


# Sistemas de Marcos (Frames)

# Sistemas de Marcos: contenidos

- Introducción
- Tipos de marcos:
  - Instancias
  - Clases
  - Meta-clases
- Atributos
  - De clase y de instancia
  - Jerarquías de atributos
- Valores calculados
- Herencia de atributos
  - Distancia inferencial
- Conclusiones

## Marcos (*frames*) [Minsky, 1975]

- Lógica de predicados
    - Conocimiento factual (terminológico y asertivo)
    - Orientado a relaciones
  - Redes semánticas
    - Conocimiento factual
    - Orientado a conceptos
  - Sistemas de producción
    - Conocimiento procedimental
  - **Marcos**
    - Conocimiento factual + cierto tipo de conocimiento procedimental
    - Orientado a conceptos
      - Una entidad o concepto se describe a través de un conjunto de pares atributo/valor (con posibles restricciones para los valores)
      - Son estructuras de ranura/relleno **débiles**
      - Evolución de las redes semánticas
- 
- Asignar más estructura a los nodos y a las conexiones

# Estructuras de relleno débiles vs fuertes

- Estructuras de ranura y relleno **débiles versus fuertes**
  - Depende de la cantidad de conocimiento específico del dominio
    - Rango amplio entre Débil y Fuerte
  - **Débil**: poco conocimiento, muy generalista → aplicable a muchos dominios
    - Aunque, para aplicarlos, quedan algunas operaciones que programar
    - El diseñador decide qué tipos de objetos y qué relaciones utilizar
    - Redes semánticas, Frames, Sistemas de producción
  - **Fuerte**: mucho conocimiento específico → aplicable sólo a ciertos dominios
    - Fijan conocimiento específico sobre los tipos de objetos y relaciones permitidos
    - El diseñador se ha de ajustar a ellos
    - Scripts, Dependencia Conceptual

# Marcos y Sistemas de Marcos

- Un **marco** es una colección de atributos y valores
  - Describe un determinado concepto o un conjunto de conceptos
  - Las propiedades de los conceptos se representan con **ranuras** (*slots*)
  - Los valores para estas propiedades son los **rellenos** (*fillers*)

<p><b>Juan</b> ejemplar: <b>Persona</b> edad: 18 estatura: 170</p>
--

- **Sistemas de marcos**
  - Las bases de conocimiento contienen una colección de marcos
  - Los marcos se conectan entre sí mediante el relleno de los slots
  - Se razona sobre clases de objetos
    - Usando representación del conocimiento prototípico (*cierto en la mayoría de los casos*)
    - Posibilidad de cambiarlo en las instancias (*representar excepciones*)

# Marcos y Sistemas de Marcos

- Los marcos representan particiones sobre el conjunto de hechos
  - Un marco agrupa hechos sobre un mismo objeto o situación
  - Permiten asociar conocimiento procedural relevante a un hecho
  - Idóneos para la organización de una gran cantidad de hechos
  
- Un Sistema de Marcos consta de dos componentes:
  - Base de conocimiento
    - Conjunto de marcos relacionados mediante rellenos de ranuras
    - Distinción de clases e instancias
    - Jerarquía de conceptos con ejemplar/subclase
    - Propiedades y relaciones entre marcos
  - Motor de inferencia
    - Herencia de propiedades, relaciones y procedimientos de cálculo a través de la estructura jerárquica
    - Clasificación de conceptos
    - Equiparación con unos slots de entrada (obtener slots salida)

# Tipos de marcos

## ● Tipos de Marcos

### – Marcos clase

- Representan conceptos, clases, estereotipos, situaciones genéricas
- Ejemplo: Herramientas, Persona, Coche

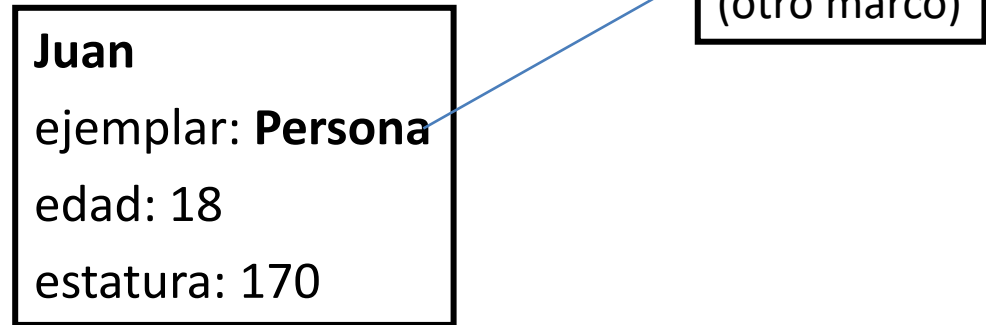
### – Marcos instancia

- Representan conceptos individuales, objetos, entidades, individuos
- Ejemplo: Martillo-1, María, M-6595-K

# Tipos de Marcos: Marcos instancia

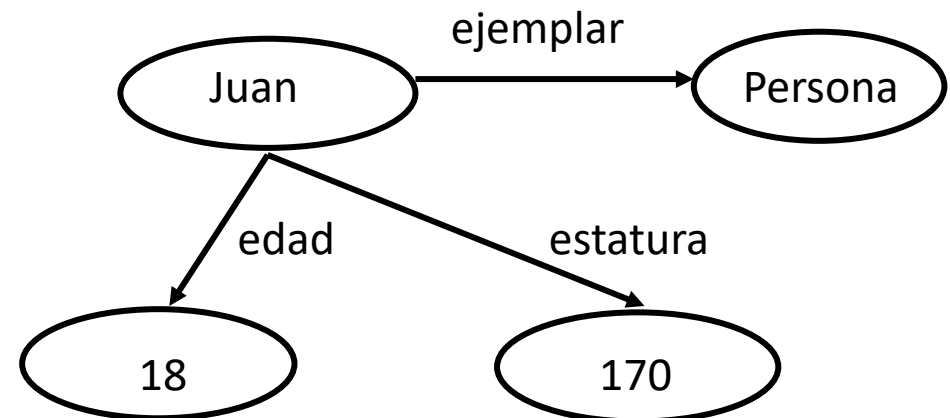
## Marco instancia

- Marco de un individuo con 3 pares atributo/valor (estructuras de ranura/relleno)
- El relleno de una ranura puede ser un enlace a otro marco
  - *Persona* es otro marco con sus propias características
  - 18 y 170 son valores



## Representación en forma de red semántica

- No hay diferencia entre individuos y clases
  - Juan es un individuo y Persona una clase de individuos
- No queda claro donde termina la descripción de una entidad





# Tipos de Marcos: Marcos clase

## ● Marco clase

- Generaliza la información acerca de varios objetos, identificando las propiedades que comparten los elementos del **conjunto**
- Describe un objeto o situación **prototípica** de una clase
- Los marcos tienen **ranuras que pueden rellenarse o no**
  - Si no se rellena los individuos y subclases heredan la existencia de la ranura
  - Si se rellena los individuos y subclases heredan además un valor por defecto que pueden sobrescribir
- **Las ranuras rellenas representan hechos**

### Barco

nombre:

número de identificación:

tipo de barco:

nacionalidad:

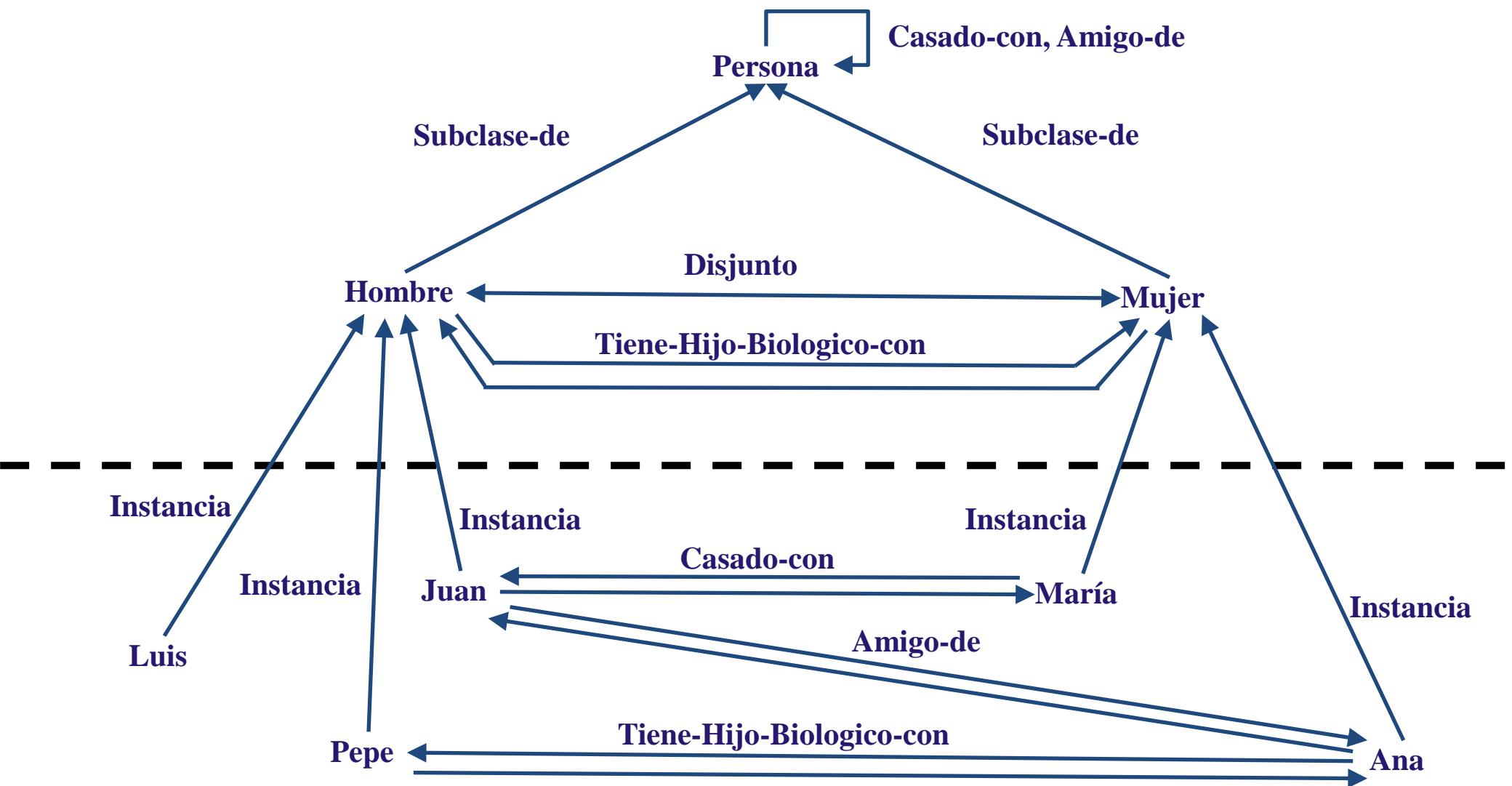
tonelaje:

lugar:

