos del dominio de discusión son instancias de las meta-clases *:_STANDARD-SHARED-CLASS* y *:_STANDARD-SHARED-SLOT*, mientras el resto de los conceptos son instancias de *:_STANDARD-CLASS* y *:_STANDARD-SLOT*.

Conceptos involucrados en la ontología compartida

Existen diferentes conceptos que están fuertemente relacionados con la edición de una ontología compartida. La propia arquitectura de meta-clases definida para los *frames* compartidos, involucra conceptos ajenos al dominio de discusión como son los *usuarios*, las *acciones*, etc.

Cada proyecto cuenta inicialmente con una cantidad de conceptos definidos en la ontología compartida y son modelados en forma ontológica; en las siguientes secciones se describe la representación de cada uno de ellos.

5.4.2. Usuarios

El concepto de usuario es fundamental dentro de CO-Protégé, con él se modelan a las personas que participan en un proyecto. Como se mencionó en el capítulo anterior un usuario posee características que tienen que ver con su información individual: nombre, e-mail, etc. e información de como se relaciona con la actividad de diseño por ejemplo: "*Guille* creo la clase *Author*", "*Alicia* actualizó la clase *Director*", "*Esteban* está interesado en la clase *Newspaper*". Las relaciones anteriores son entre un usuario y un elemento de diseño, también el usuario puede establecer relaciones con sus pares: "*Guille* está interesado en las actividades que realizan *Alicia* y *Esteban* ".

En CO-Protégé los usuarios se modelan por medio de la clase *:_USER* que incluye entre otros los *slots* del cuadro 5.1. Las relaciones que contiene un usuario son las que determinan el perfil de un usuario.

Nombre del *slot*	Descripción	Tipo	*Valores*	#
:_AWARENESS-WIDGET	atributo que determina que notificaciones desea recibir el usuario	Symbol	*:_ALL* *:_NONE* *:_USER-INTERESTED* *:_INTERESTED*	1:1
:_INTERESTED	denota el "interés" del usuario sobre los conceptos del dominio de discusión o con otros usuarios.	Instance	*:_STANDARD-SHARED-CLASS* *:_STANDARD-SHARED-SLOT* *:_USER*	0:*
:_NOT-INTERESTED	relación con los conceptos del dominio de discusión o con otros usuarios, denota el "no interés" explícito de la persona.	Instance	*:_STANDARD-SHARED-CLASS* *:_STANDARD-SHARED-SLOT* *:_USER*	0:*
:_ACTIONS	relación con las acciones que realizó	Instance	*:_ACTION*	0:*
:_HAS-PUBLISHED	relación con los elementos ontológicos que el usuario publicó, son los *frames* agregados por primera vez.	Instance		0:*
:_HAS-UPDATE	relación con los elementos ontológicos que el usuario actualizó, los *frames* ya existían en la ontología compartida.	Instance		0:*
:_NOTIFICATIONS	relación con la notificaciones que recibe. Los usuarios se enteran de los cambio del diseño recibiendo notificaciones.	Instance	*:_NOTIFICATION*	0:*
:_NAME	determina el nombre de la persona como un identificador único, es un ejemplo de la información individual.	String		0:1
:_E-MAIL	determina e-mail de la persona, que es utilizado en caso del que el sistema genere notificaciones y sean enviadas por esta vía	String		0:1

Cuadro 5.1: *Slots* principales de la clase *:_USER*

Si observamos con detenimiento las relaciones podemos observar restricciones en cuanto a los valores que se permiten. En la relación *:_INTERESTED* por ejemplo la restricción es a elementos propios del dominio de discusión e instancias de *:_USER* y se logra restringiendo la relación a instancias de *:_STANDARD-SHARED-CLASS*, *:_STANDARD-SHARED-SLOT* o *:_USER*. Las restricciones se pueden observar en la mayoría de las relaciones y es factible gracias a la arquitectura de meta-clases definida ya que podemos diferenciar los *frames* colaborativos del resto.

Los usuarios a través de instancias de la clase *:_USER* mantienen el perfil de la persona, ya sea el social como el colaborativo. El dinamismo del perfil se basa en la generación de los elementos ontológicos que se relacionan con la instancia particular de un usuario, en la sección 5.6.3 se explica como el sistema y los usuarios completan los perfiles.

5.4.3. Conflictos

La forma que los participantes tienen de comunicar al resto de la comunidad su disconformidad con los conceptos de la ontología de dominio compartida, es mediante la creación de conflictos. Los conflictos son modelados por medio la clase *:_CONFLICT*, agrupa a todos los elementos que conforman un conflicto, poseen un tipo para determinar la naturaleza del conflicto, quién es el creador, una descripción de la motivación y un conjunto de alternativas. El cuadro 5.2 describe los *slot* más importantes de la clase *:_CONFLICT*

Nombre del *slot*	Descripción	Tipo	*Valores*	#
:_CONFLICT-TYPE	representa el tipo de conflicto, no es utilizado en esta versión.	Symbol	1, 2, 3, 4	0:1
:_INVOLVED-CONCEPTS	es una relación con los elementos que conforman un conflicto.	Instance	*:_STANDARD-SHARED-CLASS* *:_STANDARD-SHARED-SLOT*	0:*
:_AUTHOR	relación con la persona que creó el conflicto, quien abrió la discusión.	Instance	*:_USER*	0:1
:_ALTERNATIVES	representa las alternativas asociadas al conflicto.	Instance	*:_ALTERNATIVE*	0:*
:_DOCUMENTATION	descripción de qué motivo el conflicto, puede verse como la pregunta inicial del modelo de discusión IBIS.	String		0:1

Cuadro 5.2: *Slots* principales de la clase *:_CONFLICT*

La relación *:_CONFLICT-TYPE* no es utilizada, pero representa la noción de que los conflictos pueden ser tipificados, por ejemplo el conflicto podría ser de:

- Jerarquía: el usuario no está de acuerdo con la relación *ES-UN*.
- Relación: el usuario no está de acuerdo con un *slot* propio de una clase.
- Nombre: el usuario concuerda con el concepto pero no con el nombre del mismo.

- Existencia: el usuario no está de acuerdo que el concepto, determina que no debe pertenecer a la ontología.
- Tipo: el usuario no está de acuerdo con el tipo de una relación.
- Otros

5.4.4. Alternativas

Fuertemente ligada a los conflictos están las alternativas. Los usuarios pueden en su espacio privado crear una nueva representación, un punto de vista distinto, a un conflicto planteado. También pueden dar a conocer él nuevo enfoque al resto de los participantes. El usuario publica su diseño en la ontología compartida como alternativa al conflicto. Las alternativas son modeladas por la clase *:_ALTERNATIVE* el cuadro 5.1 describe los *slots* más importantes.

Nombre del *slot*	Descripción	Tipo	*Valores*	#
:_ARGUMENTATIONS	relación con las argumentaciones que brindan los usuarios.	Instance	ARGUMENTATION	0:*
:_AUTHOR	relación con la persona que publicó la alternativa.	Instance	*:_USER*	0:1
:_INVOLVED-CONCEPTS-ALTERNATIVE	relación con los elementos que conforman la alternativa.	Instance	*:_ALTERNATIVE-CONFLICT-CLASS* *:_ALTERNATIVE-CONFLICT-Slot*	0:*
:_ALTERNATIVE-TO	relación que determina para que conflicto es la alternativa.	Instance	*:_CONFLICT*	0:*
:_:DOCUMENTATION	descripción de la alternativa, puede verse como una respuesta a la pregunta inicial del modelo de discusión IBIS.	String		0:1

Cuadro 5.3: *Slots* principales de la clase *:_ALTERNATIVE*

Una alternativa en la ontología compartida modela una conceptualización diferente a un conflicto, los elementos que forman parte en las alternativas no son elementos que puedan ser incorporados directamente al dominio de discusión, pero integran el espacio compartido y por tal motivo se crean como instancias de las nuevas *meta-clases*: *:_ALTERNATIVE-CONFLICT-CLASS* y *:_ALTERNATIVE-CONFLICT-SLOT* para las clases y *slots* respectivamente. Cuando una clase es creada como instancia de la meta-clase *:_ALTERNATIVE-CONFLICT-CLASS* la super-clase por defecto es *:_THING-ALTERNATIVE*.

Las meta-clases a la que pertenece una clase definen su tipo, por lo tanto se puede diferenciar cuales son los elementos usados en alternativas, de los que propios de la ontología del dominio compartida. Como los *frames* alternativos, tiene un nombre interno no visible para el usuario y un nuevo atributo para almacenar el nombre real, se permite la utilización de nombres replicados (que no están permitidos en el protege-2000).

5.4.5. Argumentaciones

Las alternativas poseen un conjunto de argumentaciones, que realizan los diferentes usuarios, son modeladas por la clase abstracta *:_ARGUMENTATION* subclase de la clase *:NOTIFICATION* (definida en Protégé-2000) y sus *slots* más importantes se describen en el cuadro 5.4.

Nombre del *slot*	Descripción	Tipo	*Valores*	#
:_CREATOR-USER	representa al usuario que creó la argumentación	Instance	*:_USER*	0:1
:_ANNOTATED-INSTANCE	relación con la alternativa que se esta argumentando, heredada	Instance	*:_ALTERNATIVE*	0:1
:_ANNOTATION-TEXT	es la descripción de la argumentación, heredada	String	*:_USER*	0:1

Cuadro 5.4: *Slots* principales de la clase *:_ARGUMENTATION*

Las argumentaciones expresan una posición de un usuario con respecto a un conflicto, pueden ser positivas o negativas. Esto se modela por medio de las clases *:_POSITIVE-ARGUMENTATION* y *:_NEGATIVE-ARGUMENTATION* que son subclases de *:_ARGUMENTATION*.

