Lógica Temporal

Jose Aguilar
Cemisid, Facultad de Ingeniería
Universidad de los Andes
Mérida, Venezuela
aguilar@ula.ve

Lógica Temporal

- Es un tipo de lógica simbólica, cuyo valores de verdad de las proposiciones dependerán del tiempo

"Tengo hambre"

- Aunque su significado es constante en el tiempo, el valor de verdad de la declaración puede variar en el tiempo.

- La declaración no puede ser verdadera y falsa al mismo tiempo.

Lógica Temporal

- **Contrasta** con el punto de vista de la **lógica clásica**,
  - Valor de verdad de una proposición es siempre el mismo
  - Declaraciones con valores constantes en el tiempo.
  - Por ejemplo:

"La luna está apareciendo" vs "La luna es un satélite de la tierra"

La primera proposición tiene una condición implícita de tiempo "**ahora**". La segunda proposición es **atemporal**

Lógica Temporal

- La lógica temporal se aplica a universos de discursos en donde sus fórmulas lógicas describen secuencias de estados

- La principal ventaja de la lógica temporal es su expresividad:
  - "Siempre estoy dormido", “A veces tengo sueño", "Voy a tener sueño en algún momento“.

- Amplias formas de razonamiento con relaciones causales:
  - Si observo A y sé que A causa B, entonces predigo B;
  - Si deseo obtener D y sé que C causa D, entonces hago C;
  - Si noto F y me desagrada y sé que E causa F, entonces trato de actuar sobre E

- El problema de modelado es determinar:
  - relaciones causales genéricas y
  - la evolución de un conjunto de eventos para un estado del mundo a partir de esas relaciones causales.

Clasificación

- *Dominio del tiempo (Continuo o Discreto) –> en general se considera que el* tiempo tiene como dominio los reales mayores o iguales a 0, con lo que el tiempo sería una función continua,

- El *tiempo lineal es cuando se considera que el tiempo tiene solo un posible* camino, aunque desconocido.

- El *tiempo paralelo es cuando existen diferentes posibles líneas del tiempo,* pero todas ellas tienen principio y fin independientes, sin que haya ninguna relación entre ellas.

- Un *tiempo ramificado en que el pasado fue lineal, pero en el* futuro se ramifica entre posibles alternativas a elegir.

- El *tiempo circular hace* coincidir el final y el principio de tiempo, de forma que el tiempo sea infinito y además cíclico.

  Esta noción del tiempo la tenían los griegos, al igual que nosotros actualmente tenemos esa noción para la energía y la materia (Un ciclo muy conocido en la naturaleza es el del agua).

Lógica Temporal

- Diferentes técnicas de modelado se pueden utilizar:
  - **Técnicas basadas en lógica**: lógica modal, crónicas, etc.;
  - Técnicas que tratan de analizar la **relación probabilística entre las acciones** (MDP),
  - Etc.

- Tres métodos principales:
  1. Uno que considera **operadores temporales** (por ejemplo: lógica modal);
  2. Otro que identifica los tiempos o **intervalos de tiempo en el que las proposiciones lógicas son válidas**;
  3. Otro introduciendo el **tiempo como argumento en los predicados** en sentencias en lógica predicado de primer orden y proposicional

Lógica Temporal Proposicional (PTL)

✓ Además de los operadores del cálculo proposicional ($\Lambda, \nu, \neg, \rightarrow$), existen tres operadores unarios temporales:

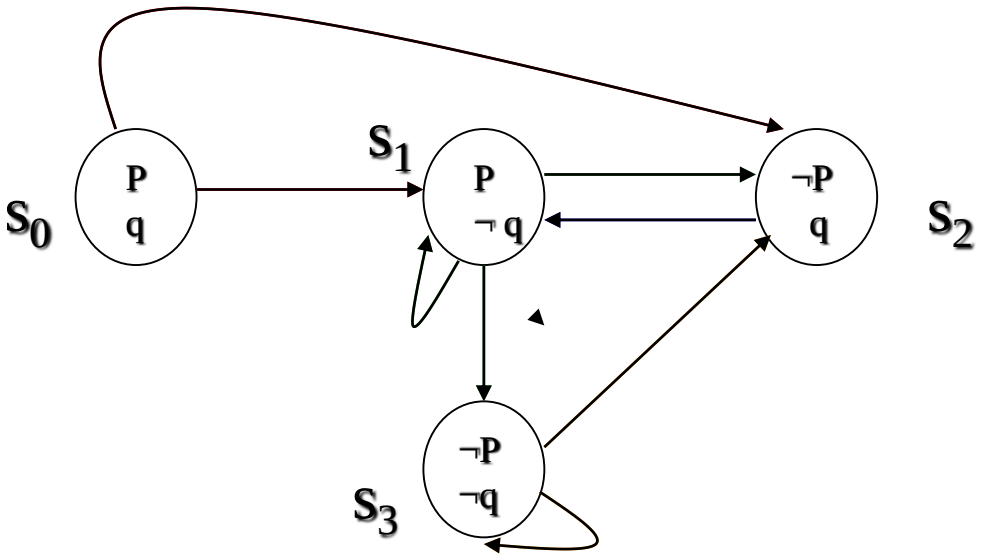
- *always,* denotado por $\Box$, "*para cualquier instante t en el futuro*",
- *eventually,* denotado por $\Diamond$, "*para algún instante t en el futuro*",
- *next,* denotado por $\circ$, "*en el instante siguiente*".

✓ La semántica de PTL da una interpretación para las proposiciones y el tiempo.
✓ Define un conjunto de estados. Cada uno contiene una interpretación de las proposiciones.
✓ Una interpretación PTL se dibuja como un diagrama de transición de estados.
✓ El tiempo se representa mediante transiciones entre estados.

PTL

- ✓ el valor cierto de la fórmula temporal $A = p \text{ v } q$ se determina para cada estado s

  - A es cierto en s_0
  - A es cierto en s_1
  - A es cierto en s_2
  - A es falso en s_3



Lógica Temporal basada en Predicados

- Los conceptos sobre el tiempo quedan fuera de la lógica de primer orden:
  - Ej: Como indicar una acción en un tiempo determinado.

    John está corriendo

    John correrá

- Estos predicados en lógica de primer orden quedan reducidos a un sujeto que ejecuta una acción

  John correr -> Corre (John).

- La importancia del tiempo radica en que, con la inclusión del tiempo, nos aporta información muy útil para consideraciones ulteriores

Lógica Temporal basada en Predicados

- Una primera aproximación se puede lograr a partir de la lógica de primer orden añadiendo algo que indique el tiempo,

  TIEMPO (t) y CORRER (John , t)

- La segunda opción es hacer uso de la lógica modal, con los predicados :
  - necesidad -> para predicados que se cumplen siempre (todo el tiempo)
  - posibilidad -> para predicados que se cumplen en algún tiempo

- Los diferentes instantes de tiempo hacen el papel de diferentes posibles mundos en la lógica modal.

Lógica Temporal basada en Predicados

- **Instantes de tiempo**
  - ANTES (T1 , T2) -> el tiempo T1 se produce antes que T2
  - DESPUES (T2 , T1) -> el tiempo T2 se produce después que T1
  - MISMO(T1 , T2) -> el instante de tiempo es el mismo.

- **Intervalos de tiempo**

  Un intervalo de tiempo es un fragmento de tiempo que empieza en un instante de tiempo y acaba en otro instante posterior.

  INTERVALO (T1 , T2) -> intervalo.

Mundos Posibles y Lógica modal

- El lenguaje para expresar la semántica de un mundo **posible o necesario** es *la lógica modal*.
- La lógica modal fue desarrollada para formalizar argumentos que incluyen *necesidad* y *posibilidad*.

Una ***proposición necesaria*** es una proposición verdadera que no puede ser falsa.

Una ***proposición posible*** es una que podría ser verdadera.

Mundos Posibles y Lógica modal

- Se basa en *lógica no-monótona (no-determinista), dada una determinada* sentencia, su valor de verdad cambia según las circunstancias.

En este caso, la circunstancia que se tiene en cuenta es únicamente **el tiempo**.

- Una sentencia puede ser verdad en un momento dado y, al instante siguiente dejar de serlo; también puede suceder a la inversa.
  - *Un estado es una instante* determinado *del universo* .
  - El tiempo se puede ordenar, por tanto, existe una *relación de orden entre* instantes de tiempo.

Mundos Posibles y Lógica modal

- La sintaxis de la lógica modal es la de la lógica clásica con la adición de dos operadores

□ necesidad

◊ posibilidad (quizás)

Lógica alética

□x significa “es necesario x“ (“no x es imposible”)

◊x significa "x es posible".

Aplicaciones Lógica

- Es posible que llueva

$$\Diamond \text{ llueva}$$

- Es necesario que el sol se levante por el este

$$\Box \text{ sol se levante por el este}$$

Lógica Modal Básica

- Se construye sobre el lenguaje de la lógica de proposiciones, con la incorporación de los dos nuevos operadores.

- Alfabeto :
  - letras proposicionales: po, p1, p2, . . .
  - símbolos lógicos:
    - constantes proposicionales: $\perp$, T
    - conectivas monarias (¬) y binarias: ^,V,->
    - operadores modales: $\Diamond$, $\Box$

Lógica Modal Básica

- Las fórmulas del lenguaje se definen según BNF:

$\phi ::= p \mid \bot \mid T \mid > \mid \neg p \mid (po \wedge p1) \mid (p0 \vee p1) \mid (po \rightarrow p1)$
$\mid \Box p \mid \Diamond p$

- Equivalencias
  - $\Box(\phi \wedge \beta) \equiv (\Box \phi \wedge \Box \beta)$
  - $\Diamond(\phi \vee \beta) \equiv (\Diamond \phi \vee \Diamond \beta)$
  - $\Box p = \neg\Diamond \neg p$
  - $p0 > p1 = \neg\Diamond(p0 \wedge \neg p1) = \Box(p0 \rightarrow p1)$

Mundos Posibles y Lógica modal

Lógica Modal

$$\Diamond p = \neg\Box\neg p$$

$$\Box p = \neg\Diamond\neg p$$

Lógica Predicado

$$\exists x\ \phi(x) = \neg\forall x\ \neg\phi(x)$$

$$\forall x\ \phi(x) = \neg\exists x\ \neg\phi(x)$$

La lógica modal también puede enriquecer a la lógica de predicado con sus operadores

- □ ∀x (A-> B) -> □ (∀x A-> ∀x B)
- ∀x □ A-> □ ∀x A

Lógica Modal Básica

Fórmulas modales que caracterizan relaciones binarias

– Reflexiva	(□ p-> p)	(T)
– Simétrica	(p-> □ ◊ p)	(B)
– Transitiva	(□ p-> □ □ p)	(4)
– Euclídea	(◊p-> □ ◊ p)	(5)
– Determinista	(◊ p-> □ p)	(Q)
– Serial	(□ p-> ◊ p)	(D)
– Inferir	□(p0-> p1) -> (□p0-> □p1)	(K)

Lewis, 1912 propuso varios sistemas lógicos aléticos definidos por un número dado de axiomas que los caracterizan

S5=KT5 S4=KT4

Lógica Modal

- Otros operadores: **Happens, Done, BEL**, y **GOAL**
  - *Happens* define una secuencia de eventos que ocurrirán
  - *Done* define una secuencia de eventos que han ocurrido
  - *e;e'* (*e* seguido de *e'*)
  - *e | e'* (*e* o *e'*)
  - *e?* (test) *e** (iteración)

Otras Lógicas Modales

- Las *creencias* pueden representar algún conocimiento sobre otros agentes (lógica epistémica).

  El agente a_1 cree φ sobre el agente a_2:

  $$\boldsymbol{BEL\ a_1\ a_2\ \varphi}$$

- *Intención:* puede representar la voluntad/obligación de cumplir un deseo o de efectuar una acción (lógica deóntica)

$$Intend(x,a) \Leftrightarrow \exists P / Goal(x,P)$$
$$\wedge Bel(x, a \supset P)$$
$$\wedge Bel(x, \neg P)$$
$$\wedge Bel(x, Do(x,a))$$

Sistemas Lógicos Temporales

Se diferencian de la lógica modal por:

- Lenguaje más rico: operadores presente, pasado y futuro
- Permiten formalizar las propiedades que dependen del tiempo

Tipos:

1. Lógica temporal arborescente (Computational tree logic, CTL),
2. Lógica temporal lineal (Linear temporal logic, LTL)
3. Lógica temporal de intervalos (Interval temporal logic, ITL).
4. Lógica temporales de acciones (Temporal Logic of Actions, TLA).

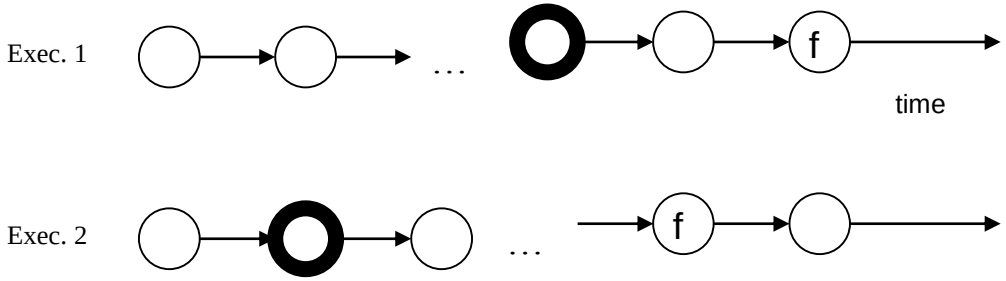
Ontología del tiempo: algunos operadores Temporales

- O or X or N (just after, next: tomorrow or immediately after),
- □ (henceforth, from now),
- ◊ (finally),
- J or U (until),
- Θ (just before),
- Δ (front),
- Φ (ever), D (from),
- F (eventually: once, sometimes),
- G (always),
- R (release).

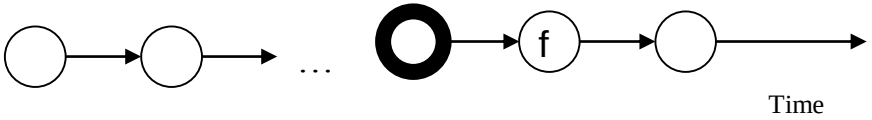
Linear-Time Temporal Logic (LTL)

Se considera los comportamientos modelados como secuencias lineales de Estados

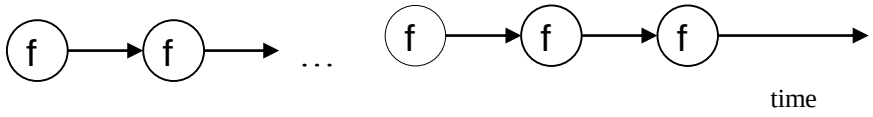
- **F(f)** Es verdad ahora si f es cierto al menos en una etapa posterior (Op F)



- **X(f)** Es verdad ahora si f es cierto en el siguiente paso (Op. X)

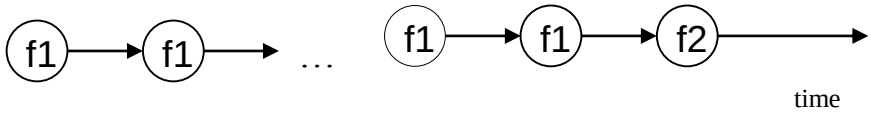


- **G(f)** Es verdad ahora si f es cierto en todas las fases posteriores, incluyendo ahora(Op. G)



LTL

– **Op. U:** f1 U f2 es verdad si f1 es verdad hasta que f2 es verdad,



- Ejemplo de sentencia LTL (llamada telefonica paga):

*u_unhook ^ **X** sys_displays(Insert_Card) ^ **X** [(u_insert_card ^ **X** sys_displays(nb_units_remaining)) ^ (**X** u_call_no V **F** (u_correction_no **U** no_correct)) ^ ((**X** u_connect V **F** (u_change_card_empty **U** units(card) > minimum)) **U** (u_hang up ^ u_remove_card))] => ltl_phone*

Donde ^ *y V son los clásicos operadores y o*

Ejemplos de propiedades en LT

- EVT2 no va antes de EVT1: $G(\neg(EVT2)\ U\ EVT1)$
- Si EVT2 entonces EVT1 será después: $G(EVT2 \Rightarrow X\ (EVT1))$
- ***Propiedad de Hambruna:*** Un recurso es garantizado al menos una vez a un proceso:

$$F(ai \wedge aj)\ i \neq j$$

- ***Propiedad de Seguridad***: Pi y Pj son 2 procesos y CSi significa que proceso Pi esta en su sección critica, la siguiente formula describe la exclusión mutua de Pi y Pj:

$$G(\neg\ (CSi \wedge CSj))$$

- ***Propiedad de Vivacidad***: TRYi expresa una solicitud de acceso a Csi del proceso Pi, la siguiente formula expresa una solicitud de acceso a la sección critica que eventualmente se le dará:

$$TRYi \Rightarrow F\ CSi$$

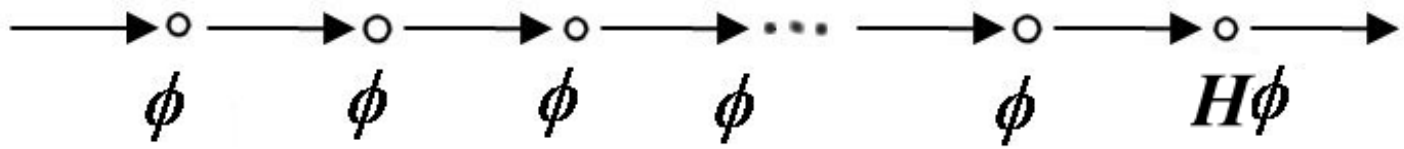
Operadores LTL

- Booleanos: negación, conjunción, etc.
- Temporales:
  - **Prevφ** : φ es verdad en el momento anterior (justo antes) (***PreviousVersion***)

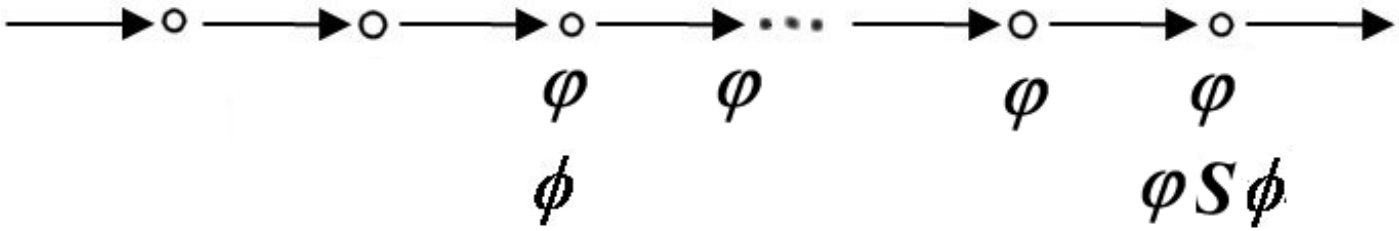
  - **Pφ**: φ es verdad en un momento anterior (algunas veces en el pasado) (***SomePriorVersion)***

Semantica en LTL

Hφ: φ es verdad en todos los momentos anteriores (siempre en el pasado) (***AllPriorVersions***)



φSϕ: φ es verdad en todos los momentos anteriores desde que ϕ fue verdad en un momento anterior (***From***)



Propiedades Formales

- ***H φ -> P φ.***
- ***H φ -> Prev φ.***
- ***Prev φ -> P φ.***
- ***Prev P φ -> P φ.***
- ***P P φ -> P φ.***
- ***H H φ -> H φ.***
- ***Prev Prev φ -> P φ.***

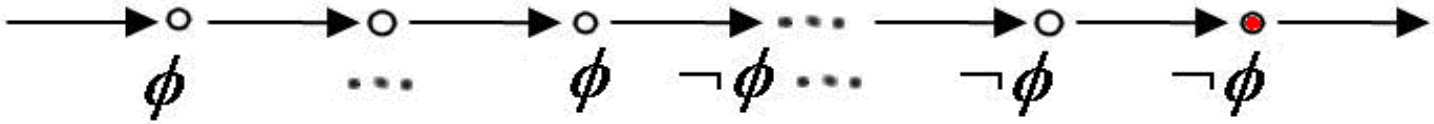
Ejemplos de Razonamiento

- φ: φ es verdad en el actual momento

- $\varphi \wedge \neg$ **Prev** φ: φ es verdad en el actual momento, pero no en el anterior.

- $\varphi \wedge$ **H**$\neg\varphi$: φ es verdad solo en el actual momento, nunca antes.

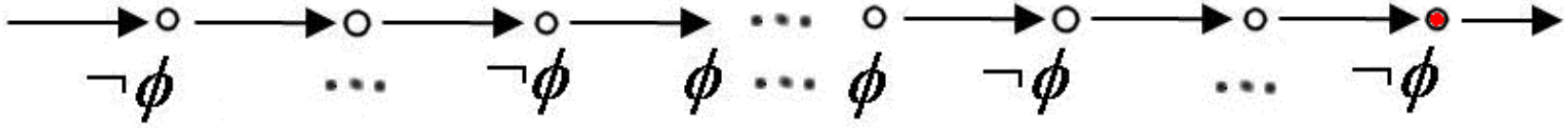
Ejemplos de Razonamiento



- Una vez ϕ cambia, nunca cambia más.

$$\neg\phi \; S \; (H\phi)$$

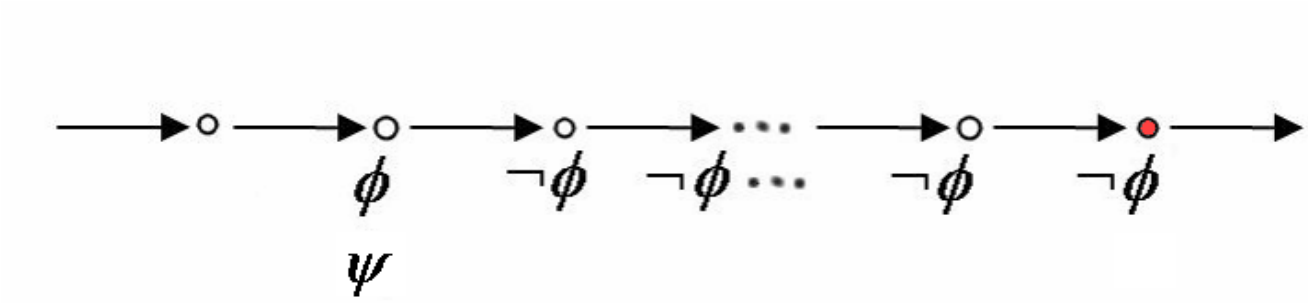
Ejemplos de Razonamiento



- ϕ cambia dos veces

$$\neg\phi \; S \; Prev(\phi \; S \; H\neg\phi)$$

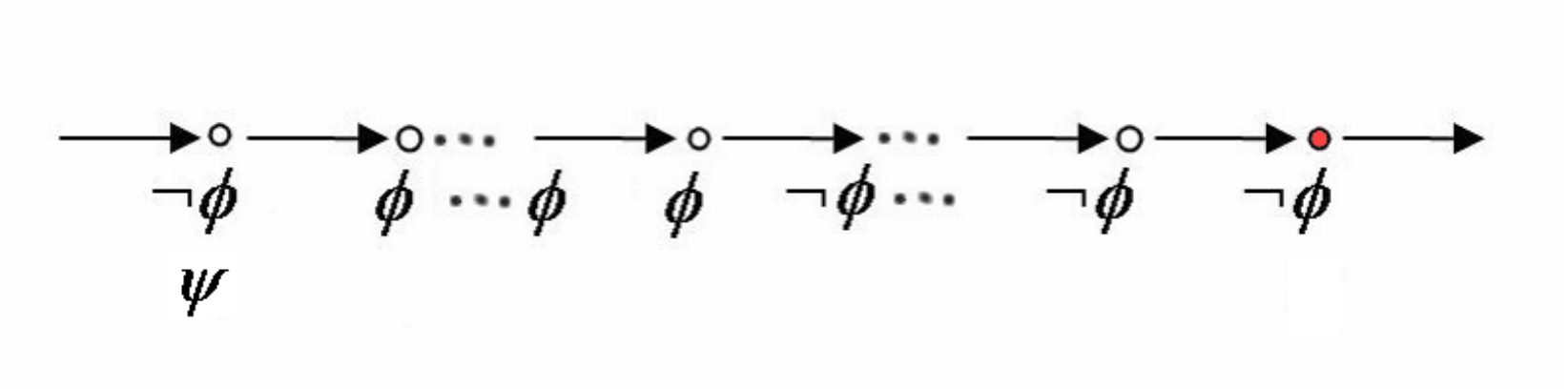
Ejemplo de Razonamiento



- ψ es verdad en la ultima ocasión en la cual ϕ fue verdad.

$$\neg\phi \, \mathbf{S} \, (\mathbf{Prev}(\phi \wedge \psi))$$

Ejemplo de Razonamiento



ψ es verdad en la última ocasión en la cual ϕ no fue verdad, antes de la última ocasión en la que ϕ fue verdad

¬ϕ S (Prev(ϕ S Prev(¬ϕ∧ ψ)))

Computation Tree Logic (CTL)

Es una lógica temporal donde se ramifican las ocurrencias de estados, lo que permite expresar propiedades en el árbol de ejecución (desde un estado inicial)

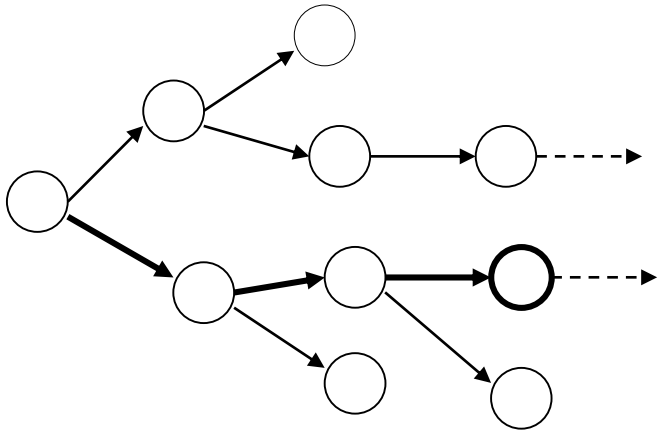
- CTL describe comportamientos más complejos que LTL

- 2 nuevos operadores:
  - **A** (Aϕ significa ϕ debe ser satisfecho en todas las rutas a partir del estado actual)
  - **E** (Eϕ significa que existe (hay) al menos un camino que comienza en el estado actual en la que ϕ es verdadero).

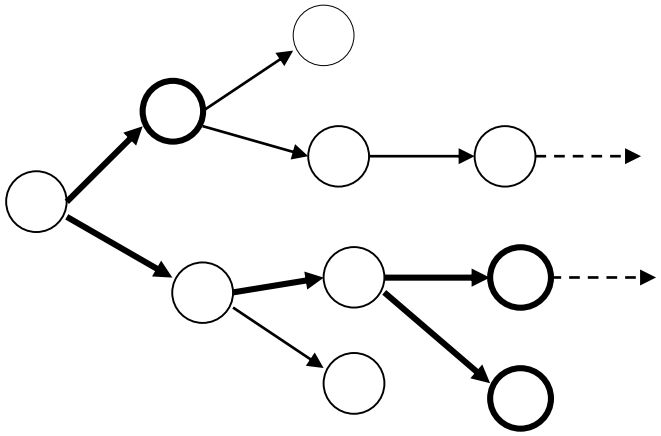
Computation Tree Logic (CTL)

Con estos operadores se pueden construir sentencias como:

- **EF(f)** significa "Es posible alcanzar un estado donde f es verificado" o “hay una ejecución (operador E) que conduce a un estado donde f es cierto (operador F)



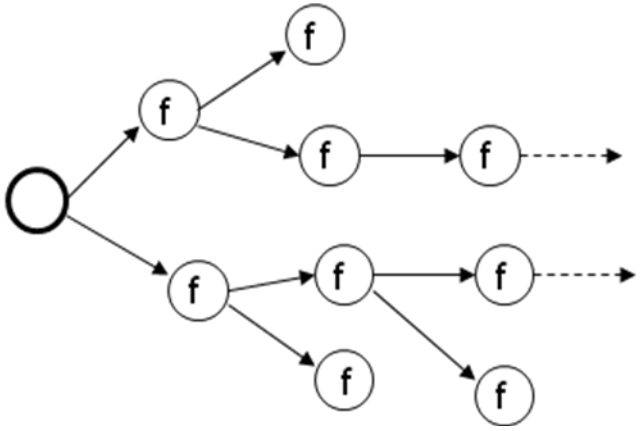
- **AF(f)** significa "f se verifica en el futuro" o "para toda ejecución (operador A), hay un estado donde f es cierto (operador F)".



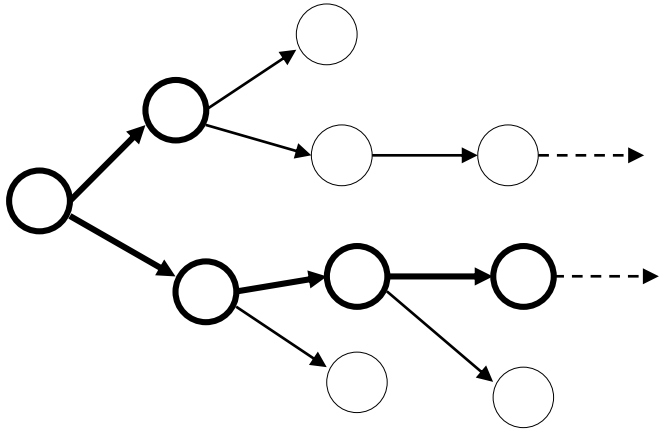
Computation Tree Logic (CTL)

AX(f) significa "todos los estados inmediatamente sucesores satisfacen f".

AG(f) significa "todas las ramas sucesoras satisfacen f".

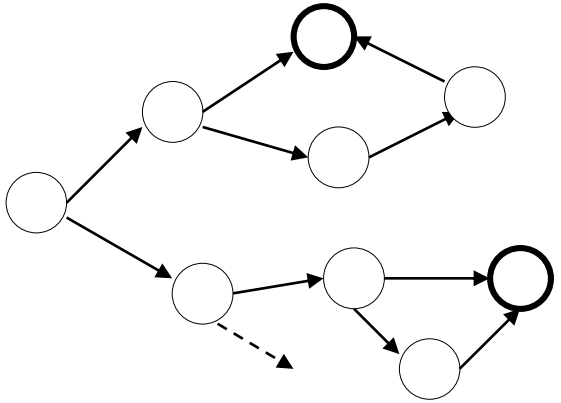


EG(f) significa “hay una ejecución (E operador) donde f es siempre verdadero (operador G)".

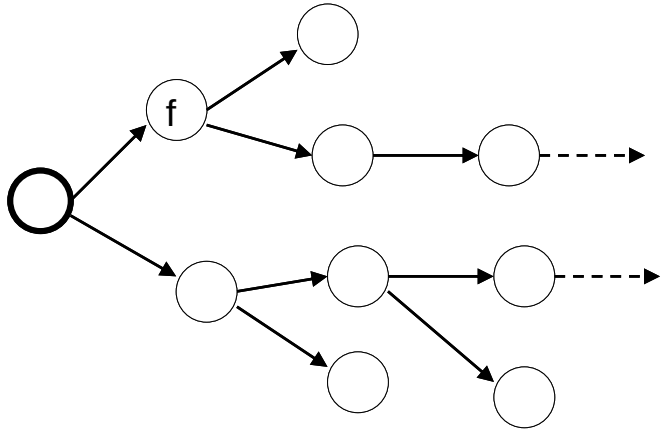


Computation Tree Logic (CTL)

AGEF(f) significa que desde cualquier estado alcanzable, es posible llegar a un estado que satisface f"

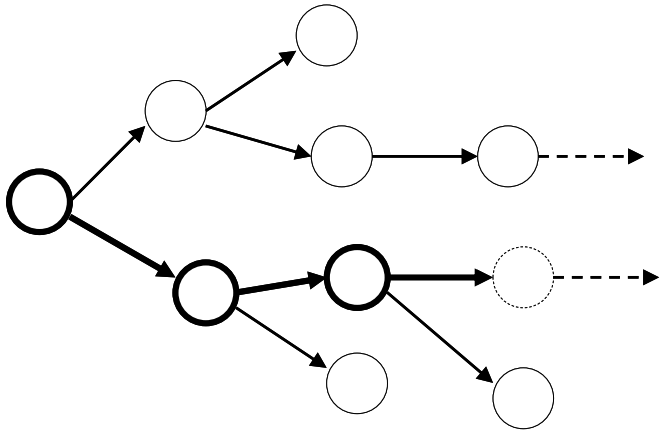


EX(f) significa que hay una ejecución (E operador) que en el siguiente estado satisface f ".



Clasificación

E(f) U g significa que “hay una ejecución (E operador) durante la cual f es verdadera hasta que g es cierto (f U g)".



- Mismo ejemplo con CTL:

u_unhook ^ u_insert_card ^ ***AX*** *[(u_call_no ^* ***EF*** *u_error_recovery) ^* ***AX*** *((u_connect ^* ***EF*** *change_card_empty)* ***U*** *(u_hang_up ^ u_remove_card)))] => ctl_phone*

lógica temporal de Intervalos (ITL)

- ITL presente tanto en la lógica proposicional como en la de primer orden los períodos de tiempo.
- Su noción clave es el intervalo (periodo) de tiempo σ
- σ es considerado como una secuencia finita de estados,

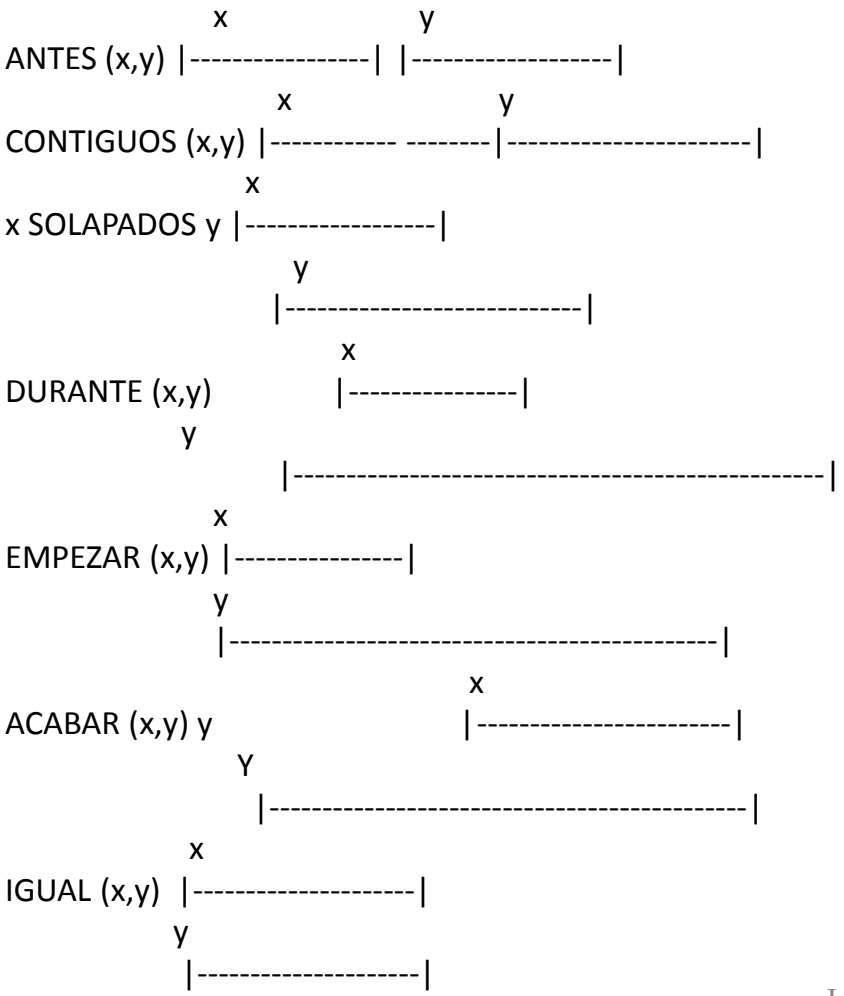
  e_1, e_2, … etc., donde cada estado es un mapeo del conjunto de variables a un conjunto de valores.
- Cada sentencia es válida en un intervalo dado σ_i
- La sintaxis es la siguiente:

$$= z|a|A| e_i \mid g(\sigma_1, \ldots \sigma_n) \mid g(e_1, \ldots e_n)$$

  - z es una variable dinámica entera que puede cambiar dentro de un intervalo,
  - a es una variable estática entera dentro de un intervalo,
  - A es una variable de estado dinámico dentro de un intervalo,
  - e_i es una variable de estado estática dentro de un intervalo,
  - g es una función de relaciones entre los intervalos o estados,

Ontología del tiempo

relaciones básicas binarias de los intervalos.



Lógica Temporal de Acciones

Es una lógica para especificar y razonar acerca de sistemas concurrentes y reactivos.

- Comenzamos especificando un sistema que comienza con x,y igual a 0 y los incrementa en 1 siempre. En un lenguaje de programación convencional, esto podría ser escrito

  initially x = 0 ; y-0

  loop forever x := x + 1 end loop

  loop forever xy:= y + 1 end loop

- Una fórmula Π de TLA es verdad o falsa en un comportamiento, que es una secuencia de estados, donde un estado es una asignación de valores a las variables.
- La fórmula es verdad en un comportamiento en el que el iesimo estado asigna el valor de i +1 a x, para i = 1, 2,. …

Lógica Temporal de Acciones

TLA es una fórmula Π:

X= x’ = x + 1

y’ = y

Y= y’= y + 1

x’ = x.

Π = (x = 0) (y = 0) Initially, x equals 0.