## Atributos: relaciones entre Marcos

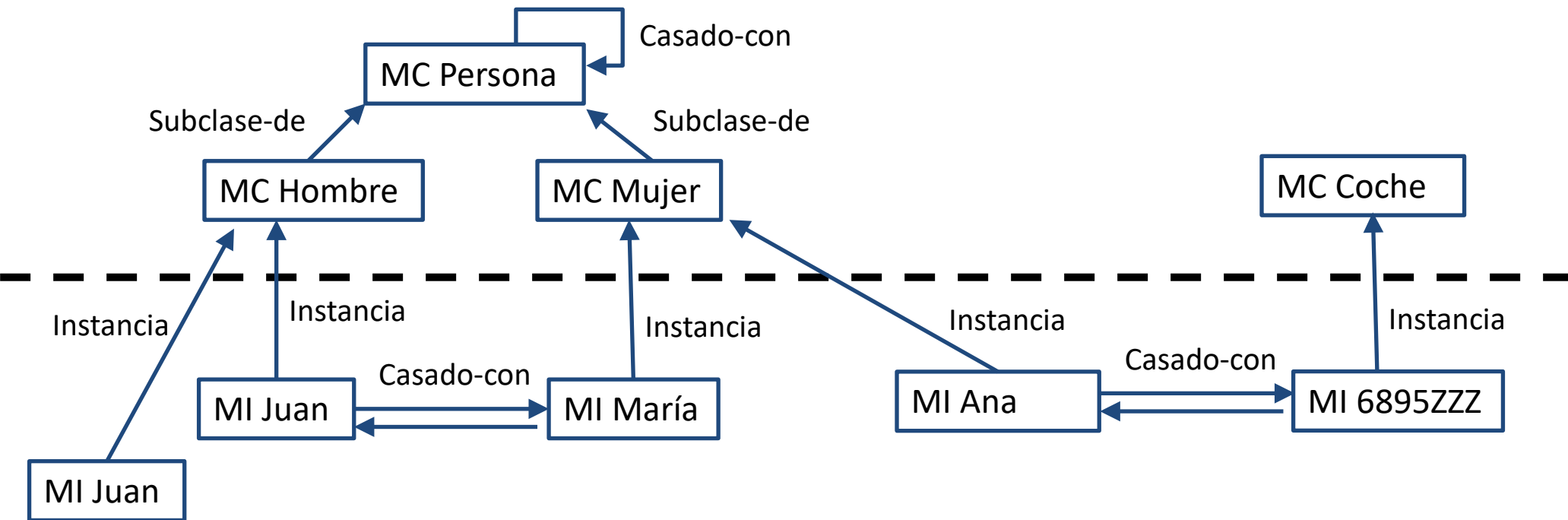
- **Relaciones estándar** (*forman jerarquías*)
  - *Subclase* y su inversa *Superclase*
  - *Ejemplar* o *Instancia* y su inversa *Contiene*
  - Se suelen manejar inversas de forma automática
- **Relaciones no estándar**
  - Disjunto/No Disjunto
  - “a medida” o “ad hoc” (relaciones dependientes del dominio)
    - amigo\_de, casado\_con, hijo\_de
  - Las inversas hay que añadirlas

# Ejemplo de Jerarquía de marcos



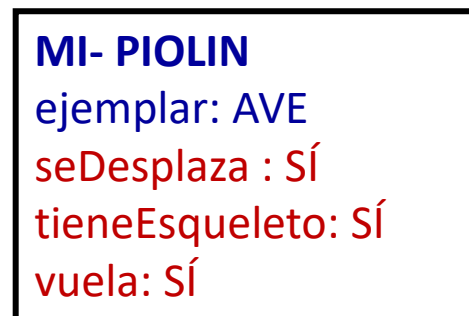
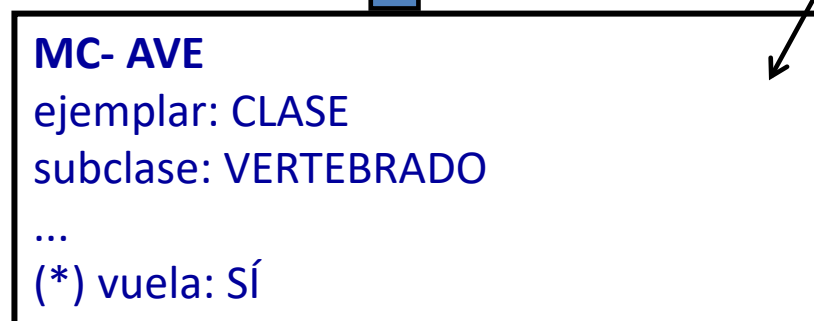
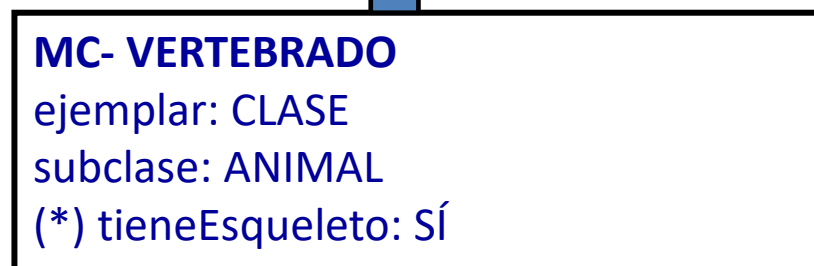
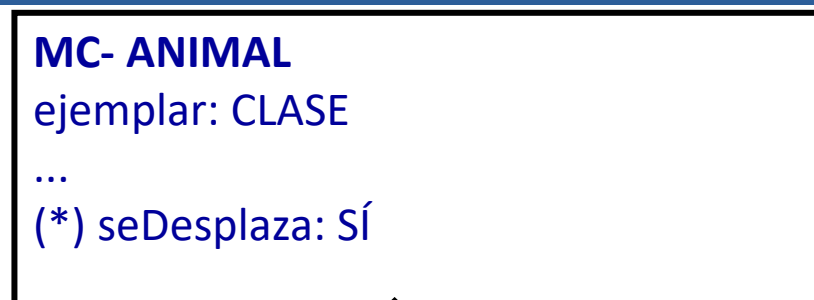
# Ejemplo de Relaciones entre marcos

- Las relaciones se definen entre marcos clase
  - Los marcos instancia son ejemplares de dichos marcos clase



Si el sistema hace comprobación de consistencia daría un error en la relación *Casado-con* entre *Ana* y *6895ZZZ* porque no se cumple la restricción de rango en la definición de la propiedad

# Ejemplo de Clases e instancias



atributos heredados automáticamente

❑ Aunque parece que la herencia sólo funciona a un nivel no es así porque **la relación subclase es transitiva**

$x \in \text{Ave} \rightarrow x \in \text{Vertebrado}$   
 $\rightarrow x \in \text{Animal}$

❑ Cualquier individuo hereda de todas sus superclases (superconceptos)

(\*): atributos heredables por los individuos pertenecientes a la clase

**CLASE**: palabra reservada que indica que se representa a un conjunto

# Atributos de clase vs Atributos de instancia

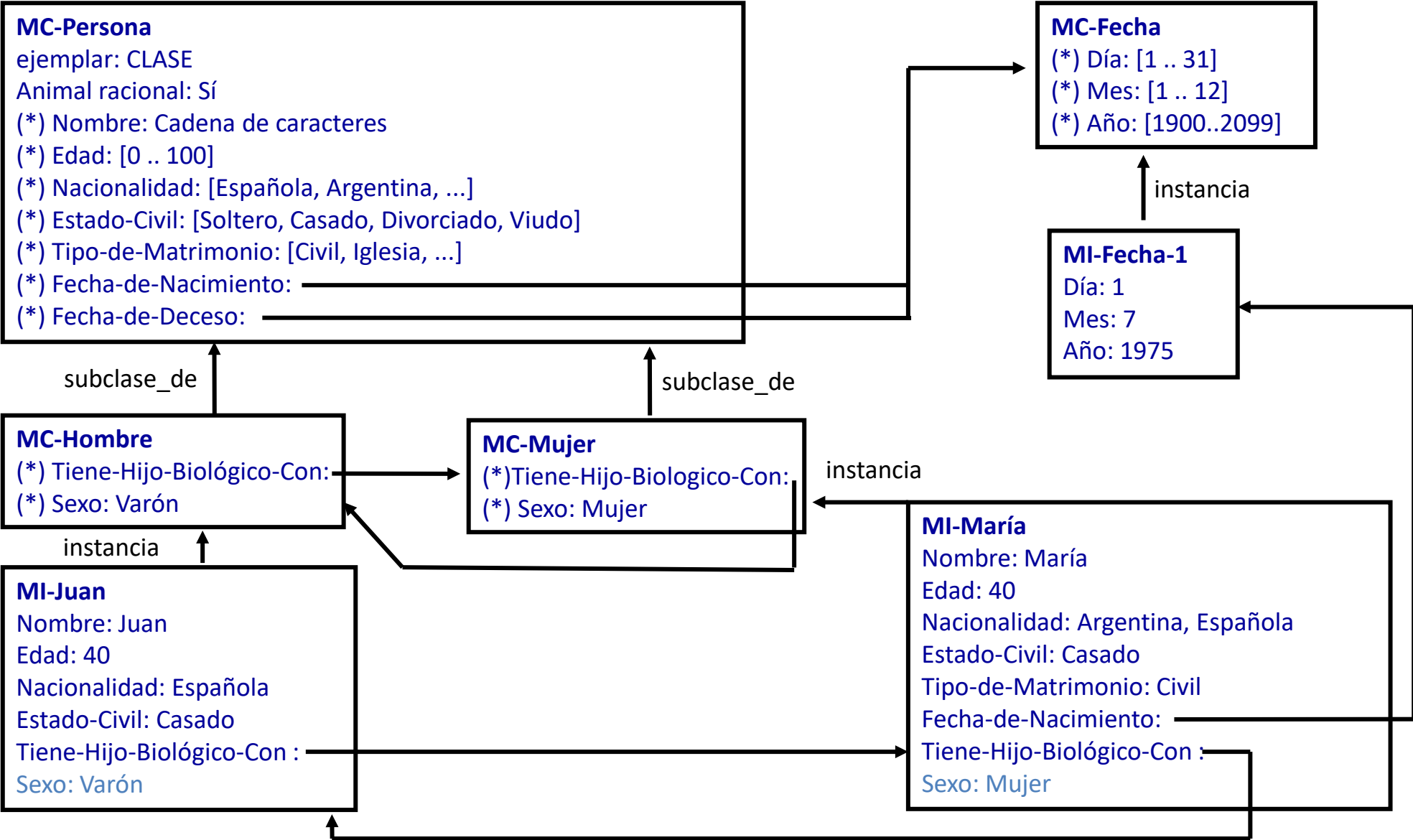
## ● Atributos de clase

- Son atributos referidos a la clase (o concepto)
- Se *definen y rellenan* en el marco clase
- No son heredables por las instancias
  - Ejemplo: cardinalidad