5.4.6. Acciones

Las acciones que realizan los participantes sobre la ontología compartida (publicar o actualizar un elemento; crear conflictos, alternativas o argumentaciones; navegar el conocimiento) son de importancia dentro del contexto colabarotivo. De las acciones se pueden sacar conclusiones de nuevos intereses ya sean en el dominio de discusión como con otros usuarios, se pueden establecer experticias y son utilizadas como el disparador del mecanismo de *awareness*. Son modeladas por medio de instancias de alguna subclase de *:_ACTION* y sus *slots* principales se muestran en el cuadro 5.5.

Nombre del *slot*	Descripción	Tipo	*Valores*	#
:_ACTION-TIMESTAMP	descripción del instante en que se realizó la acción	String		0:1
:_AUTHOR-ACTION	relación con el usuario que realizó la acción	Instance	*:_USER*	0:1
:_INVOLVED-CONCEPT	elementos del dominio involucrados en la acción	Instace	*:_STANDARD-SHARED-CLASS* *:_STANDARD-SHARED-SLOT*	0:1

Cuadro 5.5: *Slots* principales de la clase *:_ACTION*

Las acciones pueden ser catalogadas como acciones para publicar, actualizar, crear un conflicto, dar una alternativa o dar una argumentación, y se corresponden con las clases *:_PUBLISH-ACTION*, *:_UPDATE-ACTION*, *:_CONFLICT-ACTION*, *:_ALTERNATIVE-ACTION* y *:_ARGUMENTATION-ACTION*, todas subclases de *:_ACTION*.

Las acciones se utilizan además para tener un historial de las actividades que se fueron realizando y, junto con las instancias de la clase *:_NOTIFICATION*, son los elementos base para el mecanismo de *awareness*.

5.4.7. Notificaciones

La clase *:_NOTIFICATION* modelan la forma en que los usuarios se notifican de las acciones de los otros, una notificación se relaciona con la acción que se va a notificar, quien es el usuario destinatario de la notificación y un estado que permite al usuario receptor identificar si ya fue leída o no.

Nombre del *slot*	Descripción	Tipo	*Valores*	#
:_N-ACTION	relación con la acción a ser notificada	Instance	*:_ACTION*	0:*
:_N-USER	relación con el usuario a quien le llega la notificación	Instance	*:_USER*	0:*
:_N-STATE	estado de la notificación	Symbol	*:_READ, :_UNREAD*	0:*

Cuadro 5.6: *Slots* principales de la clase *:_NOTIFICATION*

Si un usuario realiza una acción pública, el sistema se encarga de crear las instancias necesarias de la clase *NOTIFICATION* y las relaciona con los usuarios que deseen ser notificados.

A la clase *NOTIFICATION* se le puede agregar un atributo que sea descripción. El nuevo atributo permitiría tener una descripción proporcionada por el sistema, y luego el usuario modificarla según su propio criterio.

Modelo completo

Para finalizar la *figura 5.4* muestra la jerarquía de clases completa del nuevo meta-modelo definido en CO-Protégé. En la figura se pueden diferenciar las clases utilizadas para representar la arquitectura de meta-clase (cuadros mas oscuros), las clases de las ontologías genéricas (cuadros claros) y las clases definidas en Protégé-2000 (cuadros sin color).

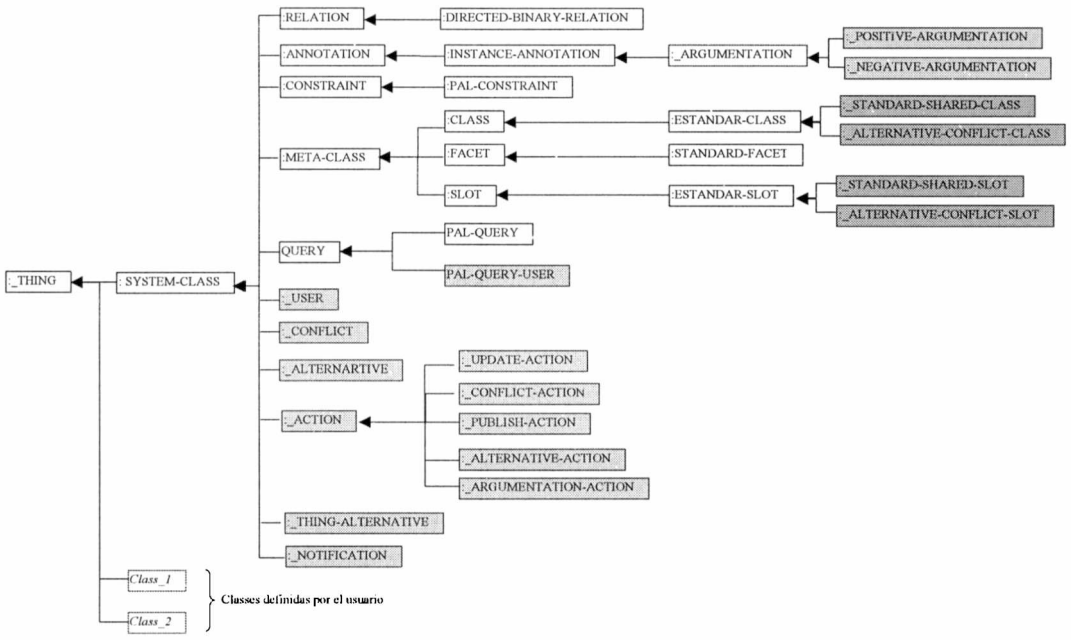


Figura 5.4: *Meta-modelo* de CO-Protégé

5.5. Espacios de trabajo

En CO-Protégé los usuarios cuentan con dos espacios de trabajos. Los elementos ontológicos del dominio que diseñan los usuarios difieren en la ontología privada y en la ontología compartida en cuanto a su representación de meta-clase.

La representación también es diferente entre los *frames* que conviven en la ontología compartida, mientras que los elementos ontológicos que modelan los usuarios sigue la nueva arquitectura de meta-clase, modelándolos como *frame* colaborativos. Los elementos que se crean para la actividad colaborativa: usuarios, acciones, notificaciones, etc. se modelan bajo la arquitectura de meta-clase estándar.

La comunicación entre un espacio privado y el espacio compartido es en ambas direcciones y está dada por diferentes acciones. Las acciones disponibles se encuentran en los diferentes *tabs*.

5.5.1. Representación de los elementos ontológicos

Un usuario crea un elemento en su ontología privada, por ejemplo *Guille* crea la clase *Author*, la representación de la clase *Author* es establecida por la meta-clase *:STANDAR-CLASS*. Cuando *Guille* publica *Author* el sistema crea la clase en la ontología compartida como instancia de la meta-clase *:_STANDAR-SHARED-CLASS*.

Luego de publicar un *frame* desde una ontología privada a la ontología compartida,

la conceptualización que modelan es igual en ambas ontologías, pero su representación a nivel de meta-clase difiere. El elemento en la ontología privada es representado como un *frame* estándar y en la ontología compartida es representado como un *frame* colaborativo.

La siguientes tablas muestran la diferencia entre el mismo concepto, la clase *Author* luego de ser publicada desde una ontología privada a la ontología compartida. La tabla de la izquierda muestra la representación en la ontología privada y la tabla de la derecha la representación en la ontología compartida.

Nombre del *slot*	Valor
:DIRECT-INSTANCES	
:DIRECT-SUBCLASSES	*Editor, Reporter, ...*
:DIRECT-SUPERCLASSES	*:THING*
:DIRECT-TEMPLATE-SLOTS	*name*
:DIRECT-TYPE	*:STANDARD-CLASS*
:DOCUMENTATION	
:NAME	*Author*
:ROLE	*Abstract*
:SLOT-CONSTRAINTS	

Ontología Privada

Nombre del *slot*	Valor
:DIRECT-INSTANCES	
:DIRECT-SUBCLASSES	*Editor, Reporter, ...*
:DIRECT-SUPERCLASSES	*:THING*
:DIRECT-TEMPLATE-SLOTS	*name*
:DIRECT-TYPE	*:_STANDARD-SHARED-CLASS*
:DOCUMENTATION	
:NAME	*Author*
:ROLE	*Abstract*
:SLOT-CONSTRAINTS	
:_PUBLISHED-BY	*Guille* [Instancia *:_USER*]
:_UPDATE-BY	
:_ACTION	*PublishGuille* [Instancia *:_ACTION*]
:_IN-CONFLICT	

Ontología Compartida

Se puede observar que ambas clases coinciden en la conceptualización: sus nombres, su rol, sus instancias, super-clases, sub-clases y *template slot* poseen los mismos valores. Difieren en *:DIRECT-TYPE* ya que pertenecen a diferentes meta-clases. Además la representación en la ontología compartida de la clase *Author* posee más relaciones *:_PUBLISHED-BY*, *:_ACTION*, ect.

Esta diferencia sucede con cada uno de los *frames* colaborativos con respecto a los privados. Para el usuario es transparente la forma en que se crean los *frames*, el sistema se encarga de crearlos según cual fuere la ontología destino.

