# Desenvolvimento de Ontologia 101: Um Guia para Criar Sua Primeira Ontologia

Natalya F. Noy e Deborah L. McGuinness Universidade de Stanford, Stanford, CA, 94305noy@smi.stanford.edu e dlm@ksl.stanford.edu

## 1 Por que desenvolver uma ontologia?

Nos últimos anos, o desenvolvimento de ontologias — especificações formais explícitas dos termos do domínio e das relações entre eles (Gruber 1993) — tem saído do âmbito dos laboratórios de Inteligência Artificial para chegar aos computadores dos especialistas na área. As ontologias tornaram-se comuns na World Wide Web. As ontologias na Web variam de grandes taxonomias que categorizam sites (como no Yahoo!) a categorizações de produtos à venda e suas características (como na Amazon.com). O WWW Consortium (W3C) está desenvolvendo o Resource Description Framework (Brickley e Guha 1999), uma linguagem para codificar conhecimento em páginas da Web para torná-lo compreensível para agentes eletrônicos que buscam informações. A Agência de Projetos de Pesquisa Avançada de Defesa (DARPA), em conjunto com o W3C, está desenvolvendo a Linguagem de Marcação de Agentes da DARPA (DAML), estendendo a RDF com construções mais expressivas destinadas a facilitar a interação de agentes na Web (Hendler e McGuinness 2000). Muitas disciplinas agora desenvolvem ontologias padronizadas que especialistas na área podem usar para compartilhar e anotar informações em seus campos. A medicina, por exemplo, produziu vocabulários grandes, padronizados e estruturados, como o SNOMED (Price e Spackman 2000) e a rede semântica do Unified Medical Language System (Humphreys e Lindberg 1993). Ontologias amplas de uso geral também estão surgindo. Por exemplo, o Programa das Nações Unidas para o Desenvolvimento e a Dun & Bradstreet combinaram seus esforços para desenvolver a ontologia UNSPSC, que fornece terminologia para produtos e serviços (www.unspsc.org).

Uma ontologia define um vocabulário comum para pesquisadores que precisam compartilhar informações em um determinado domínio. Ela inclui definições interpretáveis por máquinas de conceitos básicos do domínio e as relações entre eles.

Por que alguém desejaria desenvolver uma ontologia? Algumas das razões são:

- Para compartilhar um entendimento comum da estrutura das informações entre pessoas ou agentes de software
- Para permitir a reutilização do conhecimento do domínio
- Para tornar explícitas as suposições do domínio
- Para separar o conhecimento do domínio do conhecimento operacional
- Para analisar o conhecimento do domínio

*Compartilhar um entendimento comum sobre a estrutura da informação entre pessoas ou agentes de software* é um dos objetivos mais comuns no desenvolvimento de ontologias (Musen 1992; Gruber 1993). Por exemplo, suponha que vários sites diferentes contenham informações médicas ou forneçam serviços de comércio eletrônico médico. Se esses sites compartilharem e publicarem a mesma ontologia subjacente dos termos que todos eles usam, os agentes de computador poderão extrair e agregar informações desses diferentes sites. Os agentes podem usar essas informações agregadas para responder às consultas dos usuários ou como dados de entrada para outros aplicativos.

*Permitir a reutilização do conhecimento de domínio* foi uma das forças motrizes por trás do recente aumento na pesquisa ontológica. Por exemplo, modelos para muitos domínios diferentes precisam representar a noção de tempo. Essa representação inclui as noções de intervalos de tempo, pontos no tempo, medidas relativas de tempo e assim por diante. Se um grupo de pesquisadores desenvolver essa ontologia em detalhes, outros podem simplesmente reutilizá-la para seus domínios. Além disso, se precisarmos construir um grande

ontologia, podemos integrar várias ontologias existentes que descrevem partes do grande domínio. Também podemos reutilizar uma ontologia geral, como a ontologia UNSPSC, e ampliá-la para descrever nosso domínio de interesse.

*Tornar explícitas as suposições de domínio* subjacentes a uma implementação torna possível alterar essas suposições facilmente se nosso conhecimento sobre o domínio mudar. Codificar suposições sobre o mundo em código de linguagem de programação torna essas suposições não apenas difíceis de encontrar e entender, mas também difíceis de alterar, especialmente para alguém sem experiência em programação. Além disso, especificações explícitas de conhecimento de domínio são úteis para novos usuários que precisam aprender o significado dos termos do domínio.

*Separar o conhecimento do domínio do conhecimento operacional* é outro uso comum das ontologias. Podemos descrever uma tarefa de configuração de um produto a partir de seus componentes, de acordo com uma especificação exigida, e implementar um programa que faça essa configuração independentemente dos produtos e componentes em si (McGuinness e Wright, 1998). Podemos então desenvolver uma ontologia de componentes e características de PCs e aplicar o algoritmo para configurar PCs sob encomenda. Também podemos usar o mesmo algoritmo para configurar elevadores se "alimentarmos" uma ontologia de componentes de elevadores ( Rothenfluh et al. 1996).

*A análise do conhecimento do domínio* é possível assim que uma especificação declarativa dos termos estiver disponível. A análise formal dos termos é extremamente valiosa tanto quando se tenta reutilizar ontologias existentes como quando se pretende ampliá-las (McGuinness et al. 2000).

Muitas vezes, uma ontologia do domínio não é um objetivo em si. Desenvolver uma ontologia é semelhante a definir um conjunto de dados e sua estrutura para uso por outros programas. Métodos de resolução de problemas, aplicativos independentes de domínio e agentes de software usam ontologias e bases de conhecimento construídas a partir de ontologias como dados. Por exemplo, neste artigo, desenvolvemos uma ontologia de vinhos e alimentos e combinações adequadas de vinhos com refeições. Essa ontologia pode então ser usada como base para algumas aplicações em um conjunto de ferramentas de gerenciamento de restaurantes: uma aplicação poderia criar sugestões de vinhos para o menu do dia ou responder a perguntas de garçons e clientes. Outra aplicação poderia analisar uma lista de inventário de uma adega e sugerir quais categorias de vinhos expandir e quais vinhos específicos comprar para os próximos menus ou livros de receitas.

## Sobre este guia

Nos baseamos em nossa experiência com o uso do Protégé-2000 (Protege 2000), Ontolingua (Ontolingua 1997) e Chimaera (Chimaera 2000) como ambientes de edição de ontologia. Neste guia, usamos o Protégé-2000 para nossos exemplos.

O exemplo de vinhos e alimentos que utilizamos ao longo deste guia é vagamente baseado em uma base de conhecimento exemplar apresentada em um artigo que descreve o CLASSIC — um sistema de representação de conhecimento baseado em uma abordagem de lógica descritiva (Brachman et al. 1991). O tutorial CLASSIC (McGuinness et al. 1994) desenvolveu este exemplo ainda mais. O Protégé-2000 e outros sistemas baseados em estruturas descrevem ontologias de forma declarativa, indicando explicitamente qual é a hierarquia de classes e a quais classes os indivíduos pertencem.

Algumas ideias de design ontológico neste guia têm origem na literatura sobre design orientado a objetos (Rumbaugh et al. 1991; Booch et al. 1997). No entanto, o desenvolvimento de ontologias é diferente do design de classes e relações na programação orientada a objetos. A programação orientada a objetos se concentra principalmente em métodos em classes — um programador toma decisões de design com base nas propriedades *operacionais* de uma classe, enquanto um designer de ontologias toma essas decisões com base nas propriedades *estruturais* de uma classe. Como resultado, a estrutura de classes e as relações entre classes em uma ontologia são diferentes da estrutura para um domínio semelhante em um programa orientado a objetos.

É impossível abordar todas as questões com as quais um desenvolvedor de ontologias pode precisar lidar, e não estamos tentando abordar todas elas neste guia. Em vez disso, tentamos fornecer um ponto de partida, um guia inicial que ajude um novo designer de ontologias a desenvolver ontologias. No

final, sugerimos locais onde se pode encontrar explicações sobre estruturas e mecanismos de design mais complexos, caso o domínio assim o exija.

Por fim, não existe uma única metodologia correta de design de ontologias e não tentamos definir uma. As ideias que apresentamos aqui são aquelas que consideramos úteis em nossa própria experiência de desenvolvimento de ontologias. No final deste guia, sugerimos uma lista de referências para metodologias alternativas.

## 2 O que há em uma ontologia?

A literatura sobre Inteligência Artificial contém muitas definições de ontologia, muitas das quais se contradizem entre si. Para os fins deste guia, uma **ontologia** é uma descrição formal explícita de conceitos em um domínio de discurso (**classes**, às vezes chamadas **de conceitos**), propriedades de cada conceito que descrevem várias características e atributos do conceito (**slots**, às vezes chamados **de funções** ou **propriedades**) e restrições aos slots (**facetas**, às vezes chamadas **de restrições de função**). Uma ontologia, juntamente com um conjunto de **instâncias** individuais de classes, constitui uma **base de conhecimento.** Na realidade, há uma linha tênue entre onde termina a ontologia e começa a base de conhecimento.

As classes são o foco da maioria das ontologias. As classes descrevem conceitos no domínio. Por exemplo, uma classe de vinhos representa todos os vinhos. Vinhos específicos são instâncias dessa classe. O vinho Bordeaux na taça à sua frente enquanto você lê este documento é uma instância da classe dos vinhos Bordeaux. Uma classe pode ter **subclasses** que representam conceitos mais específicos do que a superclasse. Por exemplo, podemos dividir a classe de todos os vinhos em vinhos tintos, brancos e rosés. Alternativamente, podemos dividir uma classe de todos os vinhos em vinhos espumantes e não espumantes.

Os slots descrevem propriedades de classes e instâncias: o vinho `Château Lafite Rothschild Pauillac` tem um corpo encorpado; é produzido pela vinícola `Château Lafite Rothschild`. [1]Temos dois slots que descrevem o vinho neste exemplo: o slot corpo com o valor encorpado e o slot fabricante com o valor vinícola `Château Lafite Rothschild`. No nível da classe, podemos dizer que as instâncias da classe `Vinho` terão slots que descrevem seu `sabor`, `corpo`, `teor de açúcar`, `fabricante` do vinho e assim por diante.

Todas as instâncias da classe `Wine` e sua subclasse Pauillac têm um slot `maker` cujo valor é uma instância da classe Winery (Figura 1). Todas as instâncias da classe Winery têm um slot `produces` que se refere a todos os vinhos (instâncias da classe Wine e suas subclasses) que a vinícola produz.

Em termos práticos, o desenvolvimento de uma ontologia inclui:

- definir classes na ontologia,
- organizar as classes em uma hierarquia taxonômica (subclasse-superclasse),
- definir slots e descrever os valores permitidos para esses slots,
- preencher os valores dos slots para as instâncias.

Podemos então criar uma base de conhecimento definindo instâncias individuais dessas classes, preenchendo informações específicas sobre valores de slots e restrições adicionais de slots.

---

[1] Usamos letras maiúsculas para nomes de classes e letras minúsculas para nomes de slots. Também usamos `fonte de máquina de escrever` para todos os termos da ontologia de exemplo.