## ● Atributos de instancia

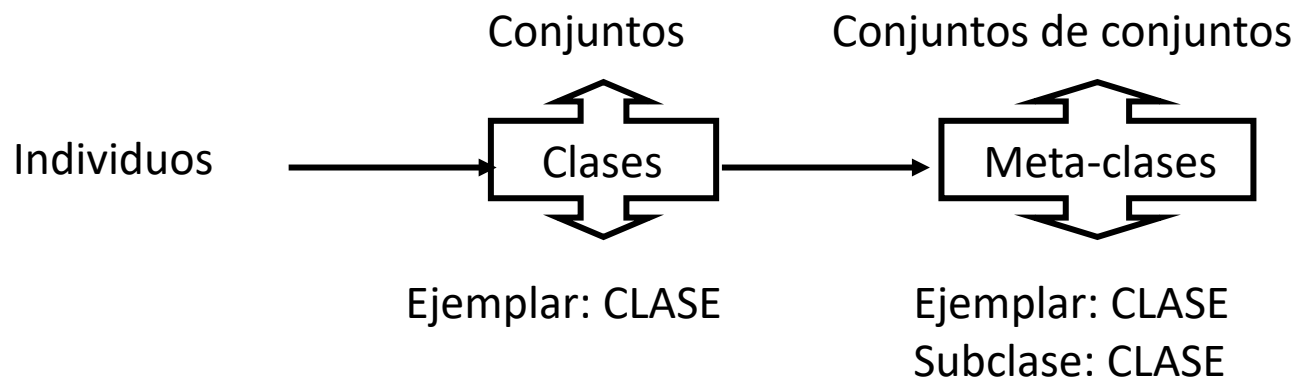
- Son atributos con valores distintos en cada instancia
- Se *definen* en el marco clase
- Si se *rellenan en el marco clase*, todas las instancias heredan su valor (**herencia de valores**)
  - **Valores prototípicos: especialmente útil para dominios con mucho conocimiento por defecto.**
  - Pueden ser redefinidos en las instancias y subclases
- Si se *rellenan en el marco instancia*, lo único que se hereda del marco clase es la existencia de la ranura (**herencia de ranuras**)
- Precedidas del símbolo (\*)

# Ejemplo de atributos



# Meta-clases

- Conjuntos de conjuntos
- Las instancias de una meta-clase son a su vez clases
- La manera de caracterizar a las meta-clases es
  - Son ejemplares de CLASE  
(como las clases regulares = conjuntos de individuos)
  - Son subclases de CLASE  
(esto hace que sus instancias sean también clases)





# Ejemplo de Meta-clases

**CLASE**  
ejemplar: CLASE  
subclase: CLASE  
...  
(\*) cardinalidad:

A diferencia de las redes semánticas, podemos definir atributos sin valor en las clases. El valor se rellenará en alguna subclase o en las instancias

**ESPECIE PROTEGIDA**  
ejemplar: CLASE  
subclase: CLASE  
cardinalidad: 300  
(\*) estudiada-por: NATURALISTAS

$\forall x \in \text{ESPECIE PROTEGIDA}$   
 $\rightarrow x \in \text{CLASE}$

**BALLENA AZUL**  
ejemplar: ESPECIE PROTEGIDA  
subclase: CETÁCEO  
origen: CRETÁCICO  
estudiada-por: NATURALISTAS  
(\*) alimentación: CRUSTÁCEOS  
(\*) tamaño: 30m  
(\*) velocidad media: 12 nudos  
(\*) peso:

Características propias de la especie; no de los individuos

Heredables porque son características de los individuos

**WILLY**  
ejemplar: BALLENA AZUL  
ejemplar: ANIMAL EN CAUTIVIDAD  
alimentación: CRUSTÁCEOS  
tamaño: 20m  
velocidad media: 12 nudos  
peso: 80 toneladas  
comportamiento: AGRESIVO  
año de nacimiento: 2008  
edad:

redefinido

heredado

damos valor

heredado de ANIMAL EN CAUTIVIDAD

Atributo propio del individuo

Procedimiento de cálculo asociado

BALLENA AZUL es una clase regular:  
sus elementos son individuos, no conjuntos

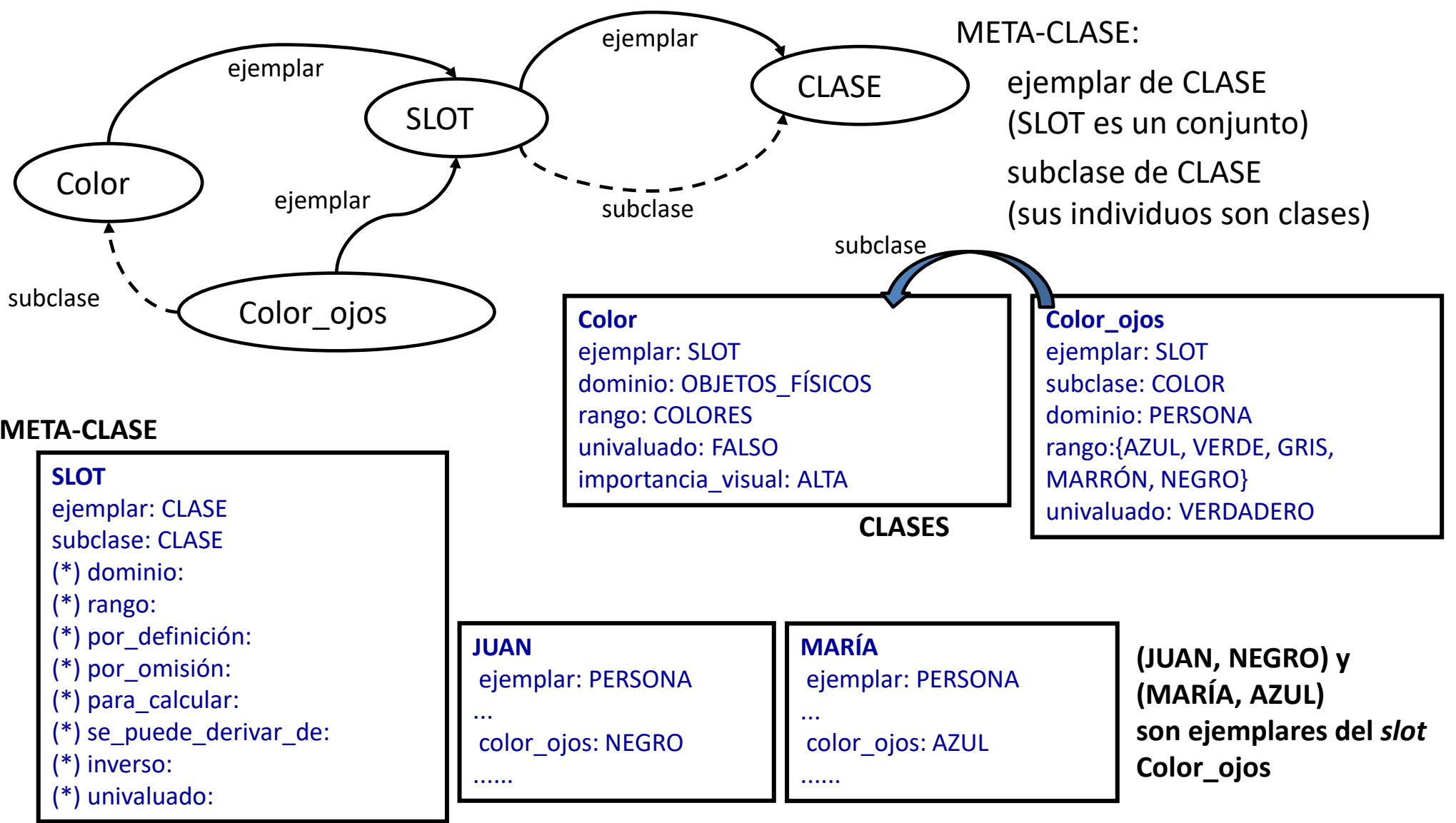
# Representación de atributos como marcos

- Podemos describir semánticamente las relaciones no estándar (específicas del dominio) representándolas como marcos
- Cada atributo (**ranura**) puede ser descrito por una serie de ranuras que se suelen denominar **facetas**:
  - **Dominio**: clases para las que se define ese atributo
  - **Rango**: posibles valores que puede tener
  - **Valor obligatorio** (*por definición*): *no puede estar vacío*
  - **Valor por omisión**: en caso de no asignar valor usa ese
  - **Reglas de herencia**: indica cuándo y qué heredar
  - **Reglas o procedimientos para calcular valores de relleno**
  - **Relaciones Inversas**
  - **Univaluado/multivaluado**: un slot con uno o varios valores
- Así representamos meta-conocimiento
  - Restricciones sobre el conocimiento a representar en los marcos
  - Los sistemas que permiten la representación de *slots* mediante marcos suelen tener restricciones sobre las ranuras definibles (facetas)