5.5.2. Comunicación entre los espacios

Una vez dentro del sistema el usuario cuenta con los dos espacios y pueden trabajar en cada uno de ellos de forma indistinta, los *tabs* proveen la visualización y la funcionalidad para manipular las diferentes ontologías y se describen en sección 5.6.

El *cuadro 5.7* muestra las acciones disponibles para la comunicación entre un espacio privado y el espacio compartido.

Icono	Nombre	Significado
	*Publish*	publica en la ontología compartida los elementos ontológicos seleccionados en la ontología privada, si pasan el chequeo de publicación aumentativa.
	*Update*	actualiza en la ontología compartida los elementos ontológicos seleccionados en la ontología privada, si pasan el chequeo de publicación aumentativa.
	*Publish/Update Anyway*	permite realizar la publicación o actualización cuando la publicación o actualización desde la ontología privada hacia la ontología compartida no satisface el chequeo de publicación aumentativa. La incompatibilidad que determina que no pase el chequeo puede ser sorteada eliminado/modificando alguno de los elementos que conforman la acción. Las modificaciones realizadas no afectan la ontología privada, por lo tanto luego de realizar la acción alguno de los elementos que participaban en la misma, no serán iguales en las dos ontologías.
	*Rebuild*	reconstruye la ontología privada con todos los elementos de la ontología compartida. Generalmente cuando un usuario ingresa por primera vez al sistema la ontología compartida contiene un conjunto de elementos del dominio ya definido, usar esta acción permite a la ontología privada traer todos los conceptos. Si la acción se aplica cuando la ontología privada contiene elementos, los elementos de la ontología privada que provoquen que la publicación no es aumentativa serán eliminados, de forma que la ontología privada incluya todos los conceptos del dominio de la ontología compartida.
	*Publish/Update*	publica en la ontología privada los elementos ontológicos seleccionados en la ontología compartida, siempre que superen el chequeo de publicación aumentativa.

Cuadro 5.7: Acciones entre los diferentes espacios de trabajo

Las acciones que involucran modificaciones en la ontología compartida determinan que en ella se creen otros elementos que son utilizados para completar las relaciones que involucran a un *frame* colaborativo. Varias de las relaciones a completar necesitan crear nuevas instancias, por ejemplo de alguna sub-clase particular de *:_ACTION*. Otras relaciones se forman con instancias ya creadas, por ejemplo la relación *:_AUTHOR-ACTION*, es establecida con alguna instancia de *User*. Hay atributos que no involucran otros elementos ontológicos y se completan en forma simple como *:_TIMESTAMP*.

Un ejemplo de las relaciones que genera el sistema se puede ver en la *figura 5.5*. En ella se observa un esquema parcial de clases (cuadrados) e instancias (elipses) luego de que *Alicia* actualiza la clase *Author* publicada por *Guille*. Las líncas remarcadas son las nuevas relaciones y las nuevas instancias creadas por el sistema.

El sistema completa la relación (*slot*) *:_HAS-UPDATE* de la clase *Author*, agregando como valor del *slot* la instancia *Alicia*. Crea una instancia *upd_n* de la clase *:_UPDATE-ACTION* y completa las relaciones de la misma. Relaciona a *upd_n* con *Author* por medio de *:_INVOLVED-CONCEPT* y con *Alicia* por *:_AUTHOR-ACTION*.

Como *Guille* determinó que tiene interés en la actividad de *Alicia*, se genera la instancia not_n de la clase *:_NOTIFCATION*, y se la relaciona con la instancia *Guille* por medio del *slot :_N-USER* y con *upd_n* por el *slot :_ACTION* .

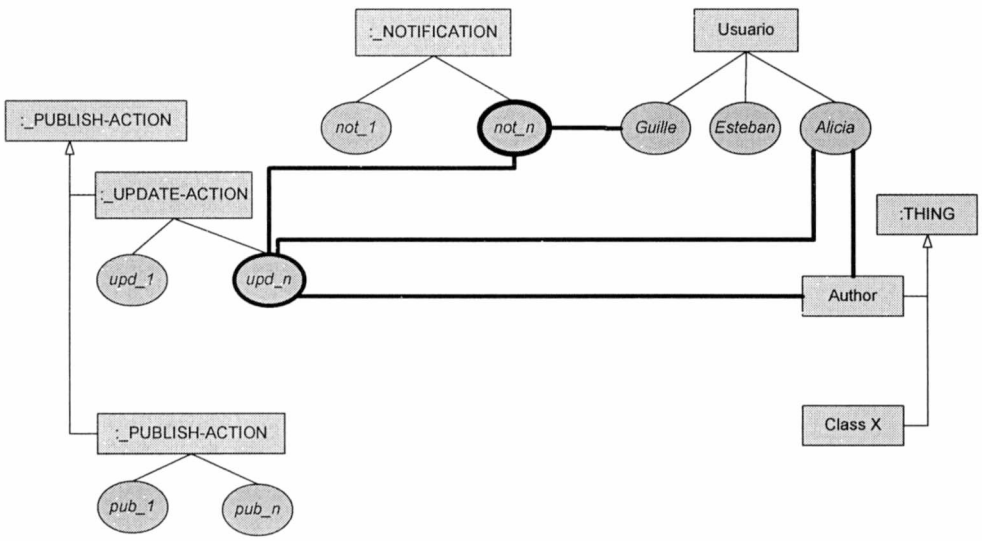


Figura 5.5: Clases e instancias parciales, luego de la actualización de *Author*

Los elementos ontológicos creados para capturar el conocimiento de la actividad colaborativa son representados con la arquitectura de meta-clase estándar de Protégé-2000.

5.6. Tab-widget *plug-ins*

Los *tabs* son la forma de comunicarse con la API CO-Protégé, visualizan las ontologías y dan acceso a la funcionalidad de diseño. Se implementaron por medio de *plug-ins* llamados Tab-widget *plug-ins*.

Se puede trabajar en el contexto de la ontología privada o en de la ontología compartida, para esto se cuentan con los *tabs* estándar (cls, *slot* e instancias) provistos en Protégé-2000. En el caso de trabajar soló con la ontología privada, la funcionalidad es la estándar. En cambio si se trabaja con la ontología compartida, la funcionalidad se restringe a las acciones de navegación (no se permite la edición directa) y a la creación de conflictos.

5.6.1. *Tabs* que involucran dos ontologías

Los *tabs*: *Private_Shared Instance*, *Private_Shared Slots* y *Private_Shared Class* permiten trabajar simultáneamente con los dos espacios de trabajo. En ellos los paneles de la izquierda se corresponden a la ontología privada y los de la derecha a la ontología compartida, el panel central es sensible a la ontología que se manipula. El usuario se da cuenta cual es el contexto de trabajo en base a aspectos visuales, el recuadro en negro, el color de la barra en el título, determinan cual es el contexto activo. El cambio entre

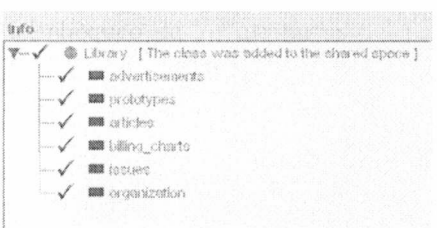
los espacios de trabajo es simplemente seleccionando un elemento de los paneles laterales. La *figura 5.2* muestra el tab *Private_Shared Class*, que permite trabajar con las clases de ambas ontologías simultáneamente.

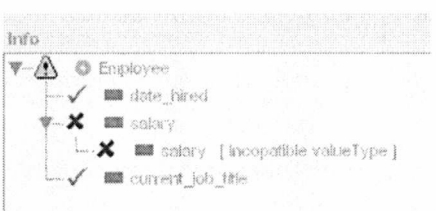
La funcionalidad de comunicación está dada en las barras de tareas correspondientes a cada panel y se habilitan solo cuando pueden realizarse, por ejemplo la acción *Publish* se habilita cuando el conjunto de clases seleccionadas no se encuentran en la ontología privada. Todas las acciones de comunicación, tienen un chequeo previo de consistencia, en la parte inferior de los *tabs* se encuentra un panel que muestra la información de los resultados de las acciones que se realizan entre ambos espacios. Los diferentes resultados se pueden recorrer en forma de árbol, donde las raíces involucran los conceptos seleccionados en la acción y cada subárbol indica el resultado del chequeo de los atributos del padre. Hay tres tipos de mensajes:

1. ✓ Indica que la acción se realizó con exito si se encuentra en la raíz. Si es en un sub-nodo interno indica que no provoca inconsistencias.
2. ⚠ Indica que la acción tiene problemas de inconsistencia, pero que se puede forzar su realización (la acción no se realizó)
3. ✘ Indica que la acción no puede ser realizada si se encuentra en la raíz. En caso de que se encuentre en un sub-nodo, indica que si se fuerza la acción del padre, los elementos involucrados en este nodo no participaran de la acción.

A continuación se muestran dos posibles ejemplos de los resultados de las acciones:

La figura de la izquierda muestra el resultado del chequeo de la publicación de la clase *Library*. Se puede observar que se ha publicado correctamente ya que la clase y sus atributos dieron correcto el chequeo.





En este caso la acción que se intentó realizar fue la actualización de la clase *Employee*. Se observa una incompatibilidad en el *slot salary* que posee un tipo distinto en ambas ontologías (en la privada es de tipo *String* y en la compartida de tipo *Int*). En este punto el usuario podría decidir retomar la acción de todas manera, en cuyo caso el sistema realiza la actualización sin tener en cuenta el *slot* incompatible.