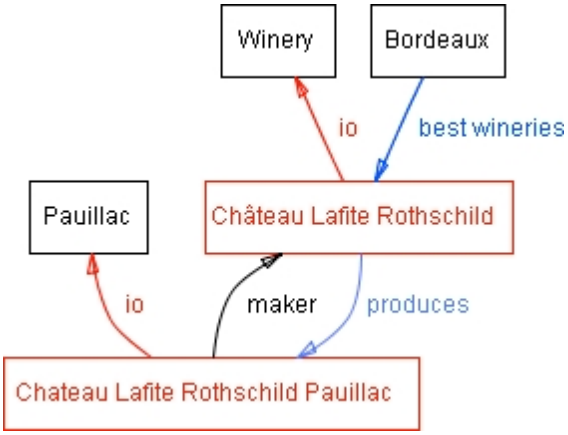


**Figura 1. Algumas classes, instâncias e relações entre elas no domínio do vinho. Usamos preto para classes e vermelho para instâncias. Links diretos representam slots e links internos, como instance-of e subclass-of.**

## 3 Uma metodologia simples de engenharia do conhecimento

Como dissemos anteriormente, não existe uma única maneira ou metodologia "correta" para desenvolver ontologias. Aqui, discutimos questões gerais a serem consideradas e oferecemos um possível processo para desenvolver uma ontologia. Descrevemos uma abordagem iterativa para o desenvolvimento de ontologias: começamos com uma primeira versão aproximada da ontologia. Em seguida, revisamos e refinamos a ontologia em evolução e preenchemos os detalhes. Ao longo do processo, discutimos as decisões de modelagem que um designer precisa tomar, bem como os prós, contras e implicações de diferentes soluções.

Primeiro, gostaríamos de enfatizar algumas regras fundamentais no design de ontologias, às quais nos referiremos muitas vezes. Essas regras podem parecer um tanto dogmáticas. No entanto, elas podem ajudar a tomar decisões de design em muitos casos.

1) *Não existe uma maneira correta de modelar um domínio — sempre há alternativas viáveis. A melhor solução quase sempre depende da aplicação que você tem em mente e das extensões que você antecipa.*
2) *O desenvolvimento da ontologia é necessariamente um processo iterativo.*
3) *Os conceitos na ontologia devem estar próximos dos objetos (físicos ou lógicos) e das relações no seu domínio de interesse. É mais provável que sejam substantivos (objetos) ou verbos (relações) em frases que descrevem o seu domínio.*

Ou seja, decidir para que vamos usar a ontologia e quão detalhada ou geral ela será.

A ontologia irá orientar muitas das decisões de modelagem ao longo do caminho. Entre várias alternativas viáveis, precisaremos determinar qual delas funcionaria melhor para a tarefa projetada, seria mais intuitiva, mais extensível e mais fácil de manter. Também precisamos lembrar que uma ontologia é um modelo da realidade do mundo e que os conceitos na ontologia devem refletir essa realidade. Depois de definirmos uma versão inicial da ontologia, podemos avaliá-la e depurá-la usando-a em aplicativos ou métodos de resolução de problemas, ou discutindo-a com especialistas na área, ou ambos. Como resultado, quase certamente precisaremos revisar a ontologia inicial. Esse processo de design iterativo provavelmente continuará durante todo o ciclo de vida da ontologia.

## Passo 1. Determine o domínio e o escopo da ontologia

Sugerimos começar o desenvolvimento de uma ontologia definindo seu domínio e escopo. Ou seja, responda a várias perguntas básicas:

- Qual é o domínio que a ontologia irá abranger?
- Para que vamos usar a ontologia?
- Para que tipos de perguntas as informações na ontologia devem fornecer respostas?
- Quem utilizará e manterá a ontologia?

As respostas a essas perguntas podem mudar durante o processo de design da ontologia, mas, a qualquer momento, elas ajudam a limitar o escopo do modelo.

Considere a ontologia do vinho e da comida que apresentamos anteriormente. A representação de alimentos e vinhos é o domínio da ontologia. Pretendemos usar essa ontologia para aplicações que sugerem boas combinações de vinhos e alimentos.

Naturalmente, os conceitos que descrevem os diferentes tipos de vinhos, os principais tipos de alimentos, a noção de uma boa combinação de vinho e comida e uma má combinação farão parte da nossa ontologia. Ao mesmo tempo, é improvável que a ontologia inclua conceitos para a gestão do inventário numa adega ou dos funcionários num restaurante, embora esses conceitos estejam de alguma forma relacionados com as noções de vinho e comida.

Se a ontologia que estamos projetando for usada para auxiliar no processamento de linguagem natural de artigos em revistas sobre vinhos, pode ser importante incluir sinônimos e informações sobre classes gramaticais para os conceitos na ontologia. Se a ontologia for usada para ajudar os clientes de restaurantes a decidir qual vinho pedir, precisamos incluir informações sobre preços de varejo. Se for usada por compradores de vinho para abastecer uma adega, os preços de atacado e a disponibilidade podem ser necessários. Se as pessoas que manterão a ontologia descreverem o domínio em uma linguagem diferente da linguagem dos usuários da ontologia, talvez seja necessário fornecer o mapeamento entre as linguagens.

### Questões de competência.

Uma das maneiras de determinar o escopo da ontologia é esboçar uma lista de perguntas que uma base de conhecimento baseada na ontologia deve ser capaz de responder, **perguntas de competência** (Gruninger e Fox 1995). Essas perguntas servirão como teste decisivo mais tarde: a ontologia contém informações suficientes para responder a esses tipos de perguntas? As respostas exigem um nível específico de detalhe ou representação de uma área específica? Essas perguntas de competência são apenas um esboço e não precisam ser exaustivas.

No domínio do vinho e da gastronomia, as seguintes são as possíveis questões de competência:

- Quais características do vinho devo considerar ao escolher um vinho?
- O Bordeaux é um vinho tinto ou branco?
- O Cabernet Sauvignon combina bem com frutos do mar?
- Qual é a melhor escolha de vinho para acompanhar carne grelhada?
- Quais características de um vinho afetam sua adequação a um prato?
- O bouquet ou o corpo de um vinho específico mudam com a safra?
- Quais foram as boas safras para o Napa Zinfandel?

A julgar por esta lista de perguntas, a ontologia incluirá informações sobre várias características e tipos de vinho, safras — boas e ruins —, classificações de alimentos que são importantes para a escolha de um vinho adequado e combinações recomendadas de vinho e comida.

## Etapa 2. Considere reutilizar ontologias existentes

Quase sempre vale a pena considerar o que outra pessoa fez e verificar se podemos refinar e ampliar as fontes existentes para nosso domínio e tarefa específicos. Reutilizando o existente

As ontologias podem ser um requisito se o nosso sistema precisar interagir com outras aplicações que já se comprometeram com ontologias específicas ou vocabulários controlados. Muitas ontologias já estão disponíveis em formato eletrônico e podem ser importadas para um ambiente de desenvolvimento de ontologias que você esteja usando. O formalismo em que uma ontologia é expressa muitas vezes não importa, uma vez que muitos sistemas de representação de conhecimento podem importar e exportar ontologias. Mesmo que um sistema de representação de conhecimento não possa trabalhar diretamente com um formalismo específico, a tarefa de traduzir uma ontologia de um formalismo para outro geralmente não é difícil.

Existem bibliotecas de ontologias reutilizáveis na Web e na literatura. Por exemplo, podemos usar a biblioteca de ontologias Ontolingua (http://www.ksl.stanford.edu/software/ontolingua/) ou a biblioteca de ontologias DAML (http://www.daml.org/ontologies/). Existem também várias ontologias comerciais disponíveis ao público (por exemplo, UNSPSC (www.unspsc.org), RosettaNet (www.rosettanet.org), DMOZ (www.dmoz.org)).

Por exemplo, pode já existir uma base de conhecimento sobre vinhos franceses. Se pudermos importar essa base de conhecimento e a ontologia na qual ela se baseia, teremos não apenas a classificação dos vinhos franceses, mas também uma primeira abordagem da classificação das características dos vinhos usadas para distinguir e descrever os vinhos. Listas de propriedades dos vinhos podem já estar disponíveis em sites comerciais, como www.wines.com, que os clientes consideram usar para comprar vinhos.

Para este guia, no entanto, vamos assumir que ainda não existem ontologias relevantes e começar a desenvolver a ontologia do zero.

## Etapa 3. Enumerar termos importantes na ontologia

É útil escrever uma lista de todos os termos sobre os quais gostaríamos de fazer afirmações ou explicar a um usuário. Quais são os termos sobre os quais gostaríamos de falar? Quais são as propriedades desses termos? O que gostaríamos de dizer sobre esses termos? Por exemplo, termos importantes relacionados ao vinho incluem `vinho`, `uva`, `vinícola`, `localização`, `cor`, `corpo`, `sabor` e `teor de açúcar` do vinho; diferentes tipos de `alimentos`, como `peixe` e `carne vermelha`; subtipos de vinho, como `vinho branco`, e assim por diante. Inicialmente, é importante obter uma lista abrangente de termos sem se preocupar com a sobreposição entre os conceitos que eles representam, as relações entre os termos ou quaisquer propriedades que os conceitos possam ter, ou se os conceitos são classes ou slots.

Os dois passos seguintes — desenvolver a hierarquia de classes e definir as propriedades dos conceitos (slots) — estão intimamente interligados. É difícil fazer um deles primeiro e depois o outro. Normalmente, criamos algumas definições dos conceitos na hierarquia e depois continuamos descrevendo as propriedades desses conceitos e assim por diante. Esses dois passos também são os mais importantes no processo de design da ontologia. Vamos descrevê-las aqui brevemente e, em `seguida`, dedicar as duas seções seguintes à discussão das questões mais complexas que precisam ser consideradas, armadilhas comuns, decisões a serem tomadas e assim por diante.

## Etapa 4. Definir as classes e a hierarquia de classes

Existem várias abordagens possíveis para desenvolver uma hierarquia de classes (Uschold e Gruninger 1996):

- Um processo de desenvolvimento **descendente** começa com a definição dos conceitos mais gerais do domínio e a subsequente especialização dos conceitos. Por exemplo, podemos começar criando classes para os conceitos gerais de `Vinho` e `Comida`. Em `seguida`, especializamos a classe `Vinho` criando algumas de suas subclasses: `Vinho branco`, `Vinho tinto`, `Vinho rosé`. Podemos categorizar ainda mais a classe `Vinho tinto`, por exemplo, em `Syrah`, `Borgonha tinto`, `Cabernet Sauvignon` e assim por diante.

- Um processo de desenvolvimento **ascendente** começa com a definição das classes mais específicas, as folhas da hierarquia, com o agrupamento subsequente dessas classes em conceitos mais gerais. Por exemplo, começamos definindo classes para os vinhos `Pauillac` e `Margaux`. Em seguida, criamos uma superclasse comum para essas duas classes — Medoc — que, por sua vez, é uma subclasse de `Bordeaux`.
- Um processo de desenvolvimento **combinado** é uma combinação das abordagens descendente e ascendente: definimos primeiro os conceitos mais salientes e, em seguida, generalizamos e especializamos adequadamente. Podemos começar com alguns conceitos de nível superior, como `Vinho`, e alguns conceitos específicos, como `Margaux`. Em seguida, podemos relacioná-los a um conceito de nível médio, como `Medoc`. Depois, podemos querer gerar todas as classes regionais de vinhos da França, gerando assim vários conceitos de nível médio.

A Figura 2 mostra uma possível divisão entre os diferentes níveis de generalidade.

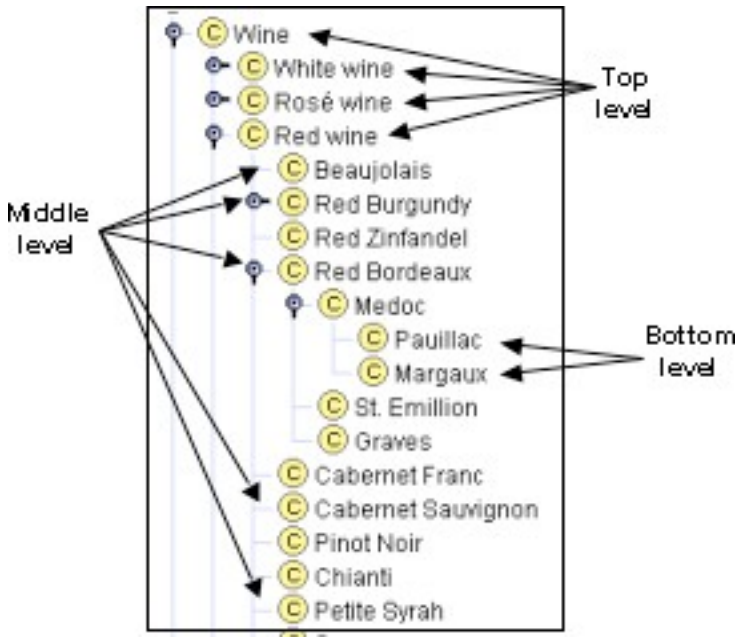


**Figura 2. Os diferentes níveis da taxonomia do `vinho`: `Vinho` é o conceito mais geral. `Vinho tinto`, `vinho branco` e `vinho rosé` são conceitos gerais de nível superior. `Pauillac` e `Margaux` são as classes mais específicas na hierarquia (ou os conceitos de nível inferior).**

Nenhum desses três métodos é inerentemente melhor do que os outros. A abordagem a ser adotada depende fortemente da visão pessoal do domínio. Se um desenvolvedor tem uma visão sistemática de cima para baixo do domínio, então pode ser mais fácil usar a abordagem de cima para baixo. A abordagem combinada é frequentemente a mais fácil para muitos desenvolvedores de ontologias, uma vez que os conceitos "no meio" tendem a ser os conceitos mais descritivos no domínio (Rosch 1978).

Se você tende a pensar em vinhos distinguindo primeiro a classificação mais geral, então a abordagem de cima para baixo pode funcionar melhor para você. Se você preferir começar com exemplos específicos, a abordagem de baixo para cima pode ser mais apropriada.

Seja qual for a abordagem escolhida, geralmente começamos definindo classes. A partir da lista criada na Etapa 3, selecionamos os termos que descrevem objetos com existência independente, em vez de termos que descrevem esses objetos. Esses termos serão classes na ontologia e se tornarão

âncoras na hierarquia de classes. [2]Organizamos as classes em uma taxonomia hierárquica perguntando se, por ser uma instância de uma classe, o objeto será necessariamente (ou seja, por definição) uma instância de alguma outra classe.

> *Se uma classe A é uma superclasse da classe B, então todas as instâncias de B também são instâncias de A.*

Em outras palavras, a classe B representa um conceito que é um "tipo" de A.

Por exemplo, todo vinho Pinot Noir é necessariamente um vinho tinto. Portanto, a classe `Pinot Noir` é uma subclasse da classe `Vinho Tinto`.

A Figura 2 mostra uma parte da hierarquia de classes para a ontologia do Vinho. A Seção 4 contém uma discussão detalhada sobre os aspectos a serem considerados ao definir uma hierarquia de classes.

Template Slots

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

**Figura 3. Os slots para a classe `Vinho` e as facetas para esses slots. O ícone "I" ao lado do `fabricante` slot indica que o slot tem um inverso (Seção 5.1)**

## Etapa 5. Defina as propriedades das classes — slots

As classes por si só não fornecem informações suficientes para responder às perguntas de competência da Etapa 1. Depois de definir algumas das classes, devemos descrever a estrutura interna dos conceitos.

Já selecionamos classes da lista de termos que criamos na Etapa 3. A maioria dos termos restantes provavelmente são propriedades dessas classes. Esses termos incluem, por exemplo, a `cor, o corpo, o sabor` e `o teor de açúcar` de um vinho e `a localização` de uma vinícola.

Para cada propriedade da lista, devemos determinar qual classe ela descreve. Essas propriedades tornam-se slots associados às classes. Assim, a classe `Vinho` terá os seguintes slots: `cor`, corpo, `sabor` e açúcar. E a classe `Vinícola` terá um slot `de localização`.

Em geral, existem vários tipos de propriedades de objetos que podem se tornar slots em uma ontologia:

- propriedades "intrínsecas", como o `sabor` de um vinho;
- propriedades "extrínsecas", como `o nome` de um vinho e `a região` de onde ele vem;
- partes, se o objeto for estruturado; estas podem ser tanto "partes" físicas como abstratas (por exemplo, os pratos de uma refeição)
- relações com outros indivíduos; são as relações entre membros individuais da classe e outros itens (por exemplo, o `produtor` de um vinho, representando uma relação entre um vinho e uma vinícola, e a `uva` da qual o vinho é feito).

Assim, além das propriedades que identificamos anteriormente, precisamos adicionar o seguinte

---

[2] Também podemos ver as classes como predicados unários — perguntas que têm um argumento. Por exemplo, "Este objeto é um vinho?" Os predicados unários (ou classes) contrastam com os predicados binários (ou slots) — perguntas que têm dois argumentos. Por exemplo, "O sabor deste objeto é forte?" "Qual é o sabor deste objeto?"

slots para a classe Wine: nome, região, produtor, uva. A Figura 3 mostra os slots para a classe Vinho.

Todas as subclasses de uma classe **herdam** o campo dessa classe. Por exemplo, todos os campos da classe Vinho serão herdados por todas as subclasses de Vinho, incluindo Vinho Tinto e Vinho Branco. Adicionaremos um campo adicional, nível de tanino (baixo, moderado ou alto), à classe Vinho Tinto. O campo nível de tanino será herdado por todas as classes que representam vinhos tintos (como Bordeaux e Beaujolais).

Um slot deve ser anexado à classe mais geral que pode ter essa propriedade. Por exemplo, o corpo e a cor de um vinho devem ser anexados à classe Vinho, pois é a classe mais geral cujas instâncias terão corpo e cor.

## Etapa 6. Defina as facetas dos slots

Os slots podem ter diferentes facetas que descrevem o tipo de valor, os valores permitidos, o número de valores (cardinalidade) e outras características dos valores que o slot pode assumir. Por exemplo, o valor de um slot de nome (como em "o nome de um vinho") é uma string. Ou seja, nome é um slot com tipo de valor String. Um slot produz (como em "uma vinícola produz esses vinhos") pode ter vários valores e os valores são instâncias da classe Vinho. Ou seja, produz é um slot com tipo de valor Instância com Vinho como classe permitida.

Descreveremos agora várias facetas comuns.

### Cardinalidade do slot

A cardinalidade do slot define quantos valores um slot pode ter. Alguns sistemas distinguem apenas entre cardinalidade única (permitindo no máximo um valor) e cardinalidade múltipla (permitindo qualquer número de valores). O corpo de um vinho será um slot de cardinalidade única (um vinho pode ter apenas um corpo). Os vinhos produzidos por uma determinada vinícola preenchem um slot de cardinalidade múltipla produzido para uma classe Vinícola.

Alguns sistemas permitem especificar uma cardinalidade mínima e máxima para descrever o número de valores de slot com mais precisão. A cardinalidade mínima de N significa que um slot deve ter pelo menos N valores. Por exemplo, o slot de uva de um vinho tem uma cardinalidade mínima de 1: cada vinho é feito de pelo menos uma variedade de uva. A cardinalidade máxima de M significa que um slot pode ter no máximo M valores. A cardinalidade máxima para o slot de uva para vinhos de uma única variedade é 1: esses vinhos são produzidos a partir de apenas uma variedade de uva. Às vezes, pode ser útil definir a cardinalidade máxima como 0 . Essa configuração indicaria que o slot não pode ter nenhum valor para uma subclasse específica.

### Tipo de valor de slot

Uma faceta do tipo valor descreve quais tipos de valores podem preencher o slot. Aqui está uma lista dos tipos de valores mais comuns:

- **String** é o tipo de valor mais simples, usado para slots como nome: o valor é uma string simples
- **Número** (às vezes, tipos de valor mais específicos, como Float e Integer, são usados) descreve slots com valores numéricos. Por exemplo, o preço de um vinho pode ter um tipo de valor Float
- Slots **booleanos** são simples sinalizadores sim-não. Por exemplo, se optarmos por não representar vinhos espumantes como uma classe separada, o fato de um vinho ser espumante ou não pode ser representado como um valor de um slot booleano: se o valor for "verdadeiro" ("sim"), o vinho é espumante e, se o valor for "falso" ("não"), o vinho não é espumante.
- Os slots **enumerados** especificam uma lista de valores específicos permitidos para o slot. Por exemplo, podemos especificar que o slot flavor pode assumir um dos três valores possíveis:

`forte`, `moderado` e `delicado`. No Protégé-2000, os slots enumerados são do tipo `Símbolo`.

- Os slots do tipo instância permitem a definição de relações entre indivíduos. Os slots com tipo de valor Instância também devem definir uma lista de classes permitidas das quais as instâncias podem vir. Por exemplo, um slot `produz` para a classe `Vinícola` pode ter instâncias da classe `Vinho` como seus valores. [3]

A Figura 4 mostra uma definição do slot `produz` na classe `Vinícola`.

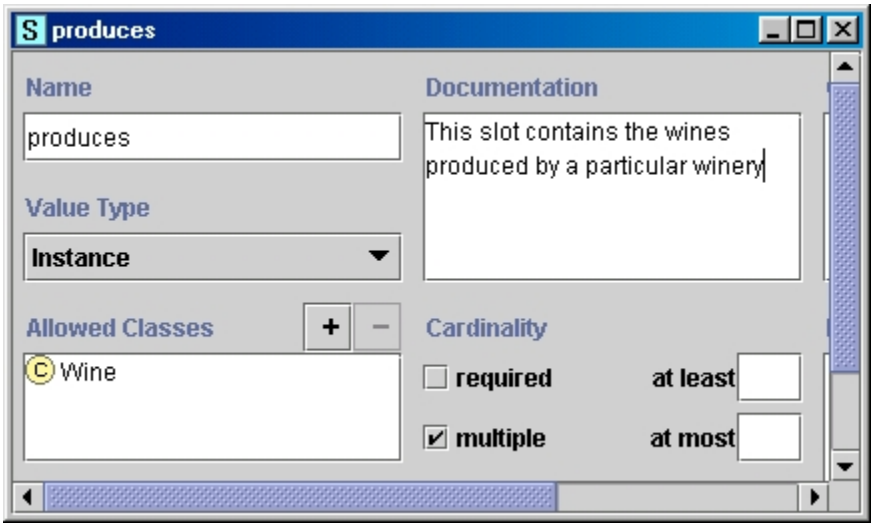


**Figura 4. A definição de um slot que descreve os vinhos produzidos por uma vinícola. O slot tem cardinalidade múltipla, tipo de valor Instância e a classe `Vinho` como a classe permitida para seus valores.**

### Domínio e intervalo de um slot

As classes permitidas para slots do tipo Instância são frequentemente chamadas de **intervalo** de um slot. No exemplo da Figura 4, a classe `Vinho` é o intervalo do slot `produzidos`. Alguns sistemas permitem restringir o intervalo de um slot quando o slot está vinculado a uma classe específica.