## Jerarquías de atributos

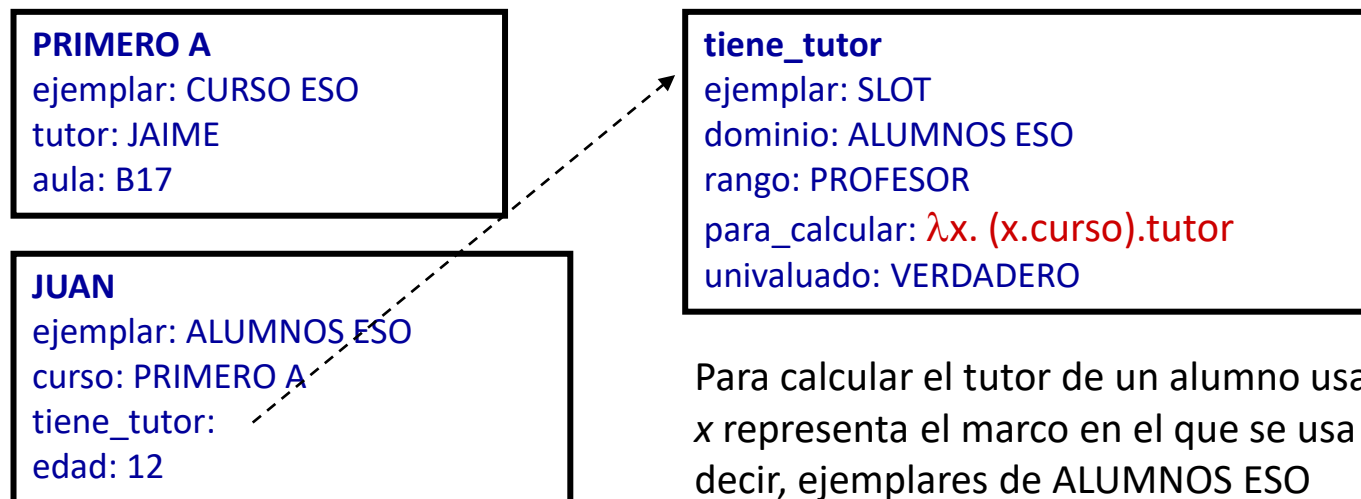
- Se pueden definir jerarquías de atributos además de jerarquías de clases
- Un atributo (ranura o *slot*) es una relación entre los elementos del dominio (las clases para las que tiene sentido) y los elementos de su rango (posibles valores)
  - Un *slot* es el conjunto de pares ordenados que cumplen esa relación
  - Atributos como conjuntos de pares:  $\{(x, y), (z, u), \dots\}$
- Un *slot*  $S1$  puede ser un subconjunto (subclase) de un *slot*  $S2$ 
  - Por ejemplo, color de ojos  $\subseteq$  color
- La relación de inclusión nos permite crear jerarquías
- Al conjunto de todos los *slots* lo denominamos SLOT (es una meta-clase)
- Las jerarquías de *slots* suelen ser bastante planas

# Jerarquías de atributos (slots)



## Valores calculados (o activos)

- Mecanismo general: se representa en el marco del *slot*

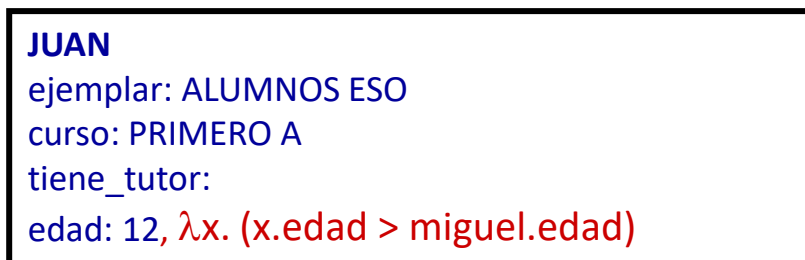


Para calcular el tutor de un alumno usamos  $\lambda$ -cálculo:  
 $x$  representa el marco en el que se usa el *slot* **tiene\_tutor**, es decir, ejemplares de ALUMNOS ESO

Por ejemplo:

$x \longrightarrow x.curso \longrightarrow (x.curso).tutor$   
 JUAN                  PRIMERO A                  JAIME

- Las restricciones particulares de un *slot* para un ejemplar particular se representan en el marco del ejemplar

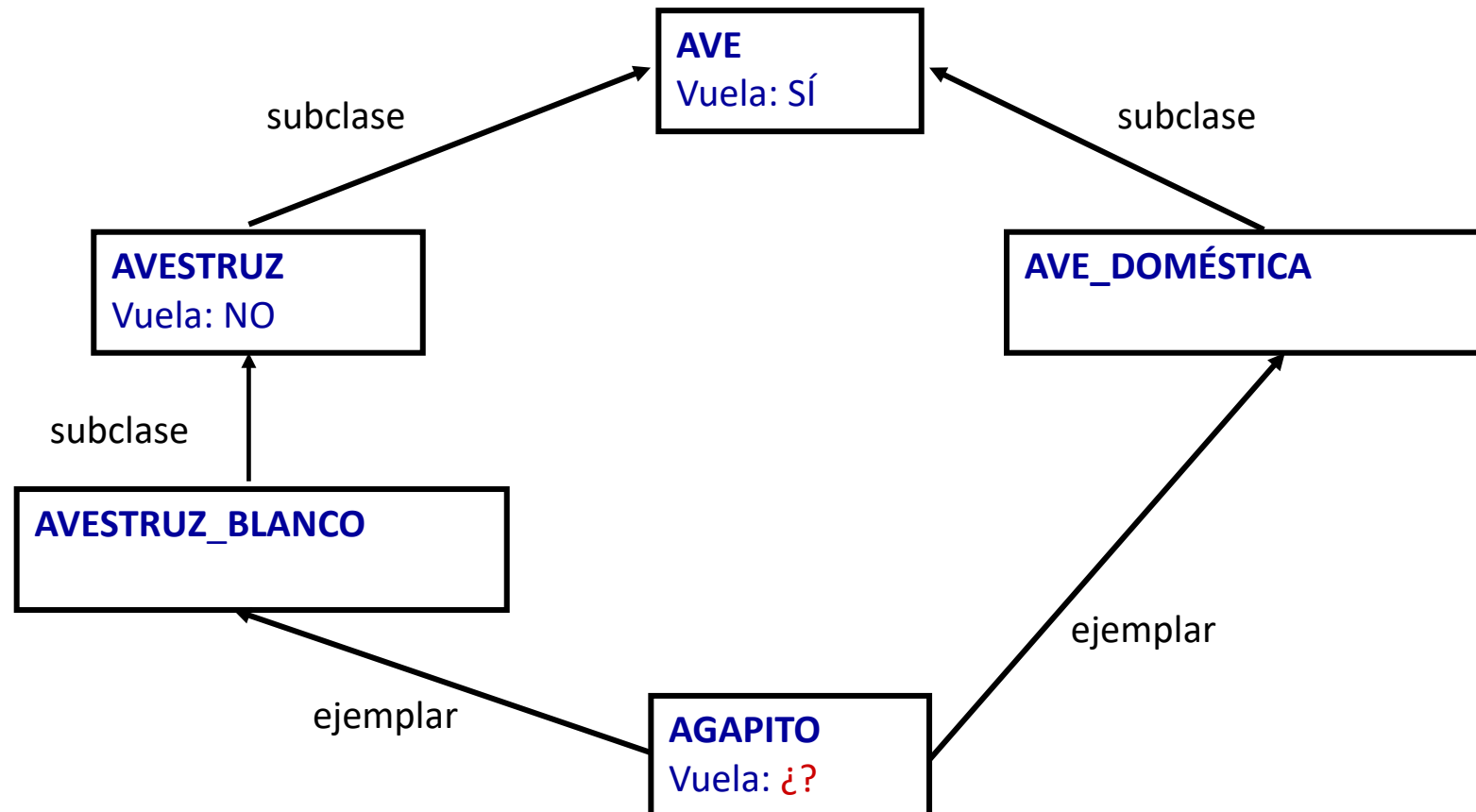


## Conocimiento sobre atributos

- La representación de **meta-conocimiento** sobre los *slots* permite a los sistemas
  - Realizar control de consistencia en el dominio y el rango de los atributos
  - Mantener la consistencia entre un atributo y su inverso cuando se cambia uno de ellos
  - Propagar los valores por definición y por omisión a través de la jerarquía de herencia (*ejemplar y subclase*)
  - Calcular el valor de un atributo cuando se necesita (*para\_calcular, se\_puede\_derivar\_de*)
  - Controlar los atributos univaluados

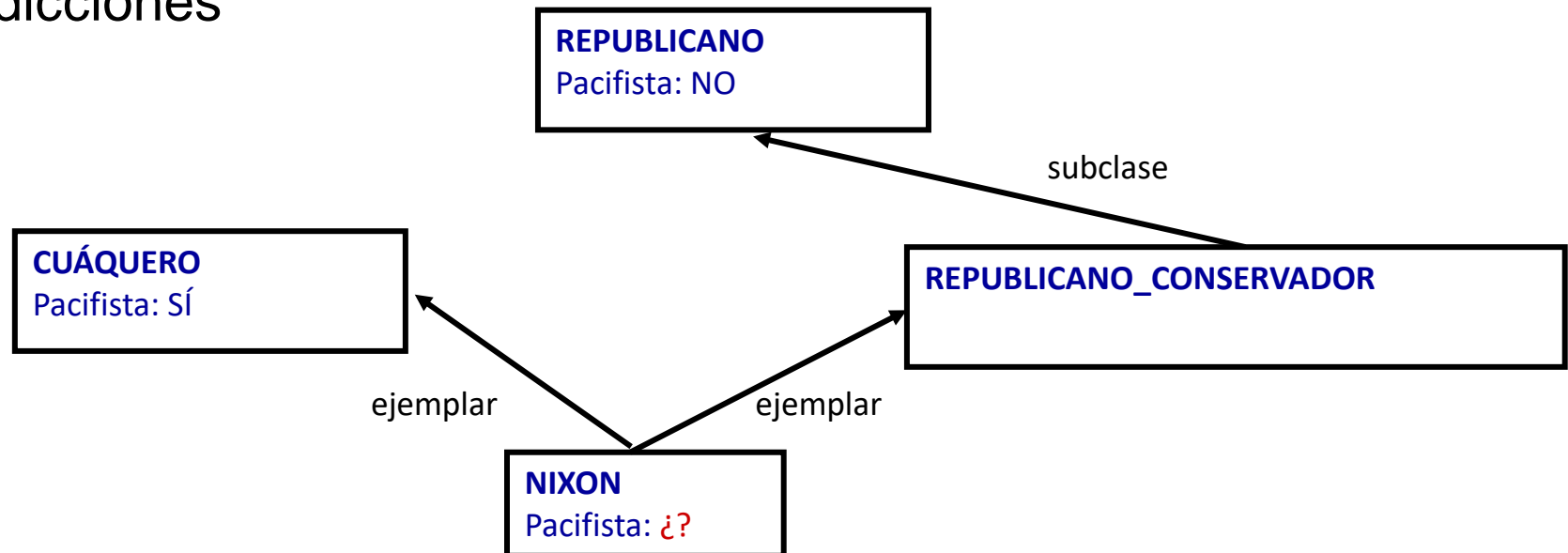
# Herencia múltiple

- Jerarquías: grafos dirigidos acíclicos, en lugar de árboles
- Distintos antepasados pueden tener distintos valores de los atributos