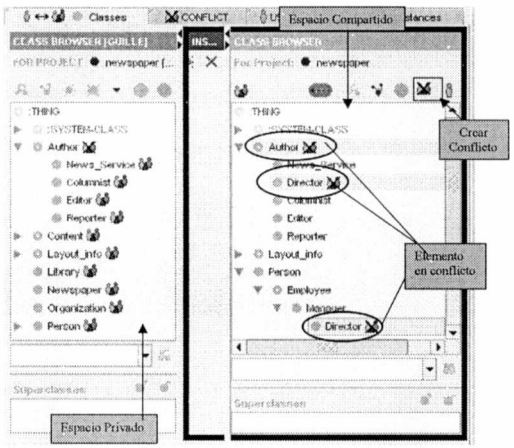


Figura 5.6: Creación de Conflictos

Los *tabs* que trabajan con ambas ontologías, además de la funcionalidad de comunicación, permiten expresar conflictos. Los conflictos se crean seleccionado los elementos correspondientes en la ontología compartida y presionando el ícono situado en la barra de tareas del espacio compartido. El usuario debe darle un nombre al conflicto y a partir de ese momento los elementos ontológicos involucrados no pueden ser modificados por ninguna acción. En la *figura 5.6* se observa nuevamente la situación planteada en la sección 4.3.1 del capítulo 4, donde el usuario *Guille* creaba un conflicto.

5.6.2. Conflict Tab

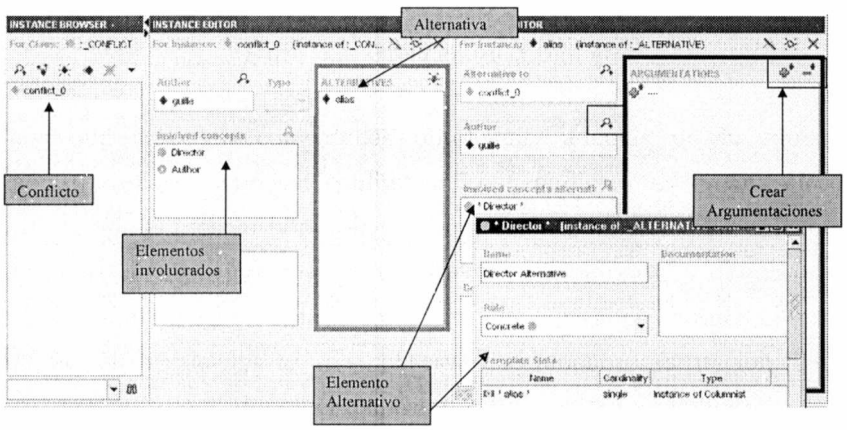


Figura 5.7: *Tab Conflict*

Los conflictos se pueden visualizar y manipular desde un *tab* especial llamado *Conflict* y permite a los usuarios "navegar" los conflictos. Por cada uno de los conflictos se puede

observar los elementos involucrados, las motivaciones que llevaron al autor a realizarlo, el conjunto de alternativas y las argumentaciones que posee.

La *figura 5.7* muestra como es la visualización de los conflictos en CO-Protégé. El usuario selecciona un conflicto de la lista de conflictos (izquierda de la ventana), puede observar los datos del conflicto que seleccionó: elementos involucrados, autor, tipo, etc. (parte central de la ventana). La lista de borde claro en la parte central permite ver las alternativas creadas para el conflicto seleccionado, los datos de la alternativa son mostrados en la derecha de la ventana.

El *tab* permite crear alternativas, cuando un usuario crea una alternativa, se abre una nueva ventana para que ingrese los elementos ontológicos que formaran parte de la alternativa, los elementos provienen de la ontología privada del participante.

Si un usuario quiere dar su opinión sobre alguna alternativa, crea una nueva argumentación negativa o positiva, presionando el botón correspondiente en la parte superior derecha de la pantalla.

La creación de una alternativa implica traer elementos ontológicos al espacio compartido que, si bien deben estar en este espacio, no forman parte de la ontología de dominio. Además estos elementos pueden llevar a la ontología compartida a un estado de incoherencia, ya que probablemente no pasarían el chequeo de publicación aumentativa. Los conceptos alternativos se crean como elementos que pertenecen a una meta-clases distinta a la de los *frames* colaborativo, la meta-clases utilizadas para tal fin son *:_ALTERNATIVE-CONFLICT-CLASS* y *:_ALTERNATIVE-CONFLICT-SLOT*. Ambas son como una "burbuja" que envuelve a los conceptos originales, tiene un nombre interno dado por el sistema y la relación *:_REAL-NAME* de tipo *String* para ver el nombre que estaba en el *frame* original.

En el ejemplo del capítulo 4 sección 4.3, *Guille* publicó una alternativa de la clase *Director*, actualizada anteriormente por *Alicia*. La siguiente tabla compara los template *slots* de las clases *Director* y la clase alternativa para *Director* publicada por *Guille*. Ambas clases pertenecen a la ontología compartida, la primera además es modelada como un elemento colaborativo y forma parte de la ontología de dominio compartida

Nombre del *slot*	Valor
:DIRECT-SUPERCLASSES	*Author, Manager*
:DIRECT-TEMPLATE-SLOTS	
:DIRECT-TYPE	*:_STANDARD-SHARED-CLASS*
:NAME	*Author*
:_PUBLISHED-BY	*Guille* [Ins *:_USER*]
:_UPDATE-BY	*Alicia* [Ins *:_USER*]
:_ACTION	...
:_IN-CONFLICT	*Conflict_1* [Ins *:_CONFLCIT*]

Clase *Director* en la ontología compartida

Nombre del *slot*	Valor
:DIRECT-SUPERCLASSES	**Manager**
:DIRECT-TEMPLATE-SLOTS	**alias**
:DIRECT-TYPE	*:_ALTERNATIVE-CONFLICT-CLASS*
:NAME	*Internal_Name_1001*
:_REAL-NAME	*Director*

Clase Alternativa *Director* en la ontología compartida

Lo primero que se destaca es que las clases comparadas pertenecen a diferentes meta-clases, esto determina que el valor de *:DIRECT-TYPE* sea distinto. La clase alternativa tiene un nombre interno definido por el sistema y posee una relación que indica cual es el nombre real. Los *slots* y los elementos que involucren, las super-clases, etc. de la clase alternativa, también son modeladas como *frames* alternativos. Por ejemplo el *slot alias* que no se encontraba en la ontología del dominio, y podía ser agregado sin problemas, igualmente es representado como una alternativa, ya que forma parte de ella.

5.6.3. User Tab

Los usuarios visualizan y editan sus perfil por medio del *tab User*. El usuario cuenta con un perfil social que manipula directamente. El perfil groupware es dinámico y se modifica con el transcurso de las actividades que un usuario realiza, el sistema completa y actualiza las relaciones: los elementos que publicó, que actualizó, notificaciones como se explicó en la sección 5.5.2. La creación de nuevas instancias y el armando de las relaciones que involucran las acciones son realizadas por el sistema, pero otras deben ser editadas por el propio usuario como los intereses y sus datos personales.

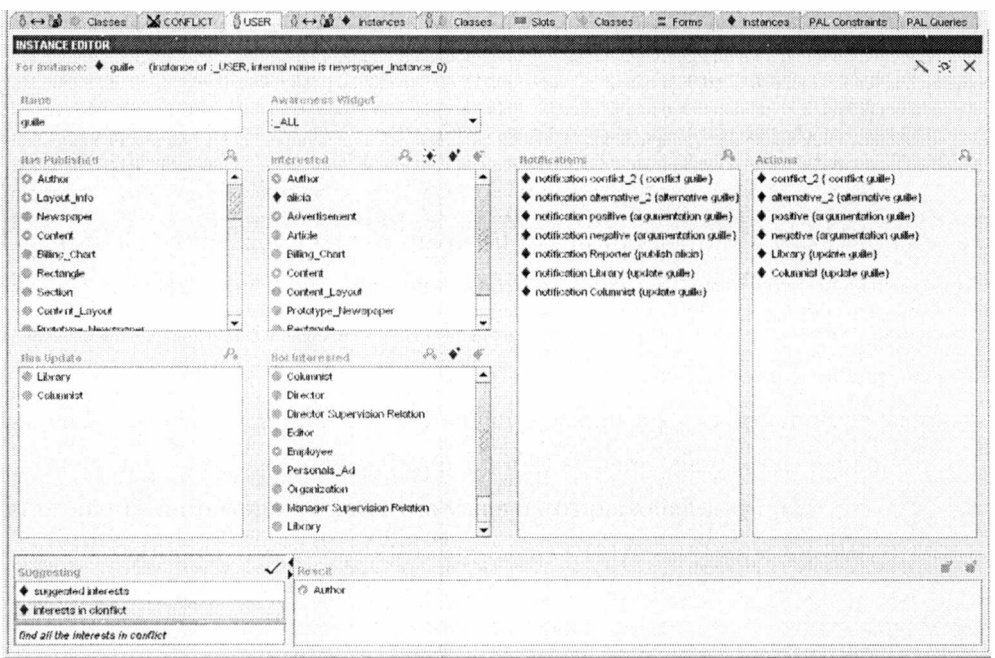


Figura 5.8: *Tab USER*

La *figura 5.8* muestra la ventana donde el usuario puede modificar su perfil, varios de los elementos que se muestran fueron creados por el sistema. Otros como el valor de *slot Awareness Widget* es editado directamente por el usuario, el valor de este *slot* determina cuales son las notificaciones que el usuario va a recibir.

