Desarrollo de Ontologías-101: Guía Para Crear Tu Primera Ontología

Natalya F. Noy and Deborah L. McGuinness
noy@smi.stanford.edu and dlm@ksl.stanford.edu
Stanford University, Stanford, CA, 94305
Traducido del inglés por: Erick Antezana*

September 19, 2005

*erant@psb.ugent.be

Contents

1 ¿Por qué desarrollar una ontología?

En los últimos años el desarrollo de ontologías (especificaciones formales y especificas de los términos y relaciones entre ellos (Gruber 1993)) ha estado moviéndose del dominio de los laboratorios de Inteligencia Artificial a los escritorios de los expertos de un dominio dado. Las ontologías has llegado a ser comunes en el World-Wide Web. Las ontologías en el Web van desde grades taxonomías que categorizan sitios Web (tales como en Yahoo!) a categorizaciones de productos para vender y sus características (tales como Amazon.com). El Consorcio WWW (W3C) está desarrollando el Resource Description Framework (Brickley y Guha 1999), un lenguaje para codificar conocimiento en páginas Web para hacerlas entendibles a los agentes electrónicos que buscan información. La agencia de proyectos de investigación avanzada en defensa (*Defense Advanced Research Projects Agency* (DARPA)), conjuntamente con el W3C, está desarrollando *DARPA Agent Markup Language* (DAML) extendiendo RDF con construcciones más expresivas buscando facilitar la interacción de agentes en el Web (Hendler and McGuinness 2000). Muchas disciplinas desarrollan ahora ontologías estandarizadas que los expertos de ciertos dominios pueden usarlas para compartir y anotar información en sus campos de trabajo. En medicina, por ejemplo, se ha producido grandes, estandarizados y estructurados vocabularios tales como snomed (Price and Spackman 2000) y la red semántica *Unified Medical Language System* (Humphreys and Lindberg 1993). Están también surgiendo otras ontologías amplias y de propósito general. Por ejemplo, el programa de desarrollo de las naciones unidas (*United Nations Development Program*) y Dun & Bradstreet unieron esfuerzos para desarrollar la ontología UNSPSC que provee terminología para productos y servicios (`www.unspsc.org`).

Una ontología define un vocabulario comun para investigadores que necesitan compartir información en un dominio. Ella contiene definiciones de conceptos basicos y sus relaciones que pueden ser interpretadas por una maquina.

¿Por qué alguien desearía desarrollar una ontología? Algunas de las razones son:

- Compartir el entendimiento común de la estructura de información entre personas o agentes de software.
- Permitir la reutilización de conocimiento de un dominio.
- Explicitar suposiciones de un dominio.
- Separar el conocimiento del dominio del conocimiento operacional.
- Analizar el conocimiento de un dominio.

Compartir el entendimiento común de la estructura de información entre personas y agentes de software es una de las más importantes metas al desarrollar ontologías (Musen 1992; Gruber 1993). Por ejemplo, supongamos que varios distintos sitios Web contengan información médica o provean servicios de e-commerce médico. Si esos sitios Web comparten y publican la misma ontología subyacente de los términos que usan, entonces agentes de software podrían extraen y agregar información de eso esos sitios diferentes. Los agentes podrían usar esta información agregada para responder solicitudes de los usuarios o servir como datos de entrada a otras aplicaciones.

Permitir la reutilización de conocimiento de un dominio fue una de las fuerzas conductoras detrás recientes trabajos en la investigación sobre ontologías. Por ejemplo, modelos para diferentes dominios necesitan representar la noción de tiempo. Esta representación incluye las nociones de intervalo de tiempos, puntos en el tiempo, medidas relativas de tiempo, y cosas por el estilo. Si un grupo de investigadores desarrollo tal ontología en detalle, otros podrian simplemente reusarla en sus dominios. Ademas, si necesitamos construir una ontología grande,

podemos integrar varias ontologías existentes que describen porciones del dominio mas grande. También podemos reusar una ontología general, tal como la ontología UNSPSC, y extenderla para describir nuestro dominio de interés.

La explicitación de suposiciones de un dominio, que subyacen bajo una implementación, permite cambiar esas suposiciones fácilmente si el conocimiento del dominio cambia. Suposiciones codificadas explícitamente acerca del mundo en algún lenguaje de programación hacen que las suposiciones no solo sean difíciles de hallar sino también dificiles de cambiar, en particular para alguien sin competencias en programación. Además, las especificaciones explicitas del dominio de conocimiento son útiles para nuevos usuarios que deben aprender el significado de los términos del dominio.

La separación del conocimiento del dominio del conocimiento operacional es otro uso común de las ontologías. Podemos describir la tarea de configuración de un producto a partir de sus componentes de acuerdo a especificaciones requeridas e implementar un programa que hace independiente esta configuración de los productos y componentes en sí (McGuinness and Wright 1998). Podemos desarrollar una ontología de componentes de PC y características y aplicar el algoritmo para configurar PCs ordenadas a medida. Podemos usar el mismo algoritmo para configurar elevadores si "alimentamos" nuestra ontología con elevador como componente (Rothenfluh et al. 1996).

Analizar el conocimiento de un dominio es posible una vez que una especificación declarativa de los términos esta disponible. El análisis formal de los términos es extremadamente valioso al intentar reusar ontologías existentes y al extenderlas (McGuinness et al. 2000).

A menudo, desarrollar una ontología de un dominio no es la meta en sí. Desarrollar una ontología es comparable a definir un conjunto de datos y sus estructuras para que otros programas los usen. Métodos de resuelven problemas, aplicaciones independientes del dominio, y agentes de software usan ontologías y bases de conocimiento construidos a partir de ontologías como datos. Por ejemplo, en esta publicación desarrollamos una ontología de vinos y alimentos y de combinaciones apropiadas de vino con comidas. Esta ontología entonces puede ser usada como una base para aplicaciones como parte un las herramientas de gestión de un restaurante: Una aplicación podría crear sugerencias de vino para el menú del día o responder a solicitudes de camareros y clientes. otra aplicación podría analizar una lista de inventario de una bodega de vino y sugerir que categorías de vino ampliar y que vinos particulares comprar para menús futuros o recetarios.

Acerca de esta guía

Nos basamos en nuestra experiencia usando Protégé-2000 (Protege 2000), Ontolingua (Ontolingua 1997), Chimaera (Chimaera 2000) como entornos de edición de ontologías. En esta guía, usamos Protégé-2000 para nuestros ejemplos.

El ejemplo de vinos y alimentos que usamos a lo largo de esta guía está aproximadamente basado en un ejemplo de base de conocimiento presentada en la publicación que describe CLASSIC (un sistema de representación de conocimiento basado en lógicas de descripción (Brachman et al. 1991)). El tutorial de CLASSIC (McGuinness et al. 1994) ha desarrollado este ejemplo ulteriormente. Protégé-2000 y otros sistemas basados en marcos describen las ontologías declarativamente, estableciendo explicitamente cual es la jerarquía de clases y a que clases individuales pertenecen.

Algunas ideas de diseño de ontologías en esta guía se originaron a partir de la literatura sobre diseño orientado a objetos (Rumbaugh et al. 1991; Booch et al. 1997). Sin embargo, el desarrollo de ontologías es diferente a diseño de clases y relaciones como en la programación orientada a objetos. La programación orientada a objetos se centra principalmente alrededor de métodos en clases - un programador toma decisiones de diseño basándose en las propiedades *operacionales*

de una clase, mientras que un diseñador de ontologías toma esas decisiones basándose en las propiedades *estructurales* de una clase. En consecuencia, la estructura de una clase y las relaciones entre clases en una ontología son diferentes de la estructura para un dominio similar en un programa orientado a objetos.

Es imposible cubrir todos los aspectos que un desarrollador de ontologías pueda necesitar para trabajar y no estamos tratando de responderlos en esta guía. En cambio, tratamos de proveer un punto de inicio; una guía inicial que ayude a los nuevos diseñadores de ontologías a desarrollar ontologías. Al final, sugerimos lugares para buscar explicaciones sobre estructuras y mecanismos de diseño más complicados si el dominio los requiere.

Finalmente, no hay una simple y correcta metodología de diseño de ontologías y no intentamos definir una. Las ideas que presentamos aqui son las que encontramos útiles dentro nuestra experiencia de desarrollo de ontologías. Al final de esta guía sugerimos una lista de referencias de metodologías alternativas.

2 ¿Qué es una ontología?

La literatura de inteligencia artificial contiene varias definiciones de ontología; muchas de ellas contradicen otras. Para los propósitos de esta guía una ontología es una descripción explicita y formal de conceptos en un dominio de discurso (**clases** (a veces llamadas **conceptos**)), propiedades de cada concepto describiendo varias características y atributos del concepto (**slots** (a veces llamados **roles** o **propiedades**)), y restricciones sobre los slots (**facetas** (algunas veces llamados **restricciones de rol**)). Una ontología junto con un conjunto de **individuos** de clases constituye una **base de conocimiento**. En realidad, hay una linea muy delgada donde la ontología termina y la base de conocimiento empieza.

Las clases son el centro de la mayoría de las ontologías. Las clases describen conceptos de un dominio. Por ejemplo, una clase de vinos representa todos lo vinos. Vinos específicos son instancias de esta clase. El vino Bordeaux en un vaso en frente tuyo mientras lees este documento es una instancia de la clase de vinos Bordeaux. Una clase puede tener **subclases** que representan conceptos que son mas específicos que la superclase. Por ejemplo, podemos dividir la clase de todos los vinos en vinos rojo, blanco, y rosado. Alternativamente, podemos dividir la clase de todos los vinos en vinos efervescentes y no-efervescentes.

Los slots describen propiedades de clases e instancias: el vino `Chteau Lafite Rothschild Pauillac` está muy bien detallado; es producido por el establecimiento vinícola `Chteau Lafite Rothschild`. Tenemos dos slots que describen el vino en este ejemplo: el slot cuerpo con el valor total y el slot productor con el valor del establecimiento vinícola Chteau Lafite Rothschild. A nivel de la clase, podemos decir que las instancias de la clase `Vino` tendran slots que describen su `sabor`, `cuerpo`, `nivel de azúcar`, el `productor` del vino, etc.[1]

Todas las instancias de la clase `Vino`, y su subclase `Pauillac`, tienen un slot `productor` cuyo valor es una instancia de la clase `Establecimiento vinícola` (Figura 1). Todas las instancias de la clase `Establecimiento vinícola` tienen un slot `produce` que se refiere a todos los vinos (instancias de la clase `Vino` y sus subclases) que el establecimiento vinícola produce.

En términos prácticos, desarrollar una ontología incluye:

- definir clases en la ontología,
- organizar las clases en una jerarquía taxonómica (subclase-superclase),
- definir slots y describir valores permitidos para esos slots,

[1]Los nombres de las clases comienzan con mayusculas y los nombres de los slots están en minúsculas. Usamos también la fuente de `typewriter` para todos los términos del ejemplo de la ontología.

- llenar los valores de los slots para las instancias.

Podemos entonces crear una base de conocimientos definiendo las instancias individuales de esas clases, precisando los valores especificos de los slots y restricciones adicionales sobre los slots.

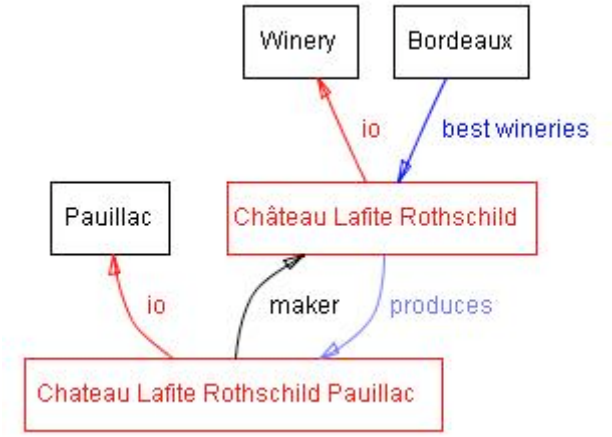

Figure 1: Algunas clases, instancias, y relaciones entre ellas en el dominio de vinos. Usamos negro para las clases y rojo para las instancias. Los enlaces directos representan los slots y enlaces internos tales como instancia-de y subclase-de.

3 Una simple metodología de ingeniería del conocimiento

Como lo dijimos antes, no existe ni una sola forma o ni una sola metodología "correcta" para desarrollar ontologías. Aquí abordamos los puntos generales que deben ser tomados en consideración y ofrecemos uno de los procedimientos posibles para desarrollar una ontología. Describimos un enfoque iterativo en el desarrollo de la ontología: comenzamos por abordar la ontología de manera frontal. A continuación volvemos sobre la ontología, que consideramos en proceso de evolución, afinándola y conpletándola con detalles. A lo argo de este proceso discutimos las decisiones de modelización que toma el diseñador, así como los pros, los contras y las implicaciones de diferentes soluciones.

Inicialmente, queremos enfatizar algunas reglas fundamentales e el diseño de ontologías a las cuales nos referiremos varias veces. Esas reglas pueden parecen algo dogmáticas. Ellas pueden ayudar, sin embargo, para tomar decisiones de diseño en muchos casos.

1. No hay una forma correcta de modelar un dominio - siempre hay alternativas viables. La mejor solución casi siempre depende de la aplicación que tienes en mente y las extensiones que anticipas.

2. El desarrollo de ontologías es un proceso necesariamente iterativo.

3. Los conceptos en la ontología deben ser cercanos a los objetos (físicos o lógicos) y relaciones en tu dominio de interés. Esos son muy probablemente sustantivos (objetos) o verbos (relaciones) en oraciones que describen tu dominio.

Es decir, decidir para que vamos a usar la ontología y cuán detallada o general será la ontología guiará a muchas de las decisiones de modelamiento a lo largo del camino. Entre

las varias alternativas viables, necesitaremos determinar cuál funcionará mejor para la tarea proyectada, cuál será más intuitiva, más extensible y más mantenible. Necesitamos también recordar que una ontología es un modelo de la realidad del mundo y los conceptos en la ontología deben reflejar esta realidad. Después de que hayamos definido una versión inicial de la ontología, podemos evaluarla y depurarla usándola en aplicaciones o métodos que resuelvan problemas o discutiéndola con expertos en el área. En consecuencia, casi seguramente necesitaremos revisar la ontología inicial. Este proceso de diseño iterativo probablemente continuara a través del ciclo de vida entero de la ontología.

Paso 1. Determinar el domino y alcance de la ontología

Sugerimos comenzar el s=desarrollo de una ontología definiendo su dominio y alcance. Es decir, responder a varias preguntas básicas:

- ¿Cuál es el dominio que la ontología cubrirá?
- ¿Para qué usaremos la ontología?
- ¿Para qué tipos de preguntas la información en la ontología deberá proveer respuestas?
- ¿Quién usará y mantendrá la ontología?

Las respuestas a esas preguntas pueden cambiar durante el proceso del diseño de la ontología, pero en cualquier momento dado ellas ayudaran a limitar el alcance del modelo.

Consideremos la ontología de vinos y alimentos que se introdujo antes. El dominio de la ontología es la representación de alimentos y vinos. Planeamos usar esta ontología en las aplicaciones que sugieran buenas combinaciones de vinos y alimentos.

Naturalmente, los conceptos que describen diferentes tipos de vinos, tipos principales de alimentos, la noción de una buena combinación de vino y alimento y la mala combinación figuraran en nuestra ontología. Al mismo tiempo, es improbable que la ontología incluya conceptos para gestionar inventarios en un establecimiento vinícola o empleados en un restaurante aunque esos conceptos están de alguna manera relacionados a las nociones de vino y alimento.

Si la ontología que estamos diseñando será usada para ayudar en el procesamiento de lenguaje natural de artículos en las tiendas de vino, seria importante incluir sinónimos e información de las varias clases de palabras a las cuales una palabra puede ser asignada para los conceptos de la ontología. Si la ontología sera usada para ayudar a los clientes de un restaurante a decidir qué vino ordenar, necesitamos incluir información del precio de venta al por menor. Si es usada por compradores de vino que almacenan el vino en bodegas, el precio de venta al por mayor y la disponibilidad serán necesarios. Si la gente que mantendrá la ontología describe el dominio en un lenguaje que es diferente del lenguaje que usan los usuarios de la ontología, tendremos que proveer el mapeo entre los lenguajes.

Preguntas de competencia.

Una de las formas de determinar el alcance de la ontología es bosquejando una lista de de preguntas que la base de conocimientos basada en la ontología debería ser capaz de responder, preguntas de competencia (Gruninger and Fox 1995). Esas preguntas servirán después como prueba de control de calidad: ¿La ontología contiene suficiente información para responder esos tipos de preguntas? ¿Las respuestas requieren un nivel particular de detalle o representación de un área particular? Las preguntas de competencia son solamente un bosquejo y no necesitan ser exhaustivas.

En el dominio de los vinos y alimentos, las siguientes preguntas son posibles preguntas de competencia:

- ¿Qué características debo considerar cuando elijo un vino?
- ¿Bordeaux es un vino rojo o blanco?
- ¿El Cabernet Sauvignon va bien con comida de mar?
- ¿Cuál es la mejor elección de vino para acompañar carne asada?
- ¿Qué características de un vino afectan su idoneidad con un pescado?
- ¿El cuerpo o aroma de un vino especifico cambia con su año de cosecha?
- ¿Cuáles fueron buenas cosechas para el Napa Zinfandel?

Juzgando a partir de esta lista de preguntas, la ontología incluirá la información de varias características de vinos y tipos de vinos, años de cosechas (buenos y malos), clasificación de alimentos que importan para elegir un vino apropiado, combinaciones recomendadas de vinos y comidas.

Paso 2. Considerar la reutilización de ontologías existentes

Casi siempre vale la pena considerar lo que otra persona ha hecho y verificar si podemos refinar y extender recursos existentes para nuestro dominio y tarea particular. Reusar ontologías existentes puede ser un requerimiento si nuestro sistema necesita interactuar con otras aplicaciones que ya se han dedicado a ontologías particulares o vocabularios controlados. Muchas ontologías ya están disponibles en forma electrónica y pueden ser importadas dentro un entorno de desarrollo de ontologías que estás usando. El formalismo en el cual una ontología está expresado a menudo no interesa, puesto que muchos sistemas de representación de conocimiento pueden importar y exportar ontologías. Aun si el sistema de representación de conocimiento no puede funcionar directamente con un formalismo particular, la tarea de traducir una ontología a partir de un formalismo a otro no es usualmente difícil.

Hay bibliotecas de ontologías reusables en la Web y en la literatura. Por ejemplo, podemos usar la biblioteca de ontologías Ontolingua (`http://www.ksl.stanford.edu/software/ontolingua/`) o la biblioteca de ontologías DAML (`http://www.daml.org/ontologies/`). También hay un cierto número de ontologías comerciales públicamente disponibles (e.g., UNSPSC (`www.unspsc.org`), RosettaNet (`www.rosettanet.org`), DMOZ (`www.dmoz.org`)).

Por ejemplo, es posible que una base de conocimientos sobre vinos Franceses exista. Si podemos importar esta base de conocimiento y la ontología sobre la cual está basada, tendremos no solamente la clasificación de vinos Franceses sino también el primero paso hacia la clasificación de caracteristicas de vinos usadas para distinguir y describir los vinos. Es posible que listas con las propiedades de los vinos esten disponibles en sitios Web comerciales tales como `www.wines.com` que los clientes consideren útilies para comprar vinos.

En esta guía, sin embargo, asumiremos que no existe ninguna ontología relevante y comenzaremos la ontología desde el principio.

Paso 3. Enumerar términos importantes para la ontología

Es útil escribir una lista con todos los términos con los que quisiéramos hacer enunciados o dar explicación a un usuario. ¿Cuáles con los términos de los cuales quisiéramos hablar? ¿Qué propiedades tienen esos términos? Por ejemplo, términos importantes relativos a los vinos incluirán `vino`, `cepaje`, `establecimiento vinícola`, `localidad`, `color` del vino, `cuerpo`, `sabor`, `contenido de azúcar`; diferentes tipos de alimentos, tales como `pescado` y `carne roja`;

subtipos de vino tales como vino blanco, etc. Inicialmente, es importante obtener una lista integral de términos sin preocuparse del recubrimiento entre los conceptos que representan, relaciones entre los términos, o cualquier propiedad que los conceptos puedan tener, o si los conceptos son clases o slots.

Los siguientes dos pasos (desarrollando la jerarquía de clases y definiendo las propiedades de los conceptos (slots)) están estrechamente relacionadas. Es difícil hacer primero uno de ellos y luego hacer el otro. Típicamente, creamos unas cuantas definiciones de los conceptos en la jerarquía y luego continuamos describiendo las propiedades de esos conceptos y así sucesivamente. Esos dos pasos son también los más importantes en el proceso de diseño de la ontología. Los describiremos brevemente y dedicaremos las siguientes dos secciones a discutir los asuntos más complicados que necesitan ser considerados, peligros comunes, decisiones a tomar, etc.

Paso 4. Definir las clases y la jerarquía de clases

Hay varios posibles enfoques para desarrollar una jerarquia de clases (Uschold and Gruninger 1996):

- Un proceso de desarrollo **top-down** comienza con la definición de los conceptos más generales en el dominio la subsecuente espicialización de los conceptos. Por ejemplo, podemos comenzar creando clases para los conceptos generales de `Vino` y `Alimentos`. Luego especializamos la clase Vino creando algunas de sus subclases: `Vino blanco`, `Vino Rojo`, `Vino rosado`. Podemos posteriormente categorizar la clase Vino rojo en, por ejemplo, `Syrah`, `Borgoña`, `Cabernet Sauvignon`, etc.

- Un proceso de desarrollo **bottom-up** comienza con la definición de las clases mas especificas, las hojas de la jerarquía, con el subsecuente agrupamiento de esas clases en conceptos más generales. Por ejemplo, comenzamos definiendo clases para los vinos `Pauillac` y `Margaux`. Luego creamos una superclase común para esas dos clases (`Medoc`) la cual a su vez es una subclase de `Bordeaux`.

- Un proceso de desarrollo **combinado** es el resultado de una combinacion de los enfoques top-down y bottom-up: primero definimos los conceptos más sobresalientes y luego los generalizamos y especializamos apropiadamente. Podríamos comenzar con unos cuantos conceptos de nivel superior como `Vino`, y unos conceptos específicos, como `Margaux`. Podemos luego relacionarlos en un concepto de nivel medio, tal como `Medoc`. Podríamos luego desear generar todas las clases de vino regional de Francia, generando en consecuencia un cierto número de conceptos de nivel medio.

La Figura 2 muestra una posible descomposición entre los diferentes niveles de generalidad.

Ninguno de esos tres métodos es inherentemente mejor que cualquiera de los otros. El enfoque a tomar depende fuertemente de la visión personal del dominio. Si un desarrollador tiene una visión sistemática top-down del dominio, entonces será más fácil usar el enfoque top-down. El enfoque combinado es a menudo es el más fácil para muchos desarrolladores de ontologías, puesto que los "conceptos del medio" tienden a ser conceptos más descriptivos en el dominio (Rosch 1978).

Si tiendes a pensar en vinos distinguiendo primero la clasificación más general, entonces el enfoque top-down podría funcionar mejor para ti. Si prefieres comenzar listando ejemplos específicos, el enfoque bottom-up podría ser el más apropiado.

Sea cual sea el enfoque que elijamos, usualmente comenzaremos definiendo las clases. De la lista creada en el Paso 3, seleccionamos los términos que describen objetos que tienen existencia independiente en lugar de términos que describen esos objetos. Esos términos serán las clases

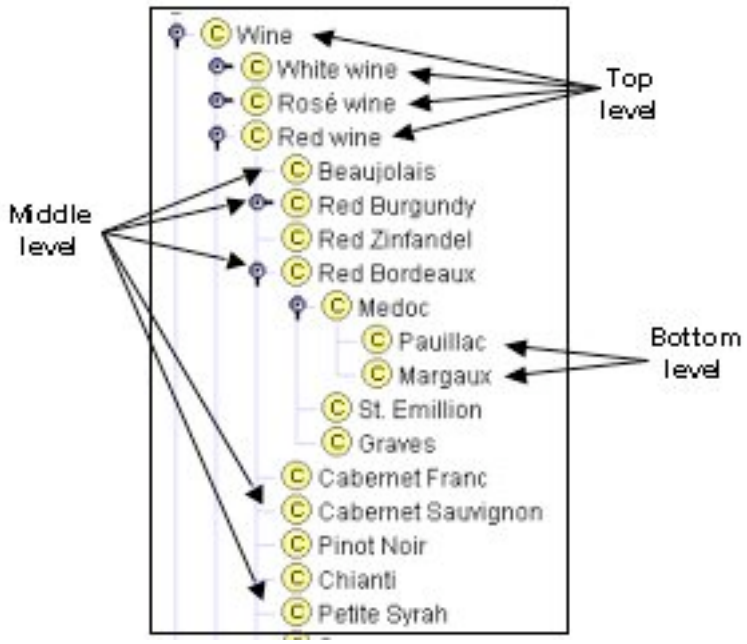

Figure 2: Los diferentes niveles de la taxonomía de los Vinos: `Vino` (Wine), `Vino rojo` (Red wine), `Vino blanco` (White wine), `Vino rosado` (Ros wine) son los conceptos más generales (nivel superior (top level)). `Pauillac` y `Margaux` son las clases más específicas en la jerarquía (nivel inferior (bottom level)).

de la ontología y llegarán a ser anclas en la jerarquía de clases[2]. Organizamos las clases en una taxonomía jeráquica preguntando si siendo una instancia de una clase, el objeto necesariamente será (i.e., por definición) una instancia de alguna otra clase.

Si una clase A es una superclase de la clase B, entonces cada instancia de B lo es también de A.

En otras palabras, la clase B representa un concepto que es un "tipo de" A.

Por ejemplo, cada vino Pinot Noir es necesariamente un vino rojo. Por lo tanto, la clase Pinot Noir es una subclase de la clase Vino Rojo.

La Figura 2 muestra una parte de la jerarquía de clases de la ontología de Vinos. La sección 4 contiene una discusión detallada de algunos aspectos a considerar cuando se está definiendo una jerarquía de clases.

Paso 5. Definir las propiedades de las clases: slots

Las clases aisladas no proveerán suficiente información para responder las preguntas de competencia del Paso 1. Una vez que hemos definido algunas de las clases, debemos describir la estructura interna de los conceptos.

Ya hemos seleccionado clases de la lista de términos creada en el Paso 3. La mayoría de los términos restantes son muy probablemente propiedades de esas clases. Esos términos incluyen,

[2]Podemos también ver a las clases como predicados unarios: preguntas que tienen un argumento. Por ejemplo, "¿Este objeto es un vino?" Los predicados unarios (o clases) se diferencian de los predicados binarios (o slots): preguntas que tienen dos argumentos. Por ejemplo, "¿El sabor de este objeto es fuerte?" "¿Cuál es el sabor de este objeto?"

por ejemplo, un `color` de vino, `cuerpo`, `sabor`, `contenido de azúcar` y `localización` de un establecimiento vinícola.

Para cada propiedad en la lista, debemos determinar qué clase es descrita por la propiedad. Esas propiedades se convierten en slots adosados a las clases. De esta forma, la clase `Vino` tendrá los siguientes slots: `color`, `sabor`, y `azúcar`. Y la clase `Establecimiento vinícola` tendrá un slot `localización`.

En general, hay varios tipos de propiedades de objeto que pueden llegar a ser slots en una ontología:

- propiedades "intrínsecas" tales como el sabor de un vino;
- propiedades "extrínsicas" tales como el nombre de un vino, y el área de donde proviene;
- partes, si el objeto es estructurado; pueden ser "partes" físicas y abstractas (e.g., los platos de una comida)
- relaciones con otros individuos; éstas son las relaciones entre miembros individuales de una clase y otros ítems (e.g., el productor de vino, representando una relaci'on entre un vino y un establecimiento vinícola, y la uva con la cual el vino está producido.)

De esta forma, además de las propiedades que hemos identificado previamente, necesitamos añadir los siguientes slots a la clase `Vino`: `nombre`, `área`, `productor`, `cepaje`. La Figura 3 muestra los slots para la clase Vino.

Todas las subclases de una clase **heredan** los slots de esa clase. Por ejemplo, todos los slots de la clase **Vino** serán heredados por todas las subclases de **Vino**, que incluyen `Vino Rojo` y `Vino Blanco`. Agregaremos un slot adicional, `nivel de tanino` (bajo, moderado, o alto), a la clase `Vino Rojo`. El slot `nivel de tanino` será heredado por todas las clases que representan vinos rojos (tales como `Bordeaux` y `Beaujolais`).

Un slot deberá estar adosado a la clase más general que pueda tener esa propiedad. Por ejemplo, el `cuerpo` y `color` de un vino deberán estar adosados a la clase `Vino`, puesto que ésta es la clase más general cuyas instancias tendrán un cuerpo y un color.

Template Slots

Name	Type	Cardinality	Other Facets
body	Symbol	single	allowed-values={FULL,MEDIUM,LIGHT}
color	Symbol	single	allowed-values={RED,ROSÉ,WHITE}
flavor	Symbol	single	allowed-values={DELICATE,MODERATE,STRONG}
grape	Instance	multiple	classes={Wine grape}
maker	Instance	single	classes={Winery}
name	String	single	
sugar	Symbol	single	allowed-values={DRY,SWEET,OFF-DRY}

Figure 3: Los diferentes niveles de la taxonomía de `Vinos`: `Vino` (`Wine`), `Vino rojo` (`Red wine`), `Vino blanco` (`White wine`), `Vino rosado` (`Rosé wine`) son los conceptos más generales, el nivel superior. `Pauillac` y `Margaux` son las clases más específicas en la jerarquía, el nivel inferior.

Paso 6. Definir las facetas de los slots

Los slots puedes tener diferentes facetas que describen el tipo de valor, valores admitidos, el número de los valores (cardinalidad), y otras características de los valores que los slots pueden tomar. Por ejemplo, el valor del slot `nombre` (como en "el `nombre` de un vino") es una cadena

de caracteres. Es decir, `nombre` es un slot con `String` como tipo de valor. El slot `produce` (como en "un establecimiento vinícola `produce` tales vinos") puede tener valores múltiples y los valores son instancias de la clase `Vino`. Es decir, `produce` es un slot con `Instance` como tipo de valor y `Vino` como clase admitida.

Describiremos ahora varias facetas comunes.

Cardinalidad del slot

La cardinalidad de un slot define cuantos valores un slot puede tener. Algunos sistemas solamente distinguen entre cardinalidad simple (admitiendo a lo sumo un valor) y cardinalidad múltiple (admitiendo cualquier cantidad de valores). Los vinos producidos por un establecimiento vinícola particular completan el slop `produce` que es de cardinalidad múltiple para la clase `Establecimiento vinícola`.

Algunos sistemas admiten la especificación de una cardinalidad mínima y máxima para describir la cantidad de valores de un slot con más precisión. Una cardinalidad mínima N significa que un slot debe tener al menos N valores. Por ejemplo, el slot `cepaje` de un Vino tiene una cardinalidad mínima de 1: cada vino esta hecho de al menos una variedad de uva. Una cardinalidad máxima M siginifica que un slot puede tener a lo sumo M valores. La cardinalidad máxima para el slot `cepaje` para vinos de simple variedad de uva es 1: esos vinos son hechos de solamente una variedad de uva. Algunas veces puede ser útil fijar la máxima cardinalidad en 0. Esta definición indicaría que el slot no puede tener ningún valor particular para una subclase particular.

Tipo de valor de los slots

Una faceta tipo de valor describe qué tipos de valores pueden llenar el slot. Aqui está una lista de los tipos de valores más comunes:

- **String** es el tipo de valor más simple el cual es usado por slots tales como nombre: el valor es una simple cadena de caracteres
- **Number** (algunas veces los tipos de valores Float e Integer son usados por ser más específicos) describe slots con valores numéricos. Por ejemplo, el `precio` que un vino puede tener es un tipo de valor Float.
- Los slots del tipo **Boolean** son simples banderas si/no. Por ejemplo, si elegimos no representar vinos espumantes como una clase separada, que el vino sea espumante o no, puede ser representado como un valor de un slot Boolean: si el valor es "true" ("si") el vino es espumante y si el valor es "false" ("no") el vino no es espumante.
- Los slots del tipo **Enumerated** especifican una lista específica de valores admitidos para el slot. Por ejemplo, podemos especificar que slot `sabor` puede tomar uno de los siguientes valores posibles: `fuerte`, `moderado` y `delicado`. En Protégé-2000 los slots enumerados son del tipo Symbol.
- Los slots del tipo **Instance** admiten la definición de relaciones entre individuos. Los slots con tipo de valor Instance deben también definir una lista de clases admitidas de las cuales las instancias pueden provenir. Por ejemplo, el slot `produce` de la clase `Establecimiento vinícola` puede tener instancias de la clase `Vino` como sus valores[3].

[3] Algunos sistemas solo especifican el tipo de valor con la clase en lugar de exigir un enunciado especial de slots de tipo instancia.

La Figura 4 muestra la definición del slot `produce` en la clase `Establecimiento vinícola` (Winery). El slot tiene cardinalidad múltiple, Instance como tipo de valor, y la clase `Vino` (Wine) como clase admitida para sus valores.

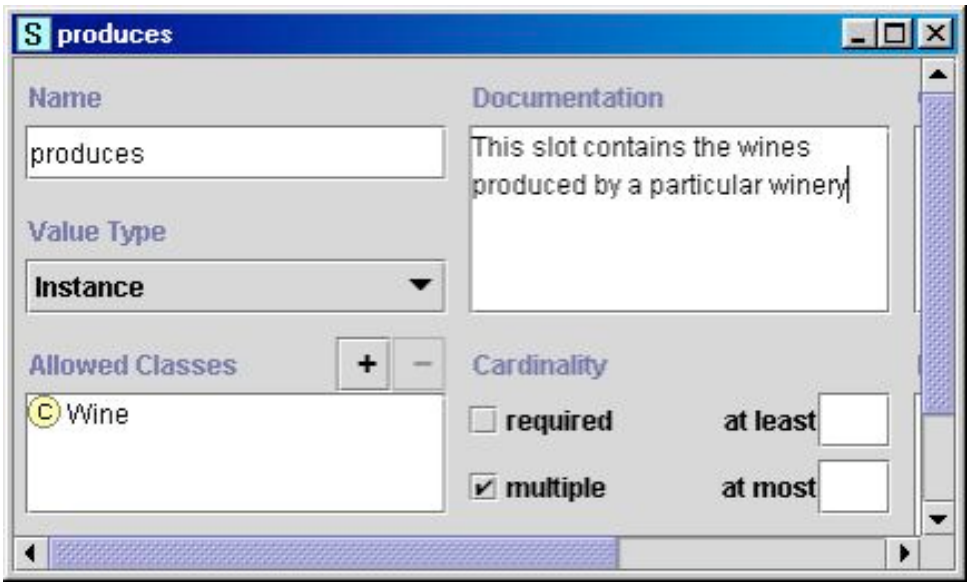

Figure 4: La definición del slot `produce` que describe los vinos producidos por un establecimiento vinícola. The slot has cardinality multiple, value type Instance, and the class Wine as the allowed class for its values.

Dominio y rango de un slot

Las clases admitidas para los slots de tipo Instance son a menudo llamadas **rango** de un slot. En el ejemplo de la Figura 4, la clase `Vino` es el rango del slot `produce`.Algunos sistemas permiten restringir el rango de un slot cuando el slot esta adosado a una clase particular.

Las clases a las cuales un slot está adosado o las clases cuyas propiedades son descritas por un slot son llamadas **dominio** del slot. La clase `Establecimeinto vinícola` es el dominio del slot `produce`. En los sistemas en los cuales adosamos slots a las clases, las clases a las cuales el slot es adosado usualmente constituye el dominio del slot. No hay necesidad de especificar el dominio separadamente.

Las reglas básicas para determinar un dominio y un rango de un slot son similares:

Cuando se define un dominio o rango de un slot, se debe encontrar las clases o clase más generales que puedan ser respectivamente el dominio o rango de los slots. Por otro lado, no definir un dominio ni rango que sea demasiado general: todas las clases en el dominio de un slot deben ser descritas por el slot y las instancias de todas las clases en el rango de un slot deben poder ser rellenos potenciales del slot. No elegir una clase demasiado general para el rango (i.e., es inútil de crear un rango COSA(THING)) pero es posible elegir una clase que cubre todos los valores de relleno.

En lugar de listar todas las las subclases posibles de la clase `Vino` para el ranfo del slop `produce` solo listar `Vino`. Al mismo tiempo, no queremos especificar el rango del slot como COSA (THING: la clase más general en una ontología).

En términos más especificos:

Si una lista de clases que definen un rango o un dominio de un slot incluye una clase y sus subclases, remover la subclase.

Si el rango de un slot contiene la clase `Vino` y la clase `Vino Rojo`, podemos remover `Vino Rojo` del rango porque no añade nueva información: El `Vino Rojo` es una subclase de `Vino` y por lo tanto el rango del slot ya lo incluye implícitamente como también todas las otras subclases de la clase `Vino`.

Si una lista de clases que definen un rango o dominio de un slot contiene todas las subclases de la clase A, pero no la clase A es sí, el rango debería contener solamente la clase A y no las subclases.

En lugar de definir el rango del slot para incluir `Vino Rojo`, `Vino Blanco` y `Vino Rosado` (enumeración de todas las subclases directas de `Vino`), podemos limitar el rango a la clase `Vino` como tal.

Si una lista de clases que definen un rango o dominio de un slot contiene unas cuantas subclases de la clase A, considerar si la clase A daría una definición de rango más apropiada.

En sistemas en los cuales adosar un slot a una clase es lo mismo que agregar la clase al dominio del slot, las mismas reglas se aplican al adosado del slot: Por un lado, debemos tratar de hacerla tan general como sea posible. Por otro lado, debemos asegurar que cada clase a la cual adosamos el slot pueda en efecto tener la propiedad que el slot representa. Podemos adosar el slot del `nivel de tanino` a cada clase que representa a los vinos rojos (e.g., Bordeaux, Merlot, Beaujolais, etc.). Sin embargo, puesto que todos los vinos rojos tienen la propiedad `nivel de tanino`, deberíamos adosar en su lugar el slot a esta clase más general de `Vinos Rojos`. Si adicionalmente, generalizamos el dominio del slot del `nivel de tanino` (adosándolo en su lugar a la clase `Vino`) no sería correcto puesto que no usamos el `nivel de tanino` para describir por ejemplo a los vinos blancos.

Paso 7. Crear instancias

El último paso consiste en crear instancias individuales de clases en la jerarquía. La definición de una instancia individual de una clase requiere (1) elegir una clase, (2) crear una instancia individual de la clase y (3) rellenar los valores del slot. Por ejemplo, podemos crear una instancia individual `Chateau-Morgon-Beaujolais` para representar un tipo específico de vino `Beaujolais`. `Chateau-Morgon-Beaujolais` es una instancia de la clase `Beaujolais` que representa a todos los vinos `Beaujolais`. Esta instancia tiene definidos los siguientes valores de slot (Figura 5):

- Cuerpo: Ligero
- Color: Rojo
- Aroma: Delicado
- Nivel de tanino: Bajo
- Cepaje: Gamay (instancia de la clase `Uva`)
- Productor: Chateau-Morgon (instancia de la clase `Establecimiento vinícola`)
- Región: Beaujolais (instancia de la clase `Región-Vino`)
- Azúcar: Seco

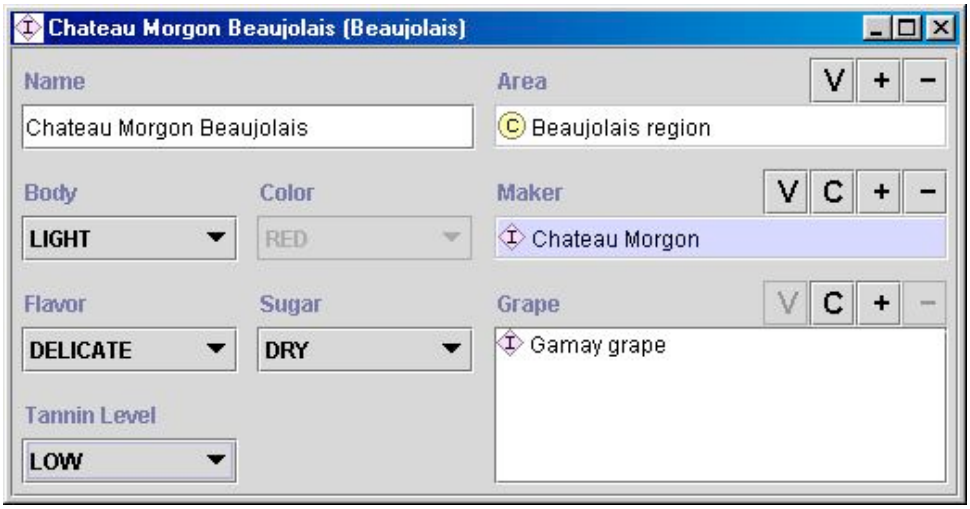

Figure 5: La definición de una instancia de la clase `Beaujolais`. La instancia es Chateau Morgon Beaujolais de la región `Beaujolais`, producida con la uva Gamay por el establecimiento vinícola `Chateau Morgon`. Tiene un cuerpo ligero, aroma delicado, color rojo y bajo nivel del tanino. Es un vino seco.

4 Definición de las clases y de la jerarquía de clases

Esta sección discute aspectos en los cuales hay que tener cuidado y errores que son fáciles de cometer cuando se definen clases y jerarquías de clases (Paso 4 de la Sección 3). Como lo mencionamos antes, no hay una jerarquía de clases correcta para un dominio dado. La jerarquía depende de los posibles usos de la ontología, el nivel de detalle que es necesario para la aplicación, preferencias personales, y algunas veces requerimientos de compatibilidad con otros modelos. Sin embargo, discutiremos varias recomendaciones para tenerlas en cuenta cuando se desarrolle una jerarquía de clases. Después de haber definido un número considerable de nuevas clases, es útil detenerse y verificar si la jerarquía emergente va de acuerdo a esas recomendaciones.

4.1 Asegurarse que la jerarquía de clases es correcta

Una relación "is-a"

la jerarquía de clases representa una relación "is-a" ("es-un, es-una"): una clase A es una subclase de B si cada instancia de B es también una instancia de A. Por ejemplo, `Chardonnay` es una subclase de `Vino Blanco`. Otra forma de pensar en la relación taxonómica es viéndola como una relación "kind-of" ("tipo-de"): `Chardonnay` es un tipo de `Vino Blanco`. Un avión comercial es un tipo de avión. Carne es un tipo de alimento.

Una subclase de una clase representa un concepto que es un "tipo de" concepto que la superclase representa.

Un simple vino no es una subclase de todos los vinos

Un error común de modelamiento es el de incluir una versión singular y plural del mismo concepto en la jerarquía haciendo esta anterior una subclase de la ultima. Por ejemplo, está mal definir una clase `Vinos` y una clase `Vino` como una subclase de `Vinos`. Cuando tu piensas en la jerarquía como representación de la relación "kind-of" ("tipo-de"), el error de modelamiento se

hace claro: un simple `Vino` no es un tipo de `Vinos`. La mejor forma de evitar ese tipo de error es utilizando siempre la forma singular o plural al nombrar las clases (ver la Sección 6 sobre la discusión del nombrado de conceptos).

Transitividad de las relaciones jerárquicas

Una relación de subclase es transitiva:

Si B es una subclase de A y C es una subclase de B, entonces C es una subclase de A

Por ejemplo, podemos definri la clase `Vino`, y luego definir la clase `Vino Blanco` como una subclase de `Vino`. Luego definimos una clase `Chardonnay` como una subclase de `Vino Blanco`. La transitividad de la relación de subclase significa que la clase `Chardonnay` es también una subclase de `Vino`. Algunas veces hacemos la distinción entre subclases directas y subclases indirectas. Una **subclase directa** es la subclase "más cercana" de la clase: no hay clases entre la clase y sus subclases directas en la jerarquía. Es decir, no hay otras clases en la jerarquía entre la clase y su superclase directa. En nuestro ejemplo, `Chardonnay` es una subclase directa de `Vino Blanco` y no es una subclase directa de `Vino`.

Evolución de una jerarquía de clases

Mantener una jerarquía consistente de clases puede llegar a ser desafiante a media que el dominio evoluciona. Por ejemplo, por muchos años, todos los vinos `Zinfandel` fueron rojos. Por lo tanto, definimos una clase de vinos `Zinfandel` como una subclase de la clase `Vino Rojo`. Algunas veces, sin embargo, los productores comenzaron a presionar las uvas y extraer inmediatamente los elementos de las uvas que producen color, modificando en consecuencia el color del vino resultante. Por lo tanto, obtenemos "zinfandel blanco" cuyo color es rosado. Ahora necesitamos dividir la clase `Zinfandel` en dos clases de zinfandel (`Zinfandel Blanco` y `Zinfandel Rojo`) y clasificarlos como subclases de `Vino Rosado` y `Vino Rojo` respectivamente.

Las clases y sus nombres

Es importante distinguir entre una clase y su nombre:

Las clases representan conceptos en el dominio y no las palabras que denotan esos conceptos.

El nombre de una clase puede cambiar si elegimos una terminología diferente, pero el término como tal representa la realidad objetiva en el mundo. Por ejemplo, podemos crear un clase `Camarón` y luego renombrarla a `Gamba` (la clase aun representa el mismo concepto). Combinaciones apropiadas de vinos que hacen referencia a platos con camarones deben hacer referencia a platos con gambas. En términos más prácticos, la siguiente regla siempre debe ser seguida:

Los sinónimos para el mismo concepto no representan clases diferentes.

Los sinónimos son solo nombres diferentes para un concepto o término. Por lo tanto, no deberíamos tener una clase llamada `Camarón` y una clase llamada `Gamba`, y posiblemente una clase llamada `Crevette`. En su lugar, hay una clase, llamada `Camarón` o `Gamba`. Muchos sistemas admiten la asociación de una lista de sinónimos, traducciones, o nombres de presentación con una clase. Si un sistema no permite estas asociaciones, los sinónimos siempre podrían ser listados en la documentación de la clase.

Evitar ciclos en las clases

Debemos evitar ciclos en la jerarquía de clases. Se dice que hay un ciclo en una jerarquía cuando una clase A tiene una subclase B y al mismo tiempo B es una superclase de A. Crear un ciclo como ese en un jerarquía equivale a declarar que las clases A y B son equivalentes: todas las instancias de A son instancias de B y todas las instancias de B son también instancias de A. En efecto, puesto que B es una subclase de A, todas las instancias de B deben ser instancias de la clase A. Puesto que A es una subclase de B, todas las instancias de A deben también ser instancias de la clase B.

4.2 Análisis de clases hermanas en la jerarquía de clases

Clases hermanas en una jerarquía de clases

Las **clases hermanas** en una jerarquía son clases que son subclases directas de la misma clase (ver Sección 4.1).

Todas las clases hermanas en una jerarquía (excepto para las que están al nivel de la raíz) deben estar al mismo nivle de generalidad.

Por ejemplo, `Vino Blanco` y `Chardonnay` no deberían ser clases de la misma clase (digamos `Vino`). `Vino Blanco` es un concepto más general que `Chardonnay`. Las clases hermanas deben representar conceptos que caen "en la misma línea" de la misma forma que las secciones de un mismo nivel en un libro están al mismo nivel de generalidad. En ese sentido, los requerimientos para una jerarquía de clases son similares a los requerimientos para una estructuración de un libro.

Sin embargo, los conceptos en la raíz de la jerarquía (los cuales son a menudo representados como subclases directas de alguna clase muy general, como `Thing` (`Cosa`)) representan divisiones principales del dominio y no tienen que ser conceptos similares.

¿Cuán mucho es demasiado y cuán poco es insuficiente?

No hay reglas que digan el número de subclases directas que una clase debería tener. Sin embargo, varias ontologías bien estructuradas tienen entre dos y una docena de subclases directas. Por lo tanto, consideremos las siguientes reglas:

Si una clase tiene solamente una subclase directa, puede existir un problema de modelamiento o sino la ontología no está completa. Si hay más de una docena de subclases para una clase dada, entonces categorías intermedias adicionales pueden ser necesarias.

La primera de las dos reglas e similar a la regla de composición tipográfica en la que las listas con viñetas nunca deberían tener solamente una viñeta. Por ejemplo, la mayoría de los vinos rojos de Borgoña son vinos Côtes d'Or. Supongamos que queremos representar solamente este tipo de mayoritario de vinos de Borgoña. Podríamos crear una clase `Borgoña Rojo` y luego una simpel subclase `Côtes d'Or` (Figura 6a). Sin embargo, si en nuestra representación los vinos rojo de Borgoña y Côtes d'Or son esencialmente equivalentes (todos los vinos rojos de Borgoña son Côtes d'Or y todos los vinos Côtes d'Or son rojos de Borgoña), la creación de la clase `Côtes d'Or` no es necesaria ya que no adiciona nueva información a la representación. Si deberíamos incluir los vinos Côtes Chalonnaise, los cuales son vinos de Borgoña más baratos de la región sur de Côtes d'Or, entonces crearíamos dos subclases de la clase `Borgoña`: `Cotes dOr` y `Cotes Chalonnaise` (Figura 6b).

Supongamos ahora que listamos todos los tipos de vinos como subclases directas de la clase `Vino`. Esta lista entonces incluiría tipos más generales de vinos tales como Beaujolais y Bordeaux, como también tipos más específicos de vinos tales como Paulliac y Margaux (Figura 7a).

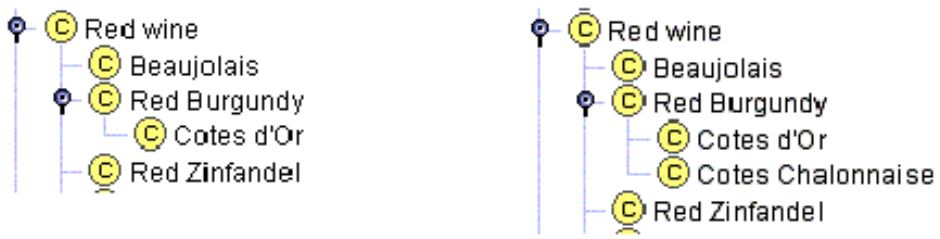

Figure 6: Subclases de la clase `Rojo de Borgoña` (`Red Burgundy`). Tener una sola subclase de la clase usualmente indica un problema de modelamiento.

La clase `Vino` tiene varias subclases directas y, de hecho, para que la ontología refleje los diferentes tipos de vino en una manera más organizada, `Medoc` debería ser una subclase de `Bordeaux` y `Cotes d'Or` debería ser una subclase de Borgoña. Tener tales categorías intermedias como `Vino Rojo` y `Vino Blanco` también reflejaría el modelo conceptual del dominio de vinos que mucha gente tiene (Figura 7b).

Sin embargo, si no existen clases naturales para agrupar los conceptos en la larga lista de clases hermanas, no hay la necesidad de crear clases artificiales (dejar las clases en la forma que están). Después de todo, la ontología es un reflejo del mundo real y si no existen categorizaciones en el mundo real, entonces la ontología debería reflejar eso.

4.3 Herencia múltiple

La mayoría de los sistemas de representación de conocimiento admiten herencia múltiple en la jerarquía de clases: una clase puede ser subclase de varias clases. Supongamos que deseamos crear una clase separada de vinos de sobremesa, la clase `Vino de Sobremesa`. El vino Porto es al mismo tiempo vino rojo y vino de sobremesa[4]. Por lo tanto, definimos una clase `Porto` con dos superclases: `Vino Rojo` y `Vino de Sobremesa`. Todas las instancias de la clase `Porto` serán instancias de la clase `Vino Rojo` y de la clase `Vino de Sobremesa`. La clase `Porto` heredará sus slots y sus facetas de sus dos superclases. De esta forma, ésta heredará el valor `DULCE` para el slot `Azúcar` de la clase `Vino de Sobremesa` y el slot `nivel de tanino` y el valor para su slot `color` de la clase `Vino Rojo`.

4.4 ¿Cuándo introducir (o no) una nueva clase?

Una de las más difíciles decisiones de tomar durante el modelamiento es cuándo introducir una nueva clase o cuándo representar una desemejanza a través de diferentes valores de propiedades. Es difícil navegar en una jerarquía extremadamente anidada con varias clases raras y en un jerarquía muy plana que tiene pocas clases con mucha información codificada en los slots. Sin embargo, no es fácil encontrar el balance apropiado.

Hay varias reglas de base que ayudan a decidir cuándo introducir nuevas clases en la jerarquía.

La subclases de un clase usualmente (1) tienen propiedades adicionales que la superclase no tiene, o (2) diferentes restricciones de las de las superclase , o (3) participan en relaciones diferentes que la superclases.

[4]Decidimos representar solamente vinos Portos rojos en nuestra ontología: existen también Portos blancos pero son sumamente raros.

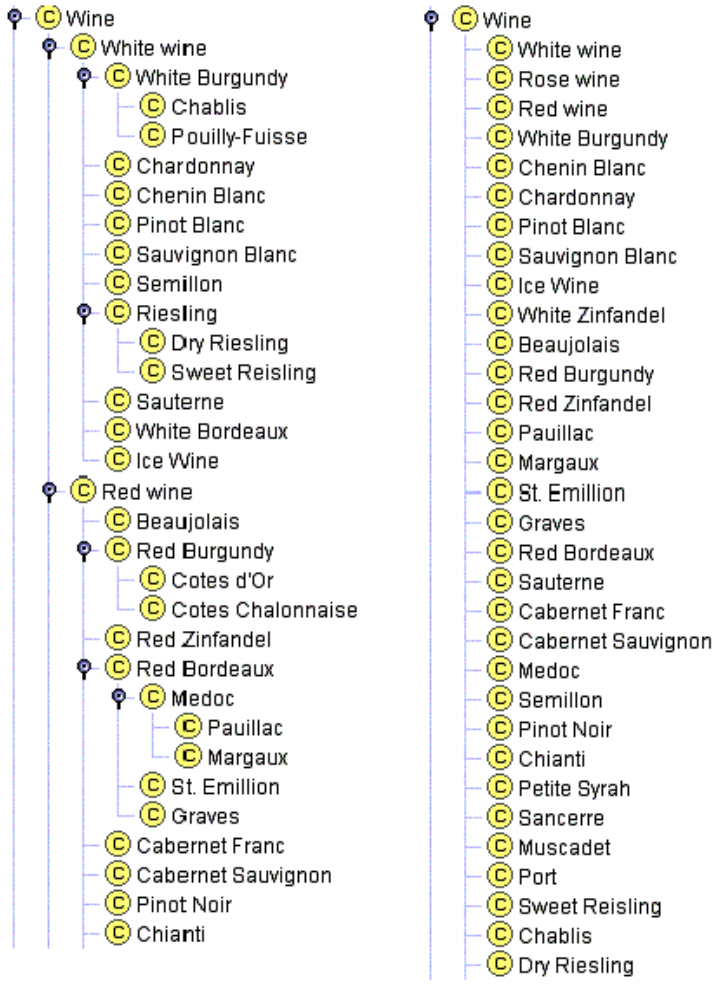

Figure 7: Categorización de vinos. Contar con todos los vinos y tipos de vino en contraste a contar con varios niveles de categorización.

Los vinos rojos pueden tener diferentes niveles de tanino, sin embargo esta propiedad no es usada para describir los vinos en general. El valor para el slot `azúcar` del `Vino de Sobremesa` es DULCE, sin embargo no es el caso para la superclase de la clase `Vino de Sobremesa`. Los vinos Pinot Noir pueden servirse con mariscos mientras que otros vinos rojos no. En otras palabras, introducimos una nueva clase en la jerarquía usualmente solo cuando hay algo hay algo que podamos decir acerca de esta clase que no podamos decir acerca de la superclase.

En la práctica, cada subclase debe tener nuevos slots a˜adidos a ésta, o tener nuevos valores definidos para el slot, o sustituir (*override*) algunas facetas de los slots heredados.

Sin embargo, puede ser útil crear nuevas clases an cuando no introduzcan nuevas propiedades.

Las clases en terminologías jerárquicas no necesitan introducir nuevas propiedades.

Por ejemplo, algunas ontologías incluyen grandes jerarquías de referencia con términos comunes usados en el dominio. Por ejemplo, una ontología que es subyacente a un sistema de registro electrónico medico puede incluir una clasificación de varias enfermedades. Esta clasificación puede ser solo eso: una jerarquía de términos sin propiedades (o con el mismo conjunto de propiedades). En ese caso, es aún útil organizar los términos en la jerarquía en lugar de en una lista plana porque (1) permitirá una exploración y navegación más fácil y (2) facilitará

al médico la fácil elección de un nivel de generalidad del término que es apropiado para la situación.

Otra razón para introducir nuevas clases sin nuevas propiedades es para modelar conceptos entre los cuales los expertos del dominio comúnmente hacen una distinción aún cuando no hayamos decidido modelar la distinción en sí. Puesto que usamos las ontologías para facilitar la comunicación entre los expertos de un dominio y entre ellos mismos y los sistemas basados en conocimiento, deseamos reflejar en la ontología la visión del experto sobre el dominio.

Finalmente, no deberíamos crear subclases de una clase para cada restricción adicional. Por ejemplo, introdujimos las clases `Vino Rojo`, `Vino Blanco`, y `Vino Rosado` porque esta distinción es natural en el mundo de los vinos. No introdujimos clases para los vinos delicado, moderado, etc. Cuando definimos una jerarquía de clases, nuestra meta es de encontrar un balance entre crear nuevas clases útiles para la organización de clases y crear demasiadas clases.

4.5 ¿Una nueva clase o un valor de propiedad?

Cuando modelamos un dominio, a menudo necesitamos decidir si modelar una distinción específica (como vino blanco, rojo o rosado) como un valor de propiedad o como un conjunto de clases, nuevamente, depende del alcance del dominio y de la tarea en mano.

¿Creamos una clase `Vino Blanco` o simplemente creamos una clase `Vino` y llenamos diferentes valores para el slot `color`?. La respuesta usualmente está en el alcance que hemos definido para la ontología. '?Qué tan importante es el concepto `Vino Blanco` en nuestro dominio? Si los vinos tienen solamente importancia marginal en el dominio y si siendo blanco o no el vino no tiene ninguna implicación particular en sus relaciones con otros objetos, entonces no deberíamos introducir una clase separada para los vino blancos. Para un modelo de dominio usado en una factoría que produce etiquetas de vinos, las reglas de las etiquetas de vino de cualquier color son las mismas y la distinción no es importante. Por el contrario, para la representación de vinos, alimentos y sus combinaciones apropiadas, un vino rojo es muy diferente de un vino blanco: está emparejada con diferentes alimentos, tiene diferentes propiedades, y assúcesivamente. De manera similar, el color del vino es importante para la base de conocimientos de vinos que podríamos usar par determinar el orden de los elementos a considerar en la degustación del vino. De esta manera, creamos una clase separada para `Vino Blanco`.

Si los conceptos con diferentes valores de slot se vuelven restricciones para diferentes slots en otras clases, entonces debemos crear una nueva clase para esta distinción. Caso contrario, representamos la distinción en un valor de slot.

De manera similar, nuestra ontología de vinos tiene clases tales como `Rojo Merlot` y `Blanco Merlot`, en lugar de una simple clase para todos los vinos Merlot: los Merlots rojos y los Merlots blancos son realmente vinos diferentes (aunque producidos del mismo cepaje) y si estamos desarrollando una ontolgía detallada de vinos, esta distinción es importante.

Si la distinción es importante en el dominio y pensamos en los objetos con diferentes valores para la distinción como diferentes tipos de objetos, entonces deberíamos crear una nueva clase para la distinción.

Considerar potenciales instancias individuales de una clase puede ser también útil al decidir si se introduce una nueva clase o no.

Una clase a la cual una instancia individual pertenece no dbería cambiar a menudo.

Usualmente, cuando usamos propiedades extrínsecas en lugar de intrínsecas de los conceptos para diferenciarlos entre las clases, las instancias de esas clases tendrán que migrar a menudo de una clase a otra. Por ejemplo, `Vino Enfriado` no debería ser una clase en una ontología que describe las botellas de vino en un restaurante. La propiedad `enfriado` debería simplemente ser un atributo del vino en una botella puesto que una instancia de `Vino Enfriado` puede fácilmente dejar de ser una instancia de esta clase y llegar a ser llegar a ser instancia de esta clase de nuevo.

Usualmente, los números, colores, localizaciones son valores de slots y no conducen a la creación de nuevas clases. En el caso del vino, sin embargo, existe una notable excepción puesto que el color del vino es primordial para la descripción del vino.

Tomemos otro ejemplo, consideremos una ontología de la anatomía humana. Cuando representamos las costillas, ¿creamos clases para "1^{ra} costilla izquierda", "2^{da} costilla izquierda" y así sucesivamente? O ¿creamos una clase `Costilla` con slots para el orden y la posición lateral (izquierda-derecha)?[5] Si al información de cada costilla que representamos en la ontología es significativamente diferente, entonces deberíamos de hecho crear una clase para cada una de las costillas. Es decir, si deseamos representar detalles de la información de adyacencia y localización (la cual es diferente para cada costilla) como también funciones específicas que cada costilla juega y órganos que proteje, entonces nos interesa las clases. Si estamos modelando la anatomía a un nivel ligeramente leve de generalidad y todas las costillas son muy similares en nuestras aplicaciones potenciales (se trata de ver cuĺ costilla está rota en los rayos X sin implicaciones en las otras partes del cuerpo), entonces podríamos simplificar nuestra jerarquía y tener simplemente la clase `Costilla`, con dos slots: posición lateral y orden.

4.6 ¿Una instancia o una clase?

Decidir si un concepto particular es una clase en la ontología o una instancia individual depende de cuáles son las aplicaciones potenciales de la ontología. Decidir dónde las clases terminan y las instancias comienzan, empieza por la decisión de cuál es el nivel más bajo de granularidad en la representación. El nivel de granularidad es a su vez determinado por una aplicación potencial de la ontología. En otras palabras, '?Cuáles son los ítems más específicos que representaremos en la base de conocimientos? Volviendo a las preguntas de competencia que hemos identificado en el Paso 1 de la Sección 3, los conceptos más específicos que constituirán respuestas a esas preguntas serán muy buenos candidatos para ser individuos en la base de conocimientos.

La instancias individuales son los conceptos más específicos representados en una base de conocimientos.

Por ejemplo, si solo hablaremos del emparejado de vinos con alimentos, no estaremos interesados en específicas botellas físicas de vino. Por lo tanto, términos como `Sterling Vineyards Merlot` serán probablemente los términos más específicos que usemos. De este modo, `Sterling Vineyards Merlot` será una instancia en la base de conocimientos.

Por otro lado, si deseamos mantener un inventario de los vinos del restaurante además de la base de conocimientos de buenas parejas vino-alimento, las botellas individuales de cada vino llegarán a ser instancias individuales en nuestra base de conocimientos.

De manera similar, si deseamos registrar las diferentes propiedades de cada cosecha específica de los `Sterling Vineyards Merlot`, entonces la cosecha específica de vino es una instancia en

[5] Asumimos que cada órgano anatómico es una clase puesto que queremos también discutir de "la 1^{ra} costilla izquierda de Juan". Los órganos individuales de toda persona serían representados individualmente en nuestra ontología.

la base de conocimientos y `Sterling Vineyards Merlot` es una clase que contiene instancias para todas sus cosechas.

Otra regla puede "desplazar" algunas instancias individuales al conjunto de clases:

Si los conceptos forman una jerarquía natural, entonces deberíamos representarlos como clases.

Consideremos las regiones de producción de vino. Inicialmente, podemos definir regiones principales producción de vino, tales como Francia, Estados Unidos, Alemania, y as sucesivamente, como clases, y regiones específicas de producción de vino dentro esas grandes regiones como instancias. Por ejemplo, la región de Borgoña es una instancia de la clase de `Región Francesa`. Sin embargo, también quisiéramos decir que la `Región Cotes d'Or` es una `Región de Borgoña`. En consecuencia, la `Región de Borgoña` debe ser una clase (para tener subclases o instancias). Sin embargo, parece arbitrario hacer que la `Región de Borgoña` sea una clase y que la `Región Cotes d'Or` sea una instancia de la `Región de Borgoña`: es muy difícil distinguir claramente qué regiones son clases y cuáles son instancias. Por lo tanto, definimos todas las regiones de producción de vino como clases. Protégé-2000 permite a los usuarios especificar algunas clases como *Abstract* (abstractas), indicando que la clase no puede tener ninguna instancia directa. En nuestro caso, todas las clases de las regiones de producción de vino son abstractas (Figura 8).

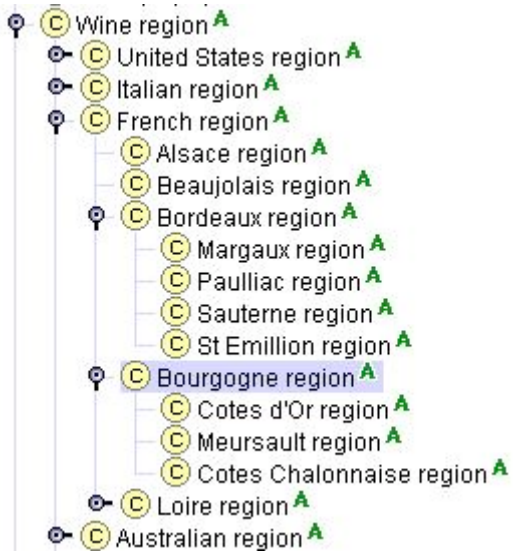

Figure 8: Jerarquía de las regiones de producción de vino. Los íconos "A" junto a los nombres de las clases indican que las clases son abstractas y no pueden tener ninguna instancia directa.

La misma jerarquía de clases sería incorrecta si omitimos la palabra "región" de los nombres de las clases. No podemos decir que la clase `Alsacia` (`Alsace`) es una subclase de de la clase `Francia`: Alsacia no es un tipo de Francia. Sin embargo, la región de Alsacia es un tipo de región de Francia.

Solamente las clases pueden ser dispuestas en una jerarquía (los sistemas de representación de conocimiento no tienen la noción de de sub-instancia). Por lo tanto, si existe una jerarquía natural entre los términos, como en las jerarquías terminológicas de la Sección 4.2, debemos definir esos términos como clases aunque no tengan ninguna instancia propia.

4.7 Limitación del alcance

Como nota final sobre la definición de una jerarquía de clases, el siguiente conjunto de reglas es siempre útil para decidir si una ontología está completa:

La ontología no debería contener toda la información posible del dominio: no necesitas especializar (o generalizar) más de lo que necesitas para tu aplicación (como máximo un nivel extra de cada lado).

En nuestro ejemplo de vinos y alimentos, no necesitamos saber qué papel es usado para las etiquetas o cómo cocinar gambas.

De manera similar, la ontología no debe contener todas las posibles propiedades de las clases ni las distinciones entre ellas en la jerarquía.

En nuestra ontología, sin duda no incluiremos todas las propiedades que un vino o alimento pueda tener. Representamos las propiedades más sobresalientes de las clases de ítems en nuestra ontología. Aunque los libros de vinos nos dirán el tamaño de las uvas, no incluimos ese conocimiento. De manera similar, no hemos agregado todas las relaciones que podríamos imaginar entre todos los términos de nuestro sistema. Por ejemplo, no incluimos las relaciones tales como `vino favorito` o `alimento preferido` en la ontología para tener una representación más completa de todas las interconexiones entre los términos que hemos definido.

Las últimas reglas también se aplican para establecer relaciones entre conceptos que ya los hemos incluido en la ontología. Consideremos una ontología que describe experimentos biológicos. La ontología probablemente contendrá un concepto de `Organismos Biológicos`. También contendrá el concepto de un `Experimentador` que ejecuta un experimento (con su nombre, afiliación, etc.). Es cierto que un experimentador es una persona, y como persona, también es casualmente un organismo biológico. Sin embargo, probablemente no debamos incorporar esta distinción en la ontología: para los propósitos de esta representación un experimentador no es un organismo biológico y probablemente nunca ejecutemos experimentos en los experimentadores como tal. Si estuvi'esemos representando todo lo que podamos decir de las clases en la ontología, un `Experimentador` llegaría a ser una subclase de `Organismo Biológico`. Sin embargo, no necesitamos incluir este conocimiento para las aplicaciones previsibles. De hecho, la inclusión de este tipo de clasificación adicional para las clases existentes en realidad es perjudicial: una instancia de un `Experimentador` tendrá slots para peso, edad, especie, y otros datos pertenecientes a un organismo biológico, pero absolutamente irrelevantes en el contexto de la descripción de un experimento. Sin embargo, debemos registrar tal decisión de diseño en la documentación para el beneficio de los usuarios que mirarán esta ontología y que no estarán enterados de la apliación que teníamos en mente.

4.8 Subclases disjuntas

Muchos sistemas nos permiten especificar explícitamente que varias clases sean disjuntas. La clases son **disjuntas** si no pueden tener ninguna instancia en común. Por ejemplo, las clases `Vino de Sobremesa` y `Vino Blanco` de nuestra ontología *no* son disjuntas: hay muchos vinos que son instancias de ambos. La instancia `Rothermel Trochenbierenauslese Riesling` de la clase `Riesling Dulce` es un ejemplo de ello. Al mismo tiempo, las clases `Vino Rojo` y `Vino Blanco` son disjuntas: ningún vino puede ser simultáneamente rojo y blanco. La especificación de clases disjuntas permite al sistema de validar la ontología de mejor manera. Si declaramos que las clases `Vino Rojo` y `Vino Blanco` son disjuntas y posteriormente creamos una clase que es una subclase de `Riesling` (una subclase de `Vino Blanco`) y `Porto` (una subclase de `Vino Rojo`), el sistema indicará que hay un error de modelamiento.

5 Definición de las propiedades (más detalles)

En esta sección discutimos muchos más detalles a tener en cuenta cuando definimos los slots de una ontología (Paso 5 y Paso 6 de la Sección 3). Principalmente, discutiremos sobre los roles inversos y valores por defecto de un slot.

5.1 Slots inversos

Un valor de un slot puede depender de un otro valor de otro slot Por ejemplo, si un vino fue `producido` por un `establecimiento vinícola`, entonces el `establecimiento vinícola` `produce` ese vino. Esas dos relaciones, `productor` y `produce`, son llamadas **relaciones inversas**. Almacenar la información "en ambos sentidos" es redundante. Cuando sabemos que un vino es producido por un establecimiento vinícola, una aplciación que usa una base de conocimientos puede siempre inferir el valor de la relación inversa: el establecimiento vinícola produce el vino. Sin embargo, desde la perspectiva de adquisición de conocimiento, es conveniente tener ambas piezas de la información disponibles explícitamente. Este enfoque permite a los usuarios introducir vinos en algunos casos y establecimientos vinícolas en otros. El sistema de adquisición de conocimiento podría entonces completar automáticamente el valor de la relación inversa asegurando la consistencia de la base de conocimientos.

Nuestro ejemplo tiene un par de slots inversos: el slot `productor` de la clase `Vino` y el slot `produce` de la clase `Establecimiento Vinícola`. Cuando un usuario crea una instancia de la clase `Vino` e introduce el valor para el slot `productor`, el sistema automáticamente añade la instancia recién creada al slot `produce` de la instancia correspondiente `Establecimiento Vinícola`. Por ejemplo, cuando decimos que el `Sterling Merlot` es producido por el establecimiento vinícola `Viñedos Sterling`, el sistema automáticamente agregaría `Sterling Merlot` a la lista de vinos que el establecimiento vinícola `Viñedos Sterling` (`Sterling Vineyard`) produce (Figura 9).

5.2 Valores por defecto

Muchos sistemas basados en marcos permiten la especificación de valores por defecto para los slots. Si un valor particular de un slot es el mismo para la mayoría de las instancias de una clase, podemos entonces definir como el **valor por defecto** para el slot. Entonces, cuando cada nueva instancia de una clase que contenga este slot sea creada, el sistema lo llenará con el valor por defecto automáticamente. Luego podemos cambiar ese valor a otro valor que las facetas lo permitan. Es decir, los valores por defecto están ahí por comodidad: no imponen ninguna nueva restricción sobre el modelo ni cambian el modelo en alguna forma.

Por ejemplo, si la mayoría de los vinos de los cuales vamos a discutir son vinos de cuerpo completo, podemos tener "completo" como valor por defecto para el cuerpo del vino. Entonces, a menos que indiquemos otra cosa, todos los vinos que definamos serán de cuerpo completo.

Notar que esto es diferente de los **valores de los slots**. Los valores de los slots no pueden ser cambiados. Por ejemplo, podemos decir que el slot `azúcar` tiene el valor `DULCE` para la clase `Vinos de Sobremesa`. Entonces, todas las subclases e instancias de la clase `Vino de Sobremesa` tendrán el valor `DULCE` para el slot `azúcar`. Este valor no puede ser cambiado en ninguna de las subclases ni instancias de la clase.

6 ¿Qué está en un nombre?

La definición de convenios sobre los nombres de los conceptos en una ontología y su uso estricto, no solamente que la ontología sea fácil de entender sino también ayuda a evitar algunos errores

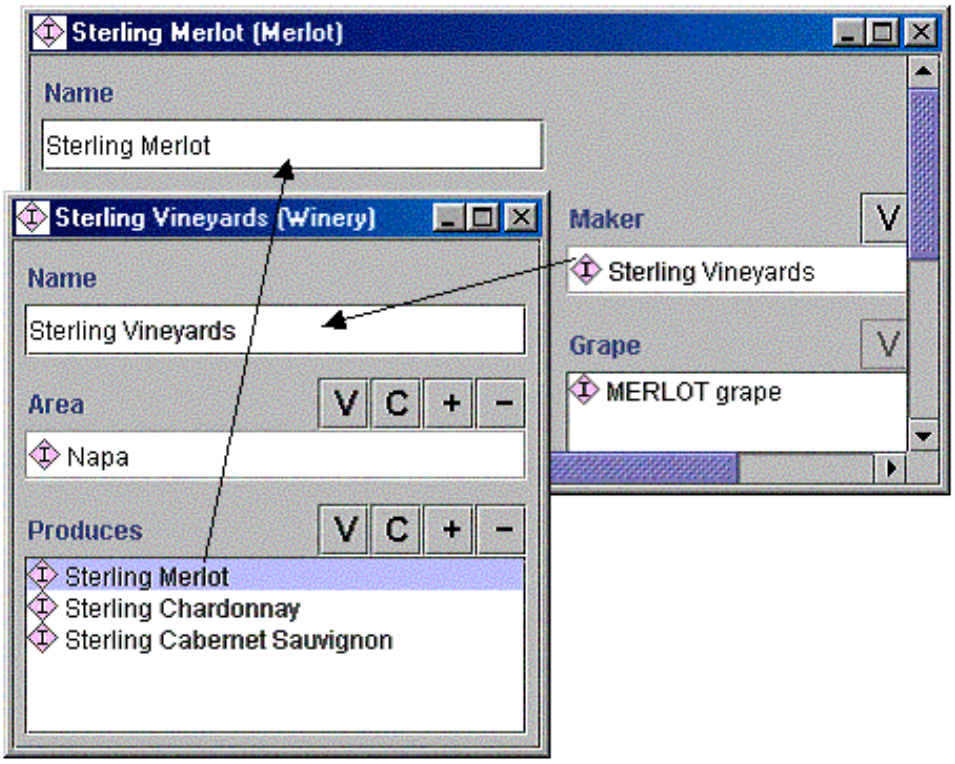

Figure 9: Instancias de slots inversos. El slot `produce` (`produces`) de la clase `Establecimiento Vinícola` (`Winery`) es un slot inverso del slot `productor` (`maker`) de la clase `Vino`. El llenado de uno de esos slots activa una actualización automática del otro.

comunes de modelamiento. Existen muchas alternativas para nombrar los conceptos. A menudo no hay ninguna razón particular para elegir un o otra alternativa. Sin embargo, necesitamos:

definir un convenio de nombrado de clases y slots, y adherirnos estrictamente a éste.

La siguientes características de un sistema de representación de conocimiento afectan la elección de convenios de nombrado:

- ¿Tiene el sistema el mismo espacio de nombres para las clases, slots e instancias? Es decir, ¿permite el sistema tener una clase y un slot con el mismo nombre (tales como la clase `establecimiento vinícola` y el slot `establecimiento vinícola`)?
- ¿El sistema diferencia entre mayúsculas y minúsculas? Es decir, ¿el sistema trata los nombres de manera diferente si si están escritos en mayúsculas o minúsculas (tales como `Establecimiento Vinícola` y `establecimiento vinícola`)?
- ¿Qué delimitadores permite el sistema en los nombres? Es decir, ¿los nombres pueden contener espacios, comas, asteriscos y así por el estilo?

Protégé-2000, por ejemplo, mantiene un solo espacio de nombres para todos sus marcos. Diferencia entre mayúsculas y minúsculas. En consecuencia, no podemos tener una clase `establecimiento vinícola` y un slot `establecimiento vinícola`. Sin embargo, podemos tener una clase `Establecimiento Vinícola` (mayúsculas) y un slot `establecimiento vinícola`. CLASSIC, por otro lado, no diferencia entre mayúsculas y minúsculas y mantiene diferentes espacios de nombres para las clases, slots e individuos. De esa manera, desde la perspectiva del sistema, no hay problema en nombrar la clase y el slot `Establecimiento Vinícola`.

6.1 Mayúsculas/minúsculas y delimitadores

Primero, podemos mejorar enormemente la legibilidad de una ontología si usamos de manera consistente los convenios sobre mayúsculas y minúsculas para los nombres de conceptos. Por ejemplo, es practica común el escribir en mayúsculas los nombres de las clases y usar minúsculas en los nombres de los slots (asumiendo que el sistema diferencia entre mayúsculas y minúsculas).

Cuando el nombre de un concepto contiene más de una palabra (tal como `Plato de comida`) necesitamos delimitar las palabras. Aquí hay algunas posibles opciones:

- Usar espacios: `Plato de comida` (muchos sistemas, incluyendo, Protégé, permiten espacios en los nombres de conceptos).
- Escribir todas las palabras juntas y poner en mayúsculas cada nueva palabra: `PlatoDeComida`.
- Usar un subrayado o guión u otro delimitador en el nombre: `Plato_De_Comida`, `Plato_de_comida`, `Plato-De-Comida`, `Plato-de-comida`. (Si usas delimitadores, necesitarás también decidir si cada nueva palabra debe comenzar con una letra en mayúsculas).

Si el sistema de representación permite espacios en los nombres, su uso sería la solución más intuitiva para muchos desarrolladores de ontologías. Es, sin embargo, importante considerar otros sistemas con los cuales tu sistema podrá interactuar. Si esos sistemas no usan espacios o si tu medio de presentación no maneja bien los espacios, sería útil usar otro método.

6.2 Singular o Plural

Un nombre de una clase representa una colección de objetos. Por ejemplo, la clase `Vino` representa a todos los vinos. Por lo tanto, sería más natural para algunos diseñadores nombrar la clase `Vinos` en lugar de `Vino`. Ninguna alternativa es mejor o peor que otra (aunque la forma singular para las clases es usada más a menudo en la prática). Sin embargo, cualquiera sea la elección, debe ser consistente a lo largo de toda la ontología. Algunos sistemas solicitan a sus usuarios declarar por adelantado si ellos usarán singular o plural para los nombres de los conceptos y no permiten desviarse de esa elección.

El uso de la misma forma todo el tiempo también previene al diseñador de cometer errores de modelamiento como crear una clase `Vinos` y luego crear una clase `Vino` como su subclase (ver Sección 4.1).

6.3 Convenios: prefijos y sufijos

Algunas metodologías de bases de conocimiento sugieren usar convenios sobre el uso de prefijos y sufijos en los nombres para distinguir entre clases y slots. Dos prácticas comunes son: añadir `tiene-` como prefijo o el sufijo `-de` a los nombres de los slots. De esta forma, nuestro slots serán `tiene-productor` y `tiene-establecimiento vinícola` si elegimos el convenio que sugiere el uso de `tiene-`. Si elegimos usar el convenio que sugiere que se emplee `-de`, los slots serán `productor-de` y `establecimiento vinícola-de`. Este enfoque permite que los usuarios vean el término para poder determinar inmediatamente si el término es una clase o slot. Sin embargo, los nombres de los términos llegan a ser ligeramente largos.

6.4 Otras consideraciones de nombrado

Aquí están unas cuantas cosas más a considerar cuando se definan los convenios de nombrado:

- No agregar cadenas tales como “clase”, “propiedad”, “slot”, y as sucesivamente a los nombres de conceptos. Siempre está claro a partir del contexto si el concepto es una clase o un slot. Además, si usas diferentes convenios de nombrado para las clases y slots (por ejemplo, mayúsculas y minúsculas respectivamente), el nombre como tal sería indicativo de qué es el concepto.

- Usualmente es buena idea evitar abreviaciones en los nombres de conceptos (es decir, usar `Cabernet Sauvignon` en lugar de `Cab`).

- Los nombres de las subclases directas de una clase deberían todas incluir o no incluir el nombre de la superclase. Por ejemplo, si estamos creando dos subclases de la clase `Vino` para representar vinos rojos y vinos blancos, los dos nombres de las subclases deberían ser `Vino Rojo` y `Vino Blanco`, y no así `Vino Rojo` y `Blanco`.

7 Otros recursos

Hemos usado Protégé-2000 como entorno de desarrollo de ontologías para nuestros ejemplos. Duineveld y sus colegas (Duineveld et al. 2000) describen y comparan otros entornos de desarrollo de ontologías.

Hemos tratado de introducir los fundamentos del desarrollo de ontologías y no hemos discutido muchos de los tópicos avanzados o metodologías alternativas para el desarrollo de ontologías. Gómez-Pérez (Gómez-Pérez 1998) y Uschold (Uschold y Gruninger 1996) presentan metodologías alternativas para el desarrollo de ontologías. El tutorial Ontolingua (Farquhar 1997) discute algunos aspectos formales del modelamiento de conocimiento.

Actualmente, los investigadores enfatizan no solamente el desarrollo de ontologías, sino también el análisis de ontologías. Por ejemplo, Chimaera (McGuinness et al. 2000) provee herramientas de diagnóstico para analizar ontologías. El análisis que Chimaera desempeña incluye una verificación de la exactitud lógica de una ontología y un diagnóstico de errores comunes de diseño de ontologías. Un diseñador de ontologías podría ejecutar diagnósticos con Chimaera sobre la ontología que evoluciona para determinar la conformidad con practicas comunes de modelamiento de ontologías.

8 Conclusiones

En esta guía, hemos descrito una metodología de desarrollo de ontologías para sistemas declarativos basados en marcos. Listamos los pasos del proceso de desarrollo de ontologías y hicimos referencia a los aspectos complejos de la definición de jerarquías de clases y propiedades de clases e instancias. Sin embargo, después de seguir todas las reglas y sugerencias, una de las cosas más importantes que recordar es la siguiente: *no hay sola ontología correcta para un dominio dado.* El diseño de ontologías es un proceso creativo y dos ontologías diseñadas por personas diferentes no serán iguales. Las aplicaciones potenciales de la ontología y el entendimiento y aspecto del dominio desde el punto de vista del diseñador afectarán sin duda las opciones de diseño de la ontología. “No se sabe si algo es bueno hasta que se lo pone a prueba”: podemos evaluar la calidad de nuestra ontología solamente usándola en aplicaciones para las cuales la diseñamos.

9 Agradecimientos

Protégé-2000 (`http://protege.stanford.edu`) fue desarrollado por el grupo de Mark Munsen en el centro de Informática Medical de Stanford (Stanford Medical Informatics). Generamos algunas de las figuras con el plug-in OntoViz de Protégé-2000. Importamos la version inicial de la ontología de vinos a partir de la biblioteca de ontologías Ontolingua (`http://www.ksl.stanford.edu/software/ontolingua/`) la cual a su vez usó una versión publicada por Brachman y sus colegas (Brachman et al. 1991) y distribuida con el sistema de representación de conocimiento CLASSIC. Luego modificamos la ontología para presentar los principios del modelamiento conceptual para ontologías declarativas basadas en marcos. Los comentarios de Ray Fergerson y Mor Peleg sobre los borradores iniciales mejoraron enormemente este artículo.

References

[1] Booch, G., Rumbaugh, J. and Jacobson, I. (1997). *The Unified Modeling Language user guide*: Addison-Wesley.

[2] Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A. and Borgida, A. (1991). Living with CLASSIC: When and how to use KL-ONE-like language. *Principles of Semantic Networks*. J. F. Sowa, editor, Morgan Kaufmann: 401-456.

[3] Brickley, D. and Guha, R.V. (1999). Resource Description Framework (RDF) Schema Specification. Proposed Recommendation, World Wide Web Consortium: `http://www.w3.org/TR/PR-rdf-schema`.

[4] Chimaera (2000). Chimaera Ontology Environment. `www.ksl.stanford.edu/software/chimaera`

[5] Duineveld, A.J., Stoter, R., Weiden, M.R., Kenepa, B. and Benjamins, V.R. (2000). WonderTools? A comparative study of ontological engineering tools. *International Journal of Human-Computer Studies* **52**(6): 1111-1133.

[6] Farquhar, A. (1997). Ontolingua tutorial. `http://ksl-web.stanford.edu/people/axf/tutorial.pdf`

[7] Gómez-Pérez, A. (1998). Knowledge sharing and reuse. *Handbook of Applied Expert Systems*. Liebowitz, editor, CRC Press.

[8] Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition* **5**: 199-220.

[9] Gruninger, M. and Fox, M.S. (1995). Methodology for the Design and Evaluation of Ontologies. In: *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95*, Montreal.

[10] Hendler, J. and McGuinness, D.L. (2000). The DARPA Agent Markup Language. *IEEE Intelligent Systems* **16**(6): 67-73.

[11] Humphreys, B.L. and Lindberg, D.A.B. (1993). The UMLS project: making the conceptual connection between users and the information they need. *Bulletin of the Medical Library Association* **81**(2): 170.

[12] McGuinness, D.L., Abrahams, M.K., Resnick, L.A., Patel-Schneider, P.F., Thomason, R.H., Cavalli-Sforza, V. and Conati, C. (1994). Classic Knowledge Representation System Tutorial. `http://www.bell-labs.com/project/classic/papers/ClassTut/ClassTut.html`

[13] McGuinness, D.L., Fikes, R., Rice, J. and Wilder, S. (2000). An Environment for Merging and Testing Large Ontologies. *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000)*. A. G. Cohn, F. Giunchiglia and B. Selman, editors. San Francisco, CA, Morgan Kaufmann Publishers.

[14] McGuinness, D.L. and Wright, J. (1998). Conceptual Modeling for Configuration: A Description Logic-based Approach. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing - special issue on Configuration*.

[15] Musen, M.A. (1992). Dimensions of knowledge sharing and reuse. *Computers and Biomedical Research* **25**: 435-467.

[16] Ontolingua (1997). Ontolingua System Reference Manual. `http://www-ksl-svc.stanford.edu:5915/doc/frame-editor/index.html`

[17] Price, C. and Spackman, K. (2000). SNOMED clinical terms. *BJHC&IM-British Journal of Healthcare Computing & Information Management* **17**(3): 27-31.

[18] Protégé (2000). The Protégé Project. http://protege.stanford.edu

[19] Rosch, E. (1978). Principles of Categorization. *Cognition and Categorization*. R. E. and B. B. Lloyd, editors. Hillside, NJ, Lawrence Erlbaum Publishers: 27-48.

[20] Rothenfluh, T.R., Gennari, J.H., Eriksson, H., Puerta, A.R., Tu, S.W. and Musen, M.A. (1996). Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTÉGÉ-II solutions to Sisyphus-2. *International Journal of Human-Computer Studies* **44**: 303-332.

[21] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). *Object-oriented modeling and design*. Englewood Cliffs, New Jersey: Prentice Hall.

[22] Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review* **11**(2).