# Distancia inferencial [Touretzky, 1986]

- Define un orden parcial:
  - Concepto1 está más cerca de Concepto2 que de Concepto3  
**si y sólo si** Concepto1 tiene un camino de inferencia a través de Concepto2 hasta Concepto3 (es decir, Concepto2 está entre Concepto1 y Concepto3)  
$$\text{distancia}(\text{Concepto1}, \text{Concepto2}) < \text{distancia}(\text{Concepto1}, \text{Concepto3}) \Leftrightarrow \exists \text{camino}(\text{Concepto1}, \text{Concepto2}, \text{Concepto3})$$
- La distancia inferencial no siempre es aplicable → permitirá detectar contradicciones





# Herencia de propiedades: algoritmo

- Para obtener el valor desconocido  $V$  de un atributo  $A$  en una instancia  $I$ 
  - $CANDIDATOS := \emptyset$
  - Búsqueda 1º en profundidad en la jerarquía a partir de  $I$  de todos los superconceptos  $SC$  (en orden ascendente)
    - Si en  $SC$  se encuentra un valor para  $A$  se añade a  $CANDIDATOS$  y se finaliza con esa rama
    - Si en  $SC$  no se encuentra ningún valor, ascendemos otro nivel. Si no hay más niveles, terminamos con esa rama
  - Para cada elemento  $C$  de  $CANDIDATOS$ :
    - Si existe algún otro elemento de  $CANDIDATOS$  que ha sido obtenido de un concepto que esté a menor distancia inferencial de  $I$  que el concepto del que se ha obtenido  $C$ , entonces sacar  $C$  del conjunto de  $CANDIDATOS$
  - Si el cardinal de  $CANDIDATOS$  es:
    - 0: no se ha obtenido ningún valor
    - 1: se devuelve el único elemento de  $CANDIDATOS$  como  $V$
    - $>1$  y todos sus elementos son iguales: devolver el valor como  $V$
    - $>1$  y elementos distintos: informar de que hay una contradicción

# Ventajas de los marcos

- Facilitan el **razonamiento basado en expectativas**
  - Un *slot* es un lugar donde se espera un cierto tipo de valor dentro del contexto de un marco
  - Proporcionando un lugar para el conocimiento, se crea la **posibilidad del conocimiento incompleto o inexistente**, permitiendo el razonamiento basado en intentar confirmar expectativas
  - Se ha aplicado en sistemas de comprensión del lenguaje natural
- Posibilidad de **asociar procedimientos de cálculo a los atributos**
  - Mecanismo hacia atrás que permite rellenar atributos “cuando se necesita” (el procedimiento “enganchado” al *slot* se dispara al preguntar por su valor)
  - Mecanismo hacia delante para rellenar atributos “cuando se añade” (cuando se rellena un *slot*, todos los *slots* de otros marcos que dependan de él se rellenan automáticamente)
- **Representación estructurada del conocimiento**, incluso en el caso del conocimiento procedimental
  - La fase de equiparación o *matching* para determinar qué procedimiento o regla aplicar se realiza aquí mediante un proceso de clasificación

# Ontologías

# Ontologías: contenidos

- Reutilizar bases de conocimiento
- Ontologías para modelar el vocabulario de un dominio
- Diseño y desarrollo de ontologías
- OWL (Web Ontology Language)
  - Clases, propiedades, individuos
  - Axiomas
  - Razonamiento
- Protege
- Ventajas e inconvenientes

# Reutilizar bases de conocimiento

- El desarrollo de Bases de Conocimiento (BC) es algo muy costoso.
  - Obtener la información de los expertos del dominio
  - Elegir un formalismo de representación adecuado (expresividad vs eficiencia)
  - Modelar el conocimiento del dominio en ese formalismo
  - Revisar y mantener las bases de conocimiento
- Necesitamos **reutilizar y combinar bases de conocimiento** ya creadas
  - Vocabularios comunes y bien definidos
  - Conocimiento factual: clases, instancias, propiedades
- Disponer de estas bases de conocimiento interoperables permite la construcción de nuevos sistemas basados en conocimiento
- Distintas iniciativas
  - Knowledge Sharing Effort, DARPA 1991
  - Web Semántica, Berners-Lee 1999 (web anotada semánticamente para las máquinas)

## Reutilizar bases de conocimiento

- “Building new Knowledge Based Systems today usually entails constructing new knowledge bases from scratch. It could instead be done by assembling **reusable components**. System developers would then only need to worry about creating the specialized knowledge and reasoners new to the specific task of their systems. This new system would interoperate with existing systems, using them to perform some of its reasoning. In this way, **declarative knowledge, problem-solving techniques, and reasoning services could all be shared between systems**. This approach would facilitate building bigger and better systems cheaply. The infrastructure to support such sharing and reuse would lead to greater ubiquity of these systems, potentially transforming the knowledge industry ...”

Neches, R.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.; Swartout, W.R.  
*Enabling Technology for Knowledge Sharing. AI Magazine. Winter 1991. 36-56.*

# ¿Qué es una ontología?

## ● RAE

- Parte de la metafísica que trata del ser en general y de sus propiedades trascendentales

## ● Informática

- **“An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary”**

Neches, R.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.; Swartout, W.R. *Enabling Technology for Knowledge Sharing*. **AI Magazine**. Winter 1991. 36-56.

- Especificaciones formales explícitas de los términos de un dominio y de las relaciones entre ellos (Gruber, 1993)
- Un sistema particular de categorías sistematizando cierta visión del mundo (Guarino, 1998)
- Una ontología es una especificación formal de una conceptualización compartida (Studer y otros, 1998)

## ¿Para qué se desarrolla una ontología?

- Formalizar explícitamente el vocabulario de un dominio (términos y relaciones)
  - Crear vocabularios estándar para representar conocimiento
  - Hacer explícitas las suposiciones sobre un dominio
  - Analizar el conocimiento de un dominio
- Permitir la construcción colaborativa e incremental de bases de conocimiento
- Permitir la reutilización de bases de conocimiento
- Establecer una interpretación compartida de la estructura de un dominio entre personas y programas
- Separar el conocimiento del dominio declarativo del conocimiento procedimental



# Componentes de una ontología

- **Clases** o conceptos
  - Definen los tipos o clases del dominio
  - Definiciones formales mediante relaciones con otras clases
- **Propiedades** o **roles** o atributos
  - Definen los tipos de relaciones entre entidades
  - Se definen mediante propiedades matemáticas (reflexividad, transitividad, ...) y restricciones sobre los valores relacionados (facetas)
- **Instancias** o individuos
  - Elementos particular de nuestro dominio
  - Se les asignan tipos mediante clases y se relacionan con otras instancias mediante propiedades
- A veces se utiliza el término **Ontología** sólo para referirse a la parte terminológica (clases y propiedades)
  - **Base de conocimiento = ontología + instancias**
- Evolución de los sistemas de Marcos con una semántica formal bien definida (respaldados por una lógica formal)

# Diseño y desarrollo de ontologías

- No hay un único modo correcto de construir una ontología
  - Depende del uso se le vaya a dar
- Los conceptos en la ontología deben ser próximos a las entidades del dominio
  - Extraer de las frases que describen el dominio (Clases ~ nombres, Propiedades ~ verbos)
- Proceso iterativo
  1. Determinar el dominio y el alcance de la ontología
  2. Considerar la posibilidad de reutilizar otras ontologías
  3. Enumerar los términos importantes
  4. Definir las clases y la jerarquía de clases
  5. Definir las propiedades de las clases
  6. Definir las facetas de las propiedades
  7. Crear instancias

**Ontology Development 101: A Guide to Creating Your First Ontology.**

N. F. Noy & D. L. McGuinness, 2001

## Consejos generales para crear una ontología

- ¿Pará qué se va a utilizar? ¿Cuánto conocimiento debe contener?
  - ¿Qué tipos de inferencia voy a realizar?
- No incluir versiones singulares y plurales de un mismo término.
- Puede haber varios nombres que identifiquen a un mismo concepto pero, en ese caso, debería haber una única clase.
- Comprobar si las relaciones de herencia están adecuadamente establecidas. Evitar ciclos en las jerarquías.
- Todas las subclases de una clase deben estar a un nivel similar de generalidad.
- Comprobar si las propiedades están adecuadamente establecidas (transitividad, inversas, dominios, rangos, ...)