Es importante destacar que las acciones pueden ser utilizada cómo una guía para un usuario. CO-Protégé provee un conjunto de "sugerencias" escritas en PAL que ayudan a completar el perfil; por ejemplo "suggested interests" intenta determinar los posibles elementos de la ontología del dominio que el usuario podría querer agregar en sus intereses; la decisión final es tomada explícitamente por el usuario. Las consultas pueden ser extendidas fácilmente, cualquier usuario puede incorporar una nueva consulta al sistema y el resto tener acceso, la consultas creadas deben ser instancias de la clase *:_PAL-QUERY-USER*.

La clase *:_PAL-QUERY-USER* es una sub-clasificación de la clase PAL-QUERY y fué creada para ejecutar consultas que amplía la sintaxis lenguaje PAL, permitiendo incorporar la sentencia: **#user** que representa al usuario actual del sistema.

```
(defrange  ?cls   :FRAME  :_STANDARD-SHARED-CLASS)
(defrange  ?user  :FRAME  :_USER)

(findall  ?cls
    (exists ?user
    (and
       ( =  (:_NAME ?user) #user )
            (:_PUBLISHED_FOR ?cls ?user)
       (not (:_INTERESTED  ?user ?cls))
       (not (:_NOT_INTERESTED  ?user ?cls)))))
```

La consulta anterior llamada "suggested interests", retorna todas las clases publicadas por el usuario actual que no estén ni en sus *intereses*, ni en sus *no intereses*. Las diferentes sugerencia pueden verse en la parte inferior de la *figura 5.8*.

5.7. Awareness en CO-Protégé

CO-Protégé brinda *awareness* de sincronización permitiendo a los usuarios observar cómo se encuentran los elementos de sus ontologías privadas con respecto a la compartida.

La visualización de los elementos de la ontología privada sigue los lineamientos especificados en el capítulo 4 sección 4.4.2. Un *frame* en la ontología privada modifica su visualización inmediatamente ocurre un cambio, ya sea por que el usuario lo edita en su ontología privada o por que alguien lo modifica en la ontología compartida. La visualización consiste en agregar a los *frames* de la ontología privada un icono que identifica los posibles estados. En los diferentes *tabs* que contienen la ontología privada se pueden observar la nueva visualización de los *frame*.

La visualización da una idea rápida de lo que ocurre con un *frame* particular. El usuario se da cuenta por ejemplo de que un concepto pertenece a ambas ontologías y además si es igual o distinto en las dos ontologías. También observa que elementos de su ontología privada están en conflicto en la compartida. Si el usuario desea saber cual es la causa de la visualización cuenta con una ventana para navegar el concepto.

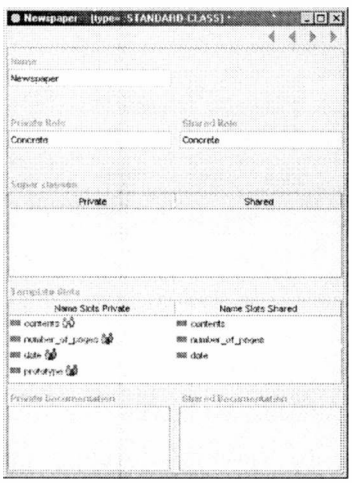


Figura 5.9: Comparación de la clase *Newspaper* en ambas ontologías

La *figura 5.9* muestra la ventana en la que un usuario puede recorrer e identificar los motivos que determinan que la clase *Newspaper* de la ontología privada es distinta que la clase *Newspaper* de la ontología compartida. En este caso no es igual, ya que el *slot prototipo* no pertenece a la clase de la ontología compartida y si a la clase de la ontología privada (notar que el *slot*, por su visualización si pertenece a la ontología compartida). Otro motivo por el que la clase no es igual es por el *slot contents* que difiere en su definición en ambas ontologías.

Si un usuario quiere ver en forma global las diferencias entre ambas ontologías, cuenta con un *tab* llamado *Difference Tab*, que permite ver la comparación de todos los elementos de ambas ontologías

El *awareness* de actividad es dado por medio las notificaciones. Las notificaciones se basan en las acciones que el grupo realiza. Un usuario personaliza cuales son los cambios, en el diseño de la ontología compartida, de los que va a ser notificado. El usuario define el valor del *slot :_AWARENESS-WIDGET* en el *tab user*. La *figura 5.10* muestra las posibles opciones a elegir.

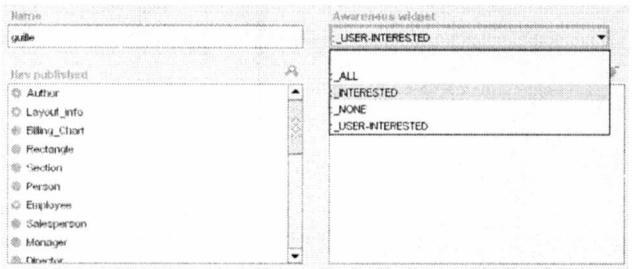


Figura 5.10: Valores posibles del *slot* *:_AWARENESS-WIDGET*

Los posibles valores para el *slot* *:_AWARENESS-WIDGET* son:

- *:_ALL*: notifica de todos los cambios.
- *:_INTERESTED*: notifica los cambios en los *frames* que el usuario agrego a sus intereses.
- *:_NONE*: no notifica.
- *:_USER_INTERESTED*: notifica de los cambios introducidos por los usuarios que estén en la lista de intereses.

Los cambios siempre son sobre *frames* de la ontología compartida. Si bien la lista contiene cuatro posibles opciones, la idea es que los usuarios puedan implementar sus propias condiciones.

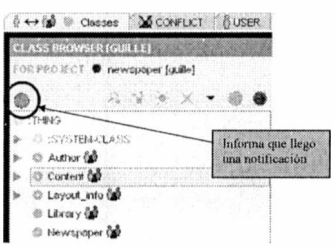


Figura 5.11: Recepción de notificaciones

Cuando un usuario realiza una acción pública, por ejemplo la modificación de las clase *Author*, el sistema crea una instancia de la clase *:_UPDATE-ACTION* y las instancias necesarias de la clase *:_NOTIFICATION* . Si la acción publica involucra conceptos, sobre los cuales un usuario quiere se notificado, el sistema le informa inmediatamente, agregando una notificación al *slot* *:_NOTIFICATION* de la instancia *User* que lo representa. Cuando a un usuario le "llega" una notificación, se entera del evento por medio de la visualización de un "disco que gira" en los diferentes *tabs*, como se muestra en la *figura 5.11*.

Si el usuario no estaba "logueado", el mecanismo anterior se activa la próxima vez que ingrese al sistema, y podrá ver todas las notificaciones recibidas. La *figura 5.12* muestra las notificaciones recibidas por el usuario *Guille*. De similar manera las acciones conforman un historial que el usuario puede visualizar en una ventana similar a la anterior.

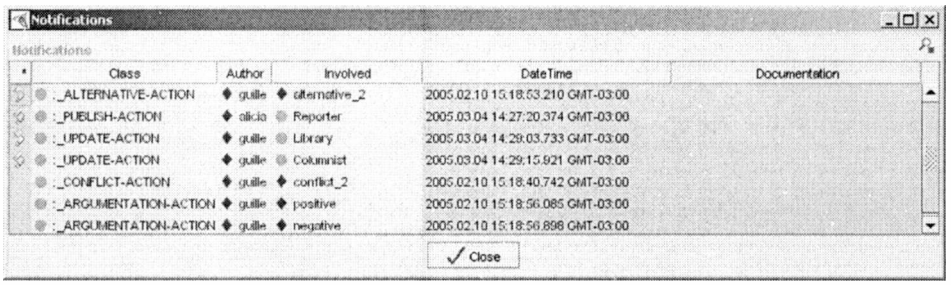


Figura 5.12: Notificaciones

5.8. Herramientas para la implementación de CO-Protégé

CO-Protégé, es un *plug-in* para la versión Protégé-2000 3.2, que utiliza la API provista por Protégé-2000. Se implemento utilizando la tecnología Java, por medio de la plataforma J2SE 1.5, JDK 1_5_0_04 (`http://java.sun.com`). El editor utilizado para el desarrollo de la aplicación fue Eclipse (`http://www.eclipse.org/`)

5.9. Comparación con otras herramientas

Para finalizar con el capítulo se realiza una breve comparación de CO-Protégé con los editores de ontologías colaborativos *Apecks* [TS05] y *webOnto* [Dom98].

APECKS, se basa en el desarrollo de ontologías personales. Los usuarios definen sus propias ontologías según sus puntos de vista personales, luego el sistema compara dos ontologías mediante la comparación de los términos que las componen. La comparación es clasificada según lo propuesto por Shaw y Gaines [TS05], que definen cuatro posible resultados: *consenso* (igual terminología, iguales atributos), *diferencia* (diferente terminología, iguales atributos), *conflicto* (igual terminología, diferentes atributos) y *contraste* (diferente terminología, diferentes atributos). APECKS además compara la jerarquía de clases, los *slots* y los valores de los *slots*. El resultado de las comparaciones se indican en una grilla, que puede ser explorada para ver en detalle el resultado. Cada usuario, en base a los resultados, puede modificar su ontología personal, o escribir una anotación que justifique la conceptualización que diseñó. Las discusiones en APECKS

surgen luego de comparar dos ontologías personales y se espera que los usuarios argumenten los caso de la comparación, justificando el modelo que ellos poseen.