As classes às quais um slot está vinculado ou as classes cuja propriedade um slot descreve são chamadas de **domínio** do slot. A classe `Winery` é o domínio do slot `produces`. Nos sistemas em que *vinculamos* slots a classes, as classes às quais o slot está vinculado geralmente constituem o domínio desse slot. Não há necessidade de especificar o domínio separadamente.

As regras básicas para determinar um domínio e um intervalo de um slot são semelhantes:

> *Ao definir um domínio ou um intervalo para um slot, encontre as classes mais gerais ou a classe que pode ser, respectivamente, o domínio ou o intervalo para os slots.*
>
> *Por outro lado, não defina um domínio e um intervalo excessivamente gerais: todas as classes no domínio de um slot devem ser descritas pelo slot e as instâncias de todas as classes no intervalo de um slot devem ser preenchimentos potenciais para o slot. Não escolha uma classe excessivamente geral para o intervalo (ou seja, não se deve definir o intervalo como COISA), mas sim uma classe que abranja todos os preenchimentos.*

Em vez de listar todas as subclasses possíveis da classe `Wine` para o intervalo dos `produtos`

[3] Alguns sistemas apenas especificam o tipo de valor com uma classe, em vez de exigir uma declaração especial de slots de tipo de instância.

slot, basta listar Wine. Ao mesmo tempo, não queremos especificar o intervalo do slot como THING — a classe mais geral em uma ontologia.

Em termos mais específicos:

> *Se uma lista de classes que define um intervalo ou um domínio de um slot incluir uma classe e sua subclasse, remova a subclasse.*

Se o intervalo do slot contiver tanto a classe Vinho quanto a classe Vinho Tinto, podemos remover o Vinho Tinto do intervalo, pois ele não acrescenta nenhuma informação nova: o Vinho Tinto é uma subclasse de Vinho e, portanto, o intervalo do slot já o inclui implicitamente, assim como todas as outras subclasses da classe Vinho.

> *Se uma lista de classes que define um intervalo ou um domínio de um slot contiver todas as subclasses de uma classe A, mas não a própria classe A, o intervalo deve conter apenas a classe A e não as subclasses.*

Em vez de definir o intervalo do slot para incluir Vinho Tinto, Vinho Branco e Vinho Rosé (enumerando todas as subclasses diretas de Vinho), podemos limitar o intervalo à própria classe Vinho em si.

> *Se uma lista de classes que define um intervalo ou um domínio de um slot contiver todas as subclasses de uma classe A, exceto algumas, considere se a classe A seria uma definição de intervalo mais adequada.*

Em sistemas em que anexar um slot a uma classe é o mesmo que adicionar a classe ao domínio do slot, as mesmas regras se aplicam ao anexo do slot: por um lado, devemos tentar torná-lo o mais geral possível. Por outro lado, devemos garantir que cada classe à qual anexamos o slot possa realmente ter a propriedade que o slot representa. Podemos anexar o slot de nível de tanino a cada uma das classes que representam vinhos tintos (por exemplo, Bordeaux, Merlot, Beaujolais, etc.). No entanto, como todos os vinhos tintos têm a propriedade de nível de tanino, devemos anexar o slot a essa classe mais geral de Vinhos Tintos. Generalizar ainda mais o domínio do slot nível de tanino (anexando-o à classe Vinho) não seria correto, pois não usamos o nível de tanino para descrever vinhos brancos, por exemplo.

## Passo 7. Criar instâncias

O último passo é criar instâncias individuais de classes na hierarquia. Definir uma instância individual de uma classe requer (1) escolher uma classe, (2) criar uma instância individual dessa classe e (3) preencher os valores dos slots. Por exemplo, podemos criar uma instância individual Chateau-Morgon-Beaujolais para representar um tipo específico de vinho Beaujolais. Chateau-Morgon-Beaujolais é uma instância da classe Beaujolais que representa todos os vinhos Beaujolais. Essa instância tem os seguintes valores de slot definidos (Figura 5):

- Corpo: Leve
- Cor: Vermelha
- Sabor: Delicado
- Nível de tanino: Baixo
- Uva: Gamay (instância da classe Uva para vinho)
- Produtor: Chateau-Morgon (exemplo da classe Vinícola)
- Região: Beaujolais (exemplo da classe Região vinícola)

- Açúcar: Seco

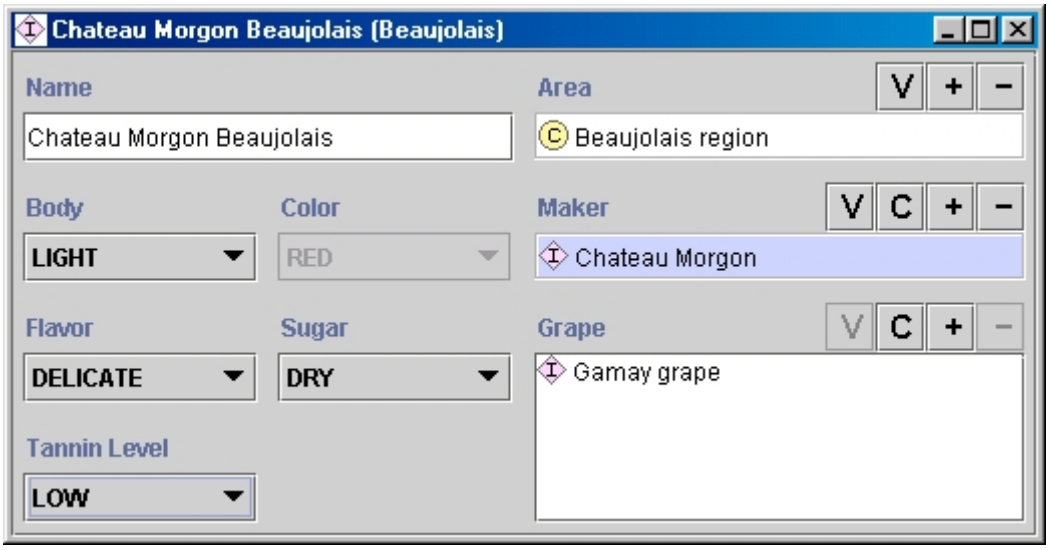


**Figura 5. A definição de uma instância da classe `Beaujolais`. A instância é `Chateaux Morgon Beaujolais` da região de Beaujolais, produzido a partir da uva Gamay pela vinícola Chateau Morgon. Tem um corpo leve, sabor delicado, cor vermelha e baixo nível de tanino. É um vinho seco.**

# 4 Definindo classes e uma hierarquia de classes

Esta seção discute os pontos a serem observados e os erros fáceis de cometer ao definir classes e uma hierarquia de classes (Etapa 4 da Seção 3). Como mencionamos anteriormente, não existe uma única hierarquia de classes correta para um determinado domínio. A hierarquia depende dos possíveis usos da ontologia, do nível de detalhe necessário para a aplicação, das preferências pessoais e, às vezes, dos requisitos de compatibilidade com outros modelos. No entanto, discutimos várias diretrizes a serem lembradas ao desenvolver uma hierarquia de classes. Depois de definir um número considerável de novas classes, é útil dar um passo atrás e verificar se a hierarquia emergente está em conformidade com essas diretrizes.

## *4.1 Garantindo que a hierarquia de classes esteja correta*

### Uma relação "é um"

A hierarquia de classes representa uma relação "é um": uma classe A é uma subclasse de B se todas as instâncias de A também forem instâncias de B. Por exemplo, `Chardonnay` é uma subclasse de `vinho branco`. Outra maneira de pensar na relação taxonômica é como uma relação "tipo de": `Chardonnay` é um tipo de `vinho branco`. Um avião a jato é um tipo de aeronave. Carne é um tipo de alimento.

> *Uma subclasse de uma classe representa um conceito que é um "tipo" do conceito que a superclasse representa.*

### Um único vinho não é uma subclasse de todos os vinhos.

Um erro comum na modelagem é incluir versões singular e plural do mesmo conceito na hierarquia, tornando a primeira uma subclasse da segunda. Por exemplo, é errado definir uma classe `Vinhos` e uma classe `Vinho` como uma subclasse de `Vinhos`. Quando se pensa na hierarquia como representando a relação "tipo de", o erro de modelagem fica claro: um único `Vinho` não é um **tipo de** `Vinhos`. A melhor maneira de evitar esse erro é sempre usar o singular ou o plural ao nomear classes ( consulte a Seção 6 para a discussão sobre conceitos de nomenclatura

conceitos).

### Transitividade das relações hierárquicas

Uma relação de subclasse é transitiva:

*Se B é uma subclasse de A e C é uma subclasse de B, então C é uma subclasse de A.*

Por exemplo, podemos definir uma classe `Vinho` e, em seguida, definir uma classe `Vinho branco` como uma subclasse de `Vinho`. Em seguida, definimos uma classe `Chardonnay` como uma subclasse de `Vinho branco`. A transitividade da relação de subclasse significa que a classe `Chardonnay` também é uma subclasse de `Vinho`. Às vezes, distinguimos entre subclasses diretas e subclasses indiretas. Uma **subclasse direta** é a subclasse "mais próxima" da classe: não há classes entre uma classe e sua subclasse direta em uma hierarquia. Ou seja, não há outras classes na hierarquia entre uma classe e sua superclasse direta. Em nosso exemplo, `Chardonnay` é uma subclasse direta de `Vinho branco` e não é uma subclasse direta de `Vinho`.

### Evolução de uma hierarquia de classes

Manter uma hierarquia de classes consistente pode se tornar um desafio à medida que os domínios evoluem. Por exemplo, durante muitos anos, todos os vinhos Zinfandel eram tintos. Portanto, definimos uma classe de vinhos `Zinfandel` como uma subclasse da classe `Vinho tinto`. Às vezes, porém, os produtores de vinho começaram a prensar as uvas e a remover imediatamente os aspectos que produzem cor, modificando assim a cor do vinho resultante. Assim, obtemos o "zinfandel branco", cuja cor é rosada. Agora precisamos dividir a classe `Zinfandel` em duas classes de zinfandel — `Zinfandel branco` e `Zinfandel tinto` — e classificá-las como subclasses de `vinho rosado` e `vinho tinto`, respectivamente.