# Lenguajes para representar ontologías

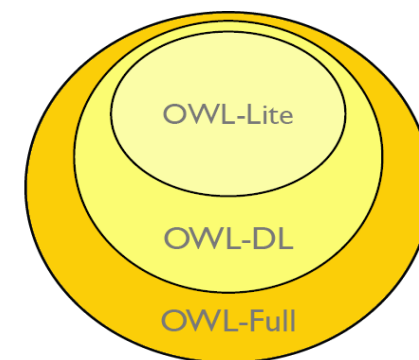
- Lenguajes basados en alguno de los paradigmas de representación de conocimiento clásicos (DLs, marcos o lógica de predicados):
  - EXPRESS [Spibey91]
  - CML el lenguaje de modelado de CommonKADS [Schreiber et al. 94]
  - Ontolingua [Gruber93]
    - Estándar de lenguaje de especificación de ontologías
    - Es una extensión de KIF y Frame Ontology [Gruber93]
    - No tiene un motor de inferencias asociado
  - KIF (Knowledge Interchange Format) [Genesereth&Fikes92]
  - CycL[Lenat&Guha90]
  - FLogic [Kifer et al. 95]
  - Loom [MacGregor&Bates87] [MacGregor88]
- Lenguajes de la Web Semántica para representar e intercambiar ontologías
  - RDFS, DAML-OIL, OWL

# Lenguajes de la Web Semántica

- RDF
  - Representación de Redes Semánticas mediante tripletas
- RDFS
  - Permite representar ontologías básicas (taxonomías)
    - Clases (subclases, superclases)
    - Propiedades (rango y dominio locales)
  - Tiene muchas limitaciones de expresividad
    - Restricciones de existencia y cardinalidad
    - Propiedades transitivas, inversas, simétricas, etc.
    - Definiciones de clases complejas mediante conectivas lógicas: and, or, not, ...
  - Semántica no estándar
- OWL
  - Estándar para representar ontologías

# OWL (Web Ontology Language)

- OWL es realmente una familia de 3 lenguajes:
  - **OWL Lite**: jerarquías de clases y restricciones simples
  - **OWL DL**: máxima expresividad manteniendo la completitud computacional
  - **OWL Full**: expresividad máxima y la libertad sintáctica de RDF sin garantía de completitud en los razonadores
- OWL Lite y OWL DL están basados en Lógicas Descriptivas (DL)
  - Subconjuntos decidibles de la Lógica de Primer Orden
  - Las DL son una familia de lenguajes
  - Nosotros nos vamos a centrar en OWL DL
    - Bastante expresivo
    - Procesos de inferencia pueden ser exponenciales



# Base de conocimiento OWL

- Una base de conocimiento se compone de
  - Parte **terminológica** o TBox
    - Definición de clases y propiedades
  - Parte **asertiva** o Abox
    - Definición de individuos
- Subconjunto de la LPO
  - Las clases o conceptos son predicados unarios
  - Los roles o propiedades son predicados binarios
  - Los individuos son constantes
- Sean  $N_C$ ,  $N_R$  y  $N_I$  conjuntos disjuntos de nombres de clases atómicas, roles e individuos. La tupla  $(N_C, N_R, N_I)$  se conoce como **signatura**.
  - $N_C = \{\text{Perro, Gato, ...}\}$        $N_R = \{\text{hasPet, livesAt, ...}\}$   
 $N_I = \{\text{Snoopy, Garfield}\}$

# Clases

Constructor	Sintaxis	Descripción
T (top)	Thing	Cualquier entidad
⊥ (bottom)	Nothing	Conjunto vacío (Clase insatisfactible)
Clases primitivas	Dog, Cat, Friendly	$A \in N_c$
AND	Dog <b>and</b> Friendly	Perros que son amigables
OR	Dog <b>or</b> Cat	Tanto los perros como los gatos
NOT	<b>not</b> Friendly	Entidades no amigables
ONE OF	{monday, tuesday, ..., sunday}	Días de la semana (enumerado)
EXISTS	hasPet <b>some</b> Dog	Entidades que tienen perros como mascota
FOR ALL	hasPet <b>only</b> Dog	Entidades que sólo tienen perros como mascota
MIN CARD	hasPet <b>min</b> 2	Entidades que tienen al menos 2 mascotas
MAX CARD	hasPet <b>max</b> 3	Entidades que tienen como máximo 3 mascotas
CARDINALITY	hasPet <b>exactly</b> 1	Entidades que tienen exactamente una mascota



## Más ejemplos de clases

- Person **and** hasPet **some** Dog
  - Personas que tienen alguna mascota que es un perro (pueden tener más mascotas perros o no perros)
- Person **and** (Friendly **or** Scared)
  - Personas amigables o asustadas
- Person **and not** Friendly
  - Personas que no son amigables
- Person **and** hasPet **exactly** 1 **and** hasPet **some** Dog
  - Personas que sólo tienen una mascota que es un perro
- Person **and** hasPet **min** 1 **and** hasPet **max** 2
  - Personas que tienen una o dos mascotas

## Más ejemplos de clases

- Person **and** hasPet **min 2 and** hasPet **some** Cat
  - Personas que tienen al menos dos mascotas y una de ellas es un gato (puede que las demás también sean gatos o no)
- Person **and** hasPet **some** (Dog **or** Cat)
  - Personas que tienen una mascota que es un perro o un gato (pero pueden tener más o incluso tener un perro y un gato)
- Person **and not** Friendly **and** hasPet **only** (Furry **and** Friendly)
  - Personas no amigables que sólo tienen mascotas peludas y amigables (pero pueden no tener ninguna mascota)
- Dog **and not** Animal
  - Si Dog es un subtipo de Animal sería insatisfactible (equivalente a Nothing)

# Axiomas de clases

Axioma	Sintaxis
Subclass	Dog <b>subclassof</b> Animal
Equivalent	Friendly <b>equivalent</b> hasFriend <b>some</b> Animal
Disjoint	<b>disjoint</b> (Dog, Cat, Person)

- En realidad sólo necesitamos el axioma de subclase
  - Pero los usaremos por comodidad
- La equivalencia es azúcar sintáctico
  - A **equivalent** B es lo mismo que A **subclassof** B and B **subclassof** A
- Las clases disjuntas también son azúcar sintáctico
  - disjoint**(Dog, Cat) es lo mismo que Dog **and** Cat **equivalent** Nothing
  - disjoint**(A, B, C, ...) indica que las clases son disjuntas 2 a 2

# Tipos de clases

## ● Clases Primitivas

- Establecen **condiciones necesarias**
- Por ejemplo: un perro es un tipo de animal
  - Dog **subclassof** Animal
- Definen subconjuntos

## ● Clases Definidas

- Establecen **condiciones necesarias y suficientes**
- Permiten definir nombres equivalentes para clases
- Por ejemplo: las personas amigables tienen al menos un amigo
  - FriendlyPerson **equivalent** Person **and** hasFriend **some** Person
- Si algo cumple las condiciones de una clase definida, será clasificado debajo de dicha clase
  - Cualquier persona que tenga algún amigo será clasificado como FriendlyPerson (y como Person, y como Friendly, ...)

# Axiomas de propiedades o roles

Axioma	Sintaxis
Subproperty	hasElectronicDevice <b>subpropertyof</b> hasDevice
Equivalent	hasEletronicDevice <b>equivalent</b> hasGatget

- Podemos definir jerarquías de propiedades
  - Considerando cada propiedad como el conjunto de pares de individuos relacionados
- Podemos definir las siguientes facetas
  - Funcional / Inversa funcional
  - Transitiva
  - Simétrica / Asimétrica
  - Reflexiva / Irreflexiva

# Axiomas de individuos

Axioma	Sintaxis
Concept assertion	Dog(snoopy)
Role assertion	hasFriend(juan, snoopy)
Same	isabel = isa
Different	juan $\neq$ isabel

- Al asertar que un individuo es instancia de una clase se infiere todo lo definido en dicha clase
  - Dog(snoopy)  $\rightarrow$  Animal(snoopy)
- Sólo las clases definidas pueden clasificar automáticamente individuos que no están asertados en su jerarquía
  - hasFriend(juan, isabel)  $\rightarrow$  Friendly(juan)
- No se asume que dos individuos sean diferentes si no se indica explícitamente (o se infiere de algún modo)
  - ¿Isabel = snoopy?

# Razonamiento en mundo abierto

- Razonamiento de **mundo cerrado**
  - Todo lo que no se conoce se asume falso
  - Se usa en sistemas de bases de datos, en programación lógica, lenguajes de restricciones, ....
- Razonamiento de **mundo abierto**
  - No se asume nada de lo que no se conoce
  - Se usa en los demostradores automáticos de teoremas, en los razonadores de DLs y en OWL
- **En OWL**
  - **usamos razonamiento en mundo abierto**
  - **distintos nombres no indican que sean entidades diferentes**
  - Necesario para integrar distintas ontologías
  - Ninguna inferencia queda invalidada si llega conocimiento nuevo (consistente con lo que ya sabíamos)
  - Es necesario añadir mucho conocimiento “de cierre” para que se realicen las inferencias que esperamos

# Ejemplo de clasificación

- Base de conocimiento:
  - Cat **subclassof** Animal
  - Dog **subclassof** Animal
  - Pet **subclassof** Cat **or** Dog
  - Person **subclassof** Animal
  - Friendly **equivalent** hasFriend **some** Animal
  - FriendlyPerson **equivalent** Person **and** hasFriend **some** Person
  - PetPerson **equivalent** Person **and** hasPet **some** Pet
  - PetPerson2 **equivalent** hasPet **some** Thing
  - hasPet: Person → Pet
- ¿Cuáles de las siguientes inferencias se pueden realizar?