WebOnto es un Java-Applet que junto con un servidor web permite a diferentes usuarios ver y editar modelos de conocimientos basados en ontologías. Cuando un usuario está dentro de WebOnto editando una ontología, la misma se bloquea al resto de los usuarios, que pueden navegar su contenido, pero no editarlo. WebOnto se extiende con una herramienta para soportar discusiones siguiendo un modelo llamado *Tadzebao* [Dom98] (palabra China para denominar a las "pancartas" utilizadas en apoyo a la discusión política durante la revolución cultural en China). Tadzebao mezcla un editor de texto e imágenes para expresar opiniones. Su funcionamiento consiste en que los usuarios creen notas, cada nota contiene en forma gráfica la jerarquía de clases o parte de ella. Sucesivas notas se van creando a partir de una anterior, donde en cada una se hacen anotaciones gráficas y textuales, por ejemplo un usuario "tacha" con una cruz roja la relación entre una clase y una super-clase. Los usuarios observan las notas para intentar llegar a un acuerdo.

WebOnto permite la construcción colaborativa de ontologías, en una forma muy primitiva, tiene en cuenta sólo el control de concurrencia, bloqueando la edición a una sola persona. El mecanismo de discusión es poco formal y la interpretación de las notas depende de como la "vea" el usuario.

APECKS, es una herramienta en desarrollo, propone un acercamiento más formal para soportar discusiones. La mayor diferencia con CO-Protégé, es que APECKS no se centraliza en una única ontología común, se basa en ontologías personales que son comparadas entre si sucesivamente. El enfoque en CO-Protégé es distinto, los usuarios tienen como base la ontología compartida, que por más que contenga conflictos, es la ontología final hasta ese instante.

Capítulo 6

Conclusiones y Trabajos Futuros

En el trabajo se explicaron las necesidades de tener una aproximación colaborativa para el diseño de ontologías, fundamentalmente por que promueve a fortalecer el compromiso ontológico, uno de los puntos centrales en la definición de una ontología.

La metodología de trabajo, definida por los tres pilares, propone un enfoque diferente al que manejan los editores de ontologías colaborativas actualmente. Los diferentes puntos de vistas de los usuarios, son tratados dentro del sistema, garantizando que los usuarios están al tanto de los cambios en el diseño, fundamentalmente cuando surgen conflictos a partir de las diferentes visiones.

Como punto destacado, se pudo observar como por medio de la manipulación del modelo de representación del conocimiento, se permitió representar los conceptos de la actividad colaborativa. El nuevo meta-modelo planteado incorpora la noción de *frames* alternativos. Los *frames* alternativos, representan visiones diferentes de los usuarios y son tratados dentro del sistema, esto permitió incluir dentro de la ontología compartida, elementos que sin el nuevo modelo serían incompatibles. *Se transformaron conceptos que antes eran incompatibles, en nuevos elementos de conocimiento.*

Otro punto importante es la unificación en la representación de los conceptos asociados a la colaboración, como son los usuarios y sus perfiles, las acciones, las notificaciones, etc. Todos los conceptos son conocimiento en si mismos, conocimientos de índole pública y se representaron por medio de ontologías genéricas. Esto permitió incorporar en la ontología compartida los conceptos involucrados en la colaboración en forma directa. Además como dichos conceptos involucran relaciones con conceptos propios del dominio de discusión de los participantes, su representación ontológica permite que "que hablen el mismo idioma", ganando las características propias de la representación ontológica, desacoplando el conocimiento de la herramienta de manipulación de la ontología.

El *awareness* propuesto, ayuda a mantener al grupo en "acción", los participantes están todo el tiempo informados de los cambios en el diseño. Esta última característica, permite que un usuario reaccione ante los estímulos recibidos por el *awareness*, ya sea

el de actividad como el de sincronización. El *awareness* informa a los usuarios de los conflictos en el diseño, mantiene siempre visible las divergencias, motivando al grupo a superarlas. Podemos concluir que el *awareness* aumenta el compromiso ontológico, un aspecto fundamental a la hora del diseño colaborativo de una ontología.

En la sección anterior se presento el prototipo CO-Protégé, donde se pudo ver una forma de implementar los tres pilares propuestos. La herramienta base elegida fue Protégé-2000, un editor de ontologías que ha ganado gran popularidad. CO-Protégé rescata los aspectos fundamentales de Protégé-2000. CO-Protégédefinió una nueva arquitectura que permite hacer el extensible su implementación y adaptarlo a necesidades futuras.

El prototipo CO-Protégé no tiene toda la funcionalidad implementada, se restringió su implementación al tratamiento de clases y *slots*, por tal motivo el primer trabajo futuro que se plantea es la necesidad de completar CO-Protégé, haciéndolo además funcional. Por ejemplo si un usuario publica una clase con un error ortográfico en el nombre, no puede cambiar el nombre de la clase en forma rápida como sería deseable. Otro ejemplo de futura funcionalidad es darle la posibilidad a un usuario de editar directamente la ontología compartida y que el sistema la trate como una edición de la ontología privada y su subsecuente publicación, teniendo en cuenta la factibilidad de la acción realizada.

La búsqueda de consensos para superar los conflictos en el diseño, es otro tema que merece un análisis profundo. La metodología de diseño propuesta brinda información sobre los conflictos y, posee mecanismos para generar discusiones. La decisión final de como superar un conflicto particular es un problema a tratar.

Uno de los avances en el área de representación del conocimiento se da en lo que se denomina web semántica. Recientemente aparición OWL [DCH+04], un lenguaje estándar para la definición de ontologías sugiere un avance en este ámbito. Protégé-2000 cuenta actualmente con un *plug-in* para soportar OWL, que permite pensar en desarrollar la metodología de diseño planteada en este trabajo teniendo como base OWL. La primer dificultad radica en que OWL modifica la representación del conociendo definiendo un *meta-modelo* propio, es necesario analizar como es el impacto en CO-Protégé. Surge la necesidad de la construcción de nuevos *tab*, ya que la visualización cambia, y definir nuevas las reglas para la comprobación de contribuciones aumentativas que tengan en cuenta la nueva representación. El resto de los conceptos son aplicables directamente, por lo que se presupone que la implementación de un nuevo *plug-in* (CO-Protégé-OWL) no impactara en el corazón de CO-Protégé.

Apéndice A

Ontología *Wine*

THING
- **Consumable thing**
  - **Drink**
    - **Soft drink**
    - **Wine**
      - **Dessert wine**
        - *Instances* *Whitehall Lane Primavera*
        - **Ice Wine**
          - *Instances* *Selaks Ice Wine*
        - **Port**
          - *Instances* *Taylor Port*
        - **Sweet Reisling**
          - *Instances* *Schloss Rothermel Trochenbierenauslese Riesling, Schloss Volrad Trochenbierenauslese Riesling*
      - **Red wine**
        - **Beaujolais**
          - *Instances* *Chateau Morgon Beaujolais*
        - **Cabernet Franc**
          - *Instances* *Whitehall Lane Cabernet Franc*
        - **Cabernet Sauvignon**
          - *Instances* *Forman Cabernet Sauvignon, Marietta Cabernet Sauvignon, Page Mill Winery Cabernet Sauvignon, Santa Cruz Mountain Vineyard Cabernet Sauvignon, Sterling Cabernet Sauvignon*
        - **Chianti**
          - *Instances* *Chianti Classico*
        - **Petite Syrah**
          - *Instances* *Marietta Petite Syrah, Sean Thackrey Sirius Petite Syrah*
        - **Pinot Noir**
          - *Instances* *Lane Tanner Pinot Noir, Mount Eden Vineyard Estate Pinot Noir, Mountadam Pinot Noir*
        - **Port**
          - *Instances* *Taylor Port*
        - **Red Bordeaux**
          - **Graves**
          - **Medoc**
            - **Margaux**
              - *Instances* *Chateau Margaux*
            - **Pauillac**
              - *Instances* *Chateau Lafite Rothschild Pauillac*
          - **St. Emillion**
            - *Instances* *Chateau Cheval Blanc St Emilion*
        - **Red Burgundy**
          - *Instances* *Clos De Vougeot Cotes D Or*
          - **Cotes Chalonnaise**
          - **Cotes dÓr**
        - **Red Merlot**
          - *Instances* *Gary Farrell Merlot, Longridge Merlot, Sterling Merlot*
        - **Red Zinfandel**
          - *Instances* *Cotturi Zinfandel, Elyse Zinfandel, Marietta Zinfandel, Saucelito Canyon Zinfandel*
      - **Rose wine**
        - **White Merlot**
        - **White Zinfandel**
      - **White wine**
        - **Chardonnay**
          - *Instances* *Bancroft Chardonnay, Forman Chardonnay, Mount Eden Vineyard Edna Valley Chardonnay, Mountadam Chardonnay, Peter Mccoy Chardonnay, Sterling Chardonnay*
        - **Chenin Blanc**
          - *Instances* *Foxen Chenin Blanc, Ventana Chenin Blanc*
        - **Ice Wine**
          - *Instances* *Selaks Ice Wine*
        - **Muscadet**
          - *Instances* *Sevre Et Maine Muscadet*
        - **Pinot Blanc**
        - **Riesling**
          - **Dry Riesling**
            - *Instances* *Corbans Dry White Riesling, Mountadam Riesling*
          - **Sweet Reisling**
            - *Instances* *Schloss Rothermel Trochenbierenauslese Riesling, Schloss Volrad Trochenbierenauslese Riesling*
        - **Sancerre**
          - *Instances* *Clos De La Poussie Sancerre*
        - **Sauterne**
          - *Instances* *Chateau D Ychem Sauterne*
        - **Sauvignon Blanc**
          - *Instances* *Corbans Private Bin Sauvignon Blanc, Corbans Sauvignon Blanc, Selaks Sauvignon Blanc, Stonleigh Sauvignon Blanc*
        - **Semilion**

- *Instances* *Congress Springs Semillon, Kalin Cellars Semillon*
- **White Bordeaux**
- **White Burgundy**
  - *Instances* *Chateau De Meursault Meursault, Corton Montrachet White Burgundy, Puligny Montrachet White Burgundy*
- **Chablis**
- **Pouilly-Fuisse**

Food
- **Dessert**
  - **Cheese/Nuts dessert**
    - *Instances* *Cheese, Nuts*
  - **Sweet dessert**
    - *Instances* *Cake, Pie*
- **Fowl**
  - **Dark meat fowl**
    - *Instances* *Duck, Goose*
  - **Light meat fowl**
    - *Instances* *Grilled chicken, Turkey*
- **Fruit**
  - **Non-sweet fruit**
    - *Instances* *Apple*
  - **Sweet fruit**
    - *Instances* *Banana, Grape, Mixed fruit, Peach*
- **Meat**
  - **Red meat**
    - *Instances* *Lamb*
    - **Non-spicy red meat**
      - *Instances* *Roast beef, Steak, Veal*
    - **Spicy red meat**
      - *Instances* *Beef curry, Garlicky roast*
  - **White meat**
    - *Instances* *Pork*
- **Pasta**
  - **Pasta with red sauce**
    - **Pasta with non-spicy red sauce**
      - *Instances* *Spaghetti with tomato sauce*
    - **Pasta with spicy red sauce**
      - *Instances* *Fra Diavolo*
  - **Pasta with white sauce**
    - **Pasta with heavy cream sauce**
      - *Instances* *Fettucine Alfredo*
    - **Pasta with light cream sauce**
      - *Instances* *Pasta with white clam sauce*
- **Seafood**
  - **Fish**
    - **Bland fish**
      - *Instances* *Flounder, Halibut, Scrod*
    - **Non-bland fish**
      - *Instances* *Salmon, Swordfish, Tuna*
  - **Shellfish**
    - *Instances* *Clams, Crabs, Lobster, Mussels, Oysters*
- **Tomato-based food**
  - *Instances* *Pizza*

Meal course
- *Instances* *Lamb, Oysters, Nuts, Salmon, Grilled chicken*

Wine grape
- *Instances* *Cabernet Franc grape, Cabernet Sauvignon grape, Chardonnay grape, Chenin Blanc grape, Gamay grape, Malbec grape, Merlot grape, Petite Syrah grape, Petite Verdot grape, Pinot Blanc grape, Pinot Noir grape,*

Wine region
- **Australian region**
  - **South Australian region**
- **French region**
  - **Alsace region**
  - **Beaujolais region**
  - **Bordeaux region**
    - **Margaux region**
    - **Paulliac region**
    - **Sauterne region**
    - **St Emillion region**
  - **Bourgogne region**
    - **Cotes Chalonnaise region**
    - **Cotes dÓr region**
    - **Meursault region**
  - **Loire region**
    - **Pays Nantes region**
    - **Sancerre region**
    - **Tours region**
- **German region**
- **Italian region**
  - **Chianti region**
- **New Zealand region**
- **Portuguese region**

Winery
- *Instances* *Bancroft, Chateau D Ychem, Chateau Lafite Rothschild, Chateau Margaux, Chateau Morgon, Clos De La Poussie, Clos De Vougeot, Congress Springs, Corbans, Corton Montrachet, Cotturi, D Anjou, Elyse, Forman, Foxen, Gary Farrell, Handley, Kalin Cellars, Kathryn Kennedy, Lane Tanner, Longridge, Marietta, Mount Eden Vineyard, Mountadam, Page Mill Winery, Peter Mccoy, Puligny Montrachet, Santa Cruz Mountain Vineyard, Saucelito Canyon, Schloss Rothermel, Schloss Volrad, Sean Thackrey, Selaks, Sevre Et Maine, Sterling Vineyards, Stonleigh, Taylor, Ventana, Whitehall Lane, Chateau Latour, Chateau Haut-Brian, Chateau Mouton-Rothschild*

Apéndice B

Ontología *Newspaper*

El apéndice describe la ontología *newspaper* en su totalidad. Primero se describe las jerarquía de clases, luego se detallan las clases con los *slots* que poseen (se indica con * los *slots* heredados).

Jerarquía de clases de la ontología *newspaper*

- **THING**
  - **Author**
    - **Columnist**
    - **Director**
    - **Editor**
    - **News_Service**
    - **Reporter**
  - **Content**
    - **Advertisement**
      - **Personals_Ad**
      - **Standard_Ad**
    - **Article**
  - **Layout_info**
    - **Billing_Chart**
    - **Content_Layout**
    - **Prototype_Newspaper**
    - **Rectangle**
    - **Section**
  - **Library**
  - **Newspaper**
  - **Organization**
  - **Person**
    - **Employee**
      - **Columnist**
      - **Editor**
      - **Manager**
        - **Director**
      - **Reporter**
      - **Salesperson**

Descripción de las clases de la ontología *newspaper*

Author			
Authors are the people or organizations which provide articles			
Rol	*SuperClases*	*SubClases*	*Slots*
Abstract	:THING	News_Service Columnist Editor Reporter	name

News_Service			
This is a use of classes as abstract placeholders. We need some way to indicate a fixed (yet changeable) list of values. In a way, this is just a list of strings, but with constraints in place.			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Author		contact_person *name

Columnist			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Author Employee		*contact_person *salary *phone_number *name *other_information *date_hired *current_job_title

Editor			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Author Employee		responsible_for sections *contact_person *salary *phone_number *name *other_information *date_hired *current_job_title

Reporter			
A reporter is an employee who writes articles			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Author Employee		*contact_person *salary *phone_number *name *other_information *date_hired *current_job_title

Content			
This is an abstract superclass of both advertisements and articles, containing the the information common to them both.			
Rol	*SuperClases*	*SubClases*	*Slots*
Abstract	:THING	Advertisement Article	expiration_date urgent containing_section layout page_number published_in

Advertisement				
Rol	*SuperClases*	*SubClases*	*Slots*	
Abstract	Content	Personals_Ad Standard_Ad	salesperson *published_in *expiration_date *containing_section	name *page_number *urgent *layout

Personals_Ad				
Rol	*SuperClases*	*SubClases*	*Slots*	
Concrete	Advertisement		text *salesperson *published_in *expiration_date *containing_section	*name *purchaser *page_number *urgent *layout

Standard_Ad				
Rol	*SuperClases*	*SubClases*	*Slots*	
Concrete	Advertisement		image *salesperson *published_in *expiration_date *containing_section	*name *purchaser *page_number *urgent *layout

Article				
Articles are included here as soon as they are written–they could go for a while without being published. For example, an article on gardening could be submitted on Monday and not be published until Thursday (when the gardening section is included in the paper).				
Rol	*SuperClases*	*SubClases*	*Slots*	
Concrete	Content		reading_level keywords article_type *expiration_date *containing_section	text author *urgent *layout *published_in

Layout_info			
Rol	*SuperClases*	*SubClases*	*Slots*
Abstract	:THING	Billing_Chart Content_Layout Prototype_Newspaper Rectangle Section	

Billing_Chart				
Rol	*SuperClases*	*SubClases*	*Slots*	
Concrete	Layout_info		cost_chart	name

Content_Layout			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Layout_info		main_rectangle breakable breakable

Prototype_Newspaper			
This is basically a reified relationship that must always hold.Entries here reflect facts like .On Wednesday, the paper has a Gardening Section and ads in it cost this much..."			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Layout_info		weekday sections prototype_billing_chart

Rectangle			
Defines a rectangle for space upon the page			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Layout_info		width height

Section			
List of possible sections of the pape			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Layout_info		name

Newspaper			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	:THING		number_of_pages date prototype contents

Organization			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	:THING		employees name employee_relationships

Person			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	:THING	Employee	phone_number name other_information

Employee			
Rol	*SuperClases*	*SubClases*	*Slots*
Abstract	:THING	Reporter Columnist Editor Salesperson Manager	date_hired salary current_job_title *name *phone_number *other_information

Salesperson			
salesperson sells, and is reponsible for the content of, advertisements.			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Employee		*date_hired *salary *current_job_title *name *phone_number *other_information

Manager			
Rol	*SuperClases*	*SubClases*	*Slots*
Abstract	Employee	Director	*date_hired *salary *current_job_title *name *phone_number *other_information

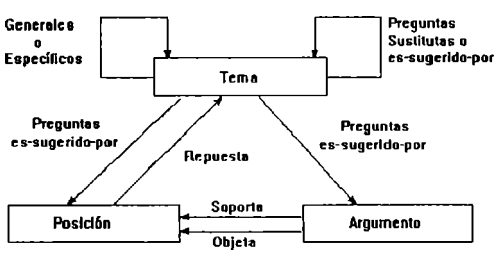
Director			
Rol	*SuperClases*	*SubClases*	*Slots*
Concrete	Employee		*date_hired *salary *current_job_title *name *phone_number *other_information

Apéndice C

IBIS - *Modelo de discusión*

Una de las características del trabajo grupal es que el grupo debe tomar decisiones para llegar a un consenso. Tener discusiones previas abre oportunidades para que los participantes encuentren un acuerdo. Uno de los modelos que se utilizan para lograr un discusión constructiva es el modelo IBIS (*Issue Based Information System*) [KR70], un modelo conversacional muy difundido.

Una discusión constructiva consiste en la presentación de ideas en una forma organizada. Las ideas deben tener algún argumento que las sustente, y que pueda ser entendido y revisado por otros miembros. En 1970 Kunz y Rittel proponen un modelo llamado IBIS, compuesto de tres elementos y nueve relaciones. El modelo va dirigido a representar los componentes cruciales de una discusión, permitiendo a las personas entender y facilitar nuevas contribuciones. La *figura C.1* muestra, a la izquierda, el esquema conceptual de IBIS y, a la derecha, un breve ejemplo de utilización del modelo.



Tema
Riesgos de Salud
Tema generado
Riesgos de Fumar
Posición
Fumar hasta los 32 años no trae riesgo
Argumento a favor
Algunos científicos dignos de confianza evidencian esta descripción
Argumento en contra
Fumar a cualquier edad siempre es perjudicial para la salud

Figura C.1: Modelo IBIS - Ejemplo de una conversación

El sistemas IBIS es un proceso que se basa fundamentalmente en la discusión entre varios participantes, es particularmente útil cuando un problema no tiene una buena definición y por su naturaleza el problema conducen a la discusión de los participantes; este tipo de problemas son llamados"malos (*wicked*)". Un problema *wicked* es uno en el cual la naturaleza del problema en sí mismo puede ser mal entendida y los términos de referencia son confusos o en contradictorios.

IBIS sigue un procedimiento que implica una descomposición del problema. Inicialmente se identifica el problema. La discusión comienza generando preguntas. Las posibles respuestas a una pregunta se identifican como posiciones. Una respuesta es una opinión de un participante individual. Los participantes pueden dar argumentos en favor o en contra de una opinión.

Dar por finalizada una pregunta, es determinar una respuesta que la satisfaga y puede realizarse a partir de una respuesta sobre cual todos los participantes están de acuerdo.

El proceso IBIS ayuda a al discusión, ya que la generación de preguntas y respuestas abstrae el problema principal y permite que los participantes contribuyan con sus habilidades de conocimiento y realicen nuevos razonamientos a aspectos del problema que de otra manera podrían quedar descartados inconscientemente.

Aunque encontrar una buena solución a cualquier problema dado no está garantizada de ninguna manera, el proceso puede revelar respuestas en las cuales un consenso puede ser alcanzado y además descubrir detalles que no serían posibles con otras técnicas de razonamiento.

Existen varias herramientas basadas en el proceso IBIS, entre las que se encuentran: Web-IBIS, gIBIS y GoDiagram [Van96].

Bibliografía

[BLD03] Guillermo Baldo, Miguel Lemus, and Alicia Díaz. Acciones como fuente del conocimiento de una comunidad. In *Proceedings of Conferencia Latinoamericana de Informatica CLEI 2003*, septiembre 2003.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[BNM95] S. Beale, S.Ñirenburg, and K. Mahesh. Semantic analysis in the mikrokosmos machine translation project, 1995.

[Bor97] W.Ñ. Borst. Construction of engineering ontologies. *PhD thesis, University of Twente, Enschede*, 1997.

[CFF+98] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. Okbc: a programmatic foundation for knowledge base interoperability. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 600–607, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[Col97] David Coleman. *Groupware: Collaborative Strategies for Corporate LANs and Intranets*. Prentice Hall PTR, 1997.

[Cro94] M. K. Crowe. cooperative work with multimedia. springer-verlag, 1994.

[DB92] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'92)*, pages 107–114, Toronto, Ontario, 1992. ACM Press.

[DC04] Alicia Díaz and Gèrôme Canals. Divergence ocurrences in knowledge sharing communities. In *Proceedings of Criwg 04*, Lecture Notes in Computer Science (LNCS), pages 17–24. Springer Verlag, 2004.

[DCH+04] M. Dean, D. Connolly, F. Van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Owl web ontology language 1.0. *W3C Recommendation 10 February 2004*, 3, 2004.

[Dom98] John Domingue. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web, 1998. `http://ksi.cpsc.ucalgary.ca/KAW/KAW98/domingue/`.

[DSW+00] A. J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa, and V. R. Benjamins. Wondertools?: a comparative study of ontological engineering tools. *Int. J. Hum.-Comput. Stud.*, 52(6):1111–1133, 2000.

[EGR91] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[EL88] Douglas C. Engelbart and Harvey Lehtman. Working togetherm. *BYTE Magazine*, pages 245–252, December 1988.

[FB] Alan Flett and Mike Brown. Enterprise-standard ontology environments for intelligent e-business.

[Gin91] Matthew L. Ginsberg. Knowledge interchange format: the KIF of death. *AI Magazine*, 12(3):57–63, 1991.

[GMF+03] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya Fridman Noy, and Samson W. Tu. The evolution of protégé: an environment for knowledge-based systems development. *Int. J. Hum.-Comput. Stud.*, 58(1):89–123, 2003.

[Gol94] A. P. Goldberg. Groupware lecture notes. *Computer Science Department,New York University , Fall*, 1994. `http://cs.nyu.edu/cs/faculty/artg/groupware/syllabus.html`.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[Gua98] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Formal Ontology in Information Systems*, pages 3–18. IOS Press, Amsterdam, 1998.

[HJ02] Clyde W. Holsapple and K. D. Joshi. A collaborative approach to ontology design. *Commun. ACM*, 45(2):42–47, 2002.

[HSW97] G. Van Heijst, A.Th. Schreiber, and B.J. Wielinga. Using explicit ontologies in kbs development. In *International Journal of Human-Computer Studies, volume 45*, pages 184–292, 1997.

[JL78] Trudy Jhonson-Lenz. Groupware definition, 1978.

[JSB+91] R. Johansen, D. Sibbet, S. Benson, A. Martin, R. Mittman, and P. Saffo. Leading business teams. addison wesley, 1991.

[KLW90] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. Technical Report TR-90-003, 1, 1990.

[KR70] Kunz and W. Y Rittel. Issues as elements of information systems. *Communications of the ACM*, Working Paper No. 131, 1970.

[Kuo93] D. Kuokka. Shade:technology for knowledge-based collaborative. *AAAI Workshop on AI in Colaborative Design*, 1993.

[Lyd02] Silvia Muñoz Lydia. *Estudo de ontologías para representação de conteúdos de ensino baseado na www.* Porto Alegre, 2002.

[Mah96] K. Mahesh. Ontology development for machine translation: Ideology and methodology. nmsu. *Computing Research Laboratory. Technical Report MCCS-96-292. New Mexico.*, 1996.

[MGS97] Rossi Mori, A. Gangemi, and A. Steve. An ontological analysis of surgical deeds, 1997.

[NR01] Sergei Nirenburg and Victor Raskin. Ontological semantics, formal ontology, and ambiguity. In *FOIS*, pages 151–161, 2001.

[NT95] I.Ñonaka and H. Takeuchi. *The Knowledge-Creating Company.* Oxford University Press, New York, New York, 1995.

[pal00] Protégé ontology editor and knowledge acquisition system, 2000.

[PFS+01] Alun D. Preece, Alan Flett, Derek H. Sleeman, David Curry, Nigel Meany, and Phil Perry. Better knowledge management through knowledge engineering. *IEEE Intelligent Systems*, 16(1):36–43, 2001.

[Pow91] David M. W. Powers. Goals, issues and directions in machine learning of natural language and ontology (bibliography). *SIGART Bulletin*, 2(1):101–114, 1991.

[PP03] H. Sofia Pinto and DuarteÑ. Peralta. Combining ontology engineering subprocesses to build a time ontology. In *K-CAP '03: Proceedings of the 2nd international conference on Knowledge capture*, pages 88–95, New York, NY, USA, 2003. ACM Press.

[PRO05] PROTEGE. Protégé ontology editor and knowledge acquisition system, 1990-2005. `http://protege.stanford.edu`.

[RE91] Gail L. Rein and Clarence A. Ellis. ribis: A real-time group hypertext system. *International Journal of Man-Machine Studies*, 34(3):349–367, 1991.

[SGP98a] G. Steve, A. Gangemi, and D. Pisanelli. Integrating medical terminologies with onions methodology, 1998. `http://saussure.irmkant.rm.cnr.it.html`.

[SGP98b] G. Steve, A. Gangemi, and D. Pisanelli. Ontology integration: Experiences with medical ontologies. *en N. Guarino (ed.)*, pages 163–178, 1998.

[Sow00] John F. Sowa. *Knowledge representation: logical, philosophical and computational foundations.* Brooks/Cole Publishing Co., 2000.

[SPKR96] Bill Swartout, Ramesh Patil, Kevin Knight, and Tom Russ. Towards distributed use of large-scale ontologies. In *Proceedings of the 10th. Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1996.

[Tat96] A. Tate. Towards a plan ontology, 1996.

[TS05] Jenifer Tennison and Nigel R. Shadbolt. Apecks: a tool to support living ontologies, 2005. `http://ksi.cpsc.ucalgary.ca/KAW/KAW98/tennison/`.

[UG96] M. Uschold and M. Gruninger. Ontologies: principles methods and applications. In *Knowledge Engineering Review, volumen 11, number 2*, pages 93–155, 1996.