□ [X vY] .

- Fórmulas X e Y se llaman acciones. Una acción puede ser verdad o falsa en un paso, tal que el par de estados sean antiguos o nuevos,

Lógica de Eventos

- **Entrada:**
  - La representación simbólica del tiempo estampillado: eventos de bajo nivel (low-level events, LLE).
  - LLE viene desde diferentes fuentes/sensores.
  - Muchas LLE.
- **Salida:**
  - Alto nivel de eventos (HLE), combinaciones de LLE y/o HLE.
  - Humanos entiende más fácil HLE que LLE.
- **Reconocimiento de Eventos puede ser:**
  - On-line (run-time).
  - Off-line (retrospective)

Lógica de Eventos: **Crónicas**

- CRP es un método de razonamiento temporal que realiza un reconocimiento temporal en línea de los patrones de eventos en un flujo de eventos viniendo o sensados desde dispositivo que observan algún sistema o el medio ambiente de interés.

- Estos patrones, llamados crónicas, representan una posible evolución (normal o anormal) del estado del sistema observada.

Crónica

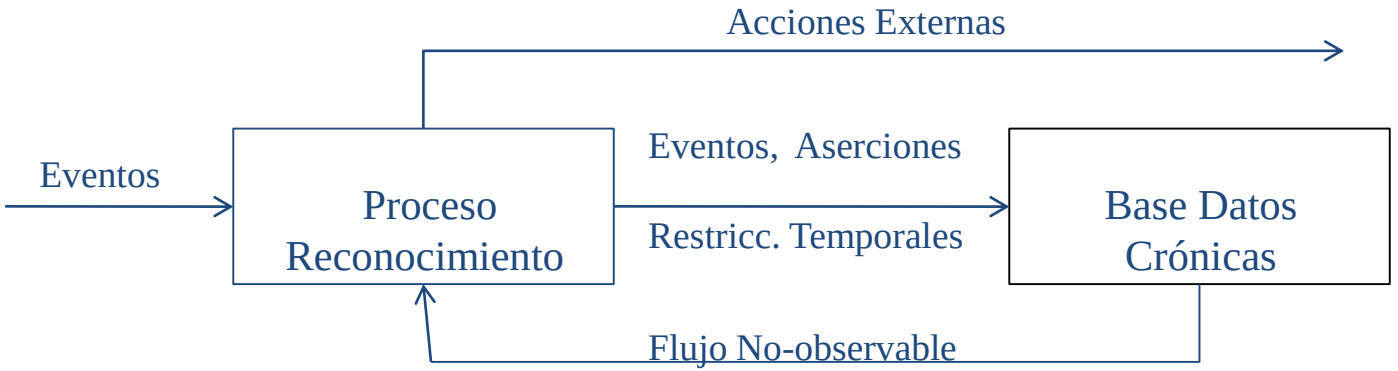
- Conjunto de eventos, unidos entre sí por restricciones temporales y contextuales

  Una crónica es un patrón de eventos.

- Representa la evolución de una parte del mundo.

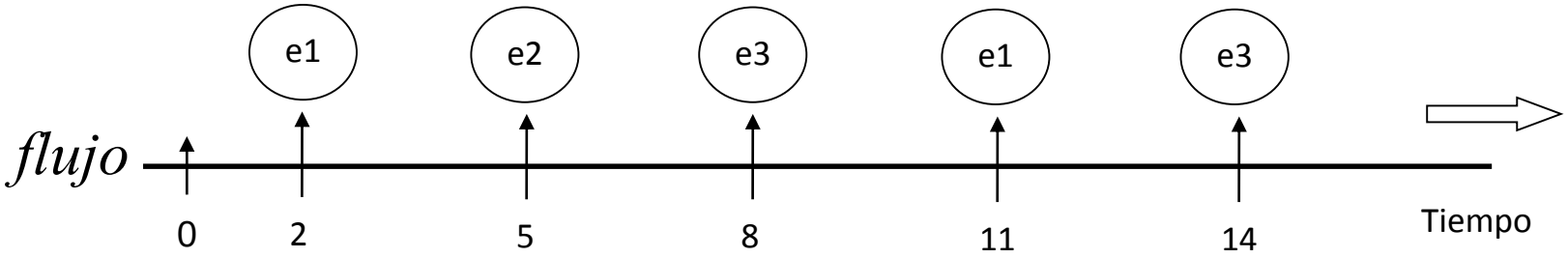
  El patrón temporal descrito por la crónica caracteriza el fenómeno a ser identificado.

- Las relaciones entre los eventos pueden ser lógica o temporal:
  - *En relaciones lógicas* se han considerado principalmente la conjunción (A y B) y la disyunción (A o B).
  - *En relaciones temporales* se consideran, principalmente, la secuencia (continuación de eventos ordenados) y la ausencia de eventos entre dos eventos (por ejemplo, C no debe ocurrir entre A y B)



Reconocimiento en las Crónicas

Reconocer es explicar los eventos de entrada (flujo) con la ayuda de patrones temporales (S), respetando las limitaciones del dominio (T)

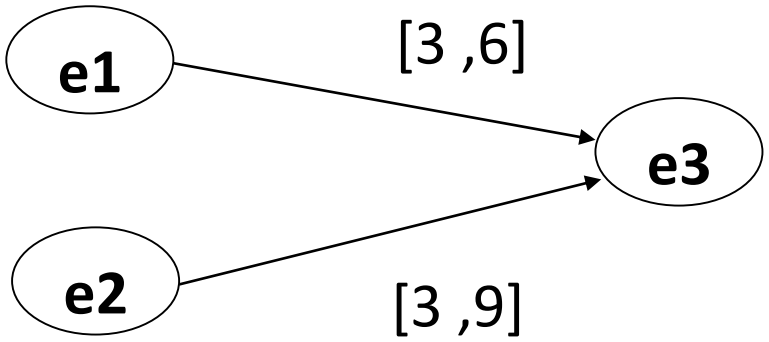


$\boldsymbol{C} = \{S,\ T\}$

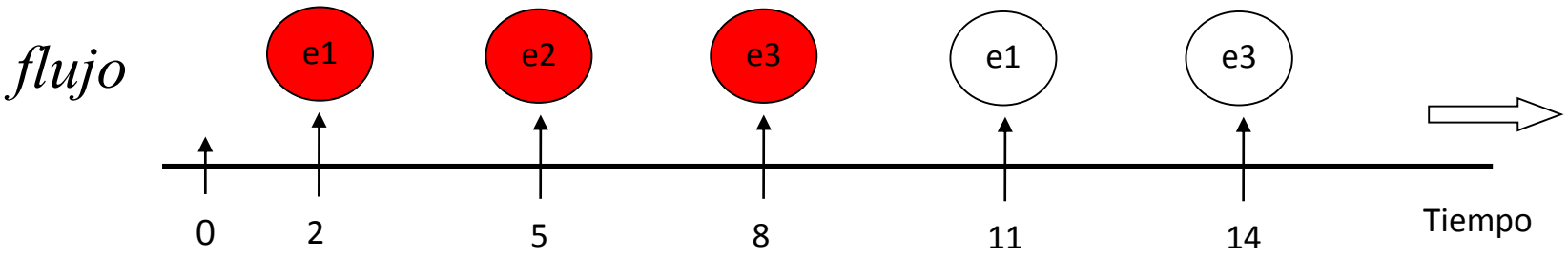
$S = \{e1,\ e2,\ e3\}$

$T = \{t1 < t3 \text{ avec } 3 \leq t3 - t1 \leq 6\ \}$

$\{t2 < t3 \text{ avec } 3 \leq t3 - t2 \leq 9\ \}$



Reconocimiento en las Crónicas



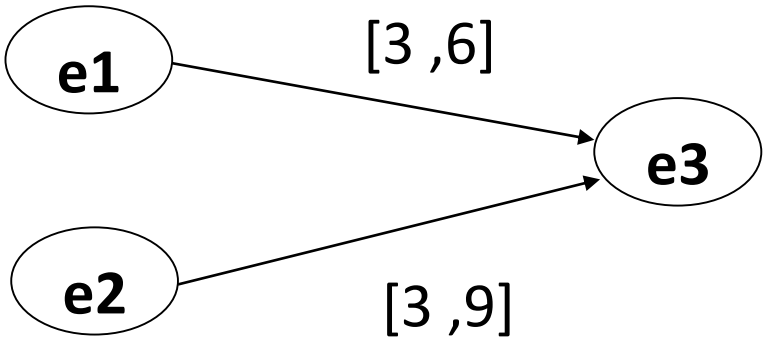
Primer Reconocimiento : Secuencia S-1

$\boldsymbol{C} = \{S, T\}$

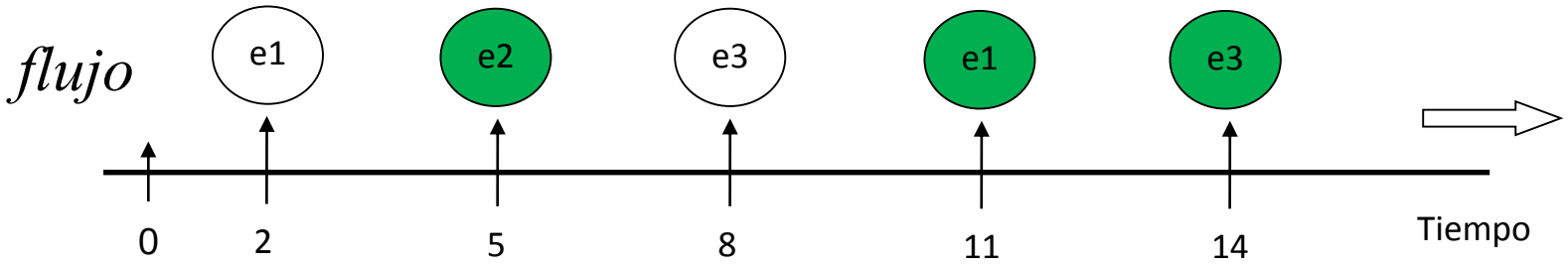
$S = \{e1, e2, e3\}$

$T = \{t1 < t3 \text{ avec } 3 \leq t3 - t1 \leq 6\}$

$\{t2 < t3 \text{ avec } 3 \leq t3 - t2 \leq 9\}$



Reconocimiento en las Crónicas



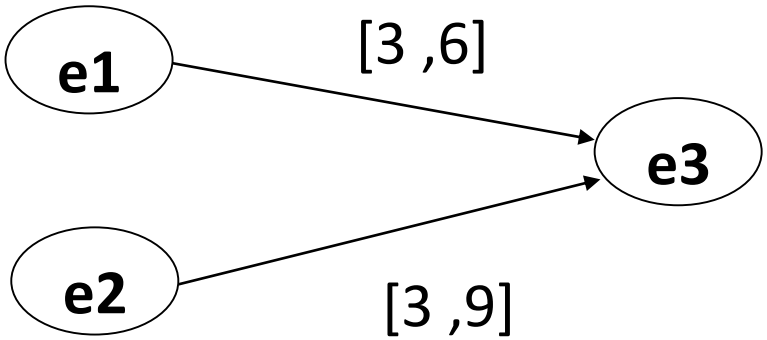
2do. Reconocimiento : Secuencia S-2

$\boldsymbol{C} = \{S, T\}$

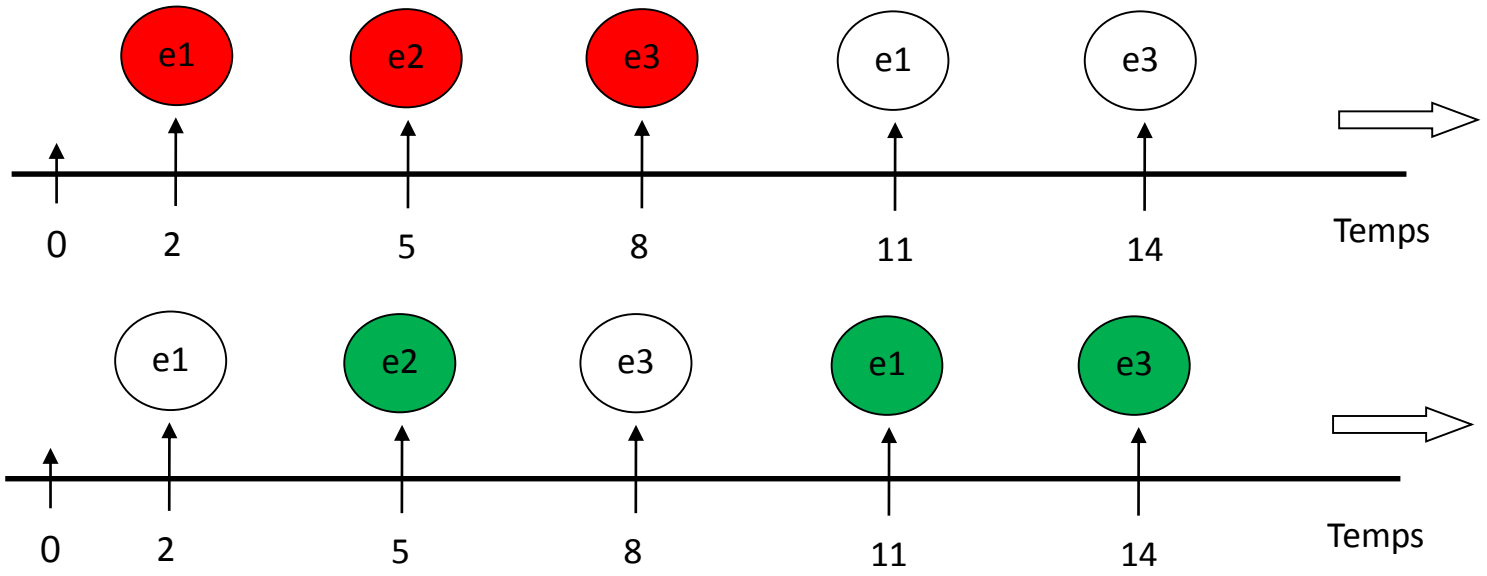
$S = \{e1, e2, e3\}$

$T = \{t1 < t3 \text{ avec } 3 \leq t3 - t1 \leq 6\}$

$\{t2 < t3 \text{ avec } 3 \leq t3 - t2 \leq 9\}$

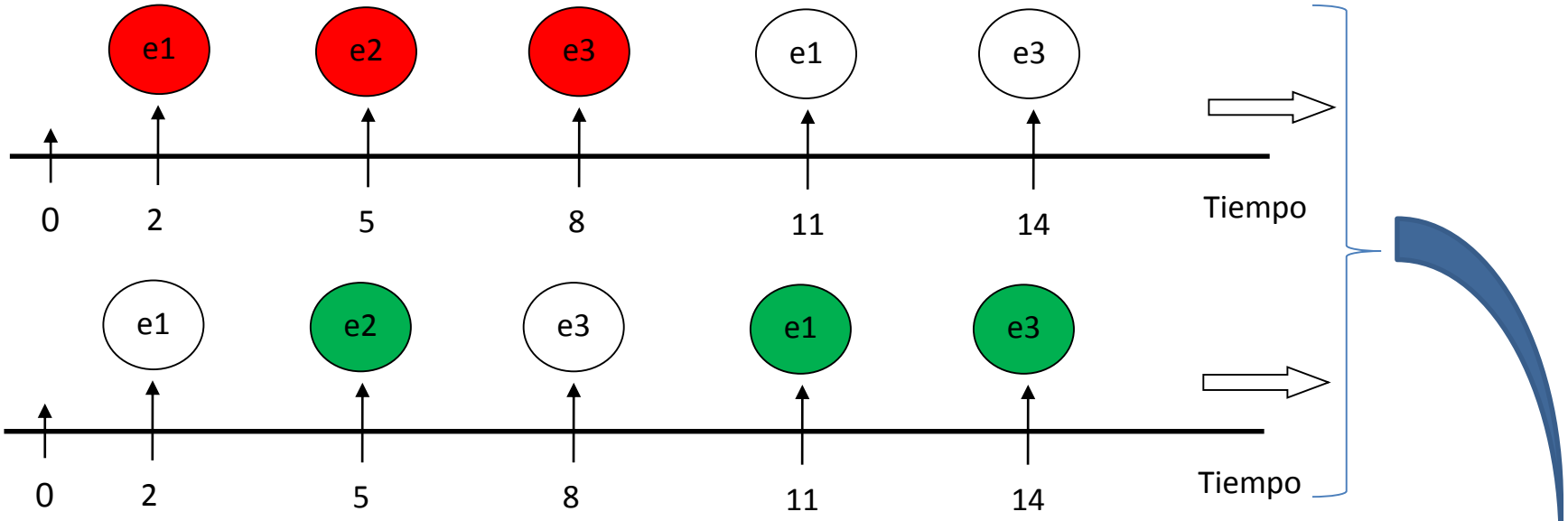


Reconocimiento en las Crónicas

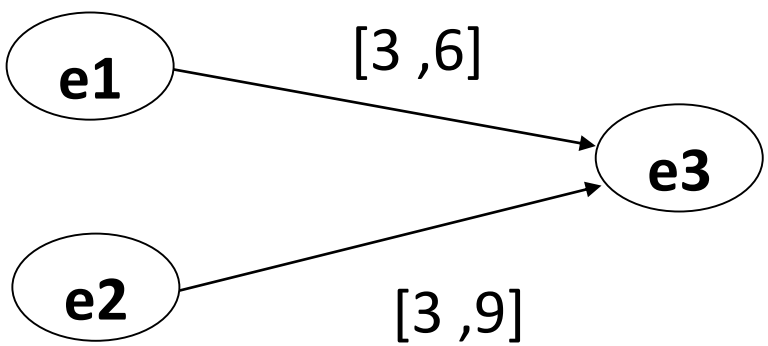


Las secuencias S-1 y S-2 son 2 instancias de la crónica C

Reconocimiento en las Crónicas



Las secuencias S-1 y S-2 son 2 instancias de la crónica C



Crónica

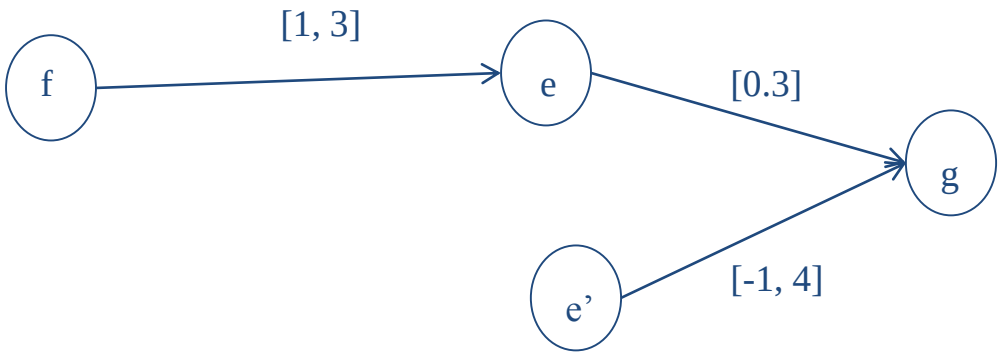
(Dousson et al.,1994) (Dousson et al., 2007)	• Usan grafos con restricciones temporales como modelo. • Los sistemas son descritos por puntos en el tiempo donde el tiempo es un conjunto linealmente ordenado de eventos discretos. • Una restricción de tiempo entre dos puntos de tiempo T1 y T2 está representado por un intervalo [I-, I +], que corresponde a los límites inferior y superior de la distancia temporal de T1 a T2 **Event** *label*(*vars*)**:** *e*s un tipo de dato donde *label* se emite durante el evento y *vars* es el conjunto de variables vinculadas al evento **Chronicle (C):** Un modelo *chronicle C e*s un par (*S, T*) donde *S e*s un conjunto de eventos y *T* el conjunto de restricciones entre las ocurrencias entre ellos

Example of chronicle:

S = *{* (*acquire*−(?*nb*), ?*t*1),
(*orderBN*−(?*nb*), ?*t*2),
(*orderBN*+(?*nbReturned*), ?*t*3),
(*nbExpected* = *nbReturned*?−
(?*nbReturned*), ?*t*4),
(*happy*(), ?*t*5)
}

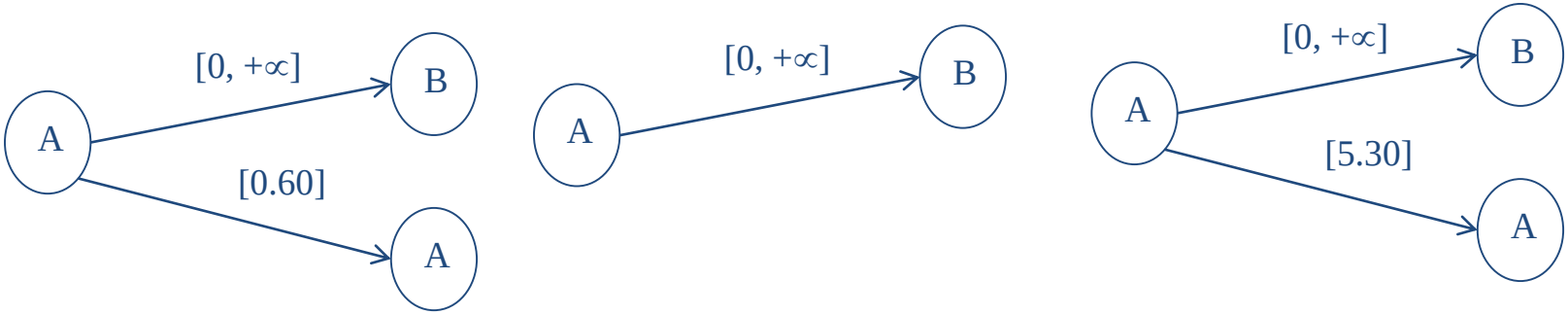
and

T = *{*?*t*1 <?*t*2, ?*t*2 <?*t*3, ?*t*3 <?*t*4, ?*t*4 <?*t*5*}*

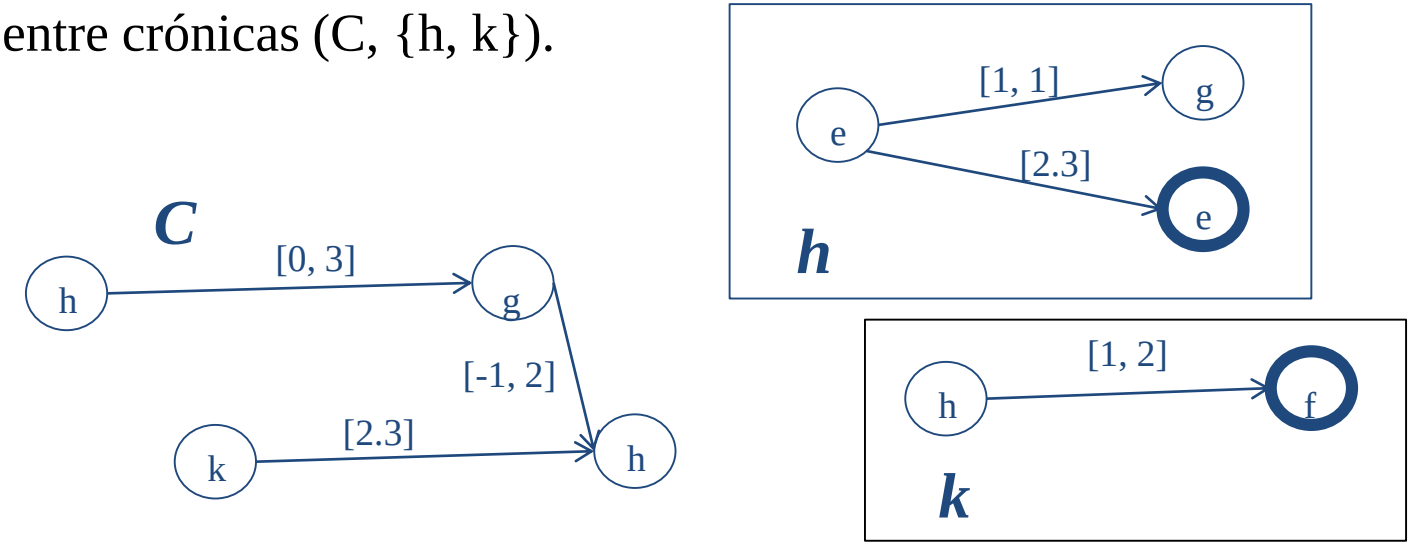


Crónica

Crónicas y subcrónicas

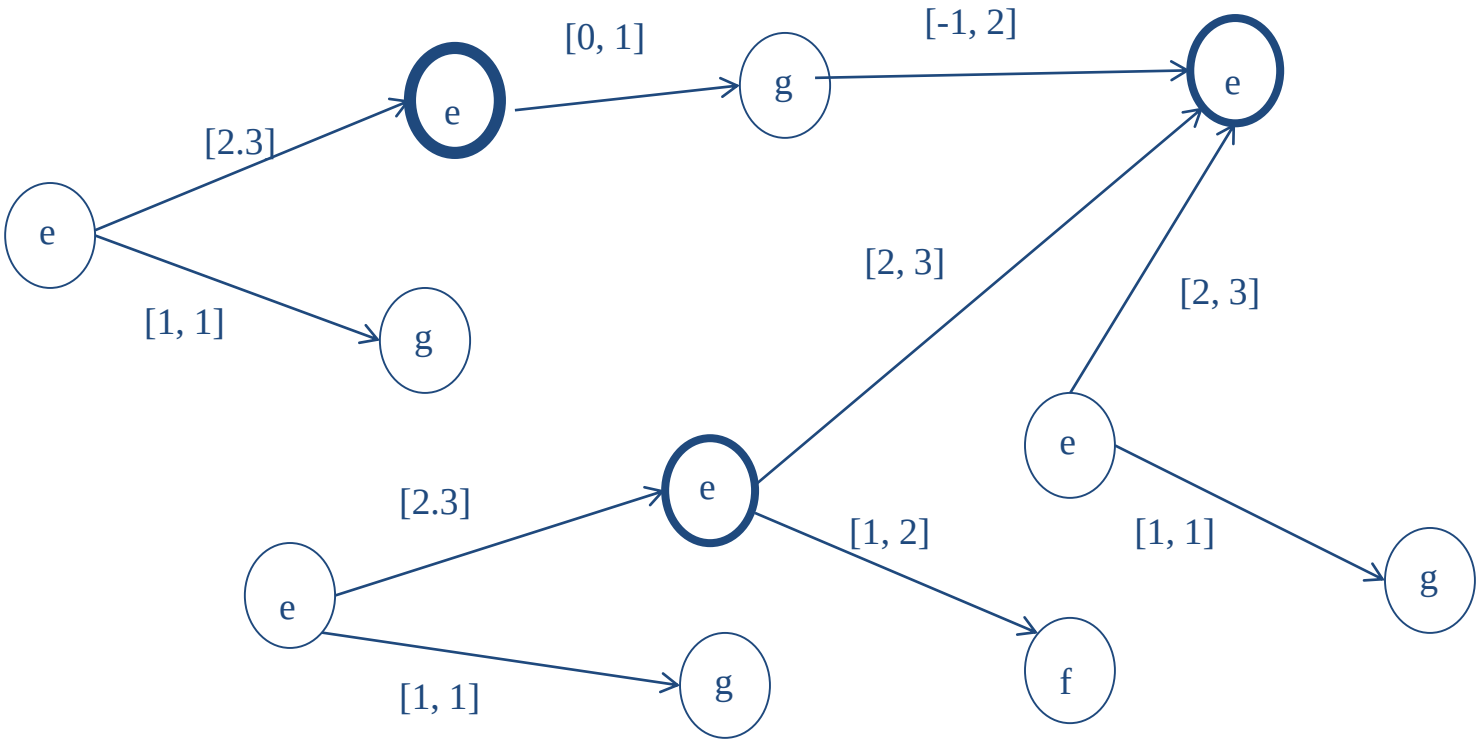


Jerarquía entre crónicas (C, {h, k}).



Crónica

Expansión f de la jerarquía de una cronica C.



Crónica

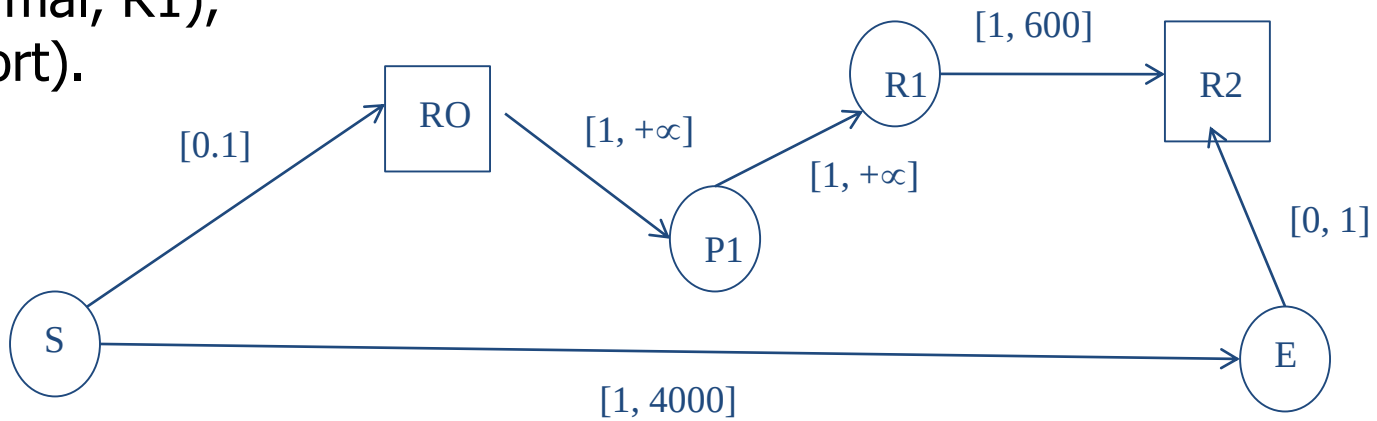
(Ghallab, 1994)	El modelo de crónica esta basado en 2 de formulas que viene de la lógica temporal:
	"**hold**" expresa que algunos atributos se mantienen durante algún intervalo de tiempo, por ejemplo:
	Hold(position (agente1, docking-site), (t5, t6)).
	"event" especifica un cambio discreto del valor de un atribuito, por ejemplo:
	Event(state (switch): (off, on), t8).
	
	Chronicle RobotLoadMachine {
	event (Robot1: (outroom, inroom), e1);
	event (Robot1: introom, outroom), e4);
	event(MachineInput: (unloaded, loaded), e2);
	event(Machine: (Stopped, Running), e3);
	e1 ≤e2;
	1 ≤e3-e2≤6;
	3 ≤e4-e2≤5;
	hold (Machine: Running, (e2, e3));
	hold (SafetyConditions: true, (e2, e3));
	when recognized { report 'successful load'; }}

Crónica

(Carrault et al., 1999), (Quiniou et al., 2001), (Carrault et al. 2003)	• Expresan crónicas usando **Prolog,** • Las relaciones temporales entre eventos son inferidas por el lenguaje de razonamiento usado: **inductive logic programming** (ILP). • Crónicas son clausulas en lógica de predicado de primer orden.

```
bigeminism :-
   qrs(R0, abnormal, _),
   p_wave(P1, normal, R0),
   qrs(R1, normal, P1),
   qrs(R2, abnormal, R1),
   rr(R1, R2, short).
```

Ejemplo de una crónica llamada bigeminism



Crónica

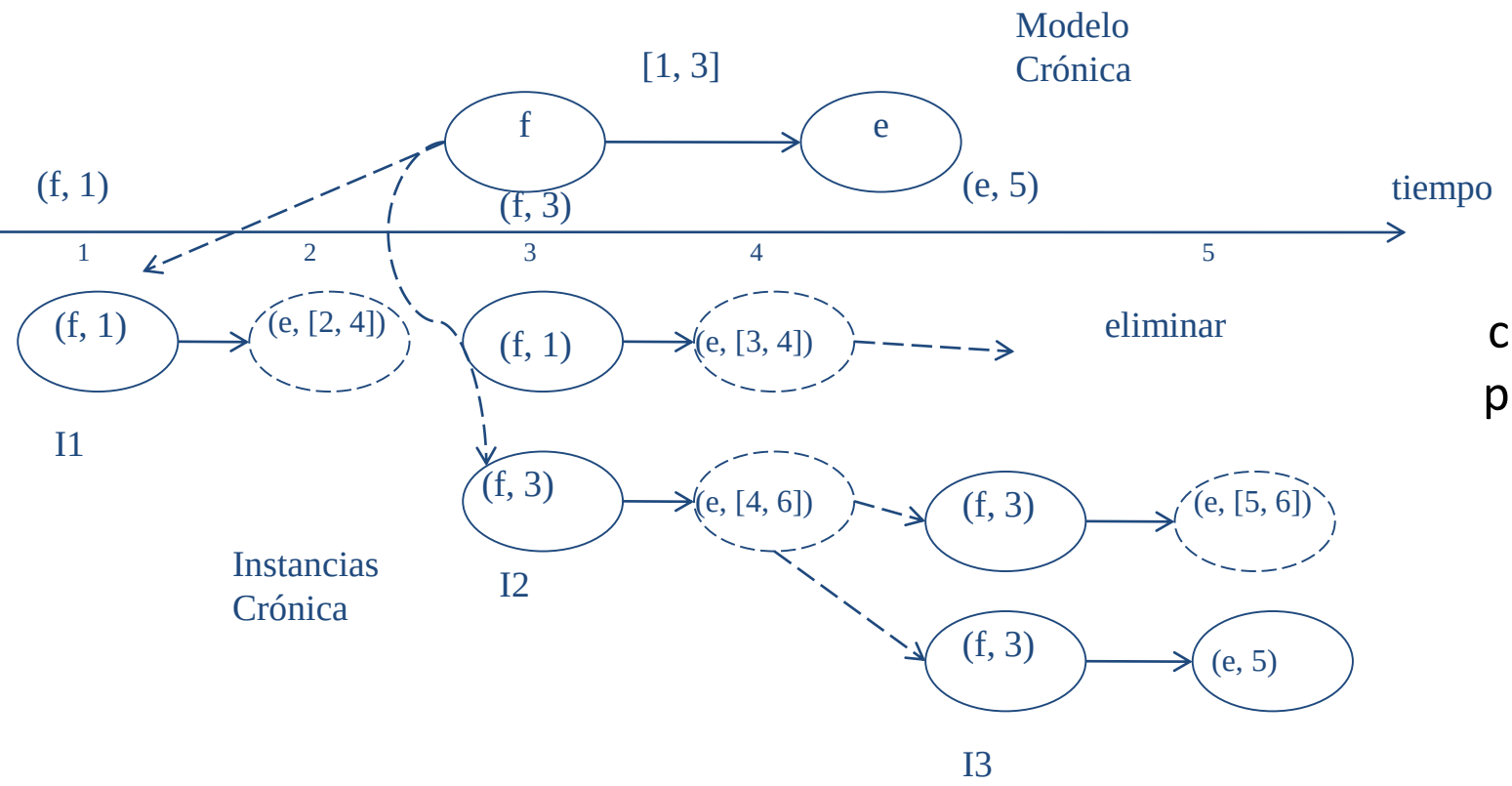
(Bertrand et al., 2007)
(Bertrand et al., 2009)
(Bertrand et al., 2009)
(Bertrand, 2009)
	• Usan un lenguaje desarrollado en ONERA (Carle et al., 1998) para describir crónicas
	
	• Los operadores son: &, || , () - [].
	
	• La gramática es:
	  • E: conjunto de eventos
	  • T: un periodo de tiempo (en unidades de tiempo)
	  • EV:: = E $\cup$ T
	  • Expresiones de crónicas:
	  C:: = E | C C | C&C | C | | C | (C C) - [C],

Ejemplos de crónicas de Bertrand et al.

1. **Crónica C1 es (A (B & C)):** patrón donde un evento A es seguido por ambos B y C , en cualquier orden,
2. **Crónica C2 es (A (B || (C D))):** patrón donde hay 2 sub-crónicas , una con evento B, y otra con evento C seguido de D. Esa sub-crónicas pueden ser reconocidas una vez reconocida A.
3. **Crónica C3 es (A||B)(C||D):** secuencia de eventos A o B, seguido por C o D.
4. **Crónica C4 es (A 5 B):** evento A seguido por 5 unidades de tiempo seguido por evento B (hay un retraso de al menos 5 unidades de tiempo entre los eventos A y B).
5. **Crónica C5 es (A B)-[5]:** noción de máximo retraso (estrictamente menos de 5 unidades de tiempo entre los eventos A y B).

Reconocimiento en las Crónicas

Algoritmo de Reconocimiento de (Dousson et al., 1994), (Dousson et al., 2007).



Instancias de crónicas creadas para los flujos de eventos

(f, 1)(f, 3)(e, 5)

Reconocimiento en las Crónicas

- Etapas del Reconocimiento:
  - Evolución de la instancia parcial de HLE.
  - Reconocimiento predictivo

- Gestión de instancias parciales de HLE
  - Con el fin de gestionar todas las instancias parciales de HLE, CRS los guarda en árboles, uno para cada HLE.
  - Cada ocurrencia de evento y cada tic tac del reloj atraviesan esos árboles con el fin de matar algunos HLE o extender algunos HLE.

- El rendimiento del CRS dependen directamente del número de casos parciales de HLE

cada evento (kn) con k el número de instancias, n el tamaño de los modelos.