FriendlyPerson <b>subclassof</b> Animal	✓	FriendlyPerson <b>subclassof</b> Friendly	✓
Pet <b>subclassof</b> Cat	✗	Cat <b>subclassof</b> Pet	✗
Animal <b>equivalent</b> Cat <b>or</b> Dog <b>or</b> Person	✗	PetPerson <b>equivalent</b> PetPerson2	✓



# Ejemplo de clasificación

## Base de conocimiento:

Cat **subclassof** Animal

Dog **subclassof** Animal

Pet **subclassof** Cat **or** Dog

Person **subclassof** Animal

Friendly **equivalent** hasFriend **some** Animal

FiendlyPerson **equivalent** Person **and** hasFriend **some** Person

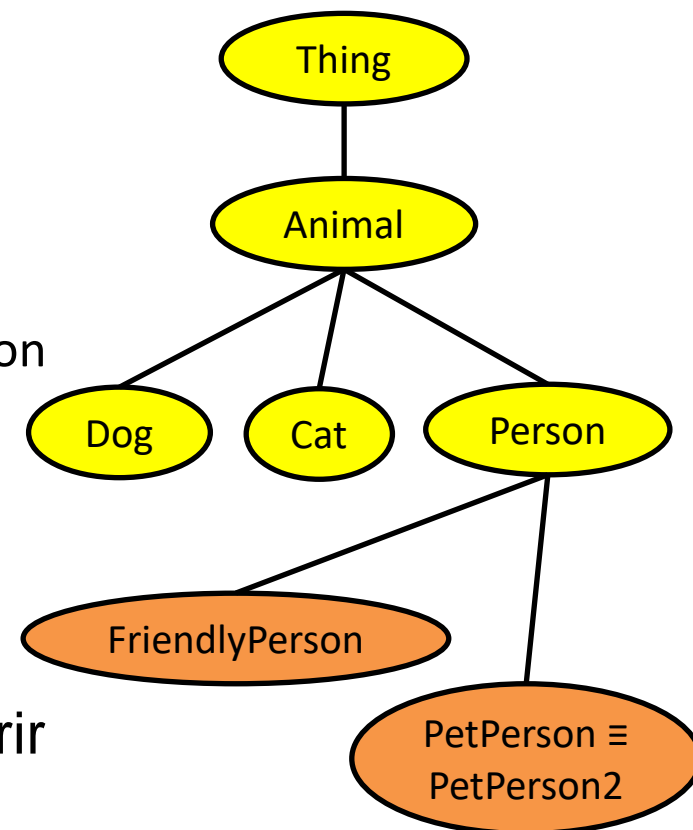
PetPerson **equivalent** Person **and** hasPet **some** Pet

PetPerson2 **equivalent** hasPet **some** Thing

hasPet: Person  $\rightarrow$  Pet

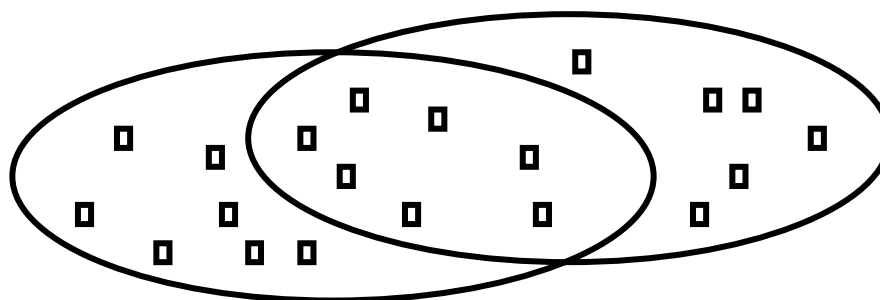
La jerarquía de clases primitivas se aserta

La jerarquía de clases definidas se puede inferir



## Clases disjuntas

- Dos clases A y B son disjuntas si ningún individuo puede ser instancia de A y de B a la vez
- En OWL las clases pueden solaparse (tener individuos comunes) a no ser que se diga explícitamente que son disjuntas



- Las siguiente base de conocimiento no es inconsistente (!)
  - Cat(pelusa), Dog(pelusa)
- Pero así sí sería inconsistente
  - Cat(pelusa), Dog(pelusa), **disjoint**(Cat, Dog)

## Particiones de valores

- Usamos particiones de valores para indicar que un conjunto es la unión de varios conjuntos disjuntos

Spiciness **equivalent** Mild **or** Medium **or** Hot  
**disjoint** (Mild, Medium, Hot)

- Ahora podemos deducir que si algo es de tipo *Spiciness* pero no es *Mild* ni *Medium* entonces debe ser *Hot*

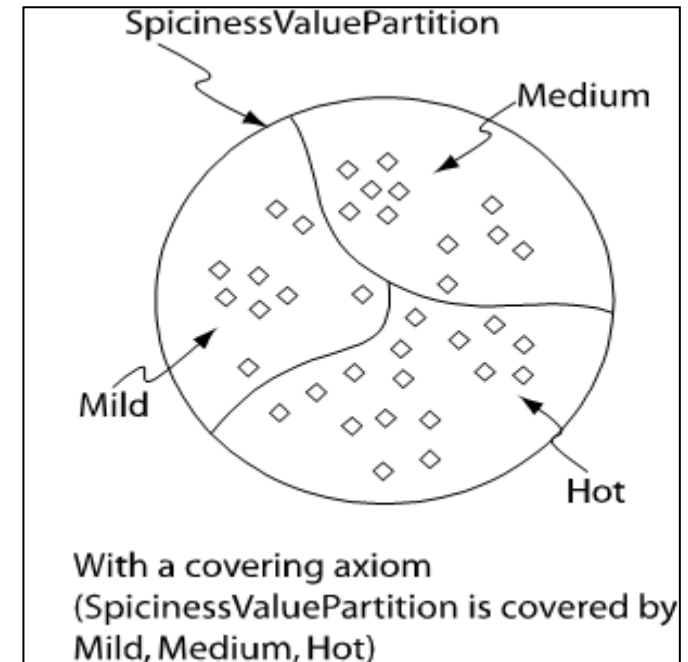
- ¡Ojo!, no es lo mismo que

Mild **subclassof** Spiciness

Medium **subclassof** Spiciness

Hot **subclassof** Spiciness

**disjoint** (Mild, Medium, Hot)



# Clasificación de individuos



## Base de conocimiento:

Cat **subclassof** Animal  
Dog **subclassof** Animal  
Pet **subclassof** Cat **or** Dog  
Person **subclassof** Animal  
Friendly **equivalent** hasFriend **some** Animal  
PetPerson **equivalent** Person **and** hasPet **some** Pet  
VeryPetPerson **equivalent** hasPet **min** 2  
hasPet: Person  $\rightarrow$  Pet  
**disjoint**(Cat, Dog, Person)

Cat(pelusa)  
Dog(leo)  
Pet(nieve)  
Persona(alicia)  
hasPet(alicia, pelusa)  
hasPet(alicia, leo)  
Persona(juan)  
hasFriend(juan, alicia)  
hasPet(juan, leo)  
hasPet(juan, nieve)



## ¿Cuáles de las siguientes inferencias se pueden realizar?

Animal(nieve)	✓	Cat(nieve)	✗
PetPerson(juan)	✓	VeryPetPerson(juan)	✗
Friendly(alicia)	✗	VeryPetPerson(alicia)	✓

# Protégé

Active ontology x

Entities x

Individuals by class x

DL Query x

Annotation properties

Datatypes

Individuals

Classes

Object properties

Data properties

Annotations

Usage

Class hierarchy: CheesyPizza

Inferred

- owl:Thing
  - owl:Nothing
  - DomainThing
    - Country
    - Food
      - Pizza
        - CheesyPizza**
        - InterestingPizza
        - NamedPizza
        - NonVegetarianPizza
        - SpicyPizza
          - AmericanHot
          - Cajun
          - PolloAdAstra
          - SloppyGiuseppe
        - SpicyPizzaEquivalent
        - ThinAndCrispyPizza
        - VegetarianPizza
      - PizzaBase
      - PizzaTopping
        - MeatTopping
        - SeafoodTopping
        - SpicyTopping
        - VegetarianTopping
    - ValuePartition
      - Spiciness

Annotations: CheesyPizza

Annotations

rdfs:label [language: en]

CheesyPizza

rdfs:label [language: pt]

PizzaComQueijo

skos:prefLabel [language: en]

Cheesy Pizza

Description: CheesyPizza

Equivalent To

Pizza

and (hasTopping some CheeseTopping)

SubClass Of

Pizza

General class axioms

SubClass Of (Anonymous Ancestor)