### Classes e seus nomes

É importante distinguir entre uma classe e seu nome:

*As classes representam conceitos no domínio e não as palavras que denotam esses conceitos.*

O nome de uma classe pode mudar se escolhermos uma terminologia diferente, mas o termo em si representa a realidade objetiva no mundo. Por exemplo, podemos criar uma classe `Camarões` e depois renomeá-la para `Lagostins` — a classe continua a representar o mesmo conceito. As combinações de vinhos adequadas que se referiam a pratos de camarão devem referir-se a pratos de lagostins. Em termos mais práticos, deve seguir-se sempre a seguinte regra:

*Sinônimos para o mesmo conceito não representam classes diferentes*

Sinônimos são apenas nomes diferentes para um conceito ou termo. Portanto, **não** devemos ter uma classe chamada `Camarão` e uma classe chamada `Lagostim` e, possivelmente, uma classe chamada `Crevette`. Em vez disso, há uma única classe, chamada `Camarão` ou `Lagostim`. Muitos sistemas permitem associar uma lista de sinônimos, traduções ou nomes de apresentação a uma classe. Se um sistema não permitir essas associações, os sinônimos sempre poderão ser listados na documentação da classe.

### Evitar ciclos de classes

Devemos evitar **ciclos** na hierarquia de classes. Dizemos que existe um ciclo em uma hierarquia quando uma classe A tem uma subclasse B e, ao mesmo tempo, B é uma superclasse de A. Criar tal ciclo em uma hierarquia equivale a declarar que as classes A e B são equivalentes: todas as instâncias de A são instâncias de B e todas as instâncias de B também são instâncias de A. De fato, como B é uma subclasse de A, todas as instâncias de B devem ser instâncias da classe A. Como A é uma subclasse de B, todas as instâncias de A também devem ser instâncias da classe B.

## 4.2 Analisando irmãos em uma hierarquia de classes

### Irmãos em uma hierarquia de classes

**Irmãos** na hierarquia são classes que são subclasses diretas da mesma classe (consulte a Seção 4.1).

> *Todos os irmãos na hierarquia (exceto aqueles na raiz) devem estar no mesmo nível de generalidade.*

Por exemplo, `Vinho branco` e `Chardonnay` não devem ser subclasses da mesma classe.
(por exemplo, `Vinho`). `Vinho branco` é um conceito mais geral do que `Chardonnay`. Os irmãos devem representar conceitos que se enquadram "na mesma linha", da mesma forma que as seções do mesmo nível em um livro estão no mesmo nível de generalidade. Nesse sentido, os requisitos para uma hierarquia de classes são semelhantes aos requisitos para um esboço de livro.

Os conceitos na raiz da hierarquia, no entanto (que são frequentemente representados como subclasses diretas de alguma classe muito geral, como `Coisa`), representam divisões principais do domínio e não precisam ser conceitos semelhantes.

### Quantos são muitos e quantos são poucos?

Não há regras rígidas para o número de subclasses diretas que uma classe deve ter. No entanto, muitas ontologias bem estruturadas têm entre duas e uma dúzia de subclasses diretas. Portanto, temos as duas diretrizes a seguir:

> *Se uma classe tiver apenas uma subclasse direta, pode haver um problema de modelagem ou a ontologia pode estar incompleta.*
>
> *Se houver mais de uma dúzia de subclasses para uma determinada classe, pode ser necessário adicionar categorias intermediárias.*

A primeira das duas regras é semelhante a uma regra tipográfica que determina que as listas com marcadores nunca devem ter apenas um marcador. Por exemplo, a maioria dos vinhos tintos da Borgonha são vinhos Côtes d'Or. Suponha que queremos representar apenas esse tipo majoritário de vinhos da Borgonha. Poderíamos criar uma classe `Borgonha Tinto` e, em seguida, uma única subclasse `Côtes d'Or` (Figura 6a). No entanto, se em nossa representação os vinhos tintos da Borgonha e os vinhos Côtes d'Or forem essencialmente equivalentes (todos os vinhos tintos da Borgonha são vinhos Côtes d'Or e todos os vinhos Côtes d'Or são vinhos tintos da Borgonha), não é necessário criar a classe `Côtes d'Or`, pois ela não acrescenta nenhuma informação nova à representação. Se incluíssemos os vinhos Côtes Chalonnaise, que são vinhos da Borgonha mais baratos da região ao sul de Côtes d'Or, criaríamos duas subclasses da classe `Borgonha`: `Cotes d'Or` e `Cotes Chalonnaise` (Figura 6b).

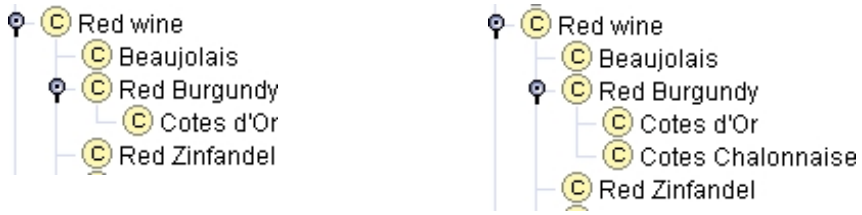


**Figura 6. Subclasses da classe `Borgonha Vermelho`. Ter uma única subclasse de uma classe geralmente indica um problema na modelagem.**

Suponha agora que listemos todos os tipos de vinhos como subclasses diretas da classe `Vinho`. Essa lista incluiria então tipos mais gerais de vinho, como Beaujolais e Bordeaux, bem como tipos mais específicos, como Paulliac e Margaux (Figura 6a). A classe `Wine` tem subclasses diretas demais e, de fato, para que a ontologia reflita os diferentes tipos de vinho de maneira mais organizada, `Medoc` deveria ser uma subclasse de `Bordeaux` e `Cotes d'Or` deveria ser uma

subclasse de `Burgundy`. Ter também categorias intermediárias como `vinho tinto` e `vinho branco` refletiria o modelo conceitual do domínio dos vinhos que muitas pessoas têm (Figura 6b).

No entanto, se não existirem classes naturais para agrupar conceitos na longa lista de irmãos, não há necessidade de criar classes artificiais — basta deixar as classes como estão. Afinal, a ontologia é um reflexo do mundo real e, se não existe categorização no mundo real, a ontologia deve refletir isso.

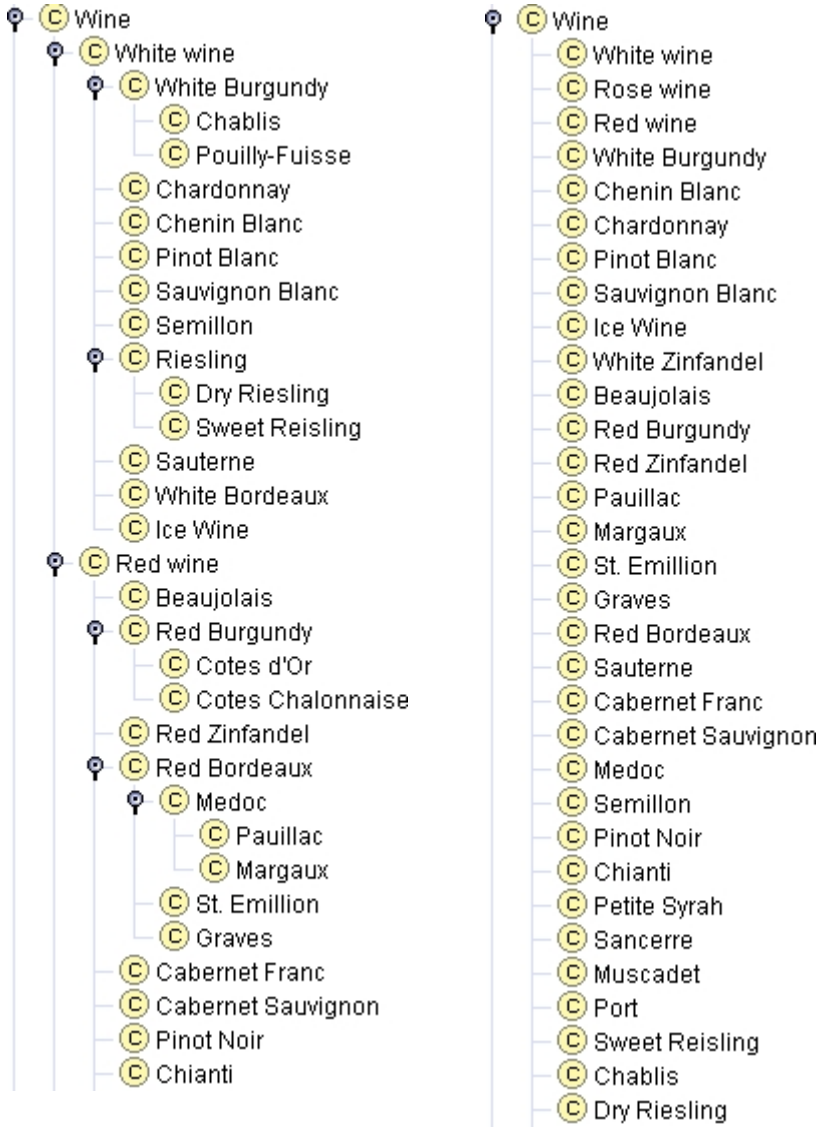


**Figura 7. Categorização de vinhos. Ter todos os vinhos e tipos de vinho versus ter vários níveis de categorização.**

## *4.3 Herança múltipla*

A maioria dos sistemas de representação de conhecimento permite **herança múltipla** na hierarquia de classes: uma classe pode ser uma subclasse de várias classes. Suponha que gostaríamos de criar uma classe separada `para` vinhos de sobremesa, a classe `Vinho de sobremesa`. O vinho do Porto é tanto um vinho tinto quanto um vinho de sobremesa.

vinho.[4] Portanto, definimos uma classe `Porto` para ter duas superclasses: `Vinho tinto` e `Vinho de sobremesa`. Todas as instâncias da classe `Porto` serão instâncias tanto da classe `Vinho tinto` quanto da classe `Vinho de sobremesa`. A classe `Porto` herdará seus slots e suas facetas de ambos os pais. Assim, ela herdará o valor DOCE para o slot `Açúcar` da classe `Vinho de sobremesa` e o slot `Nível de tanino` e o valor para seu slot Cor da classe `Vinho tinto`.

### *4.4 Quando introduzir uma nova classe (ou não)*

Uma das decisões mais difíceis a tomar durante a modelagem é quando introduzir uma nova classe ou quando representar uma distinção por meio de diferentes valores de propriedade. É difícil navegar tanto em uma hierarquia extremamente aninhada, com muitas classes irrelevantes, quanto em uma hierarquia muito plana, com poucas classes e muitas informações codificadas em slots. No entanto, não é fácil encontrar o equilíbrio adequado.

Existem várias regras práticas que ajudam a decidir quando introduzir novas classes em uma hierarquia.

> *As subclasses de uma classe geralmente (1) têm propriedades adicionais que a superclasse não tem, ou (2) restrições diferentes das da superclasse, ou (3) participam de relações diferentes das superclasses.*

Os vinhos tintos podem ter diferentes níveis de tanino, enquanto essa propriedade não é usada para descrever vinhos em geral. O valor para o slot de açúcar do vinho de sobremesa é DOCE, enquanto isso não é verdade para a superclasse da classe Vinho de sobremesa. Os vinhos Pinot Noir podem combinar bem com frutos do mar, enquanto outros vinhos tintos não. Em outras palavras, geralmente introduzimos uma nova classe na hierarquia apenas quando há algo que podemos dizer sobre essa classe que não podemos dizer sobre a superclasse.

Em termos práticos, cada subclasse deve ter novos slots adicionados a ela, ou ter novos valores de slot definidos, ou substituir algumas facetas dos slots herdados.

No entanto, às vezes pode ser útil criar novas classes, mesmo que elas não introduzam nenhuma propriedade nova.

> *As classes em hierarquias terminológicas não precisam introduzir novas propriedades.*

Por exemplo, algumas ontologias incluem grandes hierarquias de referência de termos comuns usados no domínio. Por exemplo, uma ontologia subjacente a um sistema de registros médicos eletrônicos pode incluir uma classificação de várias doenças. Essa classificação pode ser apenas isso — uma hierarquia de termos, sem propriedades (ou com o mesmo conjunto de propriedades). Nesse caso, ainda é útil organizar os termos em uma hierarquia, em vez de uma lista plana, porque isso (1) permitirá uma exploração e navegação mais fáceis e (2) permitirá que um médico escolha facilmente um nível de generalidade do termo que seja apropriado para a situação.

Outra razão para introduzir novas classes sem nenhuma propriedade nova é modelar conceitos entre os quais os especialistas da área costumam fazer uma distinção, mesmo que tenhamos decidido não modelar a distinção em si. Como usamos ontologias para facilitar a comunicação entre especialistas da área e entre especialistas da área e sistemas baseados em conhecimento, gostaríamos de refletir a visão do especialista sobre a área na ontologia.

Por fim, não devemos criar subclasses de uma classe para cada restrição adicional. Por exemplo, introduzimos as classes `Vinho tinto`, `Vinho branco` e `Vinho rosé` porque essa distinção é natural no mundo do vinho. Não introduzimos classes para delicado

---

[4] Optamos por representar apenas os vinhos do Porto tintos na nossa ontologia: os vinhos do Porto brancos existem, mas são extremamente raros.

vinho, vinho moderado e assim por diante. Ao definir uma hierarquia de classes, nosso objetivo é encontrar um equilíbrio entre a criação de novas classes úteis para a organização das classes e a criação de classes em excesso.

## 4.5 Uma nova classe ou um valor de propriedade?

Ao modelar um domínio, muitas vezes precisamos decidir se devemos modelar uma distinção específica (como vinho branco, tinto ou rosé) como um valor de propriedade ou como um conjunto de classes, o que depende novamente do escopo do domínio e da tarefa em questão.

Criamos uma classe `Vinho branco` ou simplesmente criamos uma classe `Vinho` e preenchemos valores diferentes para o campo `cor`? A resposta geralmente está no escopo que definimos para a ontologia. Qual é a importância do conceito de `vinho branco` em nosso domínio? Se os vinhos têm apenas importância marginal no domínio e se o vinho é branco ou não não tem implicações particulares para suas relações com outros objetos, então não devemos introduzir uma classe separada para vinhos brancos. Para um modelo de domínio usado em uma fábrica que produz rótulos de vinho, as regras para rótulos de vinho de qualquer cor são as mesmas e a distinção não é muito importante. Alternativamente, para a representação de vinho, comida e suas combinações apropriadas, um vinho tinto é muito diferente de um vinho branco: ele é combinado com diferentes alimentos, tem propriedades diferentes e assim por diante. Da mesma forma, a cor do vinho é importante para a base de conhecimento sobre vinhos que podemos usar para determinar a ordem de degustação. Assim, criamos uma classe separada para `vinho branco`.

> *Se os conceitos com valores de slot diferentes se tornarem restrições para slots diferentes em outras classes, devemos criar uma nova classe para a distinção. Caso contrário, representamos a distinção em um valor de slot.*

Da mesma forma, nossa ontologia do vinho tem classes como `Merlot Tinto` e `Merlot Branco`, em vez de

do que uma única classe para todos os vinhos Merlot: Merlots tintos e Merlots brancos são vinhos realmente diferentes (feitos da mesma uva) e, se estivermos desenvolvendo uma ontologia detalhada do vinho, essa distinção é importante.

> *Se uma distinção é importante no domínio e pensamos nos objetos com valores diferentes para a distinção como tipos diferentes de objetos, então devemos criar uma nova classe para a distinção.*

Considerar instâncias individuais potenciais de uma classe também pode ser útil para decidir se deve ou não introduzir uma nova classe.

> *Uma classe à qual uma instância individual pertence não deve mudar com frequência.*

Normalmente, quando usamos propriedades extrínsecas em vez de intrínsecas dos conceitos para diferenciar entre classes, as instâncias dessas classes terão que migrar frequentemente de uma classe para outra. Por exemplo, `vinho gelado` não deve ser uma classe em uma ontologia que descreve garrafas de vinho em um restaurante. A propriedade `gelado` deve ser simplesmente um atributo do vinho em uma garrafa, uma vez que uma instância de `vinho gelado` pode facilmente deixar de ser uma instância dessa classe e depois se tornar uma instância dessa classe novamente.

Normalmente, números, cores e localizações são valores de slot e não causam a criação de novas classes. O vinho, no entanto, é uma exceção notável, uma vez que a cor do vinho é tão importante para a descrição do vinho.

Para outro exemplo, considere a ontologia da anatomia humana. Quando representamos as costelas, criamos uma classe para cada uma das "1ª costela esquerda", "2ª costela esquerda" e assim por diante? Ou temos uma classe Costela com espaços para a ordem e a posição lateral (esquerda-direita)? Se as informações sobre cada uma das costelas que representamos na ontologia forem significativamente diferentes, então devemos realmente

---

[5] Aqui assumimos que cada órgão anatômico é uma classe, pois também gostaríamos de falar sobre a "1ª costela esquerda de João". Órgãos individuais de pessoas existentes seriam representados como indivíduos em nossa ontologia.

Crie uma classe para cada uma das costelas. Ou seja, se quisermos representar detalhes de adjacência e informações de localização (que são diferentes para cada costela), bem como funções específicas que cada costela desempenha e órgãos que ela protege, precisamos das classes. Se estivermos modelando a anatomia em um nível um pouco menos geral e todas as costelas forem muito semelhantes no que diz respeito às nossas aplicações potenciais (apenas falamos sobre qual costela está quebrada na radiografia, sem implicações para outras partes do corpo), podemos simplificar nossa hierarquia e ter apenas a classe `Costela`, com dois slots: `posição lateral` e `ordem`.

## *4.6 Uma instância ou uma classe?*

Decidir se um determinado conceito é uma classe em uma ontologia ou uma instância individual depende das possíveis aplicações da ontologia. Decidir onde terminam as classes e começam as instâncias individuais começa com a decisão sobre qual é o nível mais baixo de granularidade na representação. O nível de granularidade é, por sua vez, determinado por uma aplicação potencial da ontologia. Em outras palavras, quais são os itens mais específicos que serão representados na base de conhecimento? Voltando às questões de competência que identificamos na Etapa 1 da Seção 3, os conceitos mais específicos que constituirão respostas a essas questões são candidatos muito bons para indivíduos na base de conhecimento.

> *As instâncias individuais são os conceitos mais específicos representados em uma base de conhecimento.*

Por exemplo, se formos falar apenas sobre harmonização de vinhos com alimentos, não estaremos interessados nas garrafas físicas específicas de vinho. Portanto, termos como `Sterling Vineyards Merlot` provavelmente serão os termos mais específicos que usaremos. Em outras palavras, a classe Vinho é uma coleção não de garrafas individuais de vinhos, mas sim de vinhos específicos produzidos por vinícolas específicas. Portanto, `Sterling Vineyards Merlot` seria uma instância na base de conhecimento.

Por outro lado, se quisermos manter um inventário de vinhos no restaurante, além da base de conhecimento sobre boas combinações de vinhos e pratos, garrafas individuais de cada vinho podem se tornar instâncias individuais em nossa base de conhecimento.

Da mesma forma, se quisermos registrar propriedades diferentes para cada safra específica do `Sterling Vineyards Merlot`, então a safra específica do vinho é uma instância em uma base de conhecimento e o `Sterling Vineyards Merlot` é uma classe que contém instâncias para todas as suas safras.

Outra regra pode "mover" algumas instâncias individuais para o conjunto de classes:

> *Se os conceitos formam uma hierarquia natural, devemos representá-los como classes.*

Considere as regiões vinícolas. Inicialmente, podemos definir as principais regiões vinícolas, como França, Estados Unidos, Alemanha e assim por diante, como classes, e regiões vinícolas específicas dentro dessas grandes regiões como instâncias. Por exemplo, `a região de Bourgogne` é uma instância da classe `região francesa`. No entanto, também gostaríamos de dizer que a `região de Cotes d'Or` é uma `região de Bourgogne`. Portanto, `a região da Borgonha` deve ser uma classe (para ter subclasses ou instâncias). No entanto, tornar `a região da Borgonha` uma classe e `a região de Cotes d'Or` uma instância da `região da Borgonha` parece arbitrário: é muito difícil distinguir claramente quais regiões são classes e quais são instâncias. Portanto, definimos todas as regiões vinícolas como classes. O Protégé-2000 permite que os usuários especifiquem algumas classes como abstratas, significando que a classe não pode ter instâncias diretas. No nosso caso, todas as classes de região são abstratas (Figura 8).

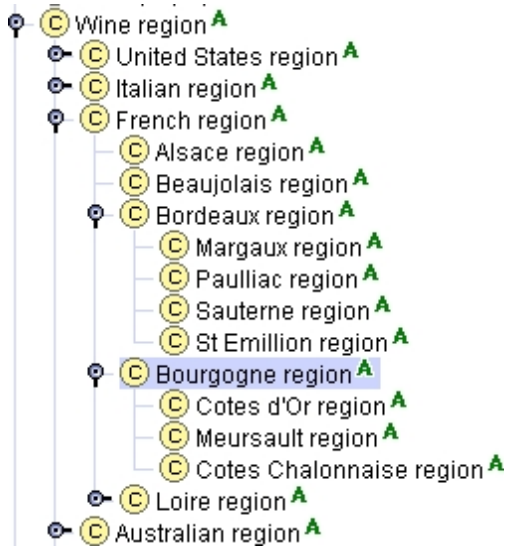


**Figura 8. Hierarquia das regiões vinícolas. Os ícones "A" ao lado dos nomes das classes indicam que as classes são abstratas e não podem ter instâncias diretas.**

A mesma hierarquia de classes seria incorreta se omitíssemos a palavra "região" dos nomes das classes. Não podemos dizer que a classe `Alsácia` é uma subclasse da classe `França`: a Alsácia não é um tipo de França. No entanto, a região da Alsácia é um tipo de região francesa.

Apenas as classes podem ser organizadas em uma hierarquia — os sistemas de representação do conhecimento não têm uma noção de subinstância. Portanto, se houver uma hierarquia natural entre os termos, como nas hierarquias terminológicas da Seção 4.2, devemos definir esses termos como classes, mesmo que eles não tenham instâncias próprias.

## *4.7 Limitando o escopo*

Como observação final sobre a definição de uma hierarquia de classes, o seguinte conjunto de regras é sempre útil para decidir quando uma definição de ontologia está completa:

> *A ontologia não deve conter todas as informações possíveis sobre o domínio: você não precisa se especializar (ou generalizar) mais do que o necessário para sua aplicação (no máximo um nível extra em cada sentido).*

Para o nosso exemplo de vinhos e alimentos, não precisamos saber que tipo de papel é usado para os rótulos ou como cozinhar pratos de camarão.

Da mesma forma,

> *A ontologia não deve conter todas as propriedades possíveis e distinções entre as classes na hierarquia.*

Em nossa ontologia, certamente não incluímos todas as propriedades que um vinho ou alimento poderia ter. Representamos as propriedades mais salientes das classes de itens em nossa ontologia. Embora os livros sobre vinhos nos informem o tamanho das uvas, não incluímos esse conhecimento. Da mesma forma, não adicionamos todas as relações que se poderia imaginar entre todos os termos em nosso sistema. Por exemplo, não incluímos relações como `vinho favorito` e `comida favorita` na ontologia apenas para permitir uma representação mais completa de todas as interconexões entre os termos que definimos.

A última regra também se aplica ao estabelecimento de relações entre conceitos que já incluímos na ontologia. Considere uma ontologia que descreve experimentos biológicos. A ontologia provavelmente conterá um conceito de `organismos biológicos`. Também conterá um conceito de um `experimentador` realizando um experimento (com seu nome, afiliação, etc.). É verdade

que um experimentador, como pessoa, também é um organismo biológico. No entanto, provavelmente não devemos incorporar essa distinção na ontologia: para os fins desta representação, um experimentador não é um organismo biológico e provavelmente nunca conduziremos experimentos nos próprios experimentadores. Se estivéssemos representando tudo o que podemos dizer sobre as classes na ontologia, um `Experimentador` se tornaria uma subclasse de `Organismo Biológico`. No entanto, não precisamos incluir esse conhecimento para as aplicações previsíveis. Na verdade, incluir esse tipo de classificação adicional para classes existentes é prejudicial: agora, uma instância de um Experimentador terá campos para peso, idade, espécie e outros dados relativos a um organismo biológico, mas absolutamente irrelevantes no contexto da descrição de um experimento. No entanto, devemos registrar essa decisão de design na documentação para o benefício dos usuários que consultarão essa ontologia e que podem não estar cientes da aplicação que tínhamos em mente. Caso contrário, as pessoas que pretendem reutilizar a ontologia para outras aplicações podem tentar usar o experimentador como uma subclasse de pessoa sem saber que a modelagem original não incluía esse fato.

### *4.8 Subclasses disjuntas*

Muitos sistemas permitem especificar explicitamente que várias classes são **disjuntas**. As classes são disjuntas se não puderem ter nenhuma instância em comum. Por exemplo, as classes `Vinho de sobremesa` e `Vinho branco` em nossa ontologia *não* são disjuntas: há muitos vinhos que são instâncias de ambas. A instância `Rothermel Trochenbierenauslese Riesling` da classe `Riesling doce` é um exemplo disso. Ao mesmo tempo, as classes `Vinho tinto` e `Vinho branco` são disjuntas: nenhum vinho pode ser simultaneamente tinto e branco. Especificar que as classes são disjuntas permite que o sistema valide melhor a ontologia. Se declararmos que as classes `Vinho Tinto` e `Vinho Branco` são disjuntas e, posteriormente, criarmos uma classe que é uma subclasse tanto de `Riesling` (uma subclasse de `Vinho Branco`) quanto de `Porto` (uma subclasse de `Vinho Tinto`), o sistema poderá indicar que há um erro de modelagem.

## 5 Definindo propriedades — mais detalhes

Nesta seção, discutimos vários outros detalhes a serem lembrados ao definir slots na ontologia (Etapa 5 e Etapa 6 na Seção 3). Principalmente, discutimos slots inversos e valores padrão para um slot.

### *5.1 Slots inversos*

O valor de um slot pode depender do valor de outro slot. Por exemplo, se um `vinho` foi `produzido por` uma `vinícola`, então a `vinícola produz` esse `vinho`. Essas duas relações, `fabricante` e `produz`, são chamadas de **relações inversas**. Armazenar as informações "em ambas as direções" é redundante. Quando sabemos que um vinho é produzido por uma vinícola, um aplicativo que usa a base de conhecimento sempre pode inferir o valor da relação inversa de que a vinícola produz o vinho. No entanto, do ponto de vista da aquisição de conhecimento, é conveniente ter ambas as informações explicitamente disponíveis. Essa abordagem permite que os usuários preencham o vinho em um caso e a vinícola em outro. O sistema de aquisição de conhecimento poderia então preencher automaticamente o valor da relação inversa, garantindo a consistência da base de conhecimento.

Nosso exemplo tem um par de slots inversos: o slot `maker` da classe `Wine` e o slot `produces` da classe `Winery`. Quando um usuário cria uma instância da classe `Wine` e preenche o valor do slot `maker`, o sistema adiciona automaticamente a instância recém-criada ao slot `produces` da instância `Winery` correspondente. Por exemplo, quando dizemos que o Sterling Merlot é produzido pela vinícola Sterling Vineyard, o sistema

adicionar automaticamente o Sterling Merlot à lista de vinhos produzidos pela vinícola Sterling Vineyard. (Figura 9).

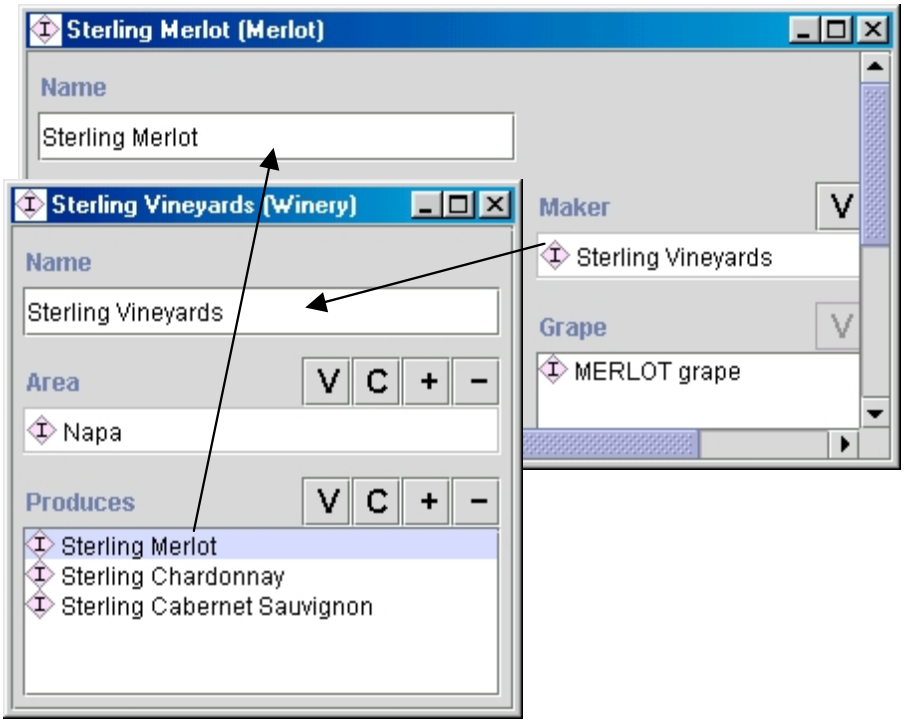


**Figura 9. Instâncias com slots inversos. O slot `produzido` para a classe `Vinícola` é um inverso do slot `maker` para a classe `Vinho`. Preencher um dos slots aciona uma atualização automática do outro.**

### *5.2 Valores padrão*

Muitos sistemas baseados em estruturas permitem a especificação de valores padrão para slots. Se um determinado valor de slot for o mesmo para a maioria das instâncias de uma classe, podemos definir esse valor como **`padrão`** para o slot. Assim, quando cada nova instância de uma classe que contém esse slot for criada, o sistema preencherá o valor padrão automaticamente. Podemos então alterar o valor para qualquer `outro` valor que as facetas permitirem. Ou seja, os valores padrão existem por conveniência: eles `não` impõem nenhuma nova restrição ao modelo nem alteram o modelo de forma alguma.

Por exemplo, se a maioria dos vinhos que vamos discutir forem vinhos encorpados, podemos ter "encorpado" como valor padrão para o corpo do vinho. Então, a menos que digamos o contrário, todos os vinhos que definirmos serão encorpados.

Observe que isso é diferente dos **valores dos slots**. Os valores dos slots não podem ser alterados. Por exemplo, podemos dizer que o slot `açúcar` tem o valor DOCE para a classe `Vinho de sobremesa`. Então, todas as subclasses e instâncias da classe `Vinho de sobremesa` terão o valor DOCE para o slot `açúcar`. Esse valor não pode ser alterado em nenhuma das subclasses ou instâncias da classe.

## 6 O que há em um nome?

Definir convenções de nomenclatura para conceitos em uma ontologia e, em seguida, seguir rigorosamente essas convenções não apenas torna a ontologia mais fácil de entender, mas também ajuda a evitar alguns erros comuns de modelagem. Existem muitas alternativas para nomear conceitos. Muitas vezes, não há uma razão específica para escolher uma ou outra alternativa. No entanto, precisamos

*Definir uma convenção de nomenclatura para classes e slots e segui-la.*

Os seguintes recursos de um sistema de representação de conhecimento afetam a escolha das convenções de nomenclatura

:

- O sistema tem o mesmo espaço de nomes para classes, slots e instâncias? Ou seja, o sistema permite ter uma classe e um slot com o mesmo nome (como uma classe `vinícola` e um slot `vinícola`)?
- O sistema diferencia maiúsculas de minúsculas? Ou seja, o sistema trata nomes que diferem apenas em maiúsculas e minúsculas como nomes diferentes (por exemplo, `Winery` e `winery`)?
- Que delimitadores o sistema permite nos nomes? Ou seja, os nomes podem conter espaços, vírgulas, asteriscos e assim por diante?

O Protégé-2000, por exemplo, mantém um único espaço de nomes para todos os seus quadros. Ele diferencia maiúsculas de minúsculas. Portanto, não podemos ter uma classe `winery` e um slot `winery`. No entanto, podemos ter uma classe `Winery` (note as letras maiúsculas) e um slot `winery`. O CLASSIC, por outro lado, não diferencia maiúsculas de minúsculas e mantém espaços de nomes diferentes para classes, slots e indivíduos. Assim, do ponto de vista do sistema, não há problema em nomear tanto uma classe quanto um slot como `Winery`.

## 6.1 Capitalização e delimitadores

Primeiro, podemos melhorar significativamente a legibilidade de uma ontologia se usarmos letras maiúsculas de forma consistente para os nomes dos conceitos. Por exemplo, é comum usar letras maiúsculas para nomes de classes e minúsculas para nomes de slots (supondo que o sistema diferencie maiúsculas de minúsculas).

Quando um nome de conceito contém mais de uma palavra (como `Prato da refeição`), precisamos delimitar as palavras. Aqui estão algumas opções possíveis.

- Use espaço: `Prato da refeição` (muitos sistemas, incluindo o Protégé, permitem espaços nos nomes dos conceitos).
- Junte as palavras e coloque cada nova palavra em maiúscula: `Prato`
- Use um sublinhado, traço ou outro delimitador no nome: `Meal_Course`, `Meal_course`, `Meal-Course`, `Meal-course`. (Se você usar delimitadores, também precisará decidir se cada nova palavra será escrita com letra maiúscula ou não)

Se o sistema de representação do conhecimento permitir espaços nos nomes, utilizá-los pode ser a solução mais intuitiva para muitos desenvolvedores de ontologias. No entanto, é importante considerar outros sistemas com os quais o seu sistema pode interagir. Se esses sistemas não utilizarem espaços ou se o seu meio de apresentação não lidar bem com espaços, pode ser útil utilizar outro método.

## 6.2 Singular ou plural

Um nome de classe representa uma coleção de objetos. Por exemplo, uma classe `Wine` representa, na verdade, todos os vinhos. Portanto, pode ser mais natural para alguns designers chamar a classe `de Vinhos` em vez de `Vinho`. Nenhuma alternativa é melhor ou pior que a outra ( embora o singular para nomes de classes seja usado com mais frequência na prática). No entanto, seja qual for a escolha, ela deve ser consistente em toda a ontologia. Alguns sistemas exigem até mesmo que seus usuários declarem antecipadamente se vão usar o singular ou o plural para nomes de conceitos e não permitem que eles se desviem dessa escolha.

Usar sempre o mesmo formato também evita que o designer cometa erros de modelagem, como criar uma classe `Vinhos` e, em seguida, criar uma classe `Vinho` como sua subclasse (consulte a Seção 4.1).

## 6.3 Convenções de prefixo e sufixo

Algumas metodologias de base de conhecimento sugerem o uso de convenções de prefixos e sufixos nos nomes para distinguir entre classes e slots. Duas práticas comuns são adicionar um `prefixo has-` ou um sufixo `-of` aos nomes dos slots. Assim, nossos slots se tornam `has-maker` e `has-winery` se escolhermos o

tem-convenção. Os slots tornam-se `fabricante-de` e `vinícola-de` se escolhermos a convenção de-. Esta abordagem permite que qualquer pessoa que veja um termo determine imediatamente se o termo é uma classe ou um slot. No entanto, os nomes dos termos tornam-se ligeiramente mais longos.

### *6.4 Outras considerações sobre nomenclatura*

Aqui estão mais algumas coisas a serem consideradas ao definir convenções de nomenclatura:

- Não adicione strings como "classe", "propriedade", "slot" e assim por diante aos nomes dos conceitos.

É sempre claro pelo contexto se o conceito é uma classe ou um slot, por exemplo. Além disso, se você usar convenções de nomenclatura diferentes para classes e slots (por exemplo, letras maiúsculas e minúsculas, respectivamente), o próprio nome indicará qual é o conceito.

- Geralmente, é uma boa ideia evitar abreviações em nomes de e es de conceitos (ou seja, use `Cabernet Sauvignon` em vez de `Cab`).
- Os nomes das subclasses diretas de uma classe devem incluir ou não incluir o nome da superclasse. Por exemplo, se estivermos criando duas subclasses da classe `Wine` para representar vinhos tintos e brancos, os dois nomes das subclasses devem ser `Red Wine` e `White Wine` ou `Red` e `White`, mas não `Red Wine` e `White`.

## 7 Outros recursos

Usamos o Protégé-2000 como ambiente de desenvolvimento de ontologia para nossos exemplos. Duineveld e colegas (Duineveld et al. 2000) descrevem e comparam vários outros ambientes de desenvolvimento de ontologia.

Procuramos abordar os fundamentos básicos do desenvolvimento de ontologias e não discutimos muitos dos tópicos avançados ou metodologias alternativas para o desenvolvimento de ontologias. Gómez-Pérez (Gómez-Pérez 1998) e Uschold (Uschold e Gruninger 1996) apresentam metodologias alternativas para o desenvolvimento de ontologias. O tutorial Ontolingua (Farquhar 1997) discute alguns aspectos formais da modelagem do conhecimento.

Atualmente, os pesquisadores enfatizam não apenas o desenvolvimento de ontologias, mas também a análise de ontologias. À medida que mais ontologias são geradas e reutilizadas, mais ferramentas estarão disponíveis para analisá-las. Por exemplo, o Chimaera (McGuinness et al. 2000) fornece ferramentas de diagnóstico para analisar ontologias. A análise que o Chimaera realiza inclui tanto uma verificação da correção lógica de uma ontologia quanto diagnósticos de erros comuns de design de ontologias. Um designer de ontologia pode querer executar diagnósticos do Chimaera sobre a ontologia em evolução para determinar a conformidade com as práticas comuns de modelagem de ontologia.

## 8 Conclusões

Neste guia, descrevemos uma metodologia de desenvolvimento de ontologias para sistemas declarativos baseados em estruturas. Listamos as etapas do processo de desenvolvimento de ontologias e abordamos as complexas questões da definição de hierarquias de classes e propriedades de classes e instâncias. No entanto, depois de seguir todas as regras e sugestões, uma das coisas mais importantes a lembrar é o seguinte: *não existe uma única ontologia correta para qualquer domínio*. O design de ontologias é um processo criativo e não há duas ontologias projetadas por pessoas diferentes que sejam iguais. As aplicações potenciais da ontologia e a compreensão e visão do designer sobre o domínio, sem dúvida, afetarão as escolhas de design da ontologia. "A prova está no pudim" — só podemos avaliar a qualidade da nossa ontologia usando-a nas aplicações para as quais a projetamos.

## Agradecimentos

O Protégé-2000 (http://protege.stanford.edu) foi desenvolvido pelo grupo de Mark Musen na Stanford Medical Informatics. Geramos algumas das figuras com o plugin OntoViz para o Protégé-2000. Importamos a versão inicial da ontologia do vinho da biblioteca de ontologias Ontolingua (http://www.ksl.stanford.edu/software/ontolingua/) que em por sua vez utilizou uma versão publicada por Brachman e colegas (Brachman et al. 1991) e distribuída com o sistema de representação de conhecimento CLASSIC. Em seguida, modificamos a ontologia para apresentar princípios de modelagem conceitual para ontologias declarativas baseadas em estruturas. Os comentários extensos de Ray Fergerson e Mor Peleg sobre versões anteriores melhoraram muito este artigo.

## Referências

Booch, G., Rumbaugh, J. e Jacobson, I. (1997). *Guia do usuário da Linguagem de Modelagem Unificada*: Addison-Wesley.

Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A. e Borgida, A. (1991). Vivendo com o CLASSIC: Quando e como usar a linguagem KL-ONE. *Princípios das redes semânticas*. J. F. Sowa, editor, Morgan Kaufmann**:** 401-456.

Brickley, D. e Guha, R.V. (1999). Especificação do Esquema da Estrutura de Descrição de Recursos (RDF). Recomendação proposta, World Wide Web Consortium**:** http://www.w3.org/TR/PR-rdf-schema.

Chimaera (2000). Chimaera Ontology Environment.www.ksl.stanford.edu/software/chimaera Duineveld,

A.J., Stoter, R., Weiden, M.R., Kenepa, B. e Benjamins, V.R. (2000). WonderTools? Um estudo comparativo de ferramentas de engenharia ontológica. *Revista Internacional de Estudos Homem-Computador* **52**(6): 1111-1133.

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

Gómez-Pérez, A. (1998). Compartilhamento e reutilização de conhecimento. *Manual de Sistemas Especializados Aplicados*. Liebowitz, editor, CRC Press.

Gruber, T.R. (1993). Uma abordagem de tradução para a especificação de ontologias portáteis. *Aquisição de conhecimento* **5**: 199-220.

Gruninger, M. e Fox, M.S. (1995). Metodologia para o projeto e avaliação de ontologias. Em: *Anais do Workshop sobre Questões Ontológicas Básicas no Compartilhamento de Conhecimento, IJCAI-95*, Montreal.

Hendler, J. e McGuinness, D.L. (2000). A Linguagem de Marcação de Agentes DARPA. *IEEE Intelligent Systems* **16**(6): 67-73.

Humphreys, B.L. e Lindberg, D.A.B. (1993). O projeto UMLS: estabelecendo a conexão conceitual entre os usuários e as informações de que necessitam. *Boletim da Associação de Bibliotecas Médicas* **81**(2): 170.

McGuinness, D.L., Abrahams, M.K., Resnick, L.A., Patel-Schneider, P.F., Thomason, R.H., Cavalli-Sforza, V. e Conati, C. (1994). Tutorial clássico sobre sistemas de representação do conhecimento. http://www.bell-labs.com/project/classic/papers/ClassTut/ClassTut.html

McGuinness, D.L., Fikes, R., Rice, J. e Wilder, S. (2000). Um ambiente para fusão e teste de grandes ontologias. *Princípios de representação e raciocínio do conhecimento: Anais da Sétima Conferência Internacional ( KR2000)*. A . G . Cohn, F . Giunchiglia e B. Selman, editores. São Francisco, Califórnia, Morgan Kaufmann Publishers.

McGuinness, D.L. e Wright, J. (1998). Modelagem conceitual para configuração: uma abordagem baseada em lógica descritiva. *Inteligência artificial para projeto, análise e fabricação de engenharia - edição especial sobre configuração*.

Musen, M.A. (1992). Dimensões do compartilhamento e reutilização do conhecimento. *Computadores e Pesquisa Biomédica* **25**: 435-467.

Ontolingua (1997). Manual de Referência do Sistema Ontolingua. http://www-ksl-svc.stanford.edu:5915/doc/frame-editor/index.html

Price, C. e Spackman, K. (2000). Termos clínicos SNOMED. *BJHC&IM-British Journal of Healthcare Computing & Information Management* **17**(3): 27-31.

Protege (2000). O Projeto Protege. http://protege.stanford.edu

Rosch, E. (1978). Princípios de Categorização. *Cognição e Categorização*. R. E. e B. B. Lloyd, editores. Hillside, NJ, Lawrence Erlbaum Publishers**:** 27-48.

Rothenfluh, T.R., Gennari, J.H., Eriksson, H., Puerta, A.R., Tu, S.W. e Musen, M.A. (1996). Ontologias reutilizáveis, ferramentas de aquisição de conhecimento e sistemas de desempenho: soluções PROTÉGÉ-II para Sisyphus-2. *Revista Internacional de Estudos Humano-Computacionais* **44:** 303-332.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. e Lorensen, W. (1991). *Modelagem e design orientados a objetos*. Englewood Cliffs, Nova Jersey: Prentice Hall.

Uschold, M. e Gruninger, M. (1996). Ontologias: princípios, métodos e aplicações. *Knowledge Engineering Review* **11**(2).