Reconocimiento en las Crónicas

El procedimiento de CRP en tiempo real es:

1. Transformar los predicados “holds” en eventos prohibidos. En este caso, eventos no pueden cambiar el valor del atributo “held” durante la duración del predicado “holds”.
2. Cuando un nuevo evento se observa,
    1. Crear una nueva instancia de las crónicas posibles que coincidan con él y
    2. Actualizar la ventana de tiempo (intervalo de tiempo para esperar los próximos eventos a fin de completar un patrón de una crónica) en cada crónica que haya coincidido.
3. Detectar los "plazos" (utilizando ventana de tiempo) y "eventos prohibidos". En estos casos, las crónicas que violen eso se eliminan de la lista de las posibles crónicas a instanciar.
4. Disparar las acciones de las crónicas instanciadas por completo.

Reconocimiento en las Crónicas

Varios técnicas se han desarrollado recientemente para mejorar la eficiencia. Por ejemplo, el enfoque temporal :

- Distinguir entre *eventos muy raros y eventos frecuentes* sobre la base de un conocimiento a priori de la aplicación supervisada.

- *Concentrarse en los eventos inusuales*:

Si de acuerdo a una definición de HLE un evento raro va después de un evento frecuente, el reconocimiento sólo se iniciará con la llegada de un evento inusitado.

De esta manera el número de instancias de HLE se reduce significativamente.

Reconocimiento en las Crónicas

Dousson et al., 1994), (Dousson et al., 2007) introduce el principio de *temporal focusing*.

- El principio es:
  - Comienza la integración de eventos del nivel n + 1 sólo cuando todos los eventos de los niveles entre 1 y n se han integrado.
  - Por lo tanto, si el nivel de (e) ≤ nivel (f), el motor integrará e en t = 5, entonces buscará en el colector de todos los eventos de f entre t = 2 y t = 4. El colector encuentra (f, 3) y lo envía al motor, esto conduce a la creación de la I3 en el ejemplo en la figura anterior.

Reconocimiento en las Crónicas

Enfoque de (Carrault et al., 1999), (Quiniou et al., 2001), (Carrault et al., 2003)

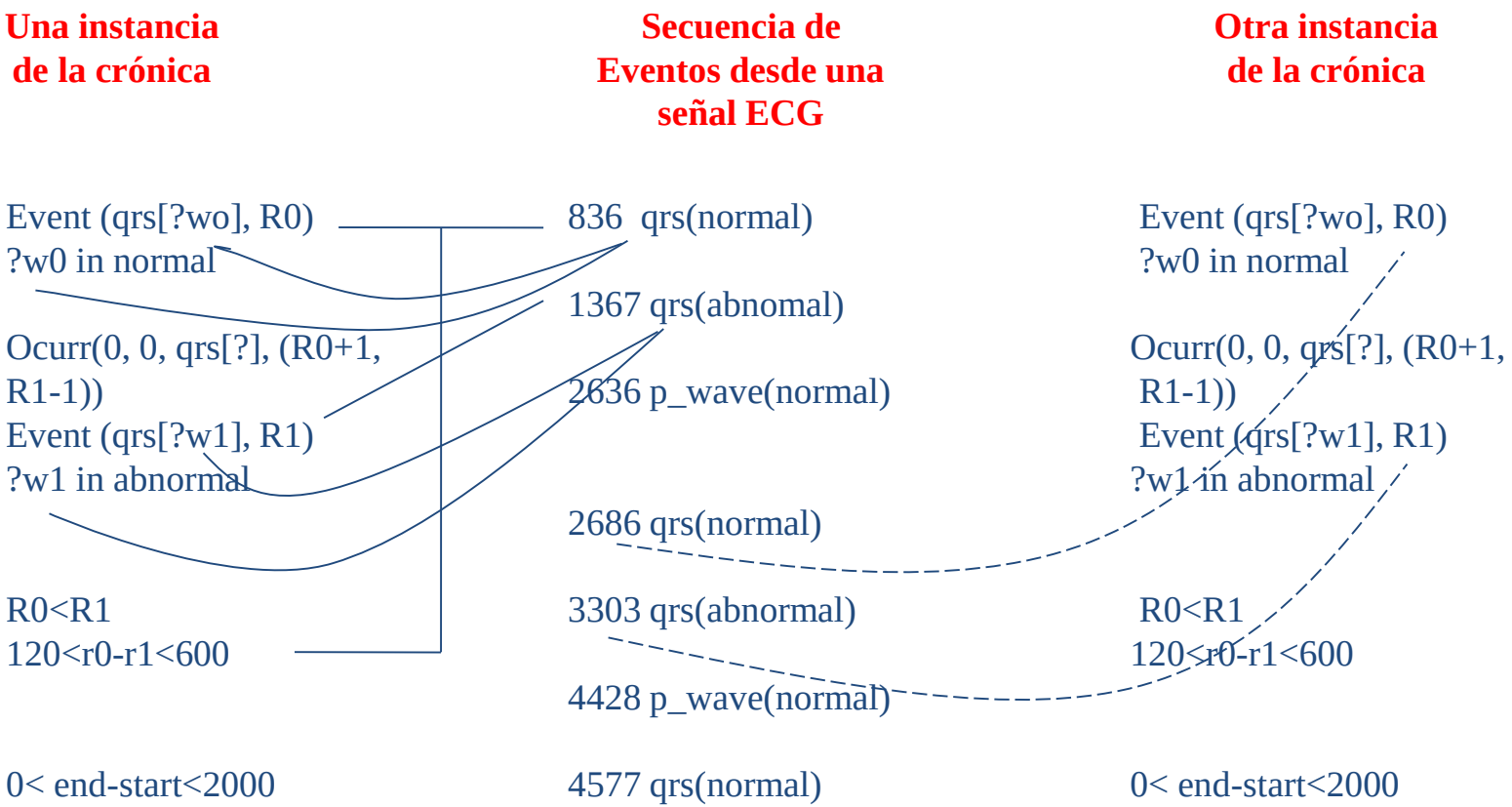
- La clase siguiente describe el concepto bigeminy usando eventos QRS:

```
class(bigeminy) :-
    qrs(R0, normal, _),
    qrs(R1, abnormal, R0),
    rr(R1, R0, short).
```

- La crónica asociada es:

```
Chronicles bigeminy(){
    Event (qrs[?w0,R0); // qrs(R0, normal, -),
    ?w0 in {normal}
    Ocurrs(0, 0, qrs [?], (R0+1, R1-1));
    Event(qrs[?w1],R1); // qrs(R1, abnormal, R0),
    ?w1 in {abnormal};
    R0<R1; // rr1(R0, R1, short),
    120 < R1-R0 < 600;
    0 < end-start < 2000}
```

Reconocimiento en las Crónicas



Reconocimiento en las Crónicas

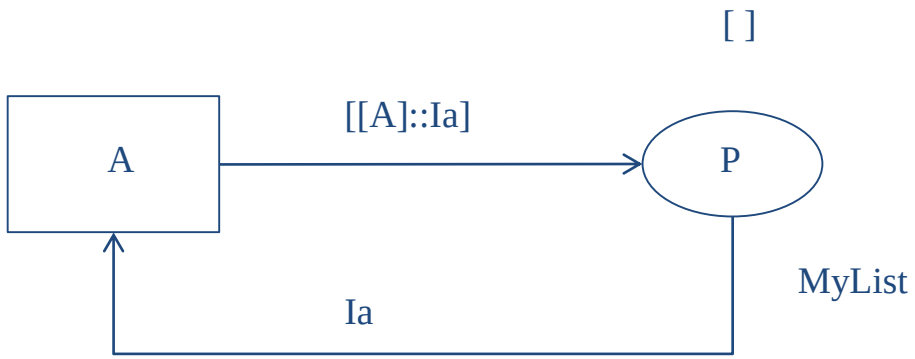
En (Bertrand et al., 2007, 2008), 2009) proponen un procedimiento para reconocer crónicas basada en redes de petri coloreadas.

- Estas redes permiten reconocer un evento y guardar un histórico (preserva todos los eventos).
- Permite la composición de los operadores del lenguaje ONERA por la composición de subredes de petri asociadas con los diferentes operadores.
- Las redes de petri se complementan bien con las necesidades del reconocimiento de crónicas, en particular múltiples ocurrencias(crónicas con repeticiones), la posibilidad de trazar los eventos que contribuyen a un reconocimiento, y la ausenciande subcrónicas

Los principios del modelado con redes de petri coloreadas son:

- Asocian una transición a cada evento que debería ser detectado, y esa transición es lanzada cuando el evento ocurre
- Sitios son usados para guardar (en un simple token) una lista de todas las instancias de crónicas identificadas (parciales o totales).
- Para cada operador del lenguaje de crónica, ella provee un modelo de red de Petri coloreada para su reconocimiento,

Reconocimiento en las Crónicas



La red de petri coloreada que reconoce un evento A.

Token en sitio P contiene la lista de eventos reconocidos (al inicio vacía [])

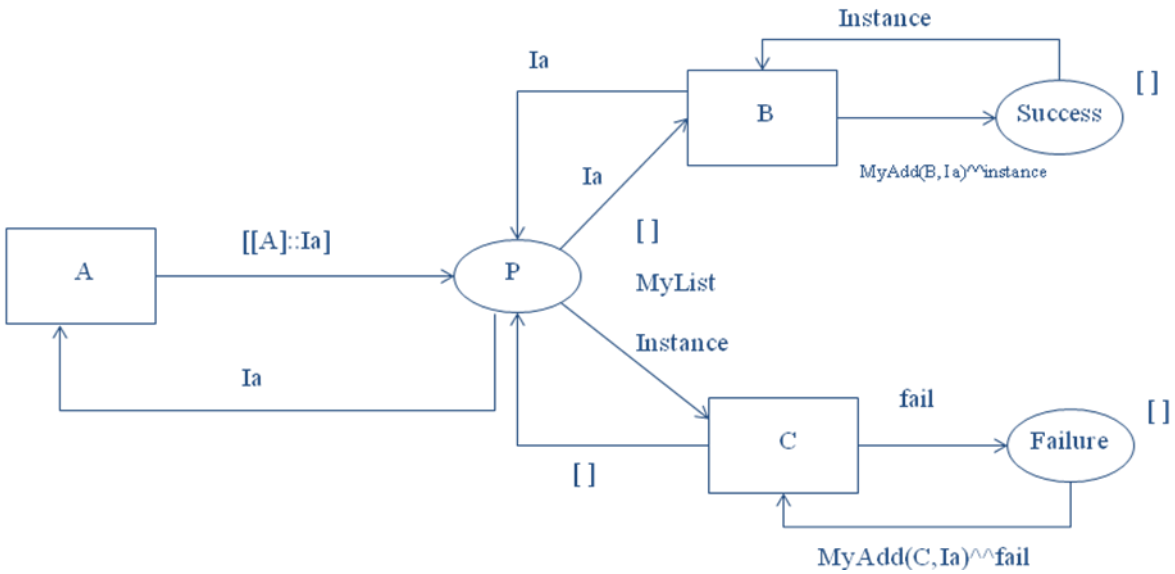
La ocurrencia del evento A es modelado por la activación de la transición A, Token en el sitio P es modificado actualizando la lista [A]

- La siguiente ejecución muestra la evolución de la red después de activarse dos veces la transición A. El resultado es que A fue reconocida dos veces.

$$[\ [\ ]\ ]\ \xrightarrow{A}\ [\ [[A]]\ ]\ \xrightarrow{A}\ [\ [[A]\ ,\ [A]]\ ]$$

Reconocimiento en las Crónicas

Ejemplo de reconocimiento de la crónica C3: (A B) – [C].



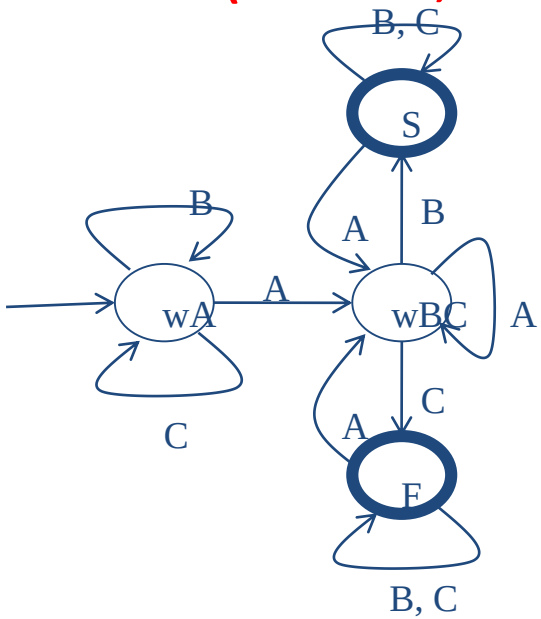
La transición A esta conectada al lugar P. La transición B modela la ocurrencia del evento B, sus entradas son ambos, sitio P (contiene ocurrencia evento A), y sitio Success (contiene la ocurrencia de [A,B], que es la crónica C3. Activando la transición B resulta en ”actualizar” todas las instancias [A,B] en Success (se hace con la función MyAdd), como se ve en siguiente flujo de eventos A A B.

P		*[[A]]*		*[[A] , [A]]*		*[[A] , [A]]*
Success	*A→*	*[]*	*A→*	*[]*	*B→*	*[[A,B] , [A,B]]*
Failure		*[]*		*[]*		*[]*

Reconocimiento en las Crónicas

Enfoque de reconocimiento de crónica basado en autómatas de estado finito (Bertrand, 2007).

El autómata reconoce una sola vez la crónica!!



Es por esta razón por la que se han utilizado otros métodos de autómatas (Bertrand et al, 2007) (counter automata and duplicating automata, diseñado para reconocer las múltiples ocurrencias de crónicas

- El estado de wA está esperando que el evento A ocurra, el estado wBC espera a los eventos B o C, en el estado S la crónica se identificó con éxito, mientras que estado F es caundo no se reconoce (a causa de la ocurrencia del evento C).
  - Si el flujo de entrada es A C B A B, la crónica debe ser reconocido una vez, que es el caso como se muestra

    → wA $_A$→ wBC $_C$→ F B→ F $_A$→ wBC $_B$→ S

  - Si el flujo de entrada es A A B, la crónica debe ser reconocido dos veces, pero el autómata sólo encuentra una ocurrencia, como se muestra en la ejecución a continuación.

    → wA $_A$→ wBC $_A$→ wBC $_B$→S

Reconocimiento en las Crónicas Distribuidas (Aguilar et al., 2013)

- El reconocimiento de una crónica distribuida debe ser completamente distribuido.

- Para ello, se coloca un sistema de reconocimiento en cada componente del sistema distribuido, el cual será responsable por reconocer las subcrónicas en ese sitio.

- Los BE le permiten a los sistemas de reconocimiento local inferir información desde sus vecinos, y usarla para reconocer las sub-crónicas locales, generar eventos a sus vecinos, y en general, esparcir lo inferido.

- El BE la da a cada sitio una global visión del sistema

Reconocimiento en las Crónicas Distribuidas (Aguilar et al., 2013)

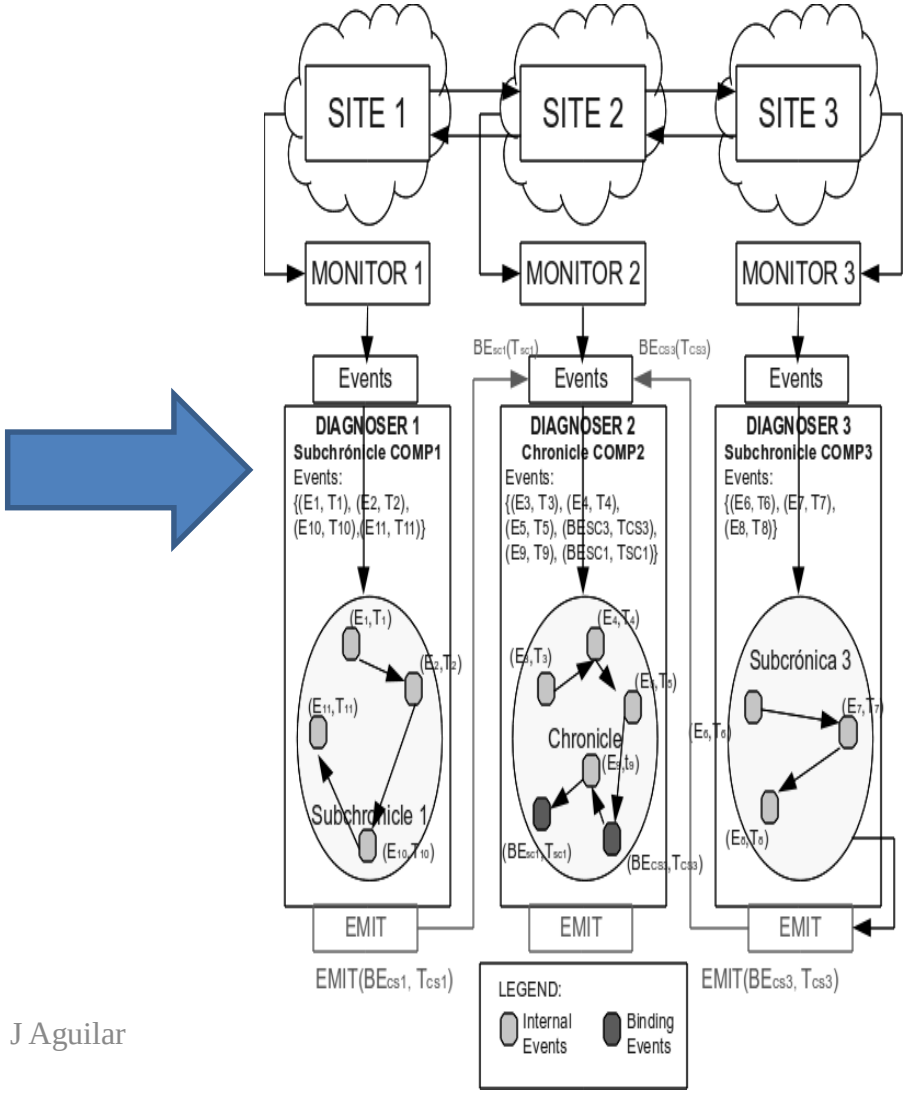
Definición como una crónica C se puede descomponer en n Subcrónicas (SC), el reconocimiento de una crónica puede ser realizada por una de sus subcrónicas SC_i, reconociendo sus eventos (Eac_i, Tac_i), uniéndolo con el reconocimiento parcial de las otras subcrónicas (SC_j $\forall$ j=1,n | j≠i) de los otros eventos (para ello, las otras subcrónicas deben enviarle un mensaje para informarle el reconocimiento de sus eventos).

Así, la subcrónica SC_i reconoce la crónica C(E, T):

$$C(E, T) = \{(Eac_i, Tac_i), UNION_{j=1, n \mid j \neq i}(SC_j(Eac_j, Tac_j))\}$$

Reconocimiento en las Crónicas Distribuidas (Aguilar et al., 2013)

Chronicle Subchronicle 1 { Events{ event(E_1, T_1), event(E_2, T_2), event(E_{10}, T_{10}), event(E_{11}, T_{11}) } Constrains{ T_2-$T_1 \leq C_1$ T_{10}-$T_2 \leq C_2$ T_{11}-$T_{10} \leq C_3$ } When recognized{ Emit event(BE_{SC1},T_{SC1}) to Diagnoser 2 } }	Chronicle Subchronicle 2 { Events{ event(E_3, T_3), event(E_4, T_4), event(E_5, T_5), event(BE_{SC3}, T_{SC3}), event(E_9, T_9), event(BE_{SC1}, T_{SC1}) } Constrains{ T_4-$T_3 \leq C_4$ T_5-$T_4 \leq C_5$ T_{SC3}-$T_5 \leq C_6$ T_{SC1}-$T_9 \leq C_8$ } When recognized{ Create log(Fault 1) } }	Chronicle Subchronicle 3 { Events{ event(E_6, T_6),event(E_7, T_7), event(E_8, T_8) } Constrains{ T_7-$T_6 \leq C_9$ T_8-$T_7 \leq C_{10}$ } When recognized{ Emit event(E_{SC3},T_{SC3}) to Diagnoser 2 } }



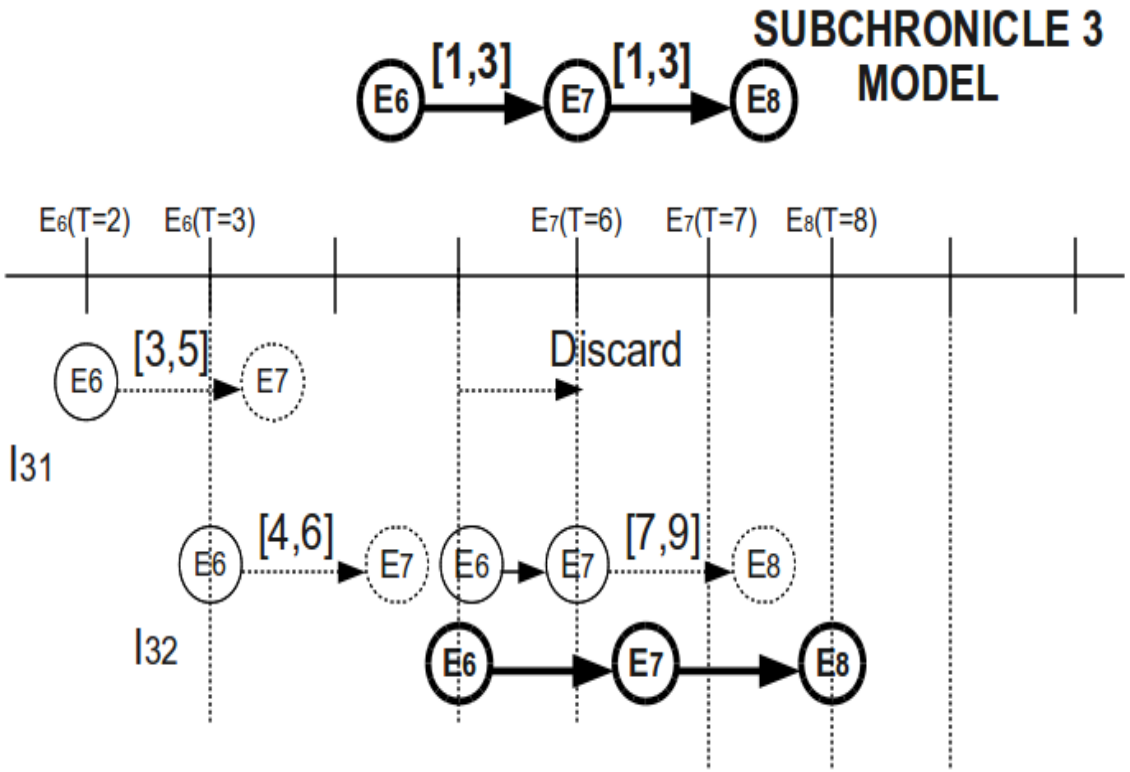
Reconocimiento en las Crónicas Distribuidas (Aguilar et al., 2013)

El programa CRS local

```
FOR EACH event received DO
  diagnoser.addEvents(chronicle c, event)
  IF chronicle c is recognized THEN
    DO c.executeAction()
```

```
diagnoser.addEvents (chronicle c, event)
FOR EACH chronicle c where event is the first event
DO
  instances = c.instances();
  FOR EACH current instances DO
    IF event match instances and temporal
       constraints are not violated THEN
         instances.addEvent(event)
    IF event match instances and temporal
       constraints are violated THEN
         instances.discard()
    IF temporal constraints are violated THEN
       instances.discard()
```

Reconocimiento en las Crónicas Distribuidas



Uso de las Crónicas

- Se utilizan para **modelar actividades** a ser detectadas, porque no son deseadas o son peligrosas.
- Una **actividad se describe** mediante una combinación de ocurrencias de eventos.
- El objetivo es identificar **todas las instancias de la crónica** dentro de un flujo de eventos observados
- **La identificación de la crónica** se logra a través de la comparación entre los eventos que van ocurriendo y los eventos que describen la crónica,
- Además, puede ser de interés guardar la información que indica **cuales acontecimientos en el flujo contribuyen** al reconocimiento de la crónica, ya que puede ayudar a encontrar las causas del fenómeno observado.

Agente Gladiador

Agente que juega COLISEUM de forma inteligente

- este juego consiste en un duelo entre dos personajes, el agente y un jugador, los cuales lucharan a muerte en la arena usando espadas.

- Durante el juego aparecerá una botiquín de forma aleatoria: conseguir la curación que él proporciona es un objetivo importante del juego

Crónicas

Relaciones

1-Short(d,e)

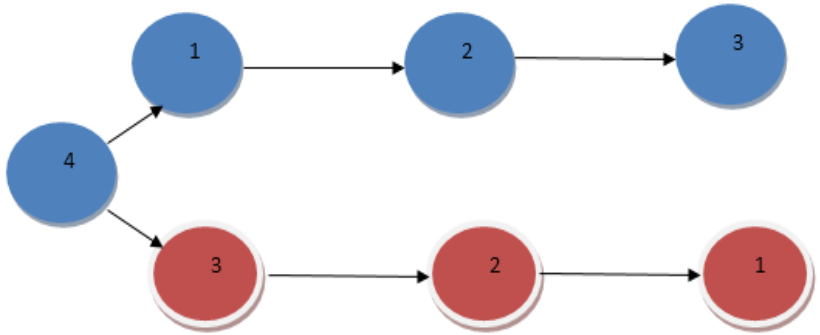
2-Mid(d,e)

3-Long(d,e)

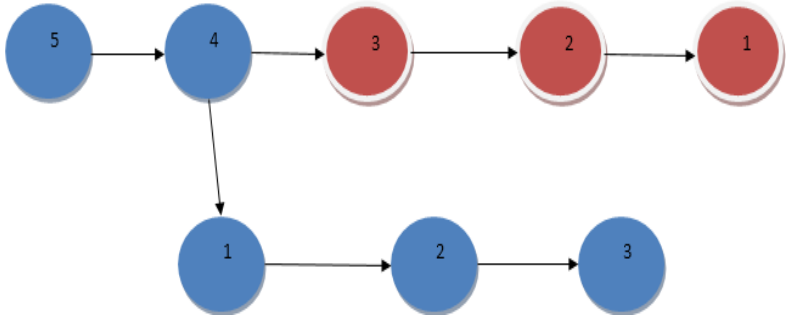
4-lowHealth(g)

5-avaLife(B)

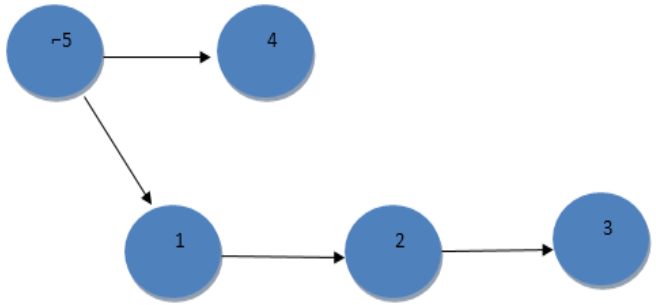
Atacando



Recuperando:



Debilitado



Sentencias en Prolog de las Crónicas

Lógica de Predicados de Primer Orden

- ⌐ lowHealth(y,t0) ^ (short(y,z,t1) ^ mid(y,z,t2) ^ long(y,z,t3)) ^ (long(x,y,t1) ^ mid(x,y,t2) ^ short(x,y,t3)) ^ (t0<t1<t2<t3) => atacando(y)
- ⌐avaLife(B, t0) ^ lowHealth(y,t1) ^ (short(x,y,t1) ^ mid(x,y,t2) ^ long(x,y,t3)) => Debilitado(y)

Prolog

atacando(PosAgenteX, PosAgenteY, PosRivalX, PosRivalY, PosVidaX, PosVidaY, PosAgente2X, PosAgente2Y, PosRival2X, PosRival2Y, PosVida2X, PosVida2Y, PosAgente3X, PosAgente3Y, PosRival3X, PosRival3Y, PosVida3X, PosVida3Y, VidaRival) :- not(lowHealth(VidaRival)), long(PosAgenteX, PosAgenteY, PosRivalX, PosRivalY), mid(PosAgente2X, PosAgente2Y, PosRival2X, PosRival2Y), short(PosAgente3X, PosAgente3Y, PosRival3X, PosRival3Y), short(PosRivalX, PosRivalY, PosVidaX, PosVidaY), mid(PosRival2X, PosRival2Y, PosVida2X, PosVida2Y), long(PosRival3X, PosRival3Y, PosVida3X, PosVida3Y).

IAm

Conformada por artefactos inteligentes que interactúan entre si para alcanzar objetivos del ambiente

- El sistema de comunicación debe aprender cuales artefactos interactúan en un ambiente dinámico, para responder a:
  - Como comunicarse,
  - Con quien cooperar,
  - Como delegar y coordinar tareas, etc,

de manera auto-organizada

⇩

Debe ser un proceso cognitivo distribuido

- **Sistemas de comunicación autónomos** aprenden crónicas que describen situaciones en el ambiente basada en el conocimiento sobre la plataforma de comunicación, requerimiento de usuarios, etc.

Crónicas para auto-adaptar estrategias en Sistemas de Comunication

Reconocimiento de diferentes situaciones es necesario para reconfigurar red.

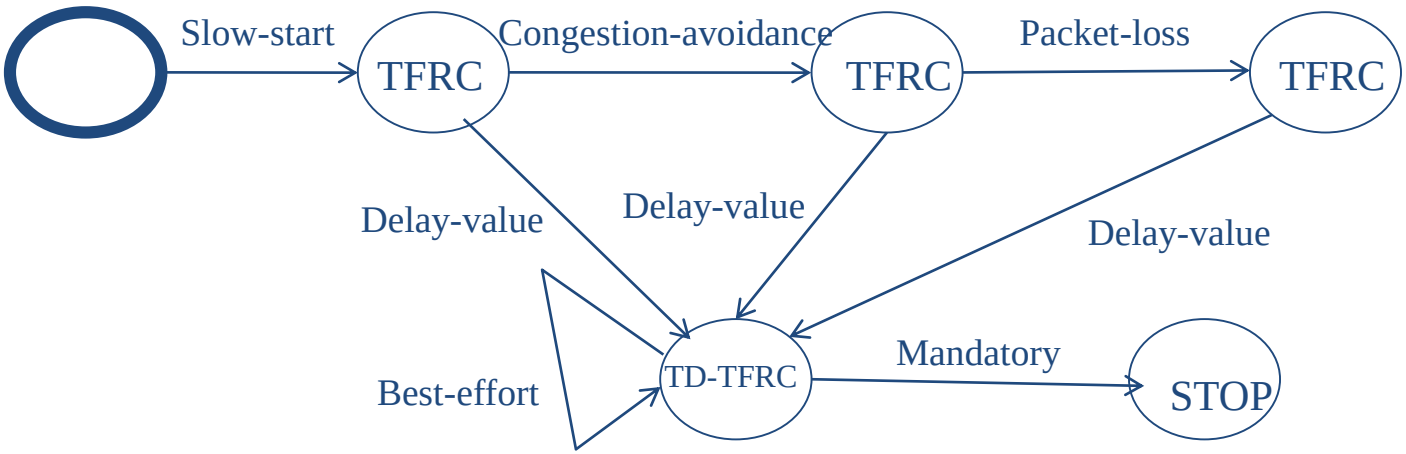
- **Una crónica para cada situación a identificar**: detección de perdida, anormal retraso , etc.
- **Ejemplo de situación**: detectar perdida de datos o notificación de congestión (ECN),
- **La crónica es:**

```
Message packet [ ?num_loss, ?seqn]{
}
Chronicle loss [?num_loss]
{
occurs ((3,3), packet [ ?num_loss,?seqn], (t1,t2))
?seqn>?num_loss
t1<t2
when recognized {
emit event(packet_loss,t2);
}
}
```

Crónicas para auto-adaptar estrategias en Sistemas de Comunication

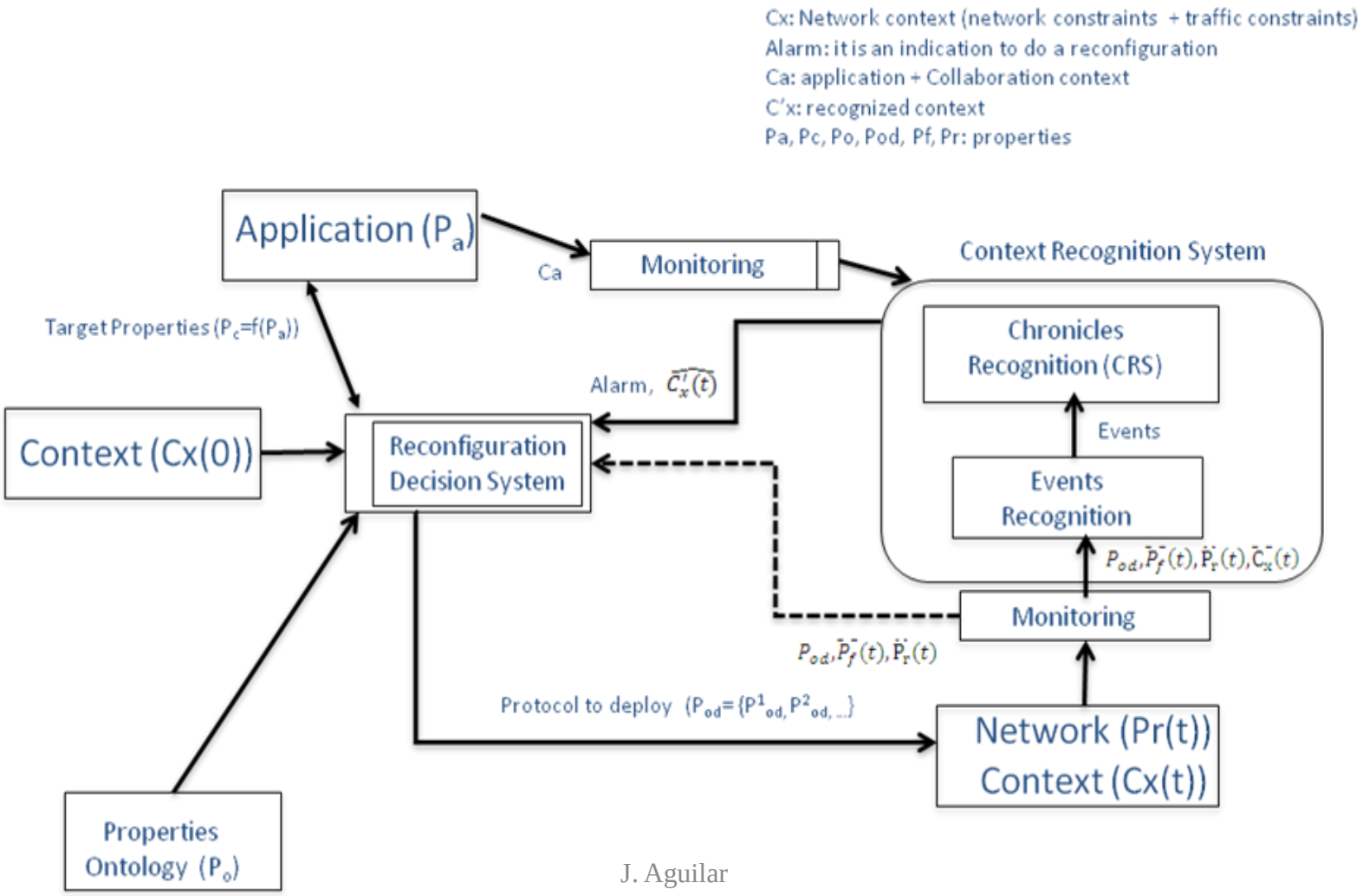
- El mecanismo de auto-adaptación es guiado por el mecanismo de reconocimiento de situaciones basado en una maquina de estados

- **Mecanismo de auto-adaptación**:
  - **Maquina de estados de mecanismos de transporte** (TFRC o TCP Friendly Rate Control, TD – TFRC o Time-constrained and Differentiated TFRC, MP – TFRC, Multipath TFRC) y
  - **Transiciones según reconocimiento de crónicas**,



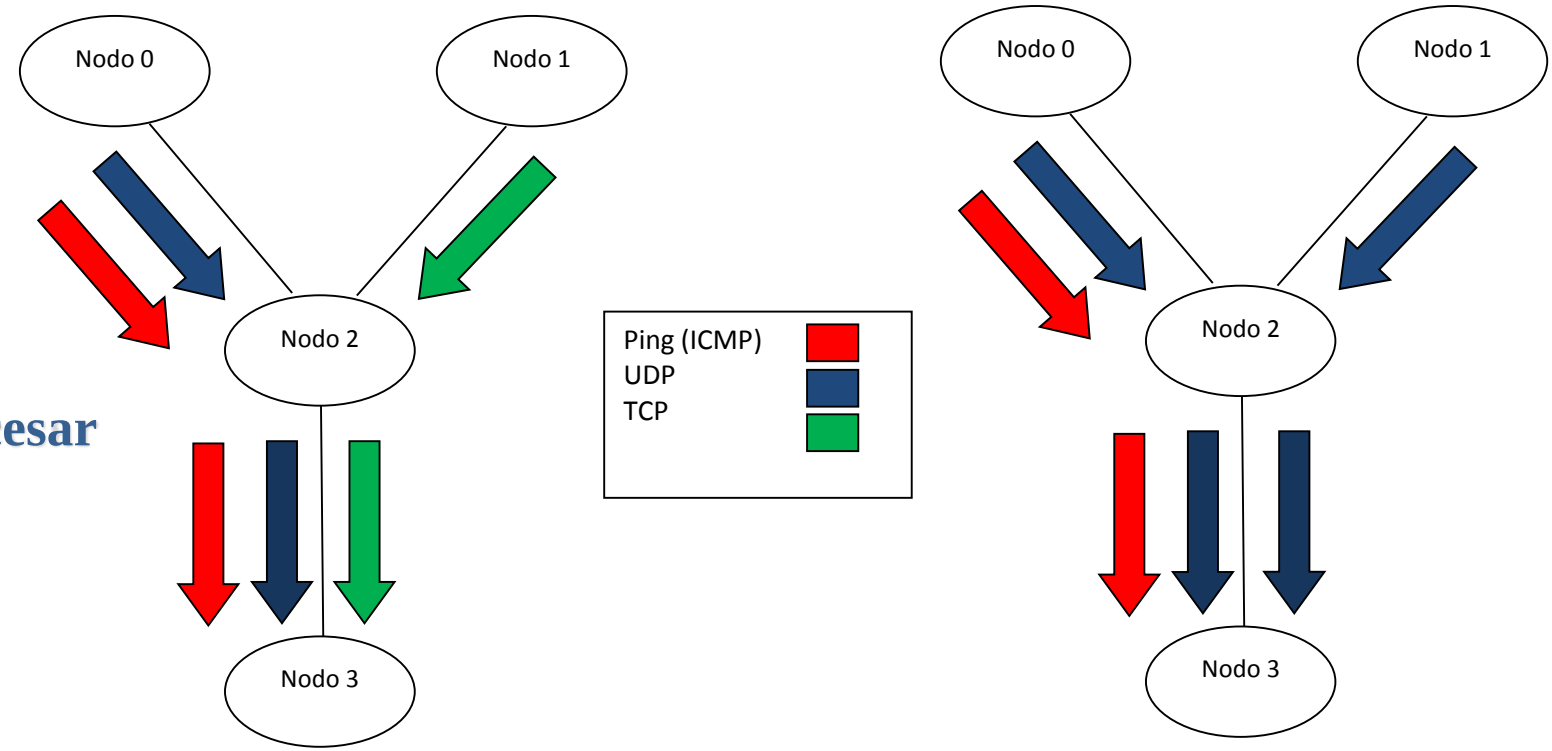
- Reconocer situaciones con crónicas
  - En termino de diagnostico: síntoma-causa
  - Diagnostico « en línea »
- Mecanismo de aprendizaje para adaptar los protocolos a un contexto
  - Aprendizaje de la base de crónicas
  - Aprendizaje de los parámetros de las crónicas
- Análisis de diagnosticabilidad para garantizar la coherencia y completitud de la base de crónicas
  - Verificación de propiedades del tipo « crónicas exclusivas »,
  - Discriminar todas las situaciones de interés

Diagnosis for AdaptIve Strategies in collaborative Systems



1. Simular situaciones de ongestión a modelar

Preprocesar



Tres congestiones en cada simulación

Preprocesar

2. Extraer descriptores : *retardo , %Perdida*

- Script de calculo para ventanas de 10 paquetes transmitidos

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
Action	Instant	Nsrc	Ndest	Type	Taille	Flag	IDflux	@src	@dest	N°Seq	IDpaquet	Timestamp

Archivo de trazas de NS-2

2. Extraer descriptores : *retardo , %Perdida*

- Script de calculo para ventanas de 10 paquetes transmitidos

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
Action	Instant	Nsrc	Ndest	Type	Taille	Flag	IDflux	@src	@dest	N°Seq	IDpaquet	Timestamp

Archivo de trazas de NS-2

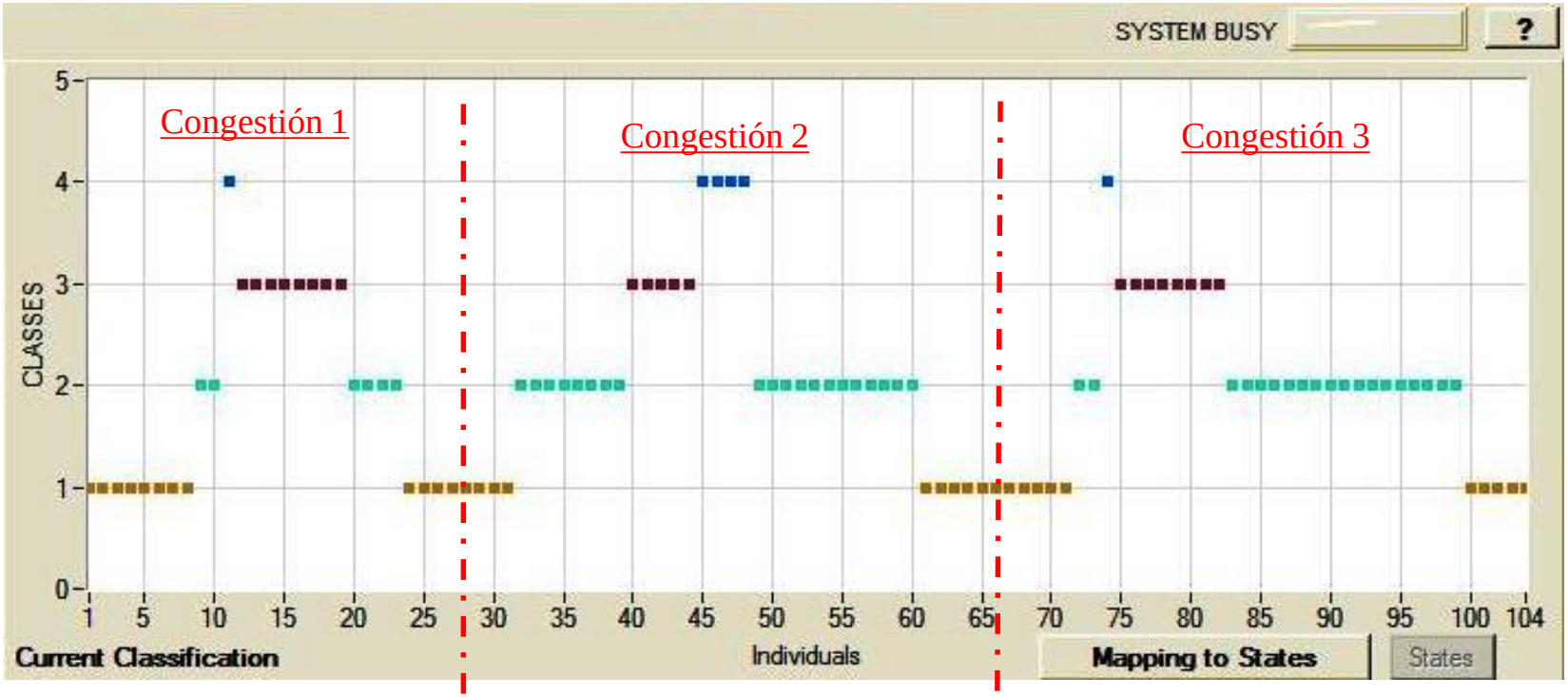
- **Resultado:**

[0]	[1]
N° Individu	Instant

[0]	[1]	[2]	[3]
gigue	Délai	Débit	% perte

Archivo de resultados

Diagnosis for AdaptIve Strategies in collaborative Systems



Secuencia N°1:

E1 (C1,C2)
E2 (C2,C3)
E3 (C3,C2)
E4 (C2,C1)

Secuencia N°2:

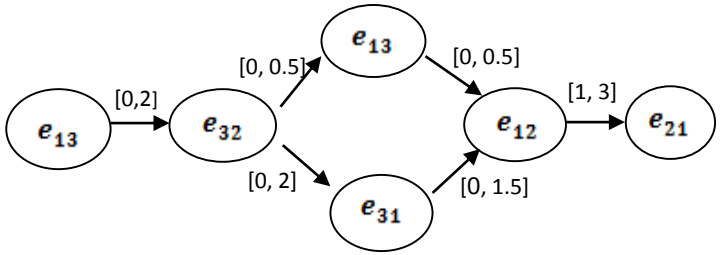
E1 (C1,C2)
E2 (C2,C3)
E5 (C3,C4)
E6 (C4,C2)
E4 (C2,C1)

Secuencia N°3:

E1 (C1,C2)
E7 (C2,C4)
E8 (C4,C3)
E3 (C3,C2)
E4 (C2,C1)

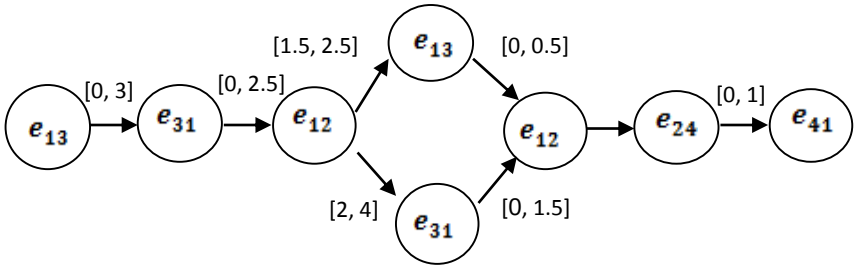
Especificar patrones temporales (crónicas)

Construir crónicas:

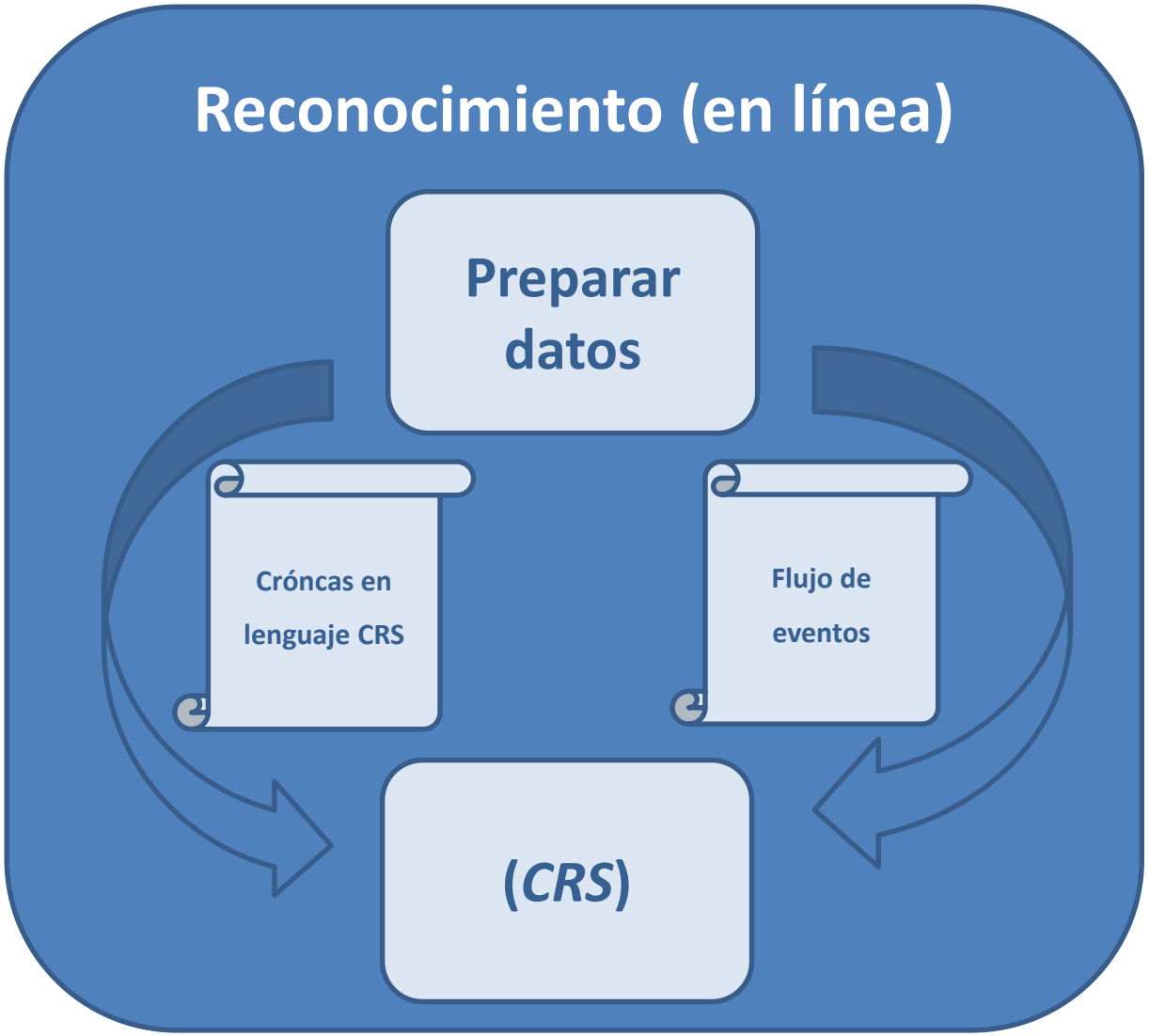


Congestión

Perdida de paquetes

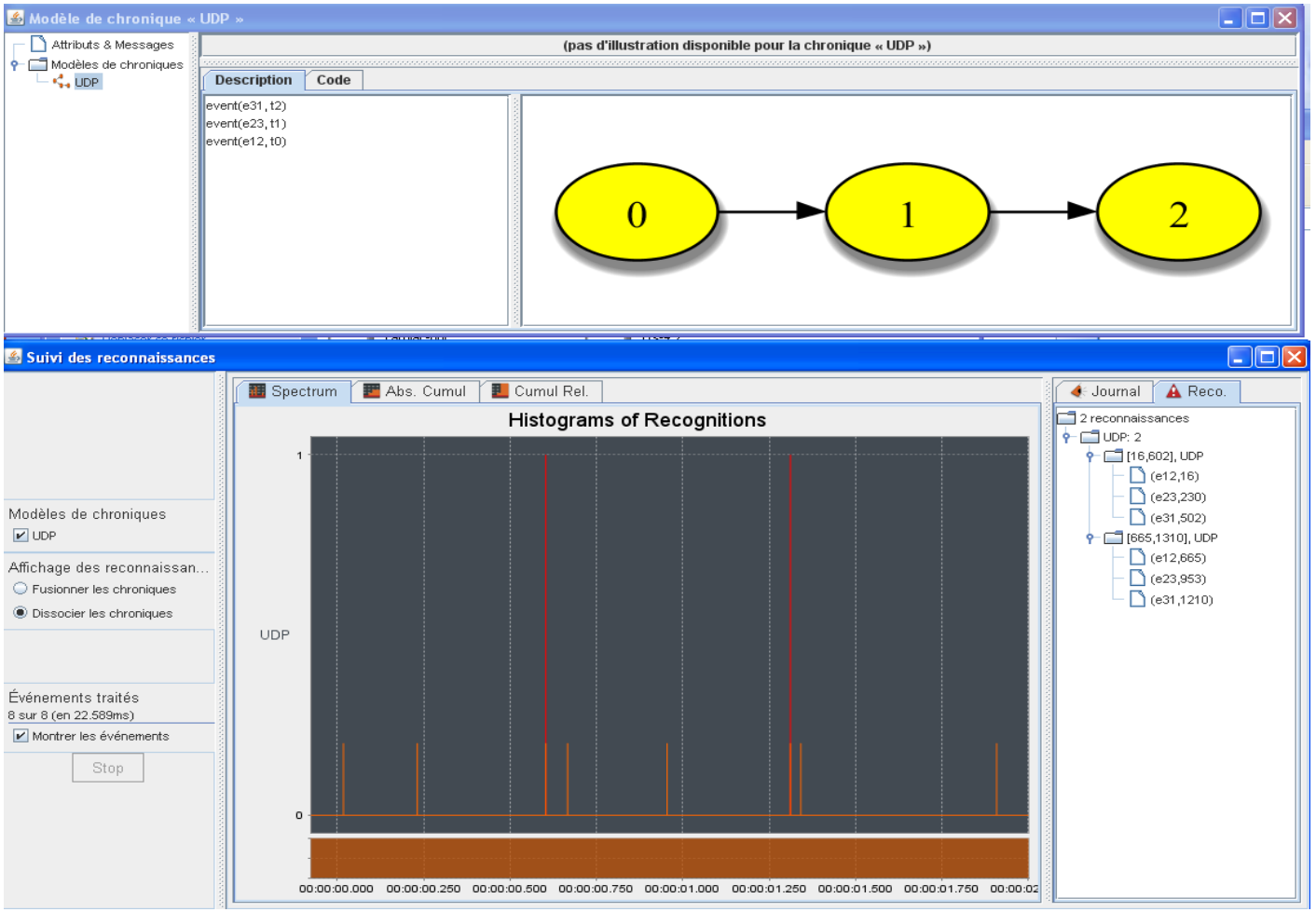


Especificar patrones temporales (crónicas)



Especificar patrones temporales (crónicas)

Resultado: *flujo1 : UDP/UDP*



Aplicaciones SOA tolerantes a fallas

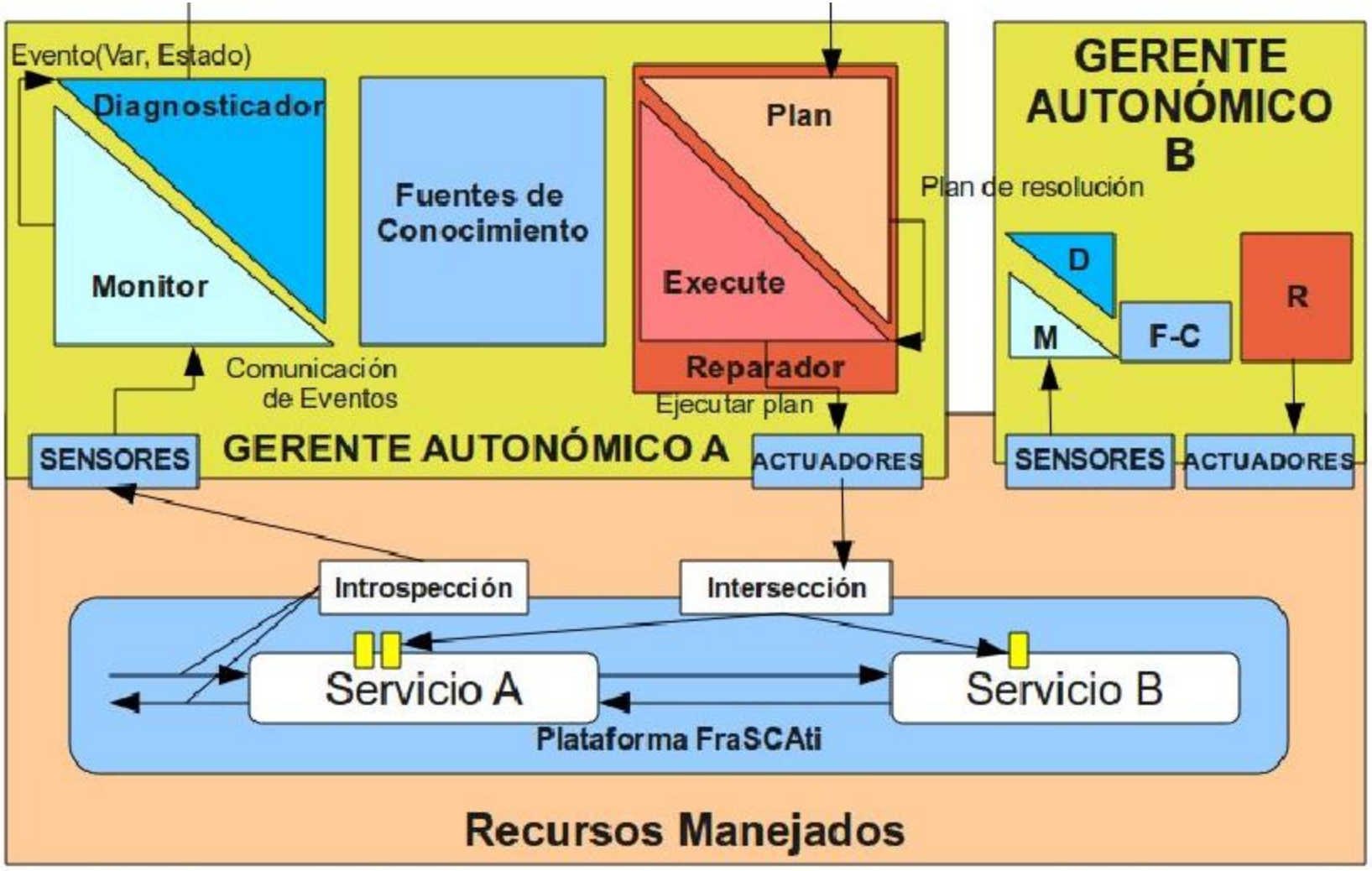
Un ejemplo muy común de aplicación SOA de comercio electrónico, el cual está compuesto por tres procesos de negocios (los cuales van a constituir nuestros servicios):

- Shops.
- Supplier.
- Warehouse.

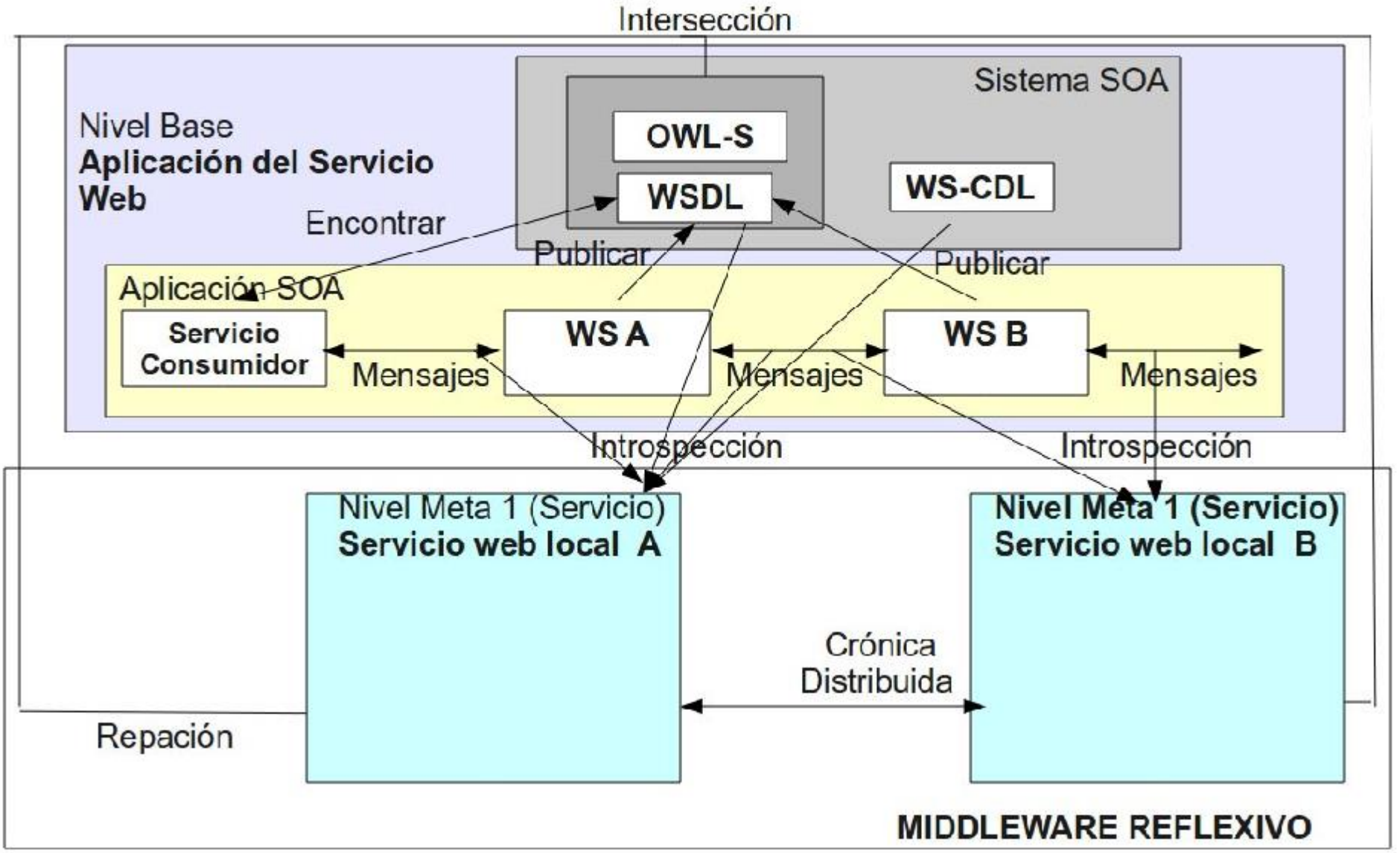
El comportamiento clásico de coreografía de esta aplicación:

(1) SuppListOut: Shop envía la lista de los productos que requiere al supplier.
(2) SuppItemIn: Supplier comprueba la disponibilidad de los productos en su depósito invocando a Warehouse.
(3) SuppItemOut: Warehouse proporciona la respuesta sobre la lista de productos en el depósito al Proveedor, el cual debe contener al menos un producto.
(4) SuppListIn: Supplier notifica a shop de los productos que puede ofrecer.

ARMISCOM
Computación autonómica



Arquitectura ARMISCOM



Caso de Estudio

Diseño de las Crónicas

•Diagnosticador Shop:

Eventos: $\{E_{11}^{+}(t_{11}, var_{11}), E_{12}^{-}(t_{12}, var_{12}), E_{13}(t_{13}, var_{13})\}$

Eventos enlazadores: $\{(E_{11}^{+}(t_{11}), var_{11}, E_{21}^{-}(t_{21})), (E_{12}^{-}(t_{12}), var_{12}, E_{26}^{+}(t_{26}))\}$

•Diagnosticador Supplier:

Eventos: $\{E_{21}^{-}(t_{21}, var_{21}), E_{22}(t_{22}, var_{22}), E_{23}^{+}(t_{23}, var_{23}), E_{24}^{-}(t_{24}, var_{24}), E_{25}(t_{25}, var_{25}), E_{26}^{+}(t_{26}, var_{26})\}$

Eventos enlazadores: $\{(E_{21}^{-}(t_{21}), var_{21}, E_{12}^{+}(t_{12})), (E_{23}^{+}(t_{23}), var_{23}, E_{31}^{-}(t_{31})), (E_{24}^{-}(t_{24}), var_{24}, E_{35}^{+}(t_{35})), (E_{26}^{+}(t_{26}), var_{26}, E_{12}^{-}(t_{12}))\}$

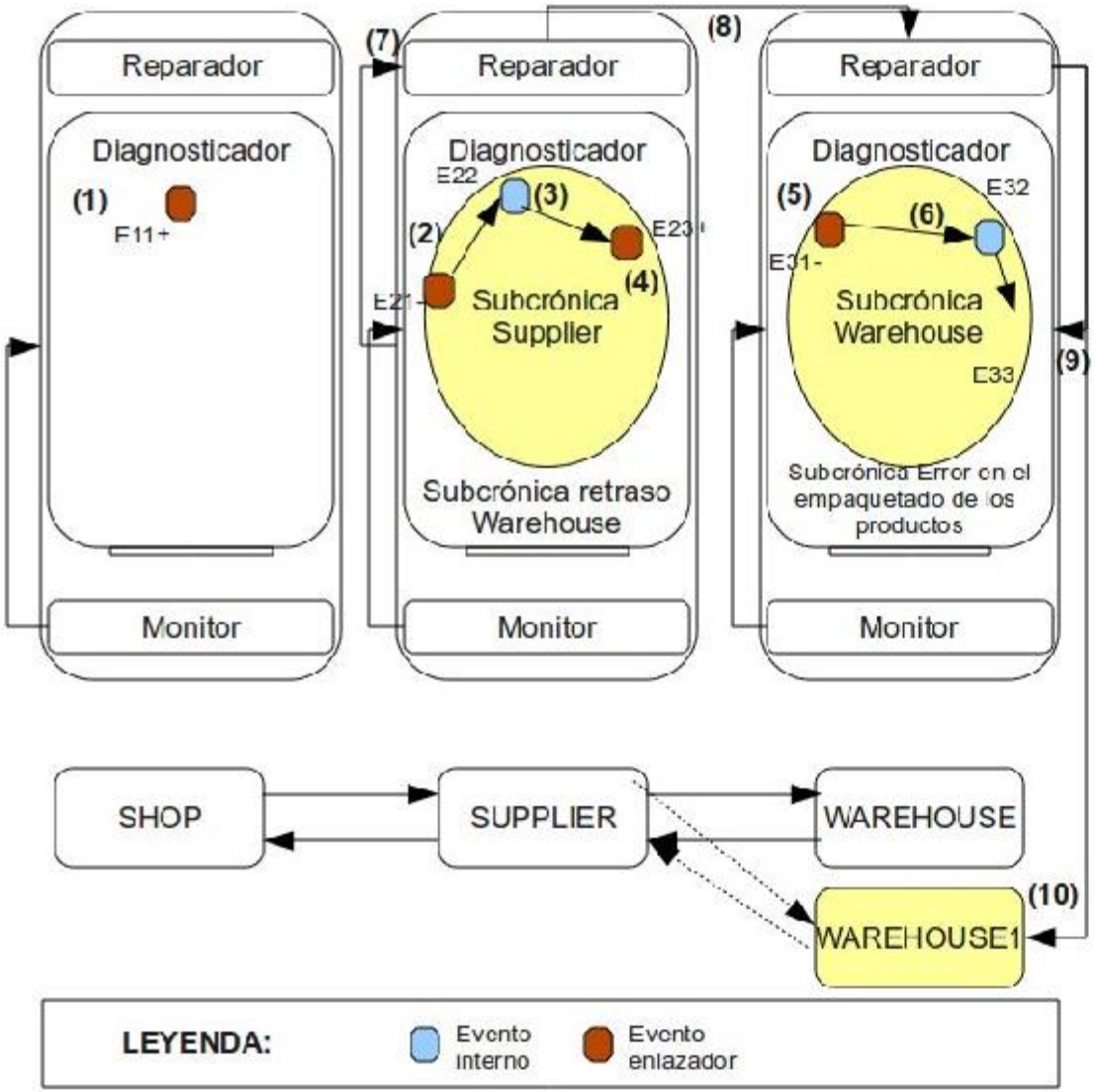
•Diagnosticador Warehouse:

Eventos; $\{E_{31}^{-}(t_{31}, var_{31}), E_{32}(t_{32}, var_{32}), E_{33}(t_{33}, var_{33}), E_{34}(t_{34}, var_{34}), E_{35}^{+}(t_{35}, var_{35})\}$

Eventos enlazadores: $\{(E_{31}^{-}(t_{31}), var_{31}, E_{23}^{+}(t_{23})), (E_{35}^{+}(t_{35}), var_{35}, E_{24}^{-}(t_{24}))\}$

Caso de Estudio

Detección de la Falla retraso del servicio Warehouse



Eventos Shop: $E_{11}^{+}(t_{11}=1.0, \neg Err)$

Eventos Supplier: $E_{21}^{-}(t_{21}=2.0, \neg Err), E_{22}(t_{22}=2.1, \neg Err), E_{23}^{+}(t_{23}=2.2, lp=3, \neg Err). E_{24}^{-}(t_{24}=8.5, lp=3, \neg Err)$

Eventos Warehouse: $E_{31}^{+}(t_{31}=2.5, lp=3, \neg Err), E_{32}(t_{32}=2.6, lp=3, \neg Err)$.

Crónicas genéricas para aplicaciones SOA Tolerantes a Fallas

Crónica para falla por Time-out

Subchrocicle S_1 Time-out {
Events{
event(E_1, T_1)
}
Constrains{
}
When recognized{
emit event(BE_{E1call}, T_{E1call}) to S_3 diagnosser
}
}

Subchrocicle S_2 Time-out {
Events{
event(E_2, T_2)
event($BE_{Timeout}$, $T_{Timeout}$)
}
Constrains{
$T_{Timeout} - T_2 > 0$
}
When recognized{
repair invoke(S_2, 'Time-out')
}
}

Subchrocicle S_3 Time-out {
Events{
noevent(E_4, ($T_{E1call} - \Delta T$, $T_{E1call} + \Delta T_1$))
event(BE_{E1call}, T_{E1call})
}
Constrains{
}
When recognized{
emit event($BE_{Timeout,}$ $T_{Timeout}$) to S_2 diagnosser
}
}

Un buen artículo para terminar

Temporal Representation and Reasoning in Artificial Intelligence: A Review

A K. PANI
Information Technology and Systems Area
XLRI, Jamshedpur-831 001, India

G. P. BHATTACHARJEE
Department of Mathematics, IIT, Kharagpur-721 302, India

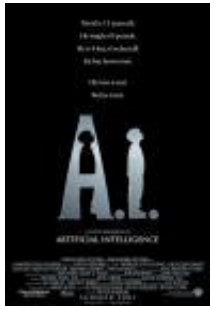
(Received August 1998, revised and accepted July 2000)

Abstract— The explicit representation and reasoning about time is an important problem in many areas of artificial intelligence Over the last 10–15 years, it has been attracting the attention of many researchers Several temporal reasoning systems, differing in design issues related to ontology of time, underlying temporal logic, temporal constraints used and algorithms employed, have been developed In this survey, important representational issues which determine a temporal reasoning system are introduced In particular, several important notions like change, causality, actions are described in terms of time For each issue different choices available in the literature are discussed The most influential approaches to temporal reasoning in artificial intelligence are analyzed in terms of these major representational issues

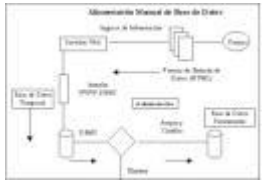
Ontología

Ontología

Explicación sistemática de la Existencia



“Especificación explícita de una conceptualización” Gruber(1993)



“Base de datos que describe los conceptos de un dominio especifico, sus propiedades y cómo estos conceptos se relacionan entre si” Weigand(1997)

Ontologías

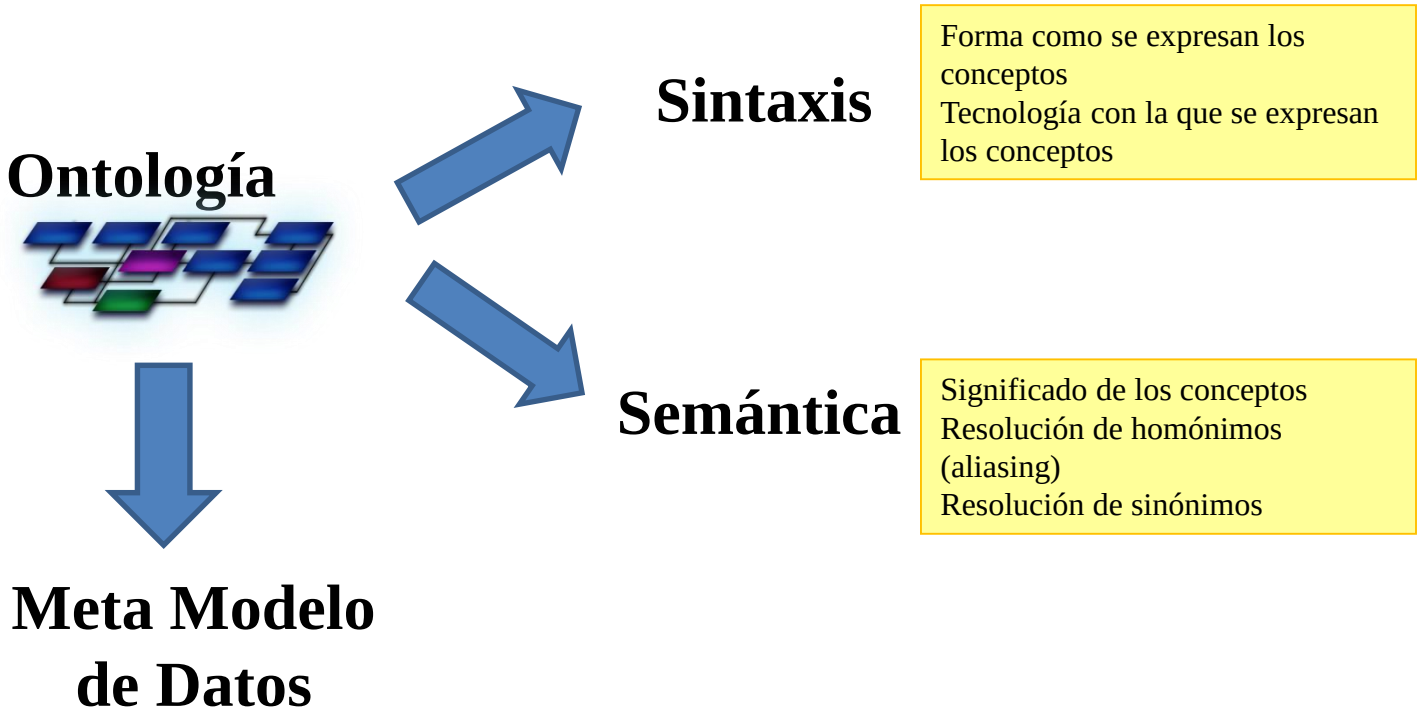
"An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary" [Neches 91]

Una *ontología* es un sistema de conceptos (o un vocabulario) usado como elemento básico (primitivo) para la construcción de sistemas basados en el conocimiento.

Ontologías

- Es una representación, a través de un modelo de datos, de un conjunto de conceptos en un dominio y de las relaciones entre ellos.
- Se utiliza comúnmente para resolver dos problemas fundamentales en grandes empresas: integración de información y gestión del conocimiento

(Soma, Bakshi, Prassanna, DaSie, & Bourgeois, 2008)



Ontologías

- Claridad y Objetividad. Una ontología debe proveer al usuario con el significado del término definido de forma objetiva.
- Completitud. Las definiciones deben ser expresadas en términos necesarios y suficientes.
- Coherencia. asegurar (*permitir*) que las inferencias derivadas de ella sean consistentes con las definiciones

Ontologías

- Máxima extensibilidad monótona. las especializaciones o generalizaciones deben ser incluidas en la ontología de tal forma que no requiera una revisión de las definiciones preexistentes.

- Principio de distinción ontológica. Las clases en una ontología *deben* ser disjuntas.

Ontologías

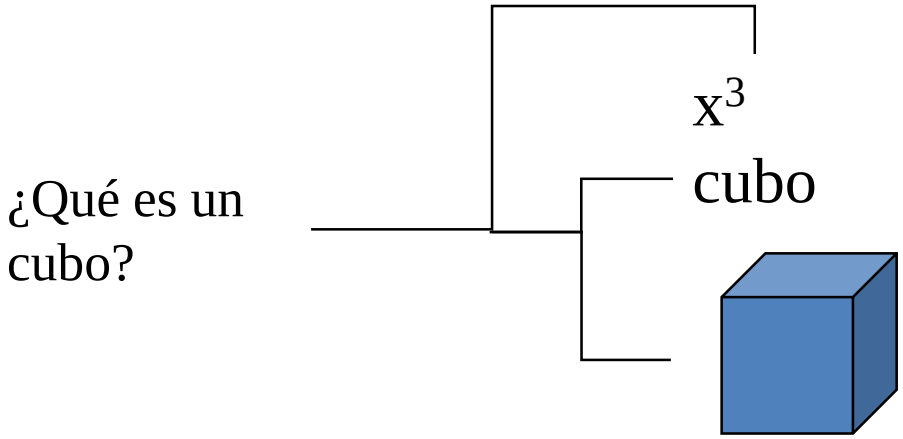
- Diversificación para aumentar la potencia de los mecanismos de herencia múltiple.
- Estandarización de nombres siempre que sea posible.
- Minimización de la distancia semántica entre conceptos emparentados. Conceptos similares estarán agrupados y representados usando las mismas primitivas.

Componentes de una Ontología

El conocimiento se representa dentro de una ontología a través de:

- Conceptos
- Relaciones
- Funciones
- Instancias
- Axiomas

Componentes de una Ontología



- Conceptos. Se emplean en un sentido amplio y pueden ser: tareas, funciones, acciones, estrategias, planes, etc.
- Relaciones. Representan un tipo de interacción entre conceptos del dominio.

Subclase-de: $Concepto_1$ x $Concepto_2$
Conectado con: Componente1 x Componente2

Componentes de una Ontología

- Funciones. Son un tipo especial de relación.

Madre-de--> Persona
Precio-objeto: Valor+Ganancia+IVA-->Precio

- Instancias. Concreciones en la ontología

- Axiomas. Proposiciones que siempre son verdaderas

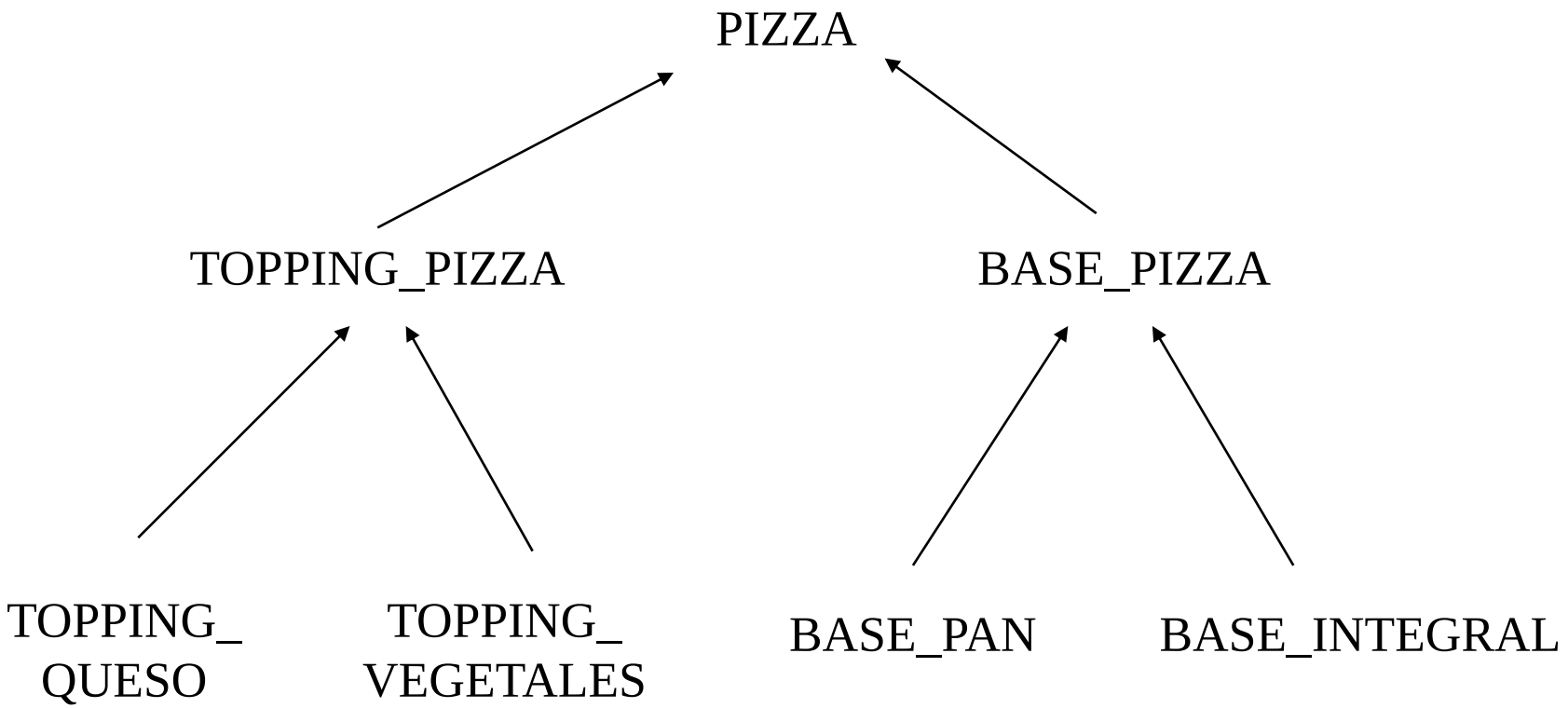
Conceptos

- Son las ideas básicas que se quieren formalizar para un determinado dominio de aplicación, y pueden estar organizadas en taxonomías.

- Pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.

- Por ejemplo:
  - Clase Pizza
  - Clase Topping:
    - Subclase ToppingQueso
    - Subclase ToppingCarne
  - Clase Base:
    - Subclase BaseIntegral
    - Subclase BaseNormal

Taxonomía de Conceptos



Relaciones

Representan las interacciones entre los conceptos del dominio.

- Forman la taxonomía del dominio

- Por ejemplo: Subclase_de, Parte_de
  - Base *es_parte_de* Pizza
  - Pizza Margarita *es_Subclase_de* Pizza

Propiedades de las Relaciones

describe a un individuo, pudiendo enlazarlo con otro

- Por ejemplo:
  - *tiene_ingrediente* tiene las subpropiedades:
    - tiene_base
    - tiene_topping
  - Pizza Margarita tiene_ingrediente
- Propiedad Inversa de *tiene_ingrediente*
  - *es_ingrediente_de*
    - es_base_de
    - es_topping_de

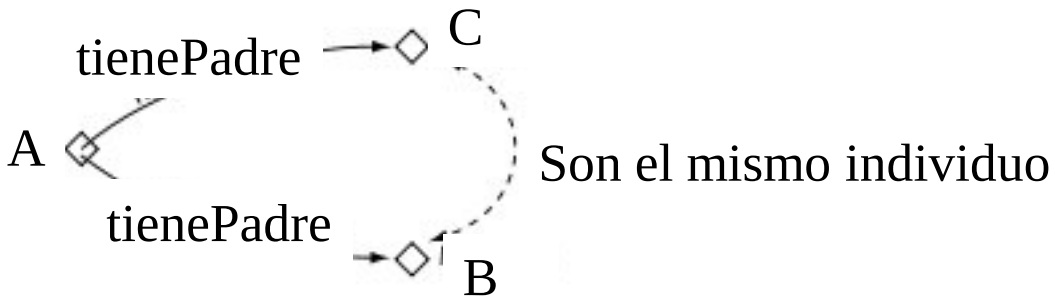
Propiedades de las Relaciones

- Propiedad Funcional, cuando un individuo se relaciona con otro por medio de la propiedad.
  - tiene_base
- Propiedad Inversa Funcional, la relación inversa es funcional
  - es_base
- Propiedad Transitiva, la propiedad relaciona al individuo *a* con *b* y *b* con *c*, entonces se infiere que *a* se relaciona con *c*
  - tiene_ingrediente
- Propiedad Simétrica, cuando un individuo a se relaciona con b y b con a por medio de la misma propiedad

Propiedades funcionales

Si una propiedad es funcional, para un individuo determinado, no puede haber más de una persona que se relaciona con el individuo a través de la propiedad. .

- Por ejemplo,
  - si se tienen tres objetos que son A, B y C y se tiene una propiedad funcional tienePadre, entonces se podrían asociar los objetos A y B por medio de la propiedad y daría como resultado A tienePadre B.
  - Igualmente se podrían asociar los objetos A y C por medio de la propiedad y daría como resultado A tienePadre C.
  - Como tienePadre es propiedad funcional, se concluye que B y C son el mismo objeto.



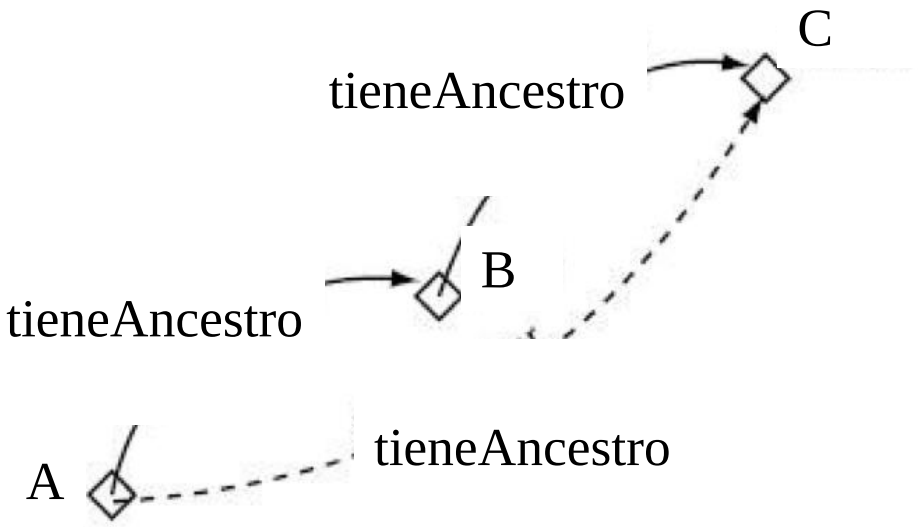
Propiedades funcionales inversa

Si una propiedad es funcional inversa, indica que puede estar a lo sumo un objeto relacionado con otro mediante esta propiedad de forma inversa a la propiedad funcional original.

- Por ejemplo,
  - si se tienen tres objetos que son A, B y C y se tiene una propiedad funcional esPadreDe, entonces se podría asociar el objeto B y A por medio de la propiedad y daría como resultado B esPadreDe A.
  - Igualmente se podrían asociar los objetos C y A por medio de la propiedad y daría como resultado C esPadreDe A.
  - Como esPadreDe es propiedad funcional inversa, se concluye que B y C son el mismo objeto.

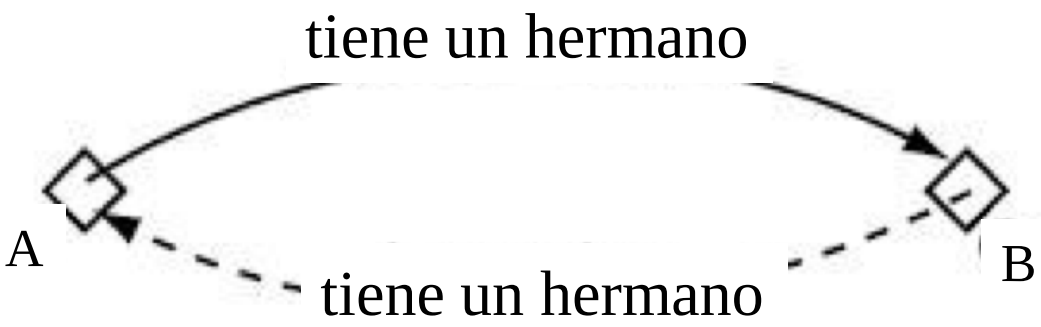
Propiedad Transitiva

Si una propiedad es transitiva y relaciona dos objetos A y B, y además hay una propiedad que relaciona al objeto B con otro C, entonces se puede inferir que el objeto A está relacionado con el objeto C mediante la propiedad transitiva.

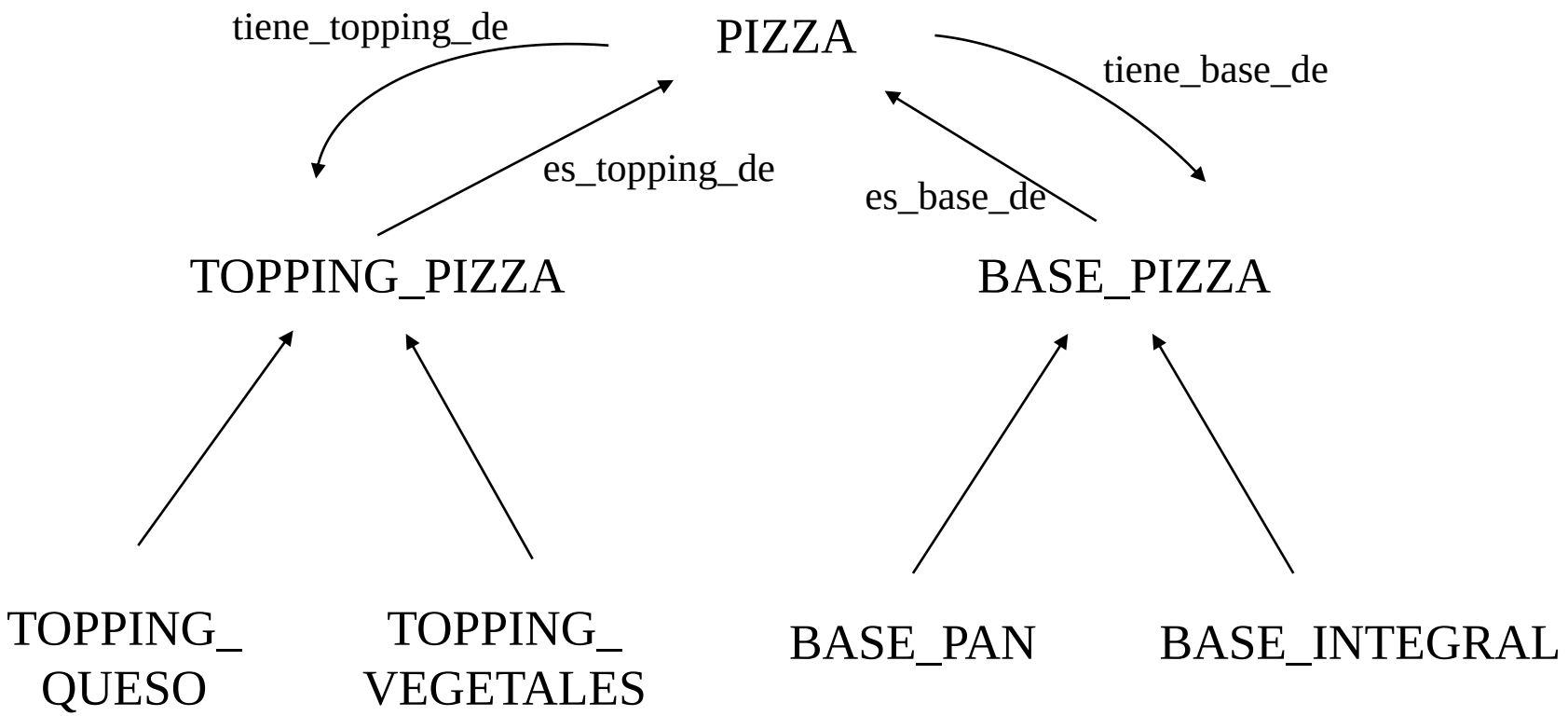


Propiedad simétrica

Si una propiedad P es simétrica y la propiedad relaciona a los objetos A y B, entonces el objeto B es relacionado por medio de la propiedad P con el objeto A.



Taxonomía de Conceptos



Funciones

Son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.

- Por ejemplo, pueden aparecer funciones como: *asignar-fecha*

Axiomas

Son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología

- Por ejemplo:
  - Toda Pizza Margarita tiene Topping_Mozarella
  - Las Pizzas Vegetarianas no tienen Topping de Carne

Instancias

Se usan para representar elementos o individuos en una ontología => hechos.

- Por ejemplo:
  - Pizza Vegetariana
  - Pizza Margarita

Características de la Ontología

- Categorías: taxonomía y herencia

  tomate$\subset$Fruta

  $\forall$x x$\in$tomate=>Rojo(x) y Redondo(x)

- Medidas

  Precio(tomate)=$(1.2)

- Composición de Objetos

  Partede(Caracas,Venezuela)

Características de la Ontología

- Tiempo, Espacio y Evento

  Subevento(BatallaCarabobo,IndependenciaVzla)

  En(Caracas,Vzla)

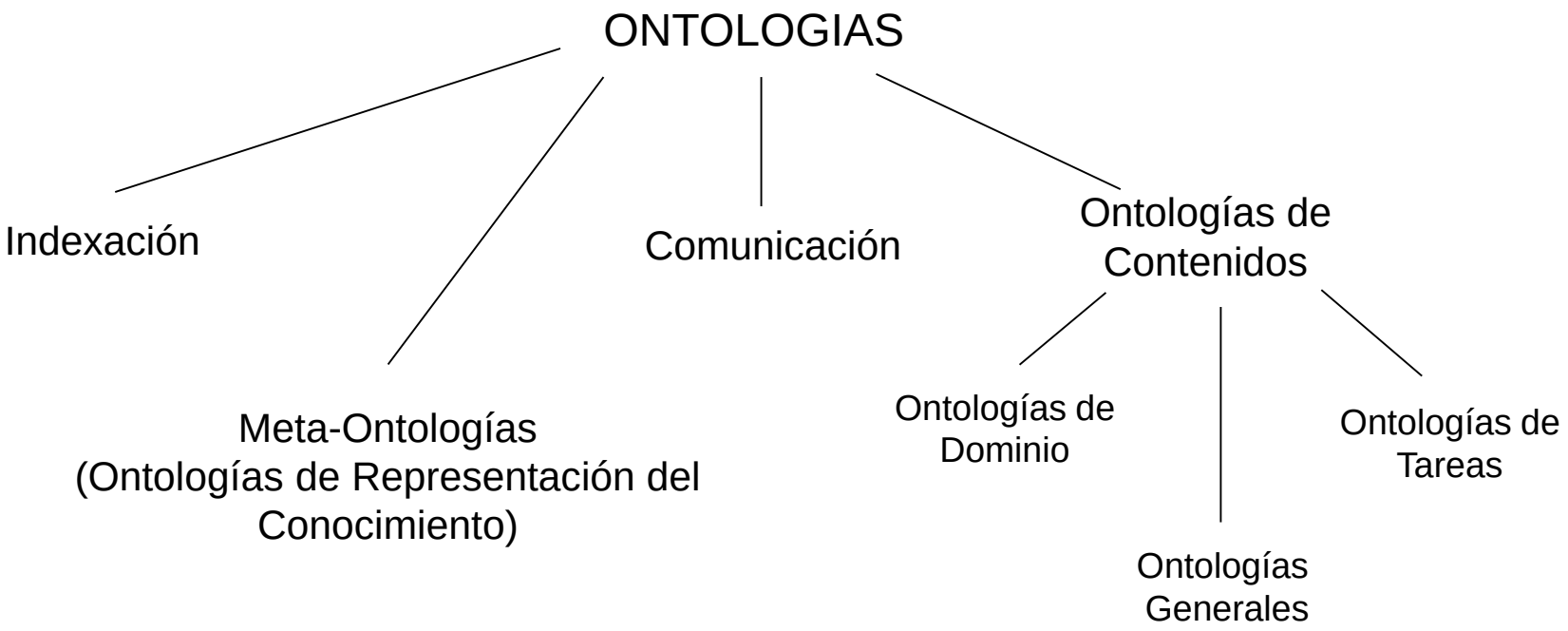
  T(En(jose, supermercado),ayer)

- Modelos Mentales

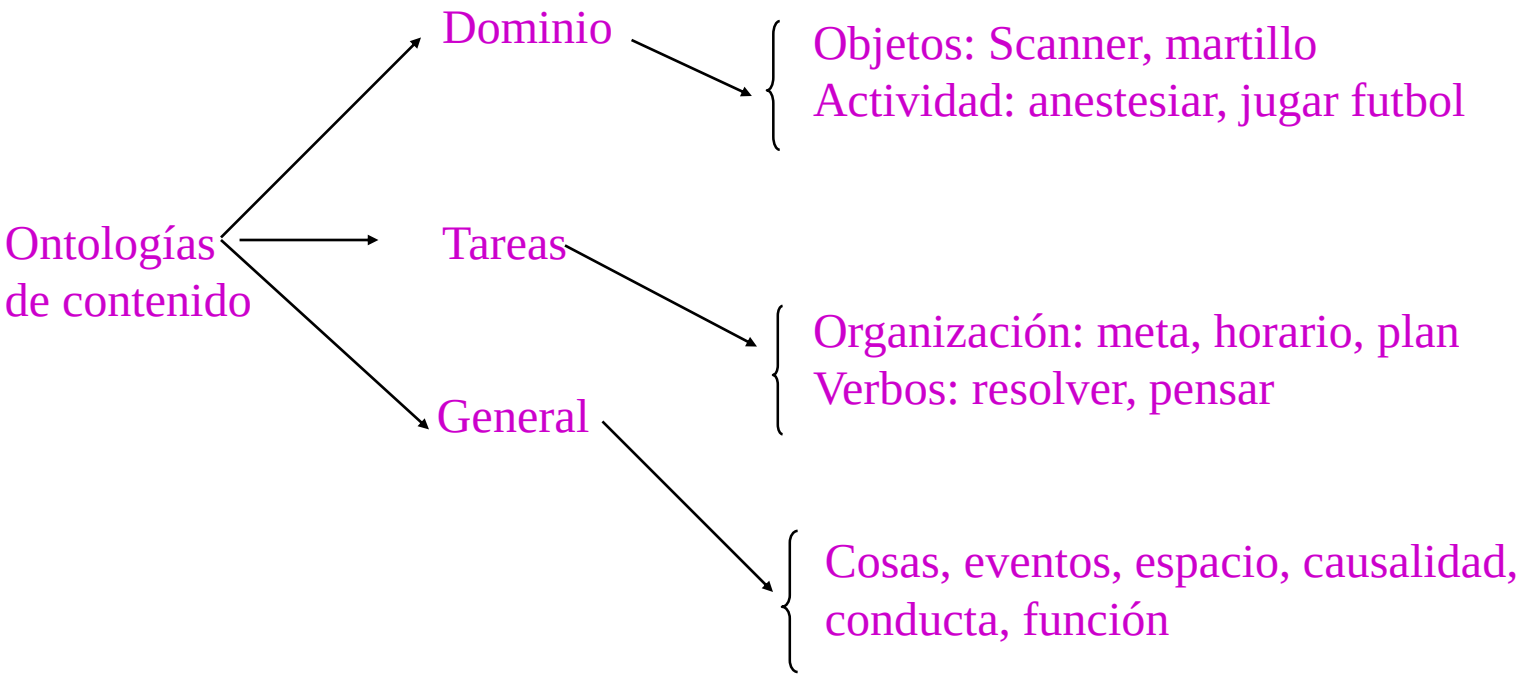
  Conoceque(Agente,numerotelefono(Jose),cadena)

  Cree(Luis,vuela(superman),mañana)

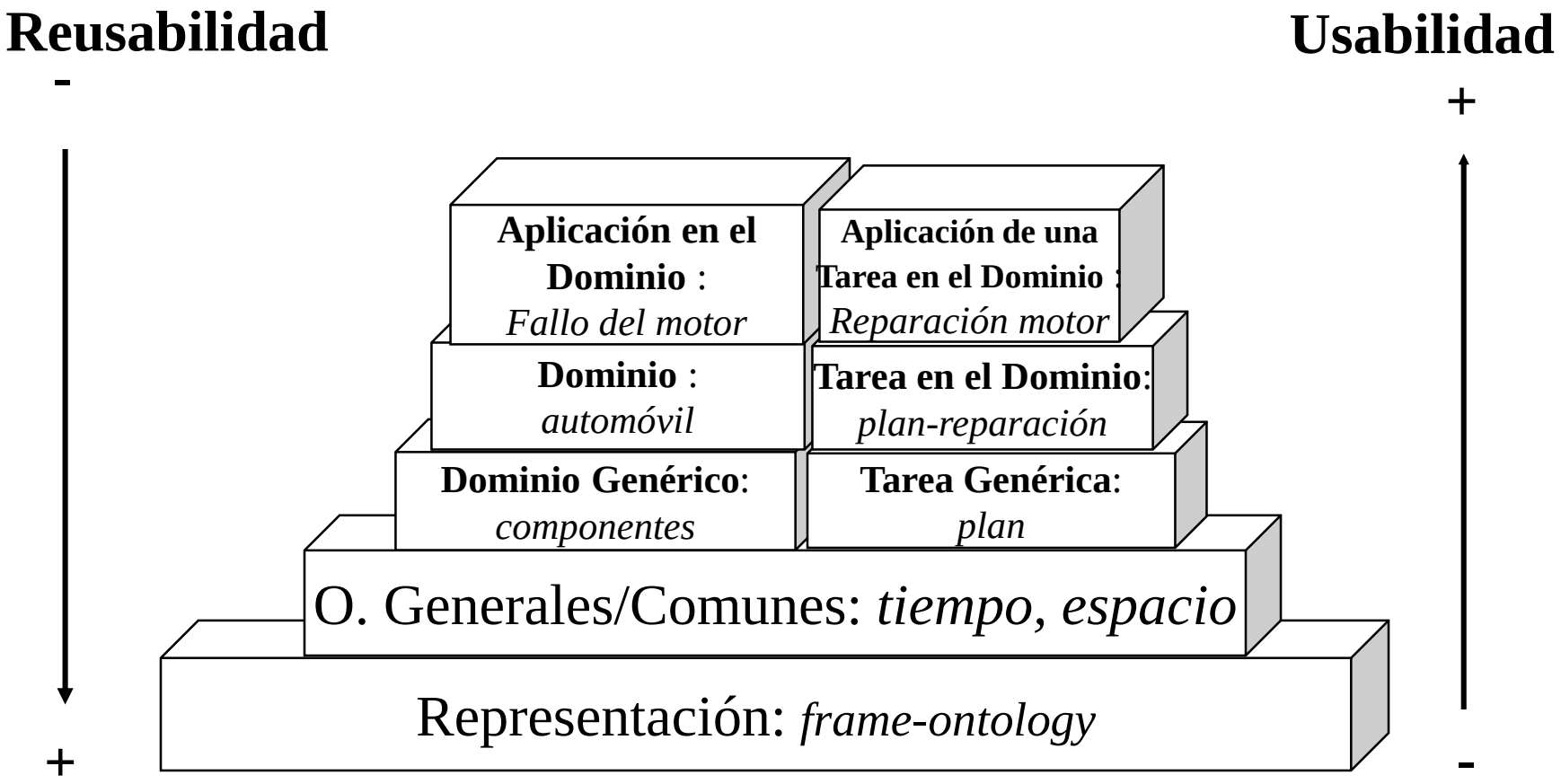
Clasificación de las Ontologías



Ontologías de Contenido



Ontologías


Reusabilidad
-
+
Usabilidad
+
-
Aplicación en el Dominio : Fallo del motor
Aplicación de una Tarea en el Dominio : Reparación motor
Dominio : automóvil
Tarea en el Dominio: plan-reparación
Dominio Genérico: componentes
Tarea Genérica: plan
O. Generales/Comunes: tiempo, espacio
Representación: frame-ontology

Tipos de Ontologías

- ***Informales*** si son expresadas en lenguaje natural
- ***Semi-informales*** si son expresadas de forma estructurada y restringida en lenguaje natural
- ***Rigurosamente formales*** si proporcionan términos definidos meticulosamente con semántica formal, teoremas y pruebas de propiedades tales como validez y completitud

Cuestiones competenciales a las que debe de responder la ontología

De acuerdo con Noy & Mc.Guiness

¿Para qué se va a utilizar ?

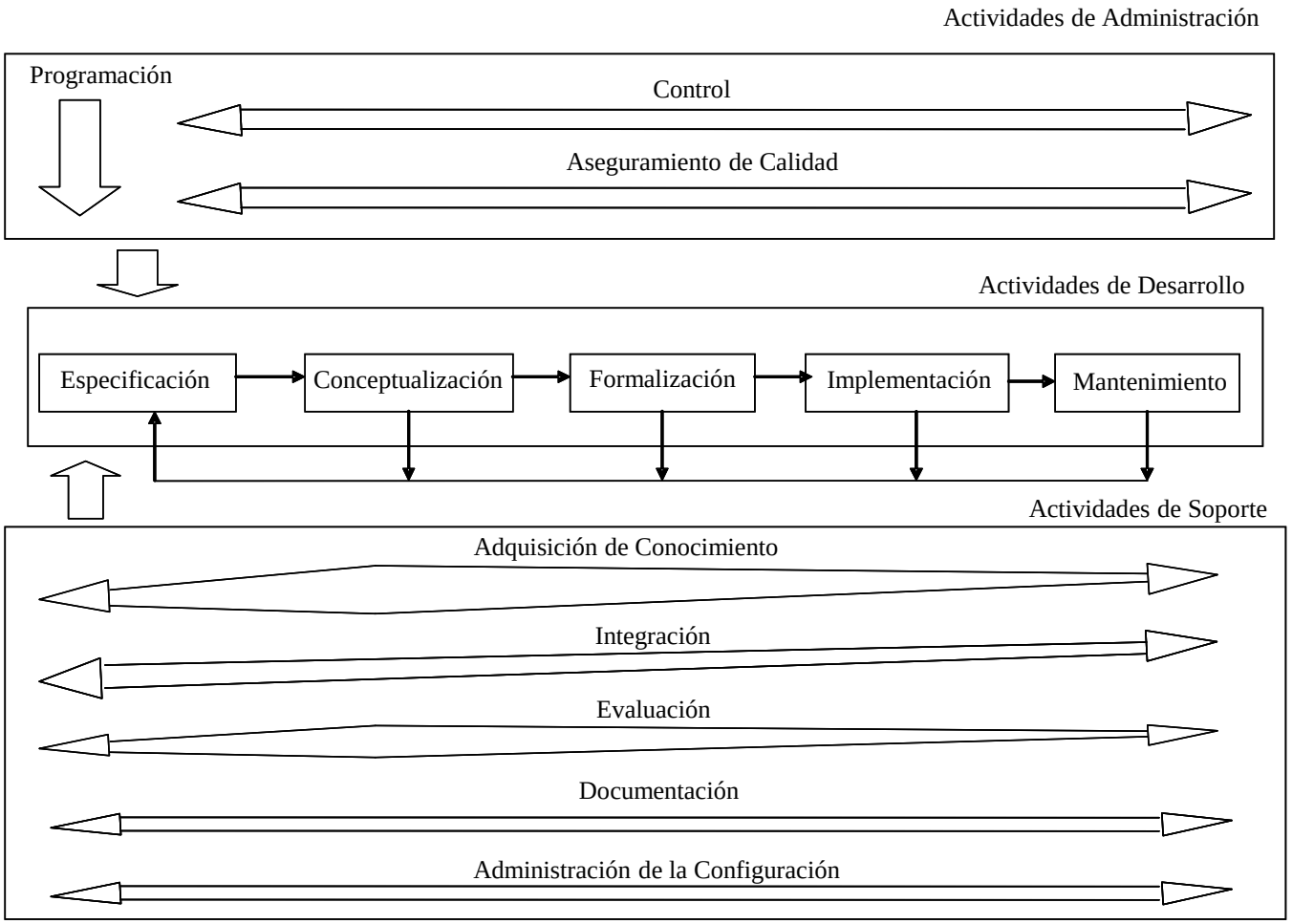
¿A qué tipo de preguntas va a dar respuesta ?

¿Quién la utilizará y mantendrá ?

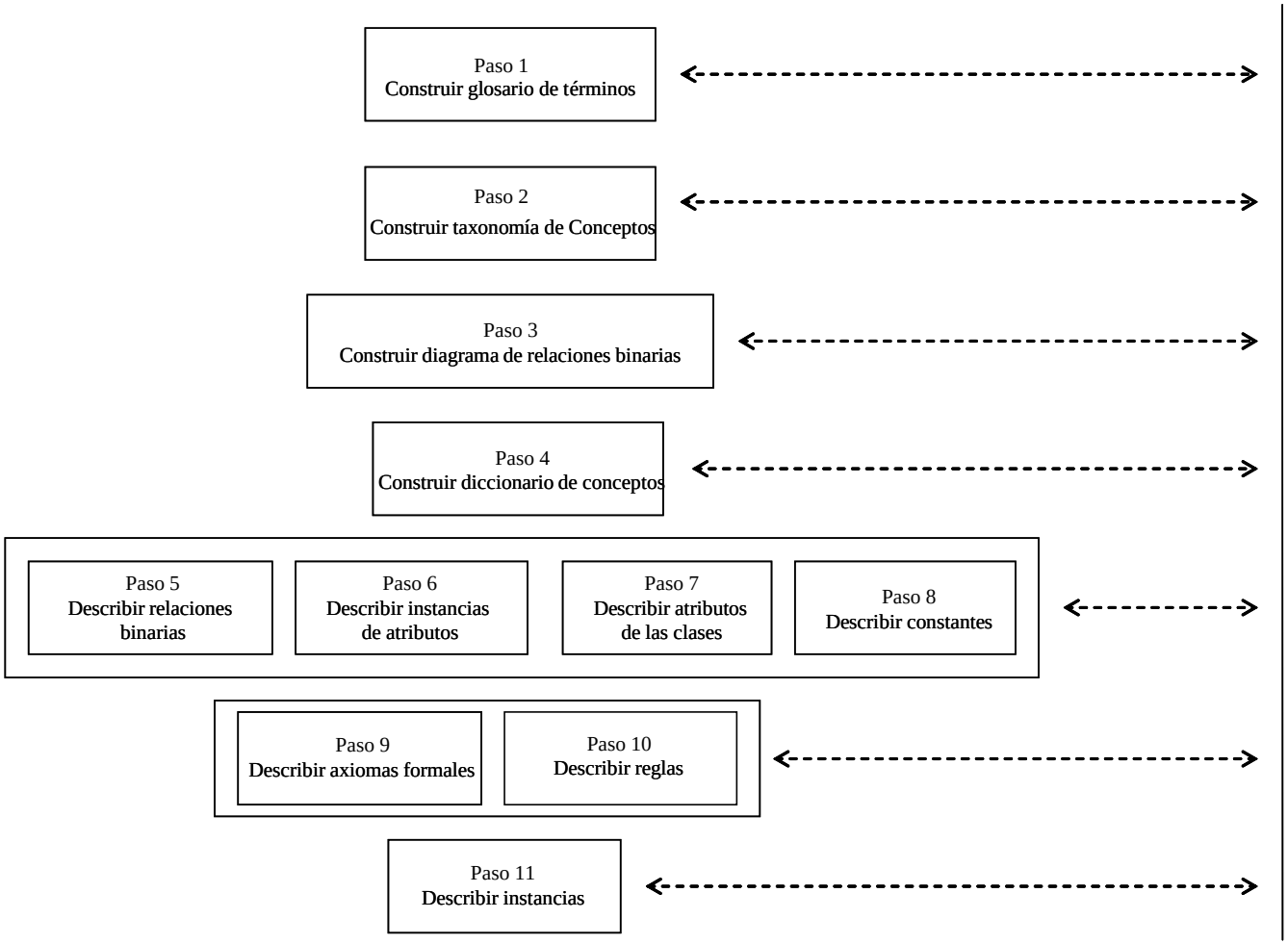
Metodologías de Diseño de Ontologías

- Metodología de Uschold y King
- Metodología de Grüninger y Fox
- Metodología Kactus
- Metodología Methontology

Methontology



Pasos en Methontology



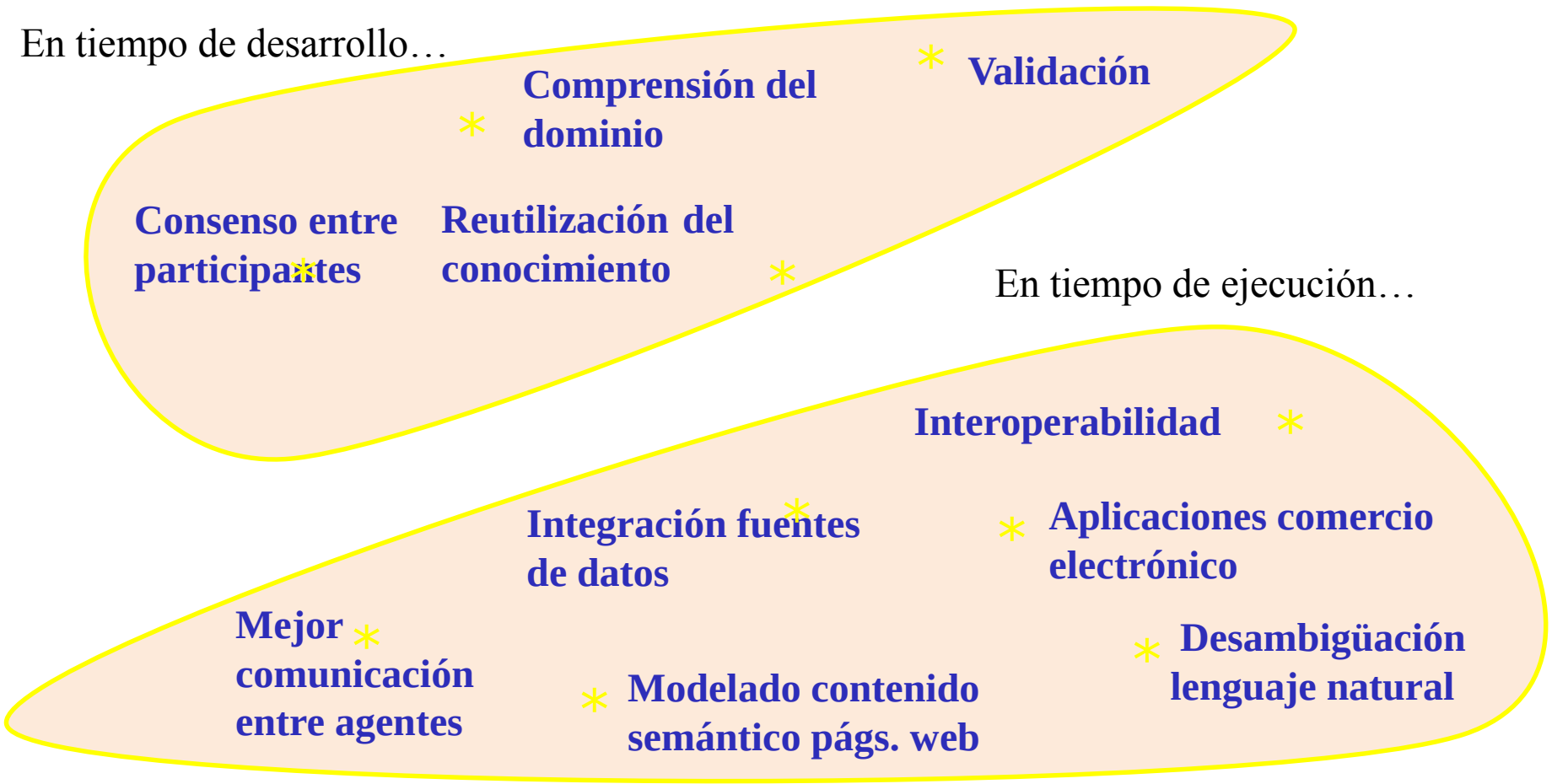
Representación de Ontologías

- Lenguajes basados en lógica de primer orden:
  - Ontolingua
  - DAML+ OIL (Darpa´s Agent Markup Language + Ontologic Inference Language)
  - OWL (Web Ontology Language)
- Ambientes:
  - Protégé: http://protege.stanford.edu/
  - WebOde: http://webode.dia.fi.upm.es/WebODEWeb/index.html
  - OntoEdit: http://ontoserver.aifb.uni-karlsruhe.de/ontoedit/

Aplicaciones de una Ontología

- Como repositorios de conocimientos e información para la organización.
- Como herramienta para la adquisición de información, en situaciones en las que un equipo de trabajo la utiliza como soporte común para la organización del dominio.
- Para permitir la reutilización del conocimiento existente en nuevos sistemas.
- Como base para la construcción de lenguajes de representación del conocimiento, junto a la formalización del cálculo que tenga lugar entre los términos

Ventajas del uso de las ontologías



Justificación de la ontología propuesta

Otras ventajas…..

1) Marco de referencia para la descripción formal de escenarios
   => garantizar el comportamiento consistente y no ambiguo

2) Definición del dominio de conocimiento
   => términos y conceptos relacionados con los escenarios educativos y su especificación

Ontologías basadas en OWL

- Las ontologías son usadas para **capturar el conocimiento s**obre algún dominio de interés.
- Una ontología **describe** los **conceptos** dentro del dominio y **la relación** que tiene entre esos conceptos.
- Un **lenguaje estándar** para hacer ontologías es OWL desarrollado por W3C.
- OWL permite describir conceptos y además cuenta con un conjunto de operadores (intercesión, unión, y negación).
- OWL esta basado en **lógica descriptiva** que permite el uso de un razonador.

Lógica de primer orden y Lógica Descriptiva

- El fundamento que garantiza la pureza lógica de la ontologías es la lógica de primer orden.

- Sobre ella se asienta las lógicas descriptivas (DL), así como OWL.

- Por qué usar lógicas descriptivas?:
  - Lógica de primer orden es indecidible (es fácil afirmar cosas de objetos, pero computacionalmente complejo)
  - Se requiere de un lenguaje formal para construir y combinar definiciones de categorías (p.ej. Relaciones de subconjunto y superconjunto)
  - Razonadores semánticos se basan en ella: FaCT++, Rancer, Pellet, …

Lógica descriptiva

- Lenguajes de representación del conocimiento
- DL se diseñó como una extensión de marcos y redes semánticas, equipada con semántica basada en lógica.
- Características :
  - Un formalismo descriptivo: conceptos, roles (relaciones), individuos.
  - Un formalismo terminológico: axiomas que describen propiedades genéricas.
  - Un formalismo asertivo: introduce propiedades de individuos.

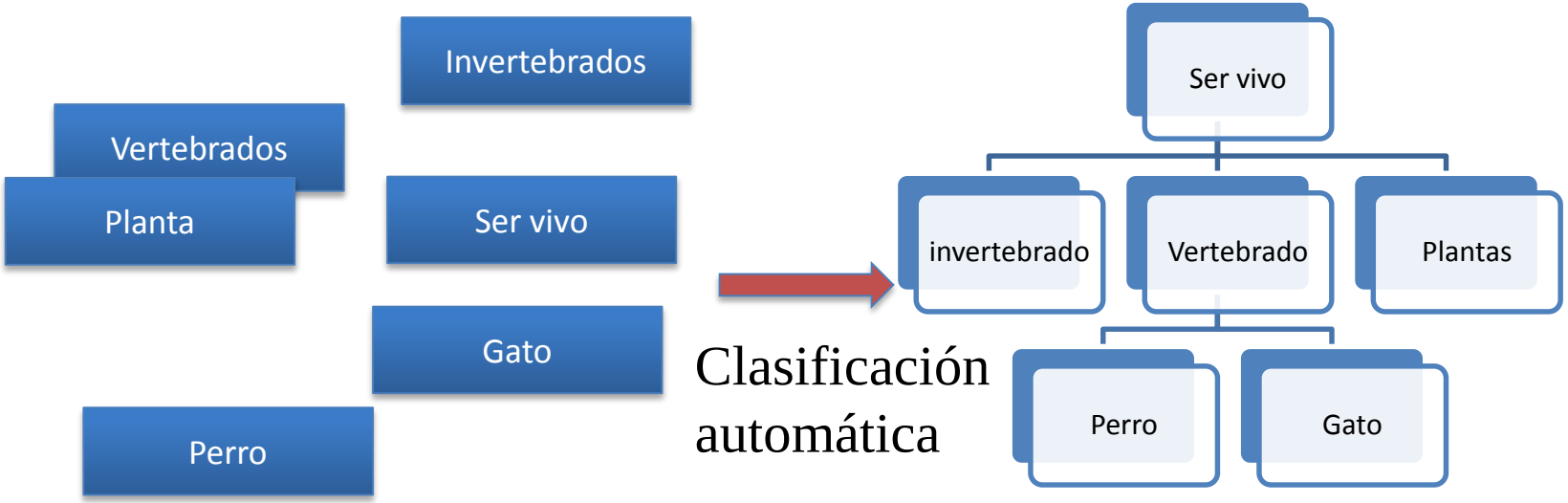
Lógica descriptiva

- Principales tareas de inferencia con lógica descriptiva:
  - Subsunción (comprobar si una categoría es subconjunto de otra)
  - Clasificación (comprobar si un objeto pertenece a una categoría)
- Ejemplo:
  - Soltero= Y(NoCasado, Adulto, Masculino)
  - Soltero(x)=>NoCasado(x)Yadulto(x)Ymasculino(x)

(lógica de primer orden)

Lógicas descriptivas

- Ejemplo: realiza clasificación automática (realizada por el motor de inferencias del lenguaje-razonador) en tiempo de ejecución

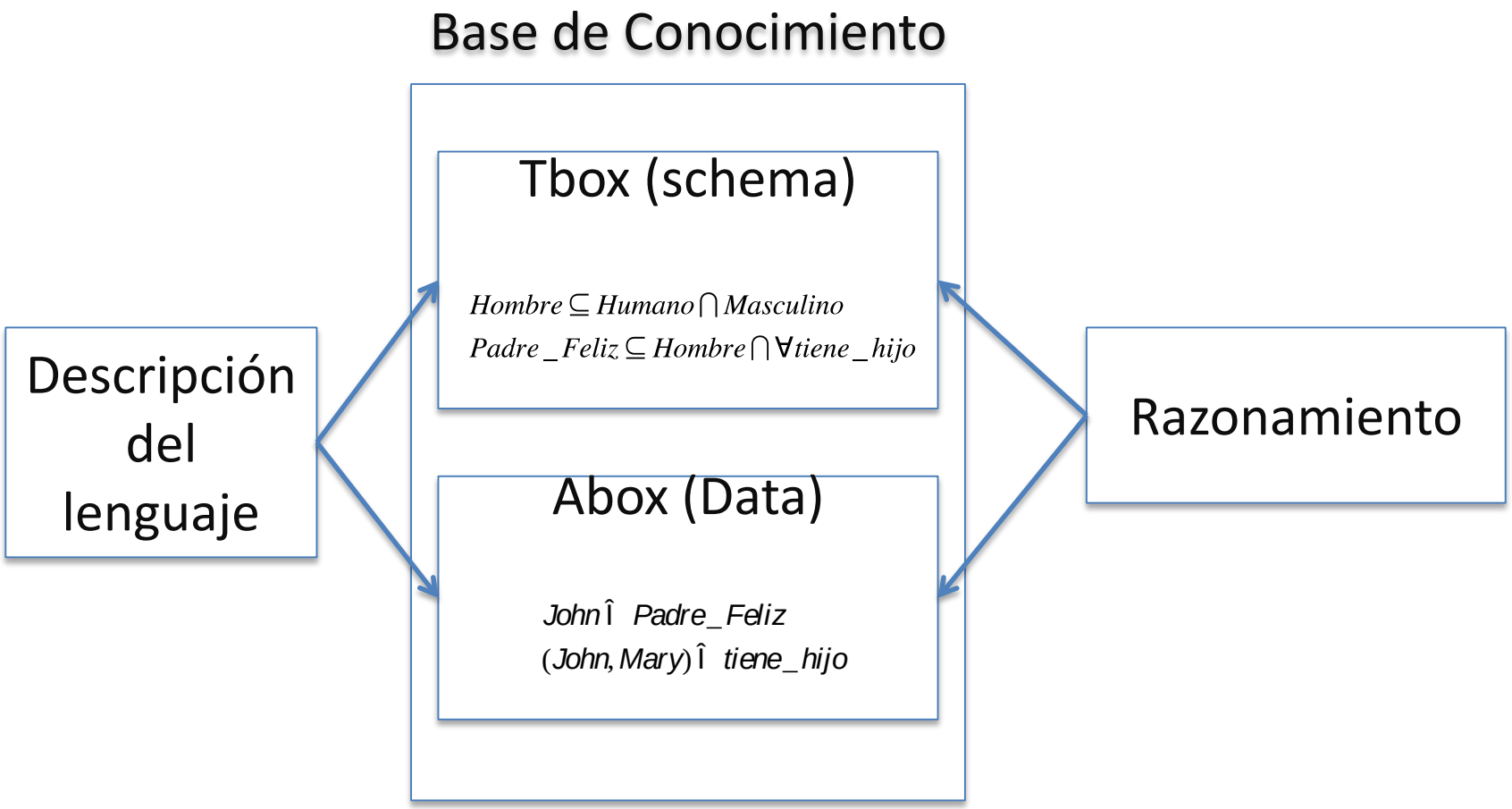


Correspondencia entre OWL y DL

Constructor OWL	Representación DL	Ejemplo
owl:equivalentTo (C,D)	$C \equiv D (C \sqsubseteq D \; y \; D \sqsubseteq C)$	$Persona \equiv Humano$
rdfs:subClassOf (C,D)	$C \sqsubseteq D$	$Padres \sqsubseteq Persona$
owl:complementOf (C,D)	$C \equiv \neg D (negacion)$	$Varon \equiv \neg Mujer$
owl:disjointWith (C,D)	$C \sqsubseteq \neg D$	$Padre \sqsubseteq \neg Madre$
owl:intersectionOf (C,D)	$C \sqcap D (conjuncion)$	$Padres \sqcap Varon$
owl:unionOf (C,D)	$C \sqcup D (disjuncion)$	$Padre \sqcup Madre$
owl:oneOf (I1, I2)	$\{I_1\} \sqcup \{I_2\}$	$\{Juan\} \sqcup \{Maria\}$
owl:someValuesFrom(P,C)	$\exists P.C (existencial)$	$\exists tieneHijo.Hija$
owl:allValuesFrom(P,C)	$\forall P.C (universal)$	$\forall tieneHijo.Hijo$
owl:hasValue (P,I1)	$\exists P.\{I_1\}$	$\exists tieneHijo.\{Juan\}$
owl:cardinality(P,n)	$= n.P$	$= 2.tienePadres$
owl:minCardinality(P,n)	$\geq n.P$	$\geq 1.tieneHija$
owl:maxCardinality(P,n)	$\leq n.P$	$\leq 2.tieneHijos$

Un *concepto* en DL se refiere a una *clase* en OWL.
Un *rol* en DL es una *propiedad* en OWL.

Arquitectura DL



Lógicas descriptivas: TBox

- Tbox: contiene declaraciones terminológicas generales. Vocabulario de un dominio de aplicación en función de: Conceptos, Roles, etc. Son de dos tipos.
  - Definición de concepto

$$A \equiv C$$

Ejemplo

$$Mujer \equiv Persona \bigcap Femenino$$

$$Madre \equiv Mujer \bigcap \exists tiene\_hijo.Persona$$

  - Axiomas descriptivos de roles, etc. $C_1 \subseteq C_2$

Ejemplo

$\$tiene\_hijo.Persona \subseteq Persona$

Lógicas descriptivas: ABox

- Abox: contiene aserciones sobre elementos y relaciones concretas del dominio. *Es decir, son a*serciones acerca de individuos usando vocabulario. Dos tipos:
  - Instancias de conceptos

$$o \equiv C$$

$$Ejemplo$$

$$Moby - Dick \equiv Ballena$$

$$Juan : Hombre \cap \exists tiene - hijo$$

  - Instancias de axiomas

$$(o_1, o_2) : R$$

$$Ejemplo$$

$$(Ana, Juan) : tiene\_hijo$$

Formalizar en Dl y luego en owl dl

- Definición de conceptos.
  - El pasto y los arboles son plantas. Las hojas son parte del árbol, pero existen otras partes de un árbol que no son hojas. Un perro debe comer al menos huesos. Una oveja es un animal solo debe comer pasto. Una jirafa es un animal que solo debe comer hojas. Las vacas locas solo se alimenta de cerebros que pertenecen a las ovejas.
- Restricciones:
  - Animales son disjuntos con plantas.
- Propiedades:
  - Comer es aplicado a los animales y su inverso es comido_por.
- Individuos
  - Tom
  - Flossie es una vaca
  - Rex es un perro y es una mascota de Mick
  - Fido es un perro
  - Tibbs es un gato

Formalizar en Dl y luego en owl dl

1. El pasto y los arboles son plantas.

2. Las hojas son parte del árbol, pero existen otras partes de un árbol que no son hojas.

3. Un perro debe comer al menos huesos.

4. Una oveja es un animal y solo debe comer pasto.

5. Una jirafa es un animal que solo debe comer hojas.

6. Una vaca loca es una vaca que se alimenta de cerebros que son parte de las ovejas.

7. Animales o parte de animales son disjuntos con plantas o parte de plantas.

- Propiedades:
  - Comer es aplicado a los animales y su inverso es comido_por.

$1.Pasto \subseteq Planta$

$1.Arboles \subseteq Planta$

$2.Hojas \subseteq Arboles$

$3.Perro \subseteq Animal \cap \exists come.Huesos$

$4.Oveja \subseteq Animal \cap \forall come.Pasto$

$5.Jirafa \subseteq Animal \cap \forall come.Hojas$

$6.VacaLocas \subseteq vacas \cap \forall come(cerebro \subseteq Oveja)$

$7.Animal \subseteq \neg Planta$

Formalizar en Dl y luego en owl

1.1.$(LD).Pasto \subseteq Planta.$

$(OWL)rdf : subClassOf(Pasto, Planta)$

1.2.$(LD).Arboles \subseteq Planta.$

$(OWL)rdf : subClassOf(Arboles, Planta)$

2.$(DL)Hojas \subseteq Arboles$

$(OWL)owl : subClassOf(Hojas, Arboles)$

3.$(DL)Perro \subseteq Animal \cap \exists come.Huesos$

$(OWL)owl : subClassOf(Perro, (\text{int}ersection(Animal, someValuesFrom(come, Huesos))))$

4.$(DL)Oveja \subseteq Animal \cap \forall come.Pasto$

$(OWL)owl : subClassOf(Animal, \text{int}er\,\text{sec}\,tionOf(Animal, owl : someValuesFrom(come, Pasto)))$

5.$(DL)Jirafa \subseteq Animal \cap \forall come.Hojas$

$(OWL)owl : subClaseOf(Animal, \text{int}er\,\text{sec}\,tion(Animal, owl : allValuesFrom(come, Hojas))$

6.$(DL)VacaLocas \subseteq Vacas \cap \forall come(cerebro \subseteq Oveja)$

$(OWL)owl : subClaseOf(Vacas, \text{int}er\,\text{sec}\,tion(Vacas, owl : allValuesFrom(come, Cerebro))$

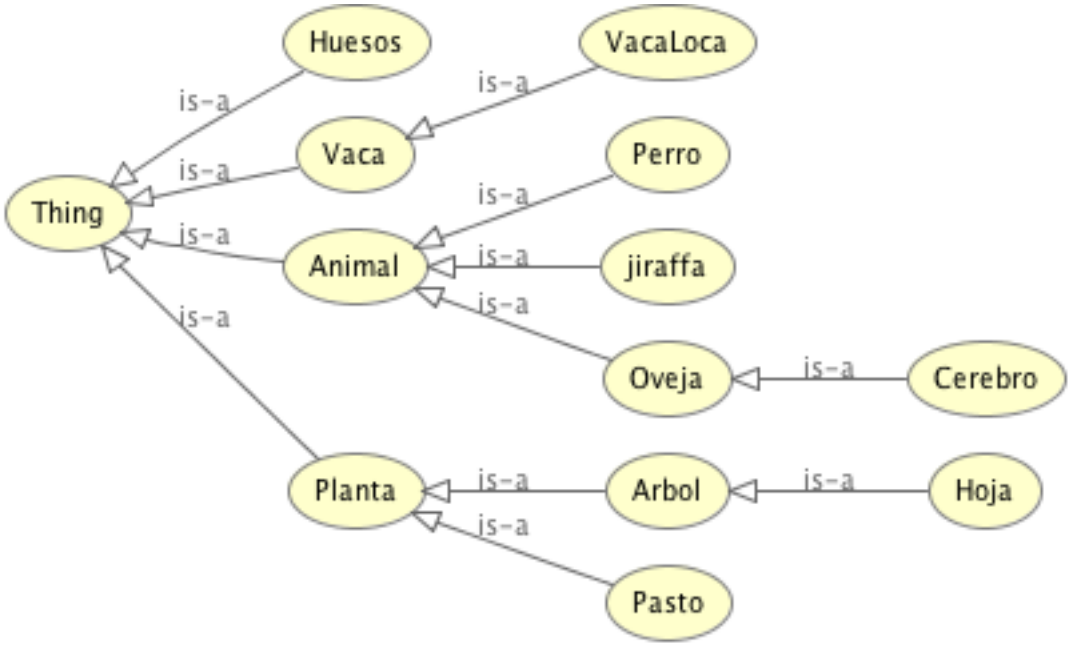
$owl : subClaseOf(Cerebro, Oveja)$

7.$(DL)Animal \subseteq \neg Planta$

$(OWL)owl : disjo\,\text{int}With(C, D)$

Resultados

1. $Pasto \subseteq Planta$
1. $Arboles \subseteq Planta$
2. $Hojas \subseteq Arboles$
3. $Perro \subseteq Animal \cap \exists come.Huesos$
4. $Oveja \subseteq Animal \cap \forall come.Pasto$
5. $Jirafa \subseteq Animal \cap \forall come.Hojas$
6. $VacaLocas \subseteq vacas \cap \forall come(cerebro \subseteq Oveja)$
7. $Animal \subseteq \neg Planta$



CONSTRUCCION DE UNA ONTOLOGÍA OWL EN PROTEGE

Componentes de una ontología owl

Ontologías	OWL	PROTÉGÉ
Individuos	Individuos	Casos (instance)
Relaciones	Propiedades	Slots
Conceptos	Clases	Clases

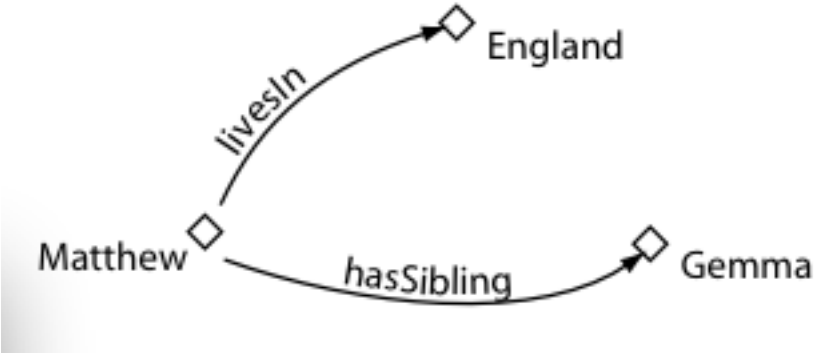
Individuos en owl

- Los individuos representan objetos del dominio de interés y son también conocidos como instancias.

Representación de Individuos

PROPIEDADES EN owl

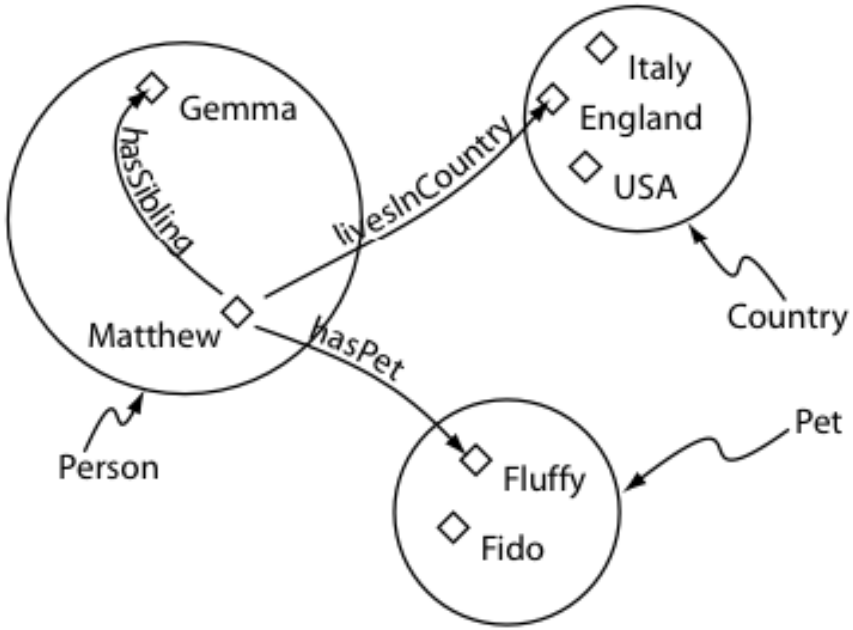
- Las propiedades son relaciones binarias sobre los individuos y pueden ser inversas, transitivas o simétricas.



Representación de Propiedades

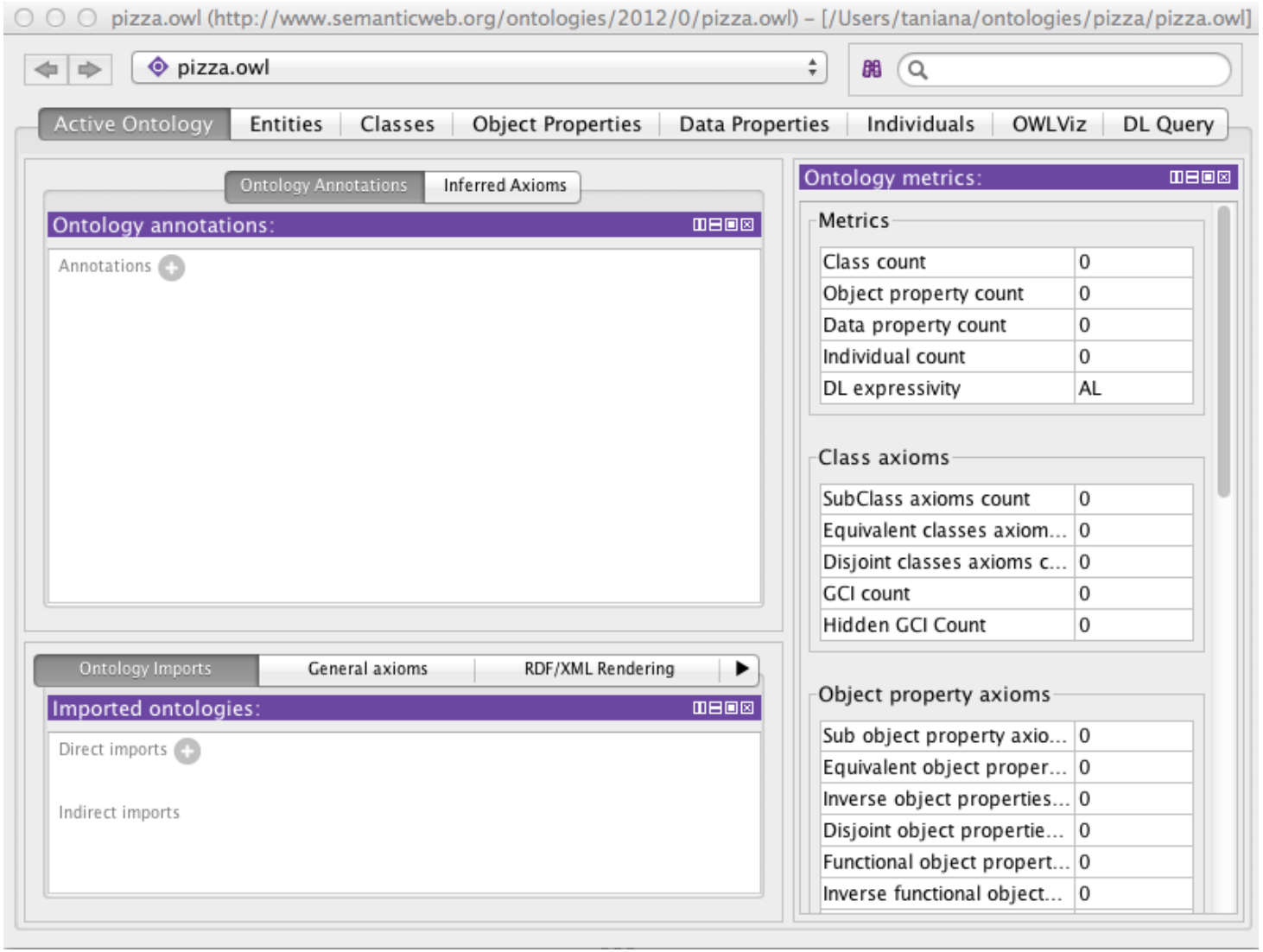
CLASES EN owl

- Las clases OWL se entienden como conjuntos que contienen individuos y pueden ser organizadas dentro de una jerarquía de clases y subclases conocida como taxonomía. Las clases también son conocidas como conceptos, pues son una representación concreta de éstos.



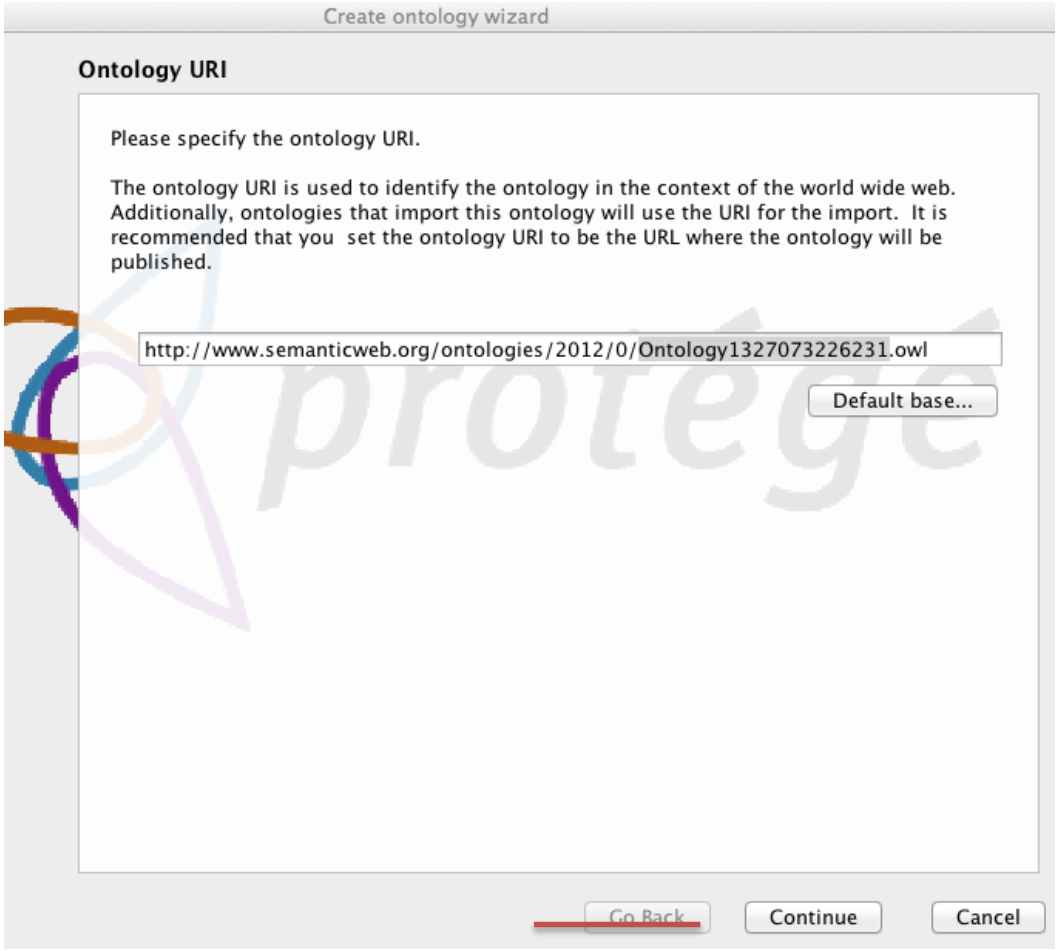
Representación de clases

Interfaz del protégé



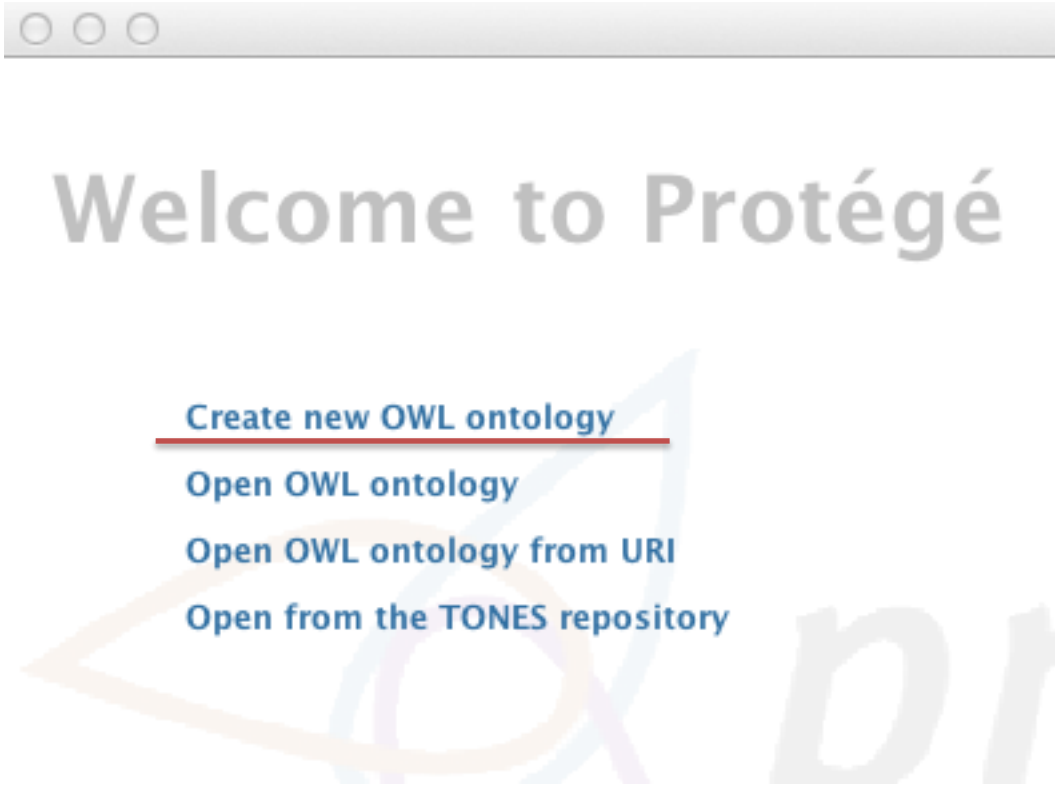
Crear una nueva ontología OWL

1. Toda ontología usa Unique Resource Identifier (URI)
2. Coloque el nombre de la ontología y presione Continue para seguir.
3. En nuestro caso colocaremos pizza.owl



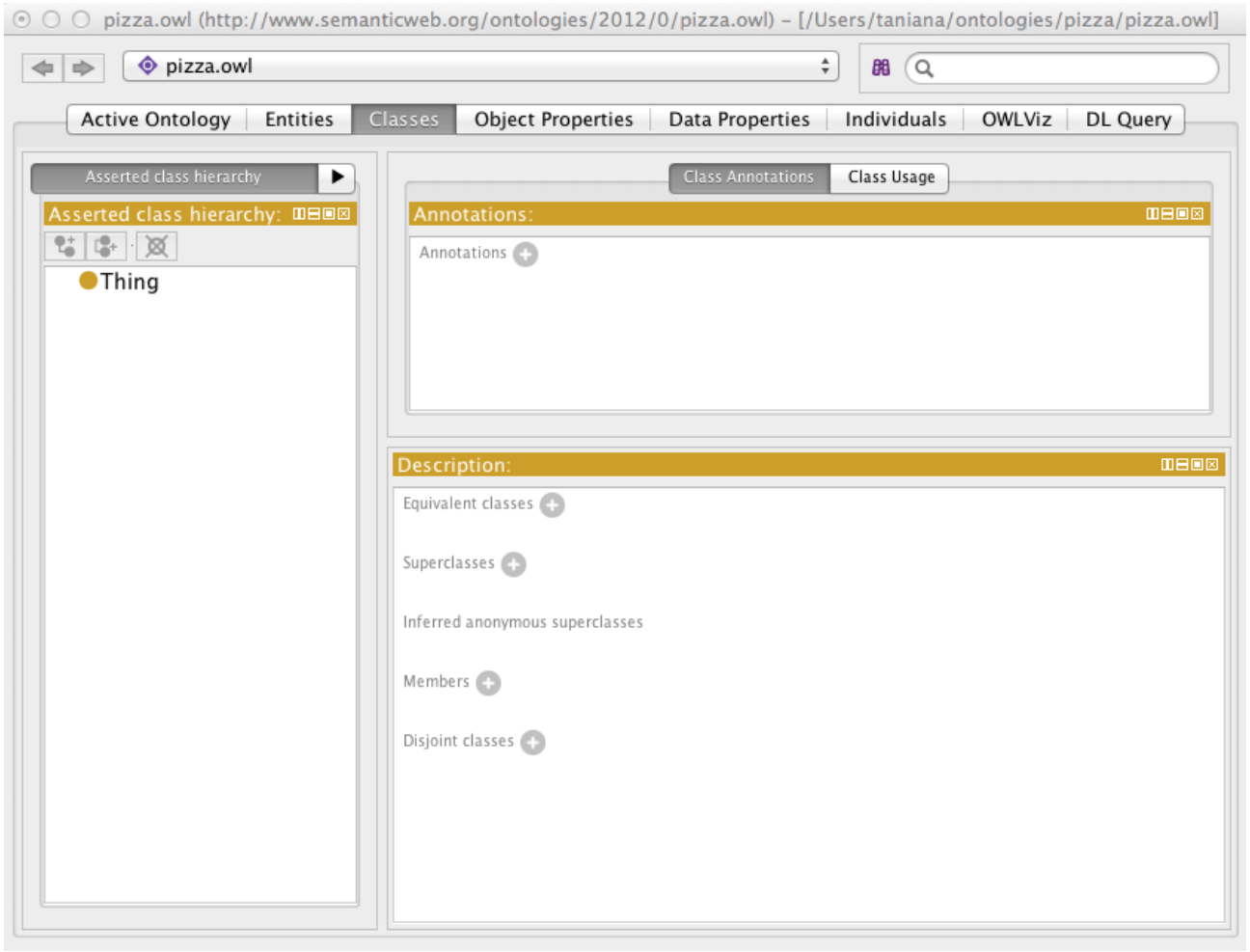
Crear una nueva ontología OWL

- Inicie la aplicación protégé.
- En la pantalla de bienvenida, seleccione “Create New OWL Ontology”

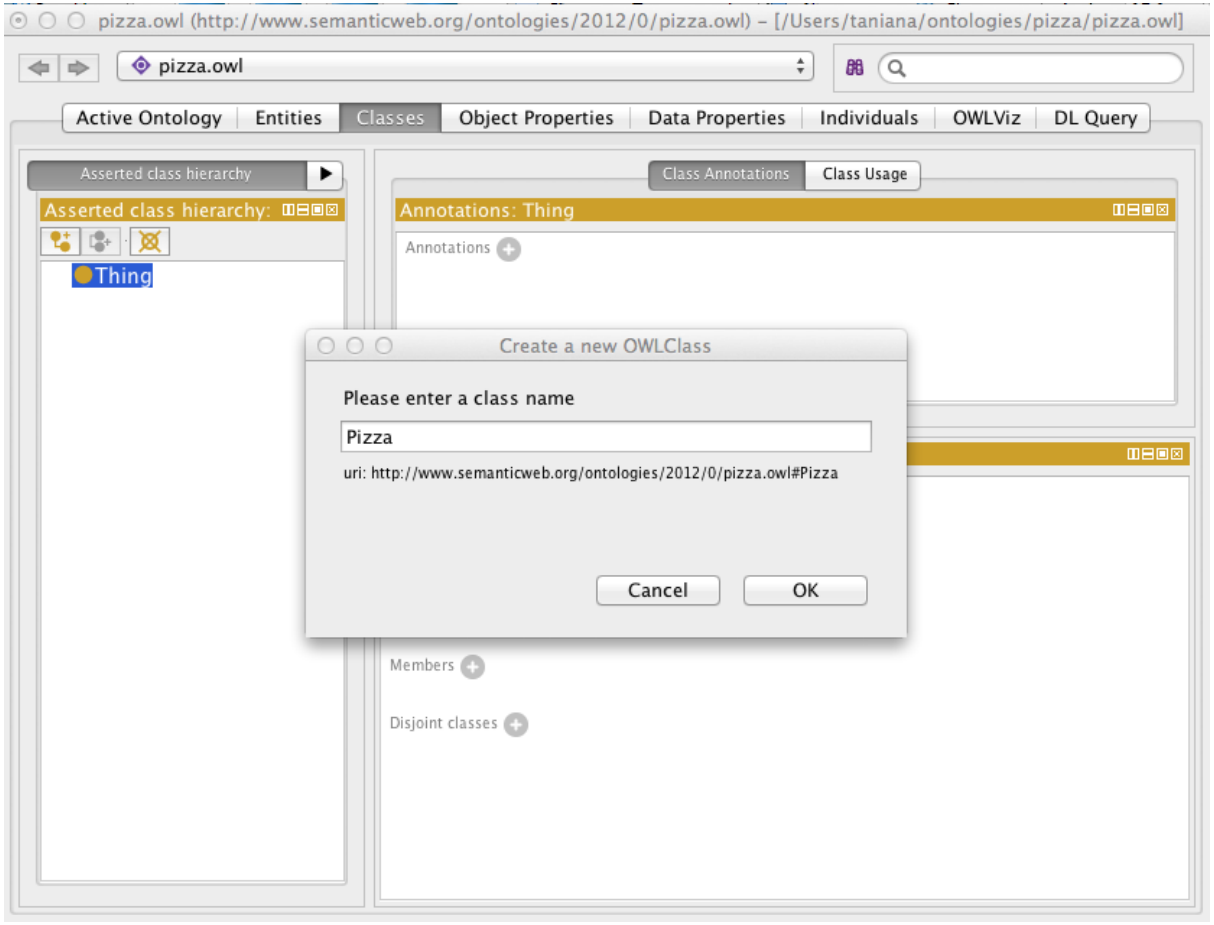


CREAR LAS CLASES

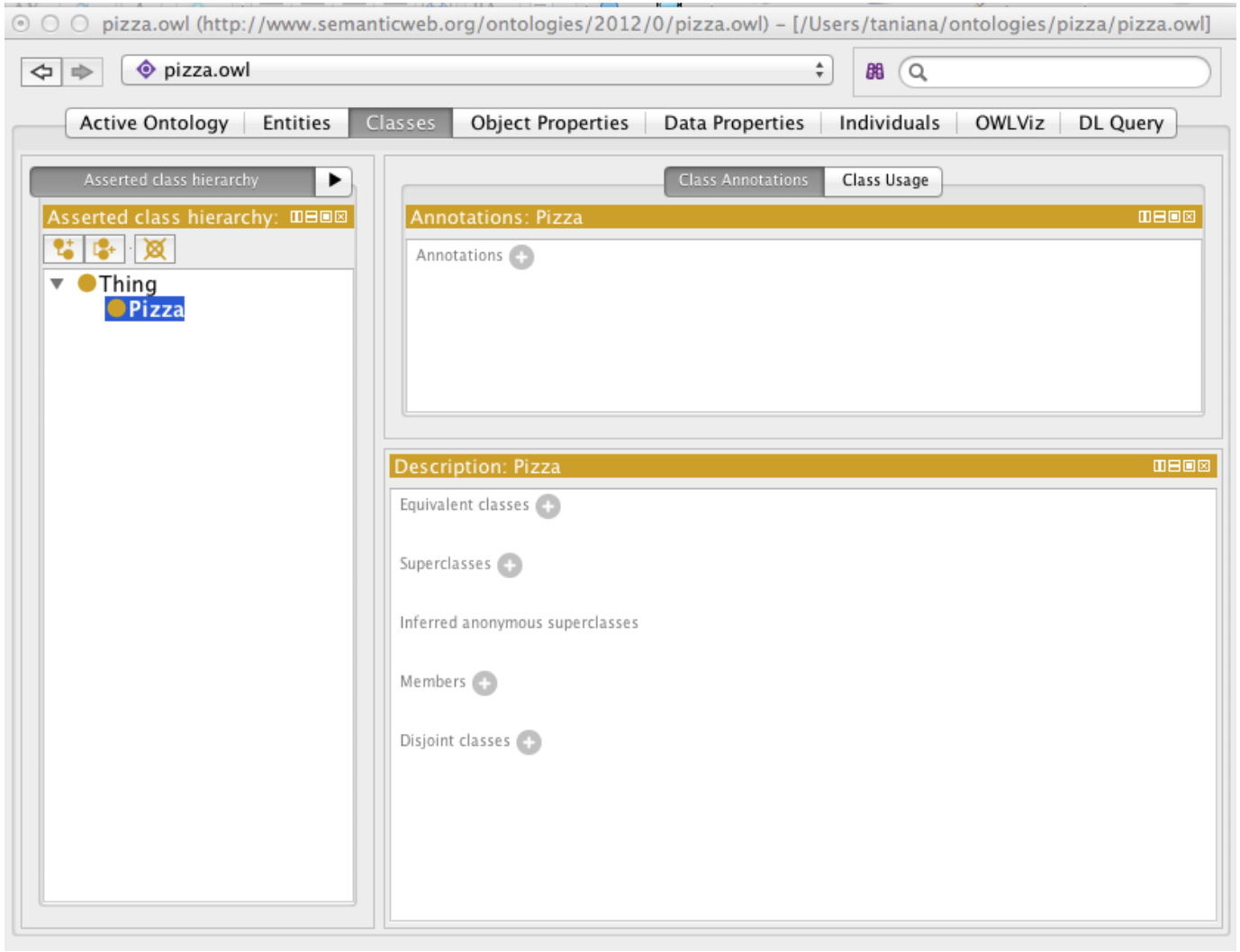
Clases en protégé



Clases en protégé

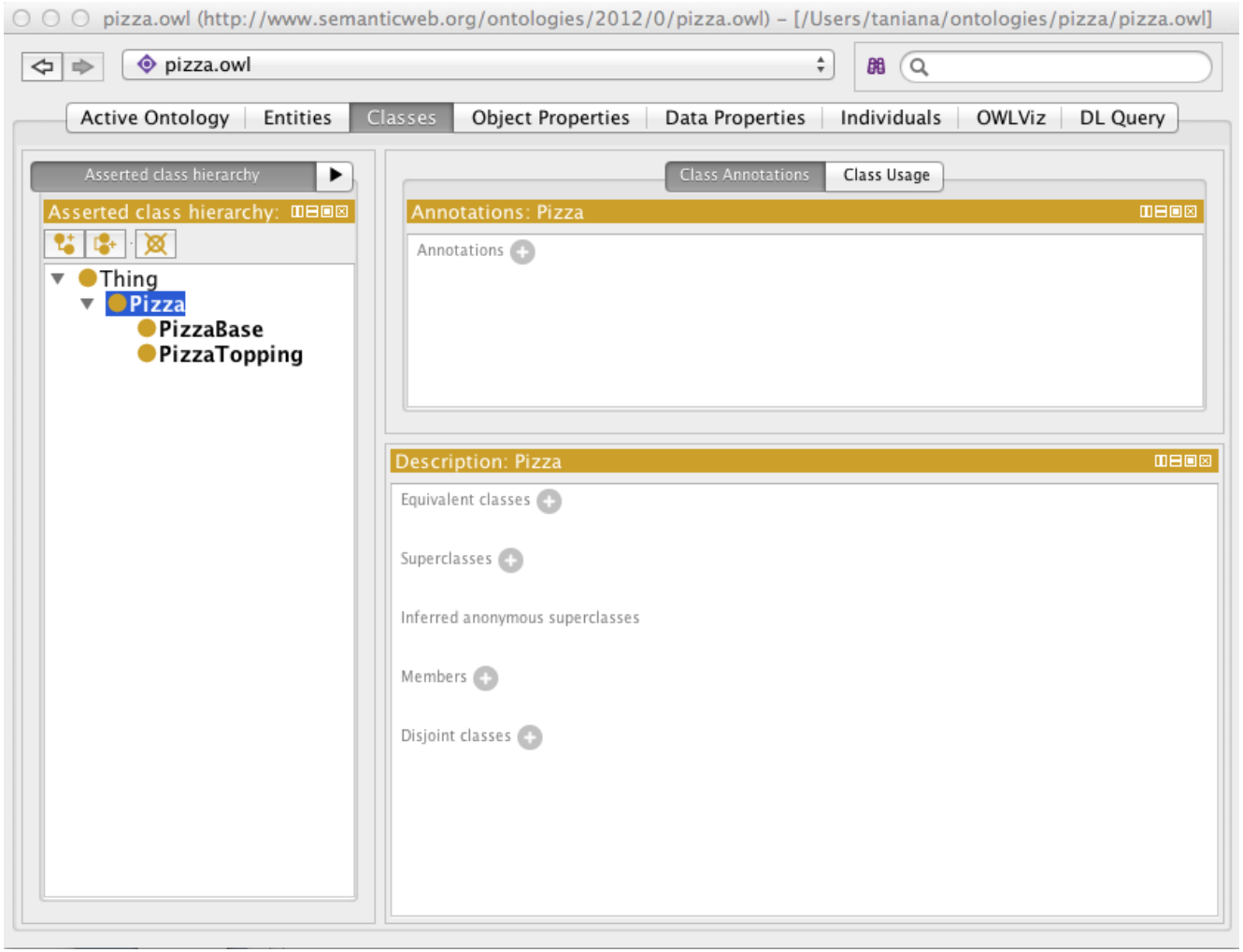


Clases en protégé



Clases en protégé

Se repite los pasos anteriores para crear:
PizzaTopping
PizzaBase



PROPIEDADES

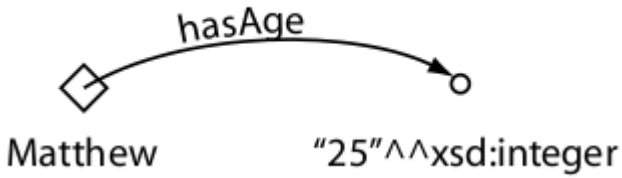
Propiedades owl

- Las propiedades OWL representan relaciones entre dos objetos o individuos.
- Existen dos tipos de propiedades en OWL:
  - "ObjectProperties", que permite relacionar un individuo con otro, y
  - "DatatypeProperties", que relaciona un individuo con un XML Schema Datatype value o un literal RDF

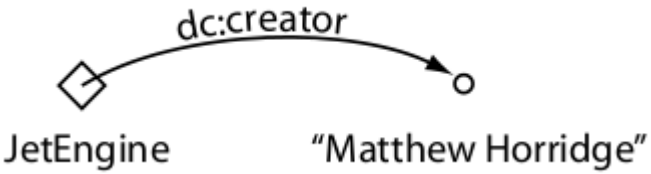
Propiedades owl

Object Properties

DataType Properties



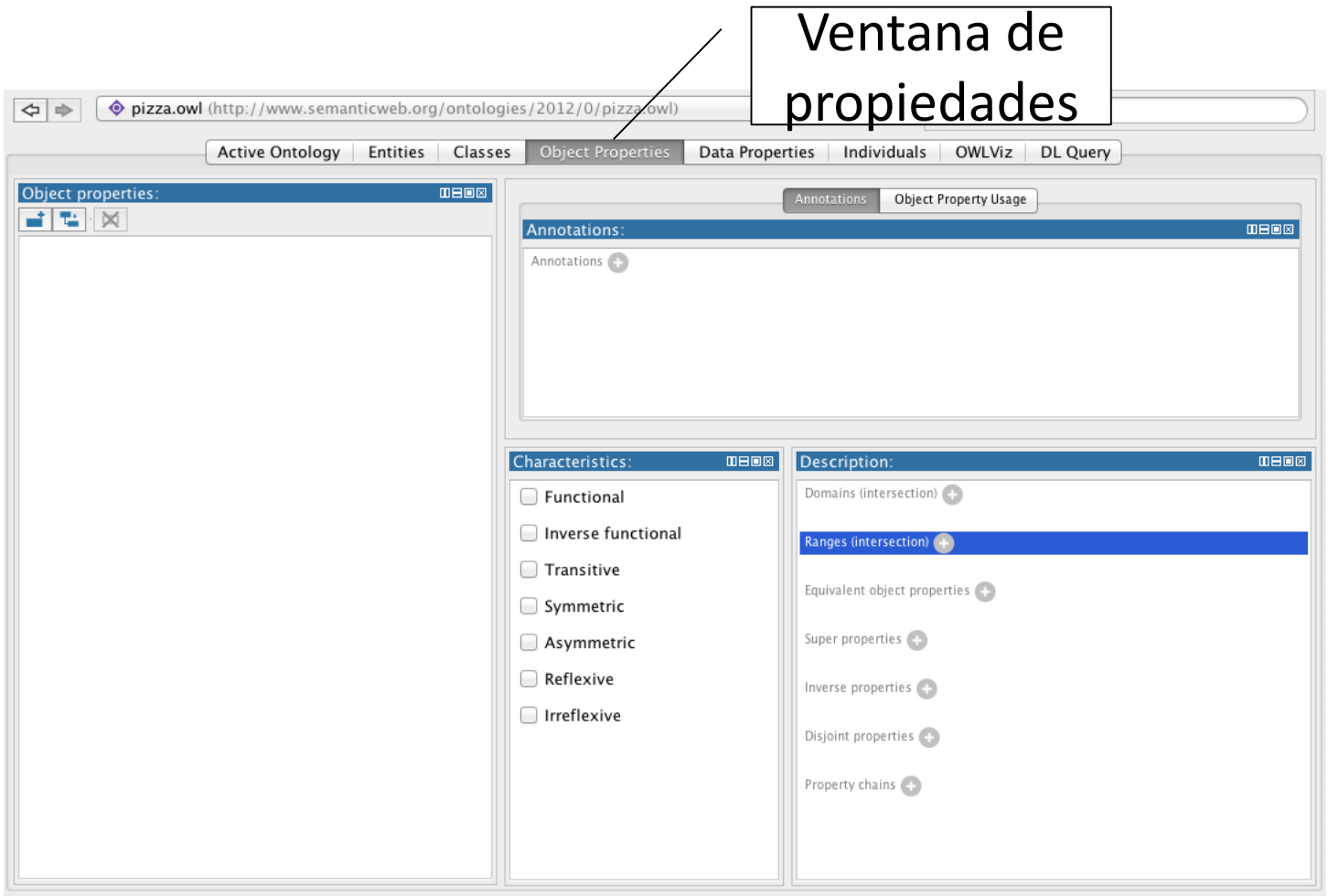
Annotation Properties*



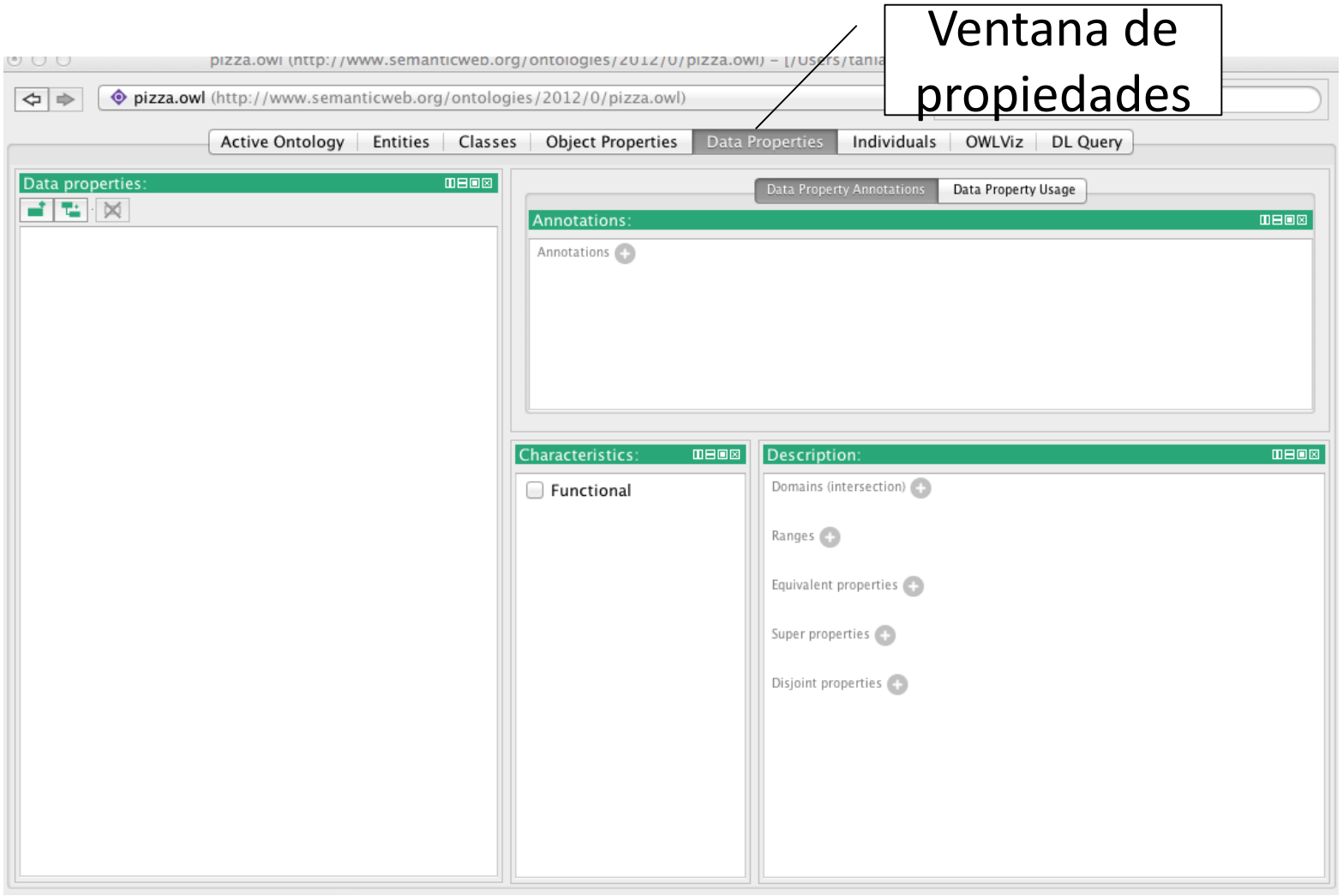
Propiedades owl

- Los elementos que debe tener un ObjectProperty son:
  - nombre,
  - Dominio: hace referencia a la clase o clases iniciales y
  - Rango: hace referencia a la clase o clases finales.
- Por ejemplo la relación es tutor
  - Nombre: es_tutor
  - Dominio: Docente
  - Rango: Estudiante

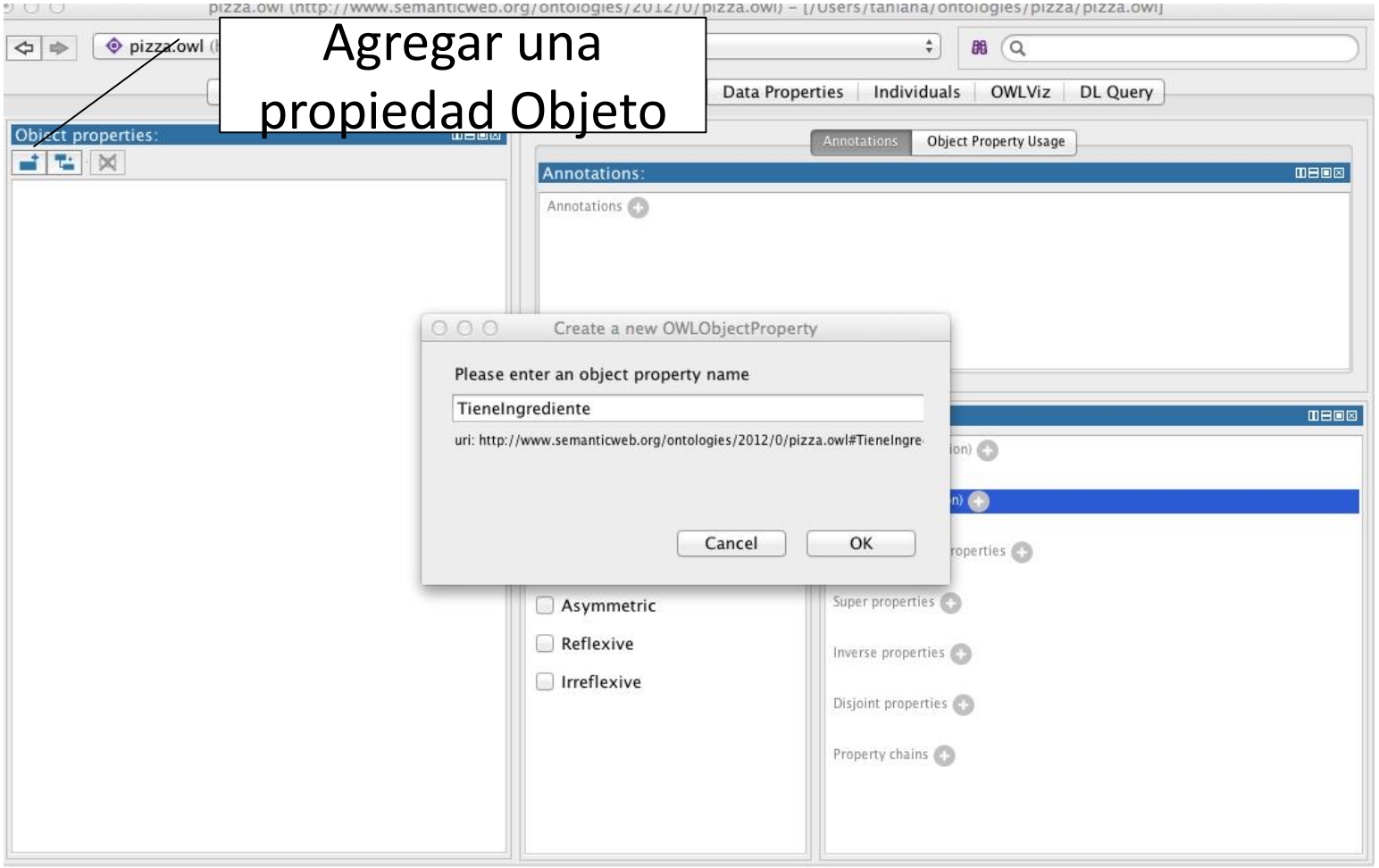
Tab de Propiedades Objetos



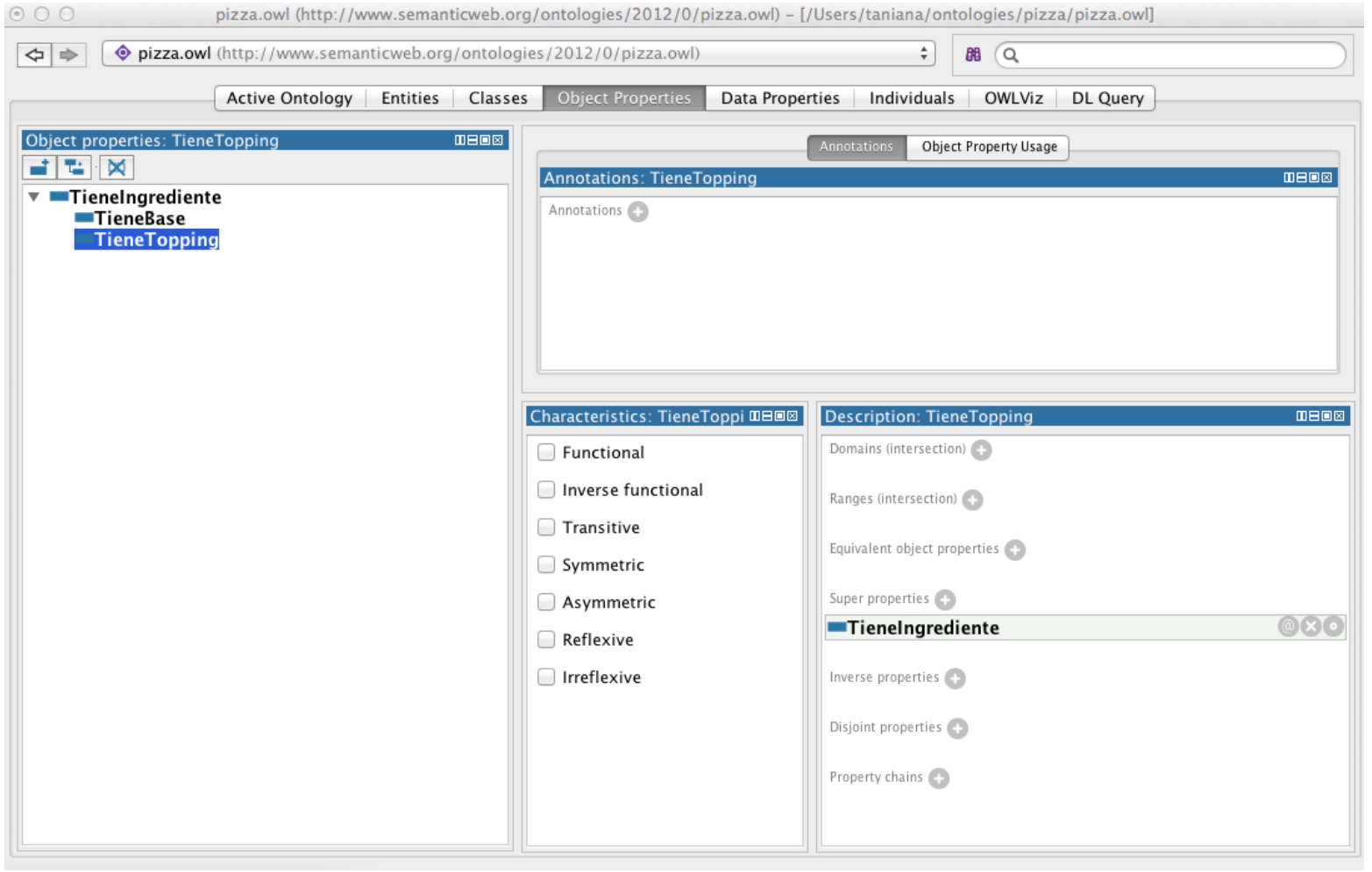
Tab de Propiedades Datos



Crear la propiedad objeto llamada tieneIngrediente



Crear otras propiedades objeto



CARACTERÍSTICAS DE LAS PROPIEDADES

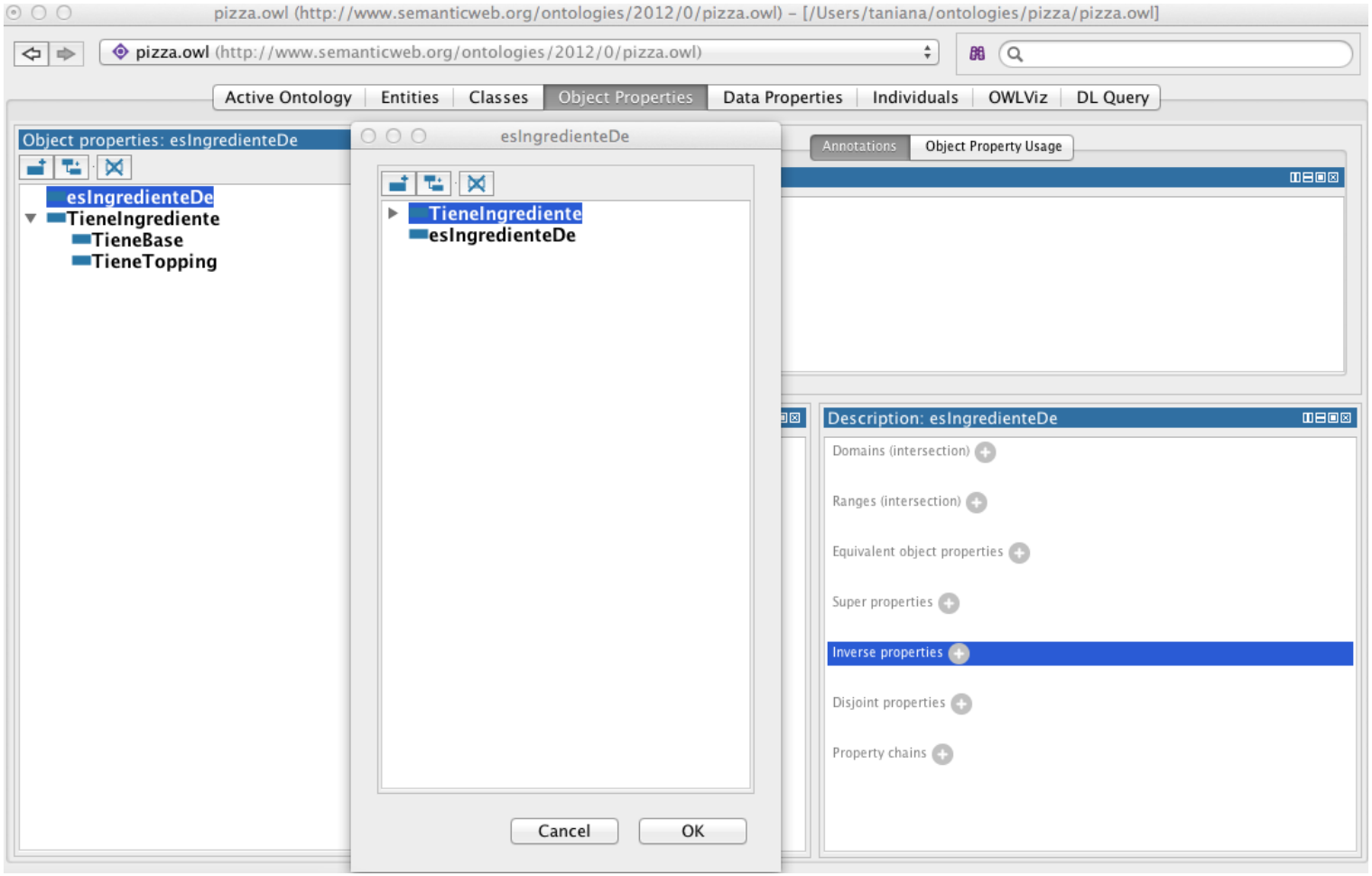
Características de las propiedades

- OWL permite que el significado de las propiedades sea enriquecido con las características de las propiedades, que son:
  - Propiedades funcionales
  - Propiedades funcionales inversas
  - Propiedades transitivas
  - Propiedades simétricas

Propiedades inversa

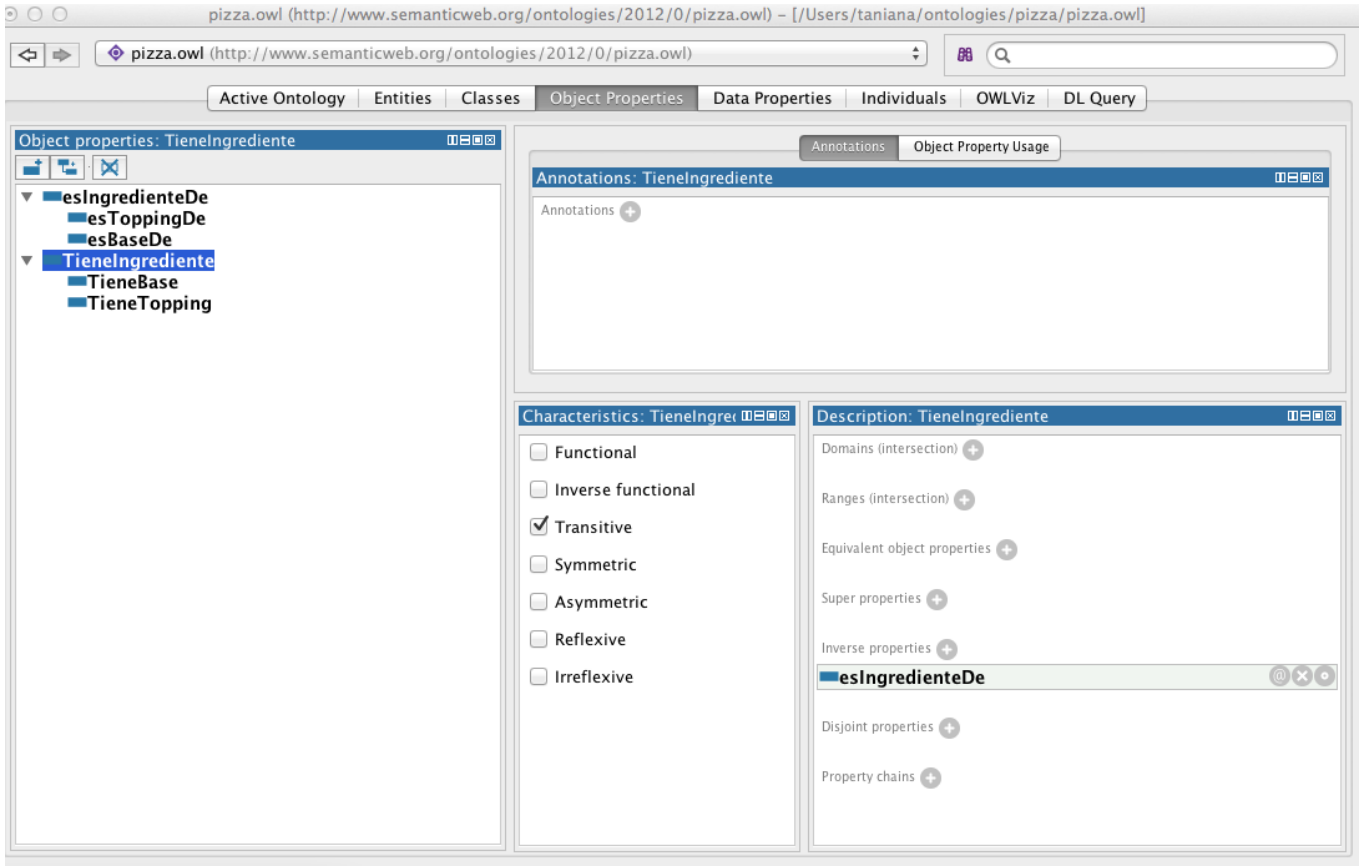
- Cada ObjectProperty debe tener su correspondiente propiedad inversa. Si una propiedad enlaza un objeto A con otro B, entonces la propiedad inversa enlaza el objeto B con el A.

Crear propiedades inversas



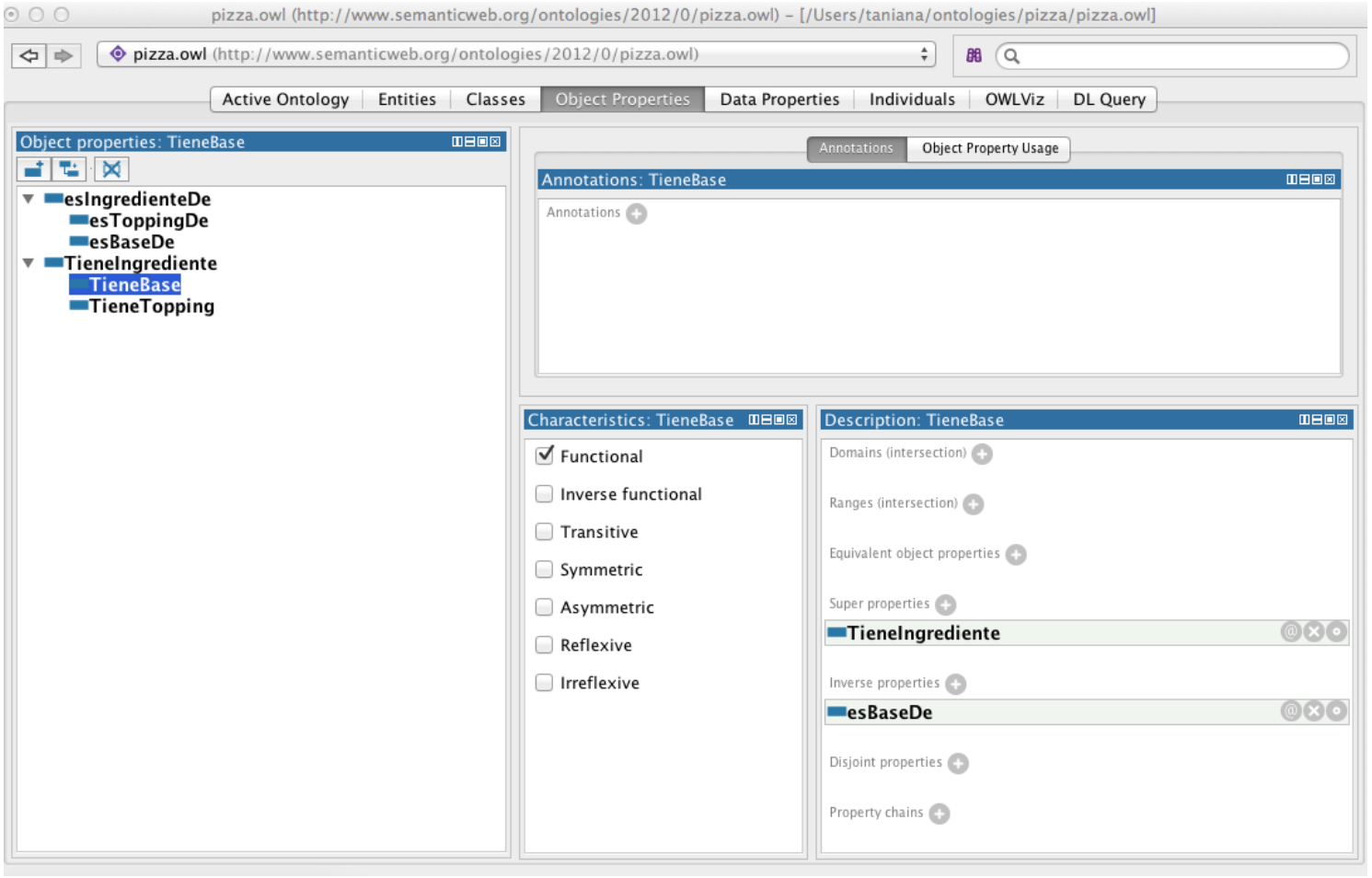
Propiedad Transitiva

Crear la propiedad tieneIngrediente como propiedad Transitiva



Propiedad funcional

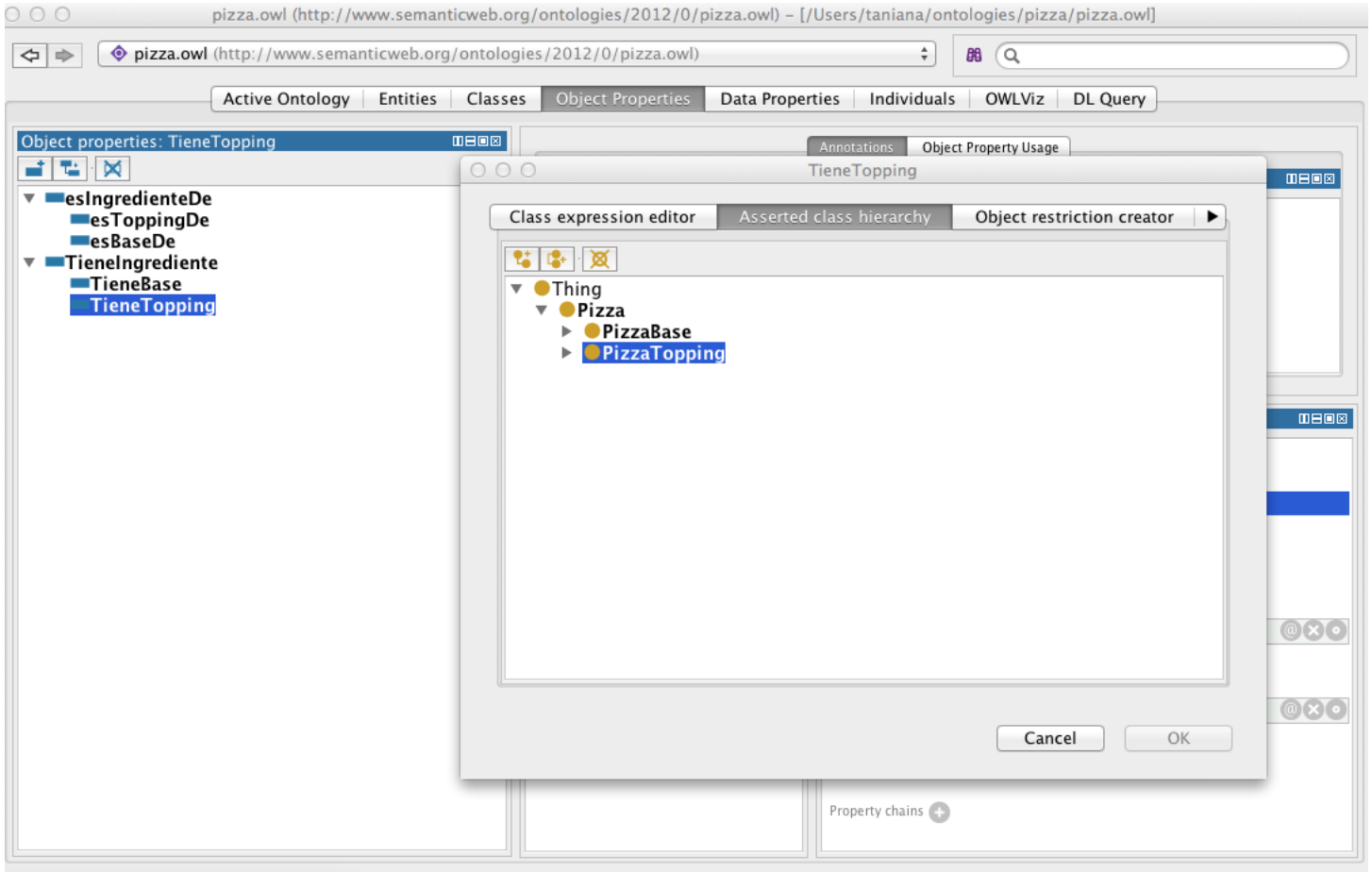
- Crear TieneBase como propiedad funcional



PROPIEDADES DE DOMINIO Y RANGO

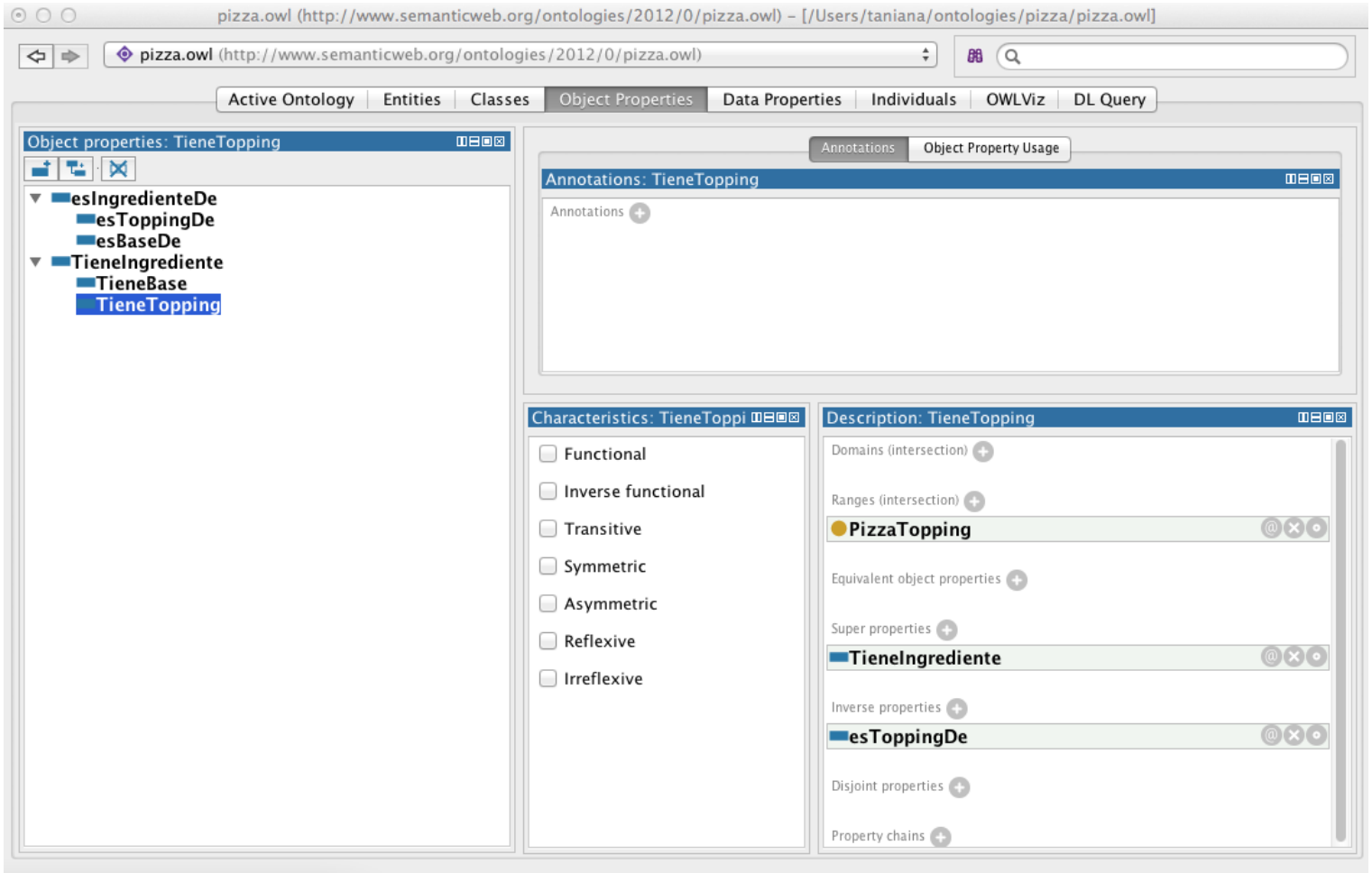
Especificar el rango

Especificar el rango de la propiedad tieneTopping



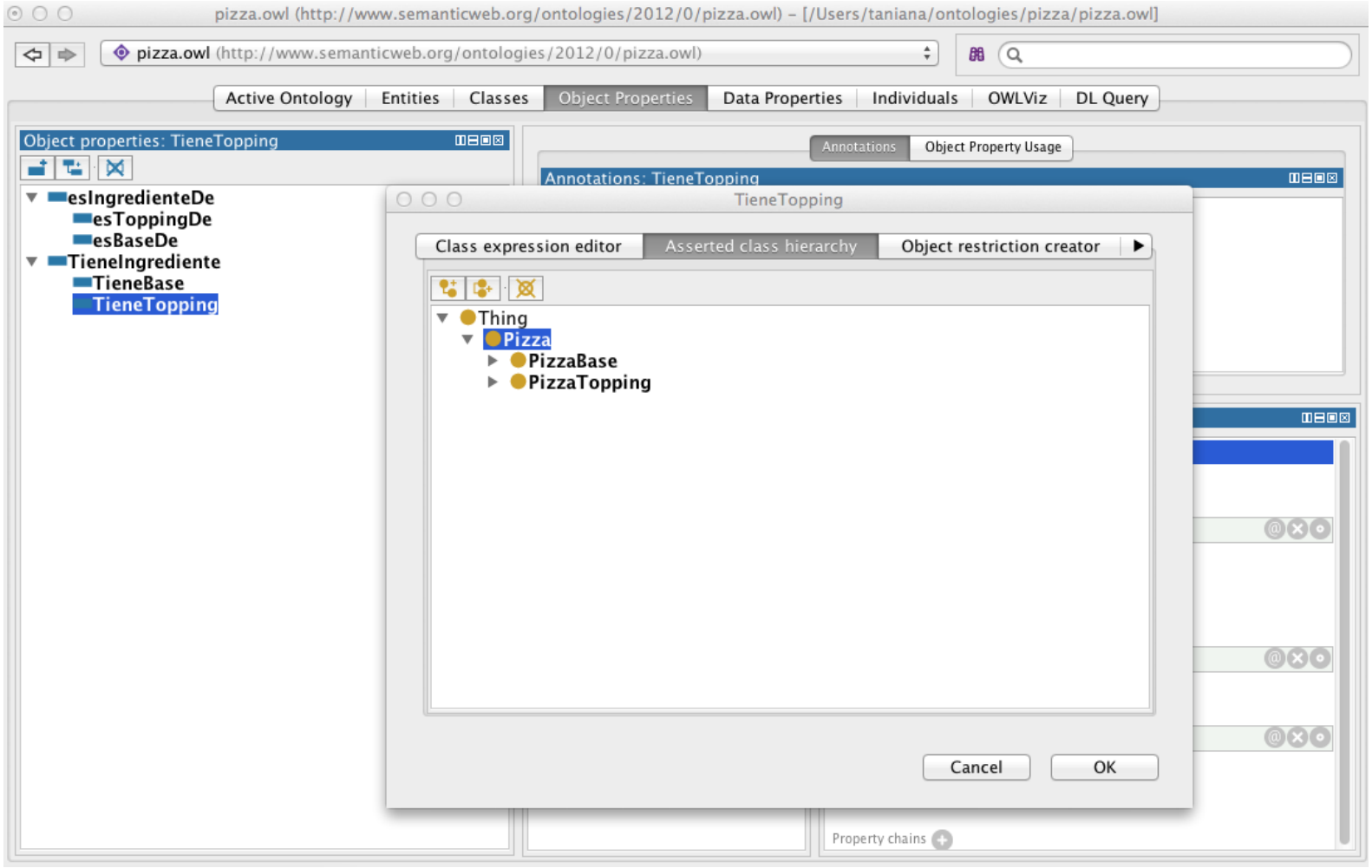
Especificar el rango

Especificar el rango de la propiedad tieneTopping



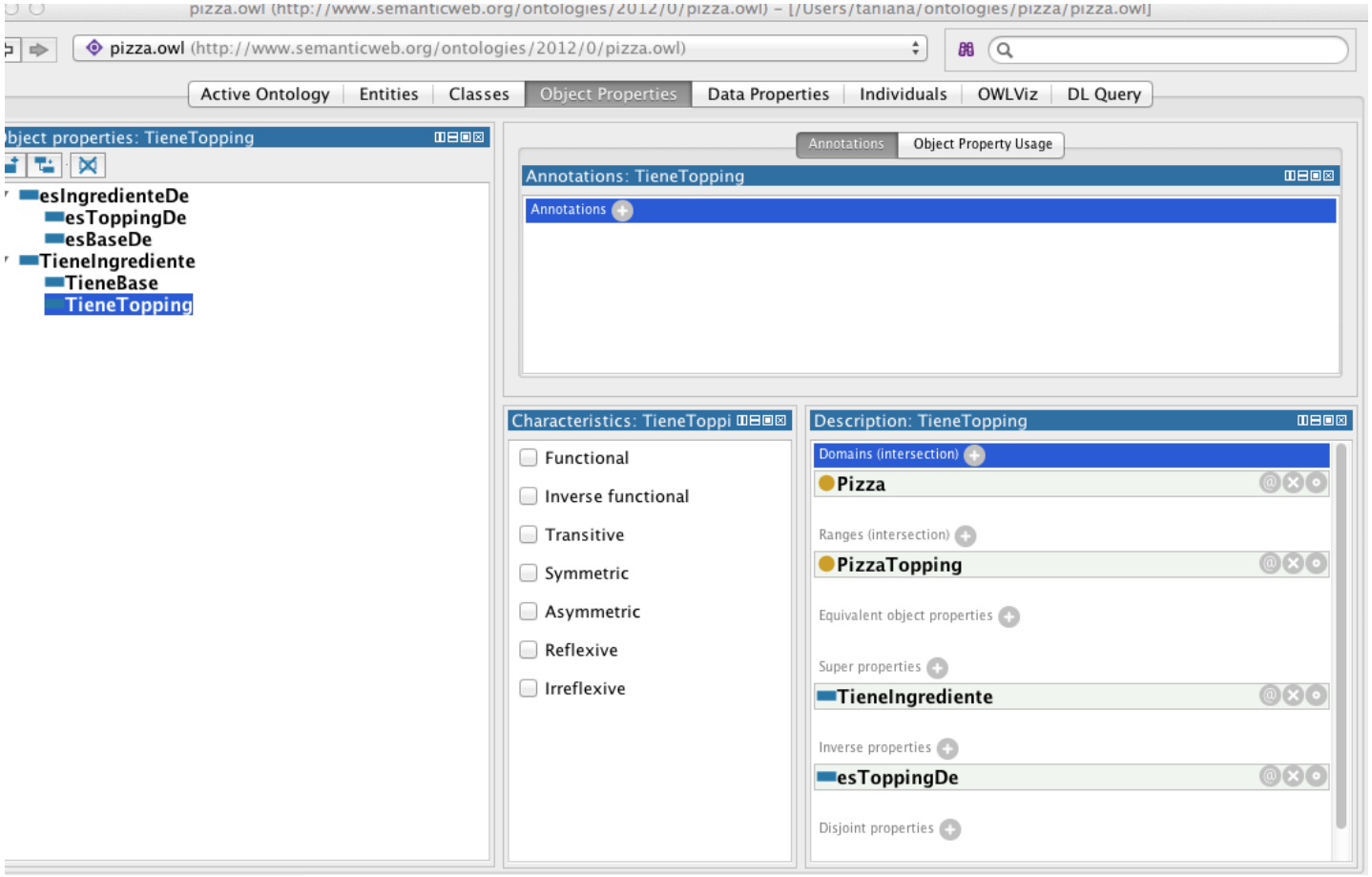
Especificar el Dominio

Especificar el dominio de la propiedad tieneTopping



Especificar el Dominio

Especificar el dominio de la propiedad tieneTopping



DESCRIBIENDO Y DEFINIENDO LAS CLASES

OWL: Constructores sobre la definición de clases y axiomas para las clases, propiedades y individuos

Intersection:	$C_1 \cap ... \cap C_n$	intersectionOf	Human ∩ Male
Union:	$C_1 \cup ... \cup C_n$	unionOf	Doctor ∪ Lawyer
Negation:	¬C	complementOf	¬Male
Nominals:	$\{x_1\} \cup ... \cup \{x_n\}$	oneOf	{john} ∪ ... ∪ {mary}
Universal restriction:	∀P.C	allValuesFrom	∀hasChild.Doctor
Existential restriction:	∃P.C	someValuesFrom	∃hasChild.Lawyer
Maximum cardinality:	≤nP	maxCardinality	≤3hasChild
Minimum cardinality:	≥nP	minCardinality	≥1hasChild
Specific Value:	∃P.{x}	hasValue	∃hasColleague.{Matthew}
Subclass	C1 ⊆ C2	subClassOf	Human ⊆ Animal ∩ Biped
Equivalence	C1 ≡ C2	equivalentClass	Man ≡ Human ∩ Male
Disjointness	C1 ∩ C2 ⊆ ⊥	disjointWith	Male ∩ Female ⊆ ⊥
Subproperty	P1 ⊆ P2	subPropertyOf	hasDaughter ⊆ hasChild
Equivalence	P1 ≡ P2	equivalentProperty	cost ≡ price
Inverse	P1 ≡ P2-	inverseOf	hasChild ≡ hasParent-
Transitive	P+ ⊆ P	TransitiveProperty	ancestor+ ⊆ ancestor
Functional	T ⊆ ≤1P	FunctionalProperty	T ⊆ ≤1hasMother
InverseFunctional	T ⊆ ≤1P-	InverseFunctionalProperty	T ⊆ ≤1hasPassportID-
Equivalence	{x1} ≡ {x2}	sameIndividualAs	{oeg:OscarCorcho}≡{img:Oscar}
Different	{x1} ≡ ¬{x2}	differentFrom, AllDifferent	{john} ≡ ¬{peter}

Fuente: Oscar Corcho García. Representación de Conocimiento: Lógica Descriptiva

Describiendo y definiendo las clases

- Una vez creadas varias propiedades, se pueden utilizar para definir y describir el comportamiento de las clases.
- Restricciones de propiedades
  - Las propiedades son utilizadas para crear restricciones en las clases en una ontología OWL. Usualmente el nombre de la propiedad debería sugerir las restricciones impuestas a los objetos de la clase. Las restricciones OWL se presentan en las siguientes tres categorías:
    - Restricciones de cuantificación.
    - Restricciones de cardinalidad.
    - Restricciones de valor.

Restricciones de Cuantificación

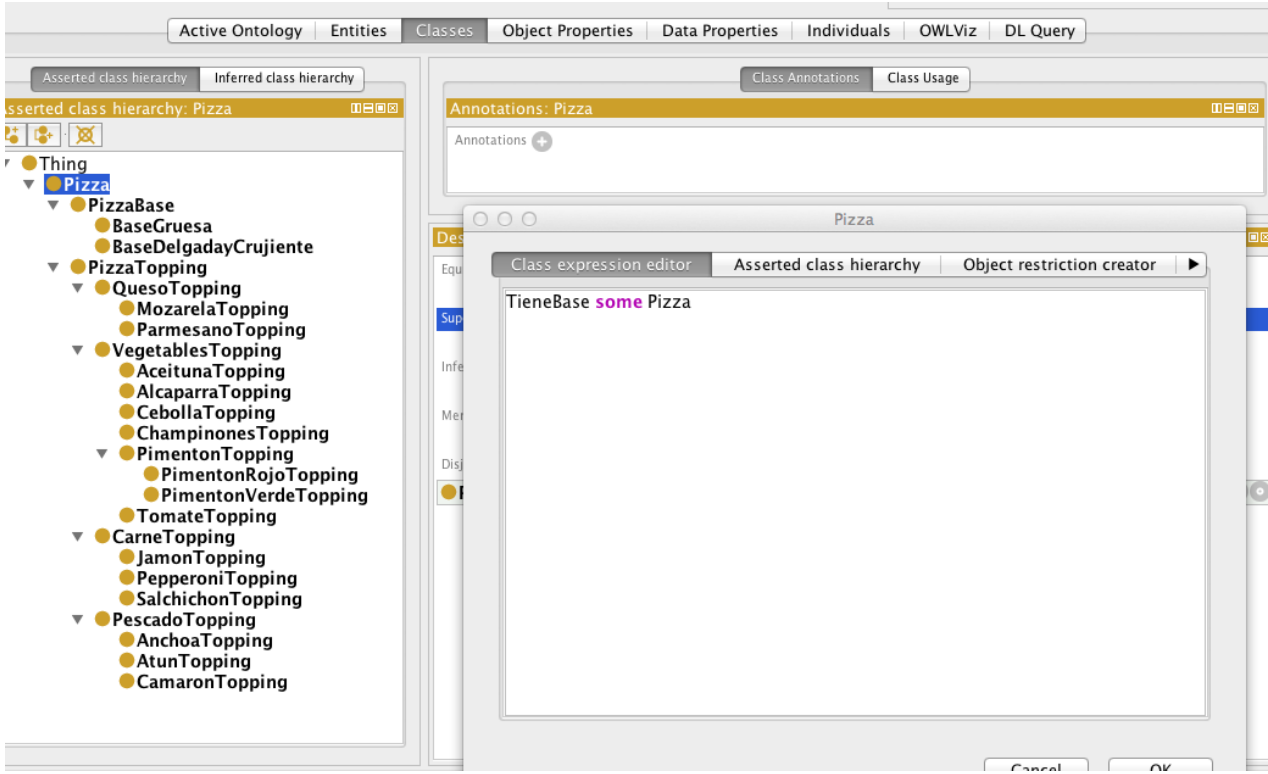
- Las restricciones de cuantificación se componen de los siguientes elementos:

  - Cuantificador existencial ($), el cual permite indicar la existencia de al menos un objeto. En protégé 4. la palabra clave **some** es usado para denotarlo
  - Cuantificador universal ("), el cual permite indicar la existencia de todos los objetos. En protégé 4. la palabra clave es **only** es usado para denotarlo

Cuantificador UNIVERSAL

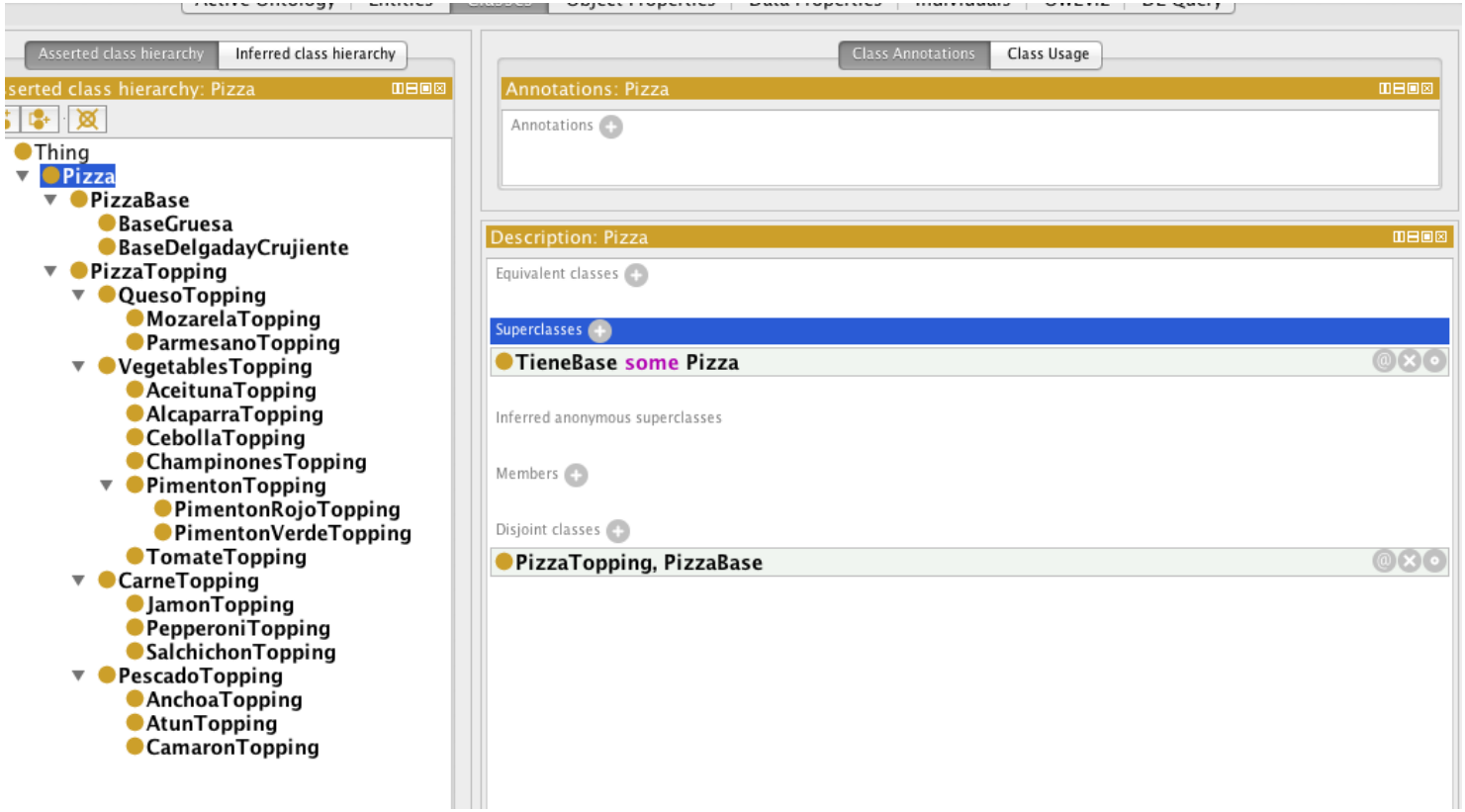
Por ejemplo la restricción para pizza que especifica que una pizza puede tener una PizzaBase

1. Seleccione Pizza
2. Seleccione en el icon de agregar al lado de Superclasse.
3. En la ventana Class expression editor. Coloque TieneBase some Pizza



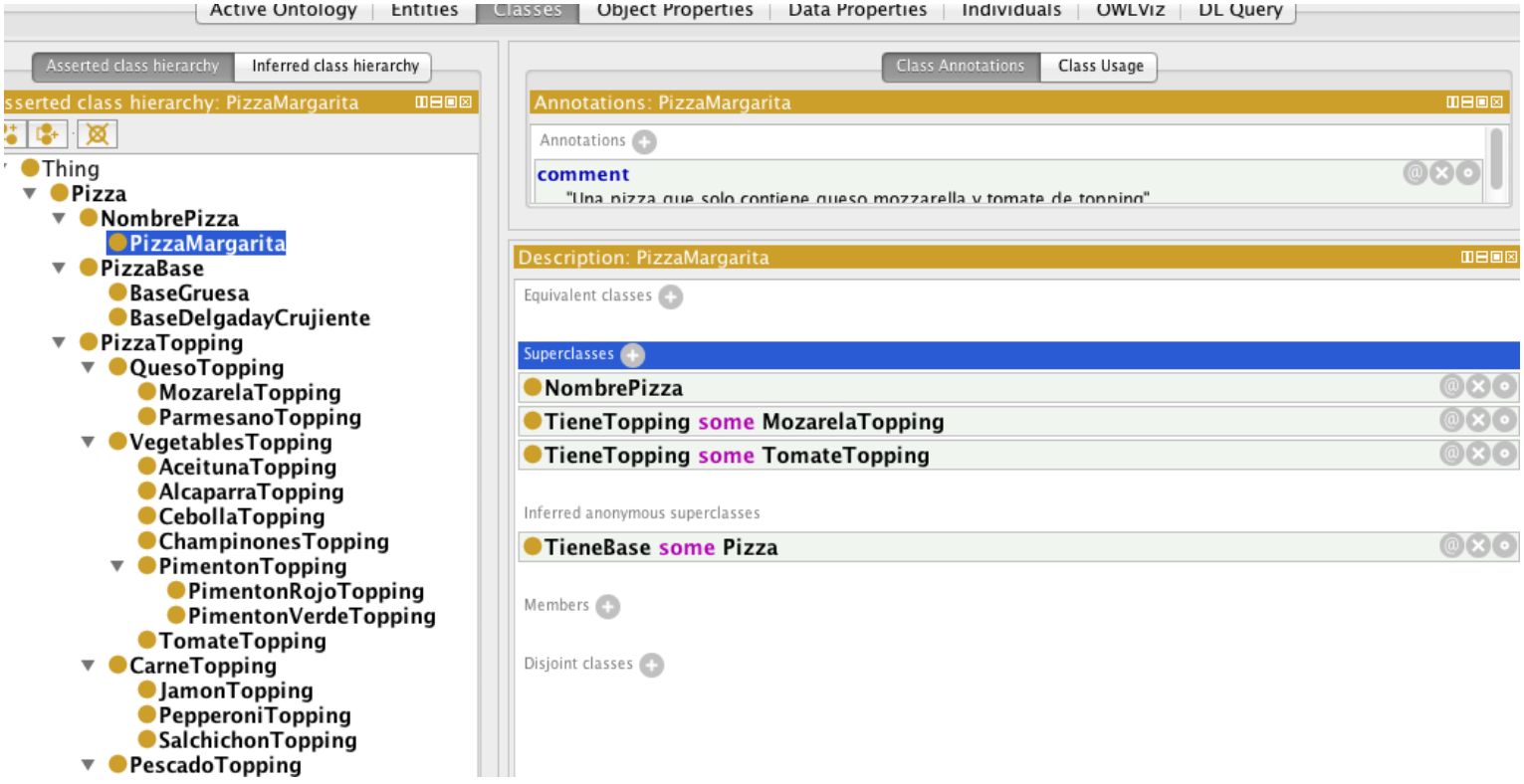
Cuantificador UNIVERSAL

Por ejemplo la restricción para pizza que especifica que una pizza puede tener una PizzaBase



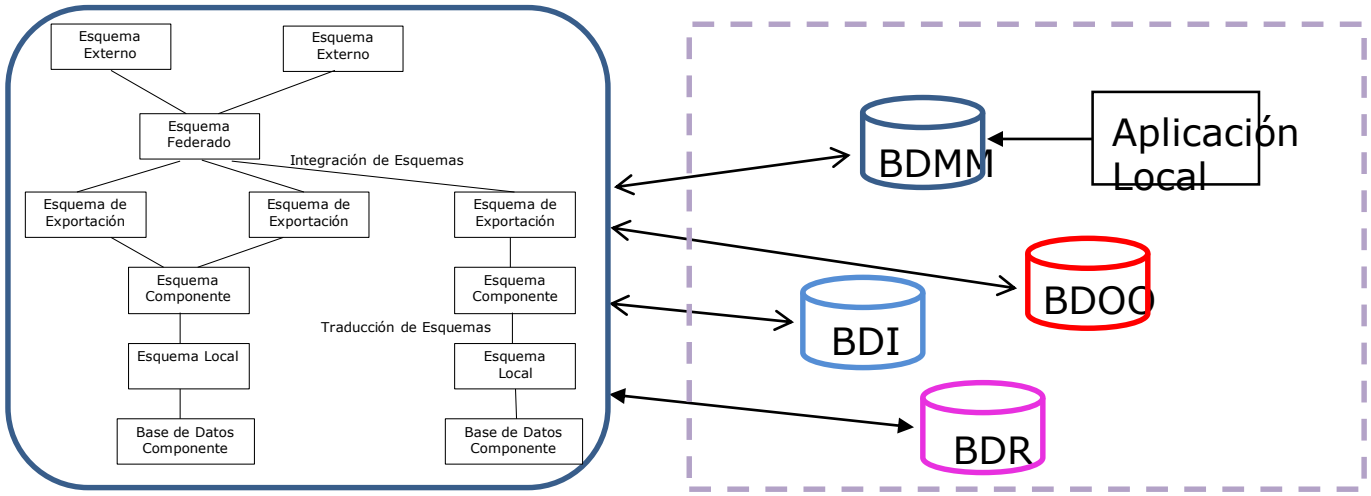
Creando algunas clases de Pizza

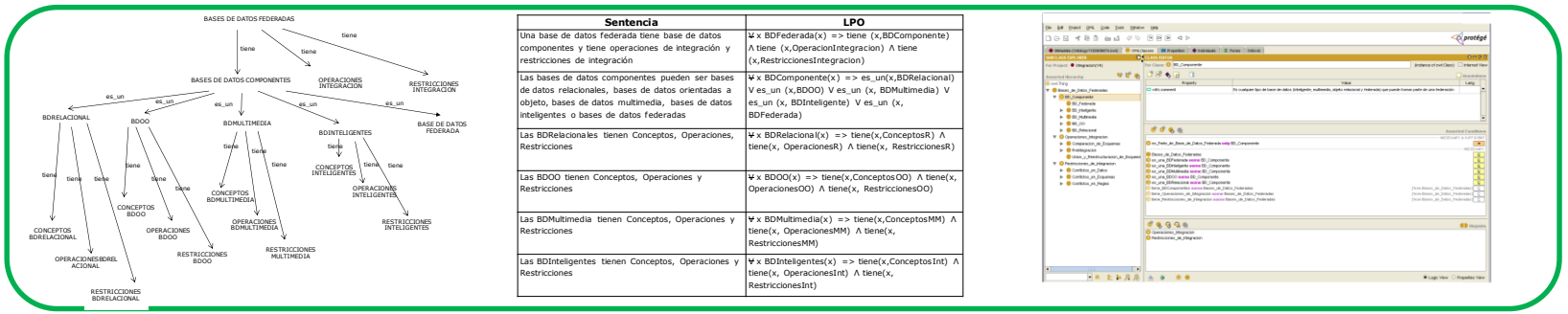
Ya teniendo creada la clase PizzaMargarita Necesitamos especificar que tipo de topping tiene. Por lo tanto necesitamos dos restricciones que diga que tiene MozarellaToping y otra que diga que tiene tomatesTopping



Problema

- Modelar el proceso de integración de bases de datos heterogéneas utilizando ontologías
  - Bases de datos federadas: integran información sintáctica
  - Ontologías: integran información semántica





Sentencia	LPO
Una base de datos federada tiene base de datos componentes y tiene operaciones de integración y restricciones de integración	∀ x BDFederada(x) => tiene (x,BDComponente) ∧ tiene (x,OperacionIntegracion) ∧ tiene (x,RestriccionesIntegracion)
Las bases de datos componentes pueden ser bases de datos relacionales, bases de datos orientadas a objeto, bases de datos multimedia, bases de datos inteligentes o bases de datos federadas	∀ x BDComponente(x) => es_un(x,BDRelacional) ∨ es_un (x,BDOO) ∨ es_un (x, BDMultimedia) ∨ es_un (x, BDInteligente) ∨ es_un (x, BDFederada)
Las BDRelacionales tienen Conceptos, Operaciones, Restricciones	∀ x BDRelacional(x) => tiene(x,ConceptosR) ∧ tiene(x, OperacionesR) ∧ tiene(x, RestriccionesR)
Las BDOO tienen Conceptos, Operaciones y Restricciones	∀ x BDOO(x) => tiene(x,ConceptosOO) ∧ tiene(x, OperacionesOO) ∧ tiene(x, RestriccionesOO)
Las BDMultimedia tienen Conceptos, Operaciones y Restricciones	∀ x BDMultimedia(x) => tiene(x,ConceptosMM) ∧ tiene(x, OperacionesMM) ∧ tiene(x, RestriccionesMM)
Las BDInteligentes tienen Conceptos, Operaciones y Restricciones	∀ x BDInteligentes(x) => tiene(x,ConceptosInt) ∧ tiene(x, OperacionesInt) ∧ tiene(x, RestriccionesInt)

Bases de Datos Componentes