hasBase some PizzaBase

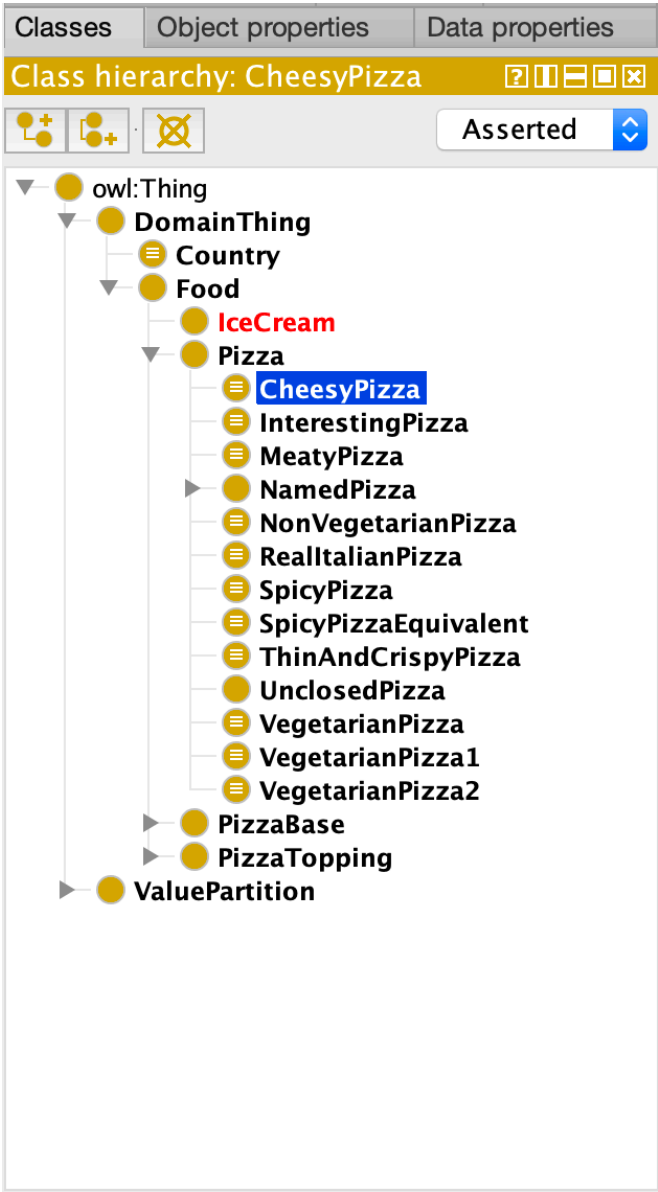
Instances

Target for Key

Reasoner active

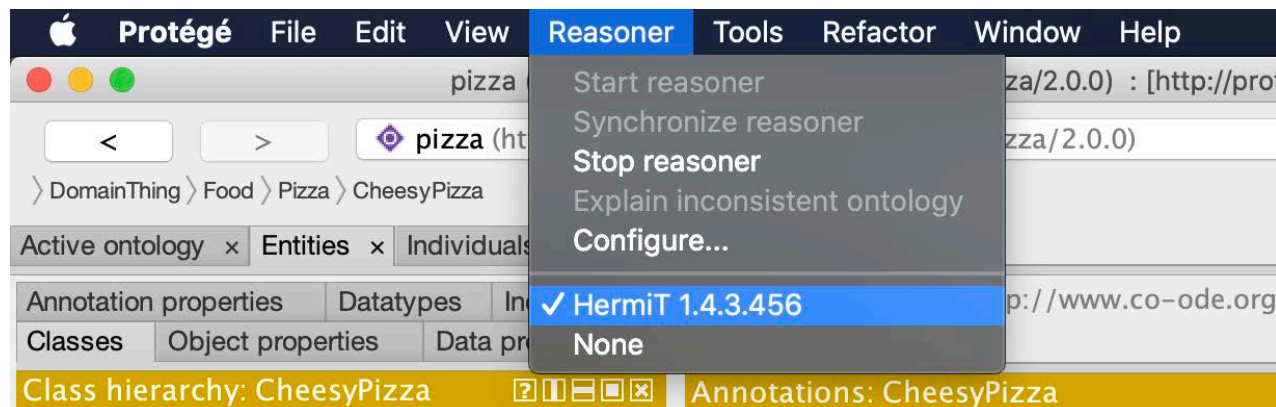
☒ Show Inferences

# Protégé: jerarquía asertada vs inferida

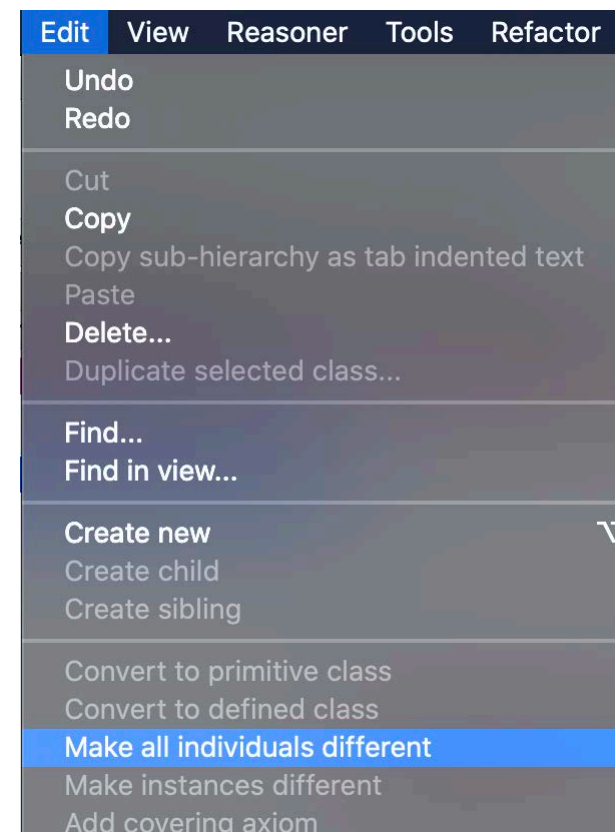


# Protégé: razonador

- Para clasificar tenemos que iniciar el razonador (motor de inferencia)



- Podemos indicar que todos los individuos son entidades distintas de una vez



# Construcción de ontologías en OWL-DL

- Comenzar con una taxonomía de clases primitivas
  - Deben formar grafos acíclicos
  - Recordar que la condición de clases disjuntas debe establecerse explícitamente
- Usar clases definidas y el razonador para crear jerarquías múltiples
  - Usar la cuantificación existencial (some) por defecto
  - Sólo se clasifican conceptos por debajo de los conceptos definidos
- Hay que tener cuidado con:
  - El razonamiento de mundo abierto
    - Usar axiomas de cierre cuando sea necesario
  - Cuantificadores (some/only)
  - Restricciones de dominio y de rango
  - Propiedades transitivas, simétricas, etc.
  - Hacer las disyunciones explícitamente



# Tutorial de las pizzas

- Tutorial paso a paso que explica los conceptos básicos relativos a la construcción de ontologías usando OWL-DL en el dominio de las pizzas
  - [A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools](#). Edition 1.3. Matthew Horridge. The University Of Manchester.

A Practical Guide To Building OWL Ontologies  
Using Protégé 4 and CO-ODE Tools  
Edition 1.3

Matthew Horridge

## Contributors

v 1.0 Holger Knublauch , Alan Rector , Robert Stevens , Chris Wroe  
v 1.1 Simon Jupp, Georgina Moulton, Robert Stevens  
v 1.2 Nick Drummond, Simon Jupp, Georgina Moulton, Robert Stevens  
v 1.3 Sebastian Brandt

# Ontologías: ventajas

- Potencian la construcción y reutilización de bases de conocimiento
  - Lenguajes de representación estándar
  - Semántica formal bien definida
  - Mecanismos de inferencia bien estudiados
  - Ya existen ontologías bastante asentadas en ciertos dominios
  - Herramientas visuales
  - Metodologías para construir ontologías
- Una gran comunidad de investigadores
- Representación declarativa
- Separación explícita entre instancias y clases
- Distintos niveles de expresividad y eficiencia

## Ontologías: inconvenientes

- Muy lejos de alcanzar el objetivo original
  - Anotar la web con etiquetas semánticas para construir agentes que puedan combinar información de distintas fuentes para resolver problemas complejos
- Sigue siendo difícil construir grandes bases de conocimiento
  - Y llegar a consensuar el vocabulario de un dominio
- Para tener tiempos de razonamiento polinómicos es necesario limitar mucho la expresividad (por ej. sólo conjunción y existencial)
- Dificultad para expresar conocimiento procedimental
- Dificultad para expresar excepciones, conocimiento por defecto, propiedades sólo de las clases (y no de los individuos), etc.

## Similitud semántica en taxonomías

- Podemos calcular la similitud (o distancia) semántica entre dos conceptos a partir de la topología de una taxonomía.
- Similitud topológica
  - Entre conceptos:
    - Basada en aristas (Pekar)
    - Basada en nodos (distancias al LCS)
  - Entre individuos:
    - Por parejas: combinando las similitudes semánticas de los conceptos que representan
    - Por grupos: similitud entre grupos de individuos (Jaccard)

## Similitud semántica en taxonomías

- La **similitud de Resnik** se basa en la teoría de la información y considera que dos conceptos son más similares cuanto más información compartan

$$\text{sim}(A, B) = \max_{C \subseteq A \cap B} [-\log p(C)]$$

- donde:
  - A y B son los conceptos a comparar
  - C es cada concepto subsumido por la intersección de A y B
  - $p(C)$  es la probabilidad de que una instancia pertenezca a C
- Cuando más abstracto sea un concepto C, mayor será  $p(C)$  y por tanto menos información aportará ( $-\log p(C)$ )

*Philip Resnik (1995). Chris S. Mellish (ed.). [Using information content to evaluate semantic similarity in a taxonomy](#). Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95). 1: 448–453.*

## Similitud semántica en taxonomías

- La **similitud de Jaccard** mide la similitud entre dos conceptos como la razón entre el número de instancias comunes y totales

$$\text{sim}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

– donde:

- $|A \cap B|$  representa el número de instancias comunes
- $|A \cup B|$  representa el número de instancias de la unión

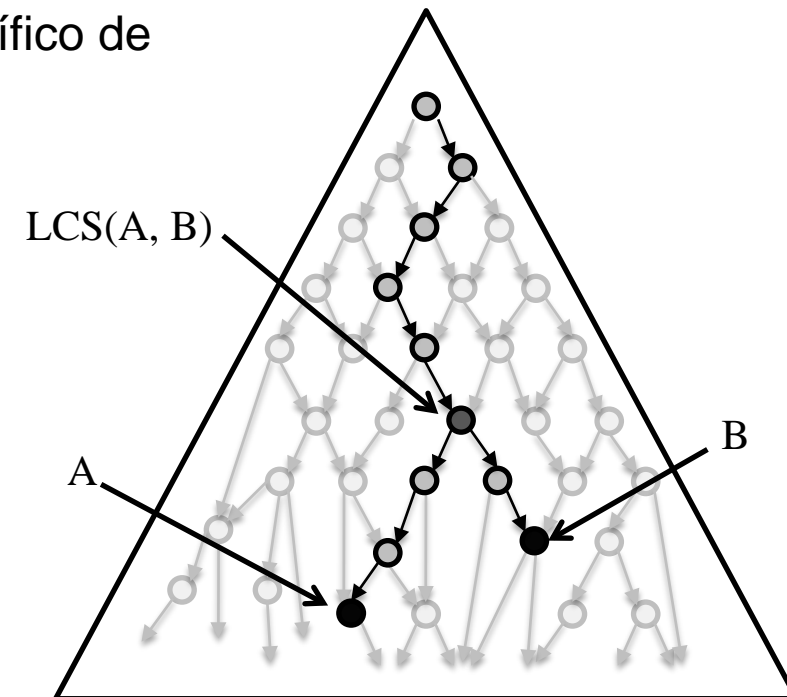
# Similitud semántica en taxonomías

- La similitud basada en **distancias al LCS** mide la similitud entre dos conceptos como la razón entre las siguientes longitudes

$$\text{sim}(A, B) = \frac{\partial(\text{root}, C)}{\partial(\text{root}, C) + \partial(C, A) + \partial(B, C)}$$

– donde:

- $\partial(A, B)$  es el mínimo número de aristas que conecta A y B
- $C = \text{LCS}(A, B)$  es el concepto más específico de la jerarquía que es más general que A y B (*least common subsummer*).
  - Puede haber varios



Pekar, Viktor; Staab, Steffen (2002). [Taxonomy learning](#). *Proceedings of the 19th international conference on Computational linguistics* -. **1**. pp. 1–7.

# Bibliografía

- Palma Méndez, J.T., Marín Morales, R., [Inteligencia Artificial. Métodos, técnicas y aplicaciones](#). McGraw-Hill, 2008 (capítulos 1, 2 y 3)
- D. Allemang and J. Hendler. [Semantic Web for the Working Ontologist : Effective Modeling in RDFS and OWL](#). Elsevier Science & Technology, 2011
- Apache Jena [SPARQL Tutorial](#)
- [W3C Semantic Web Standards](#)
- [Protégé ontology editor](#)
- [Ontology Development 101: A Guide to Creating Your First Ontology](#). N. F. Noy & D. L. McGuinness, 2001
- [A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools](#). Edition 1.3. Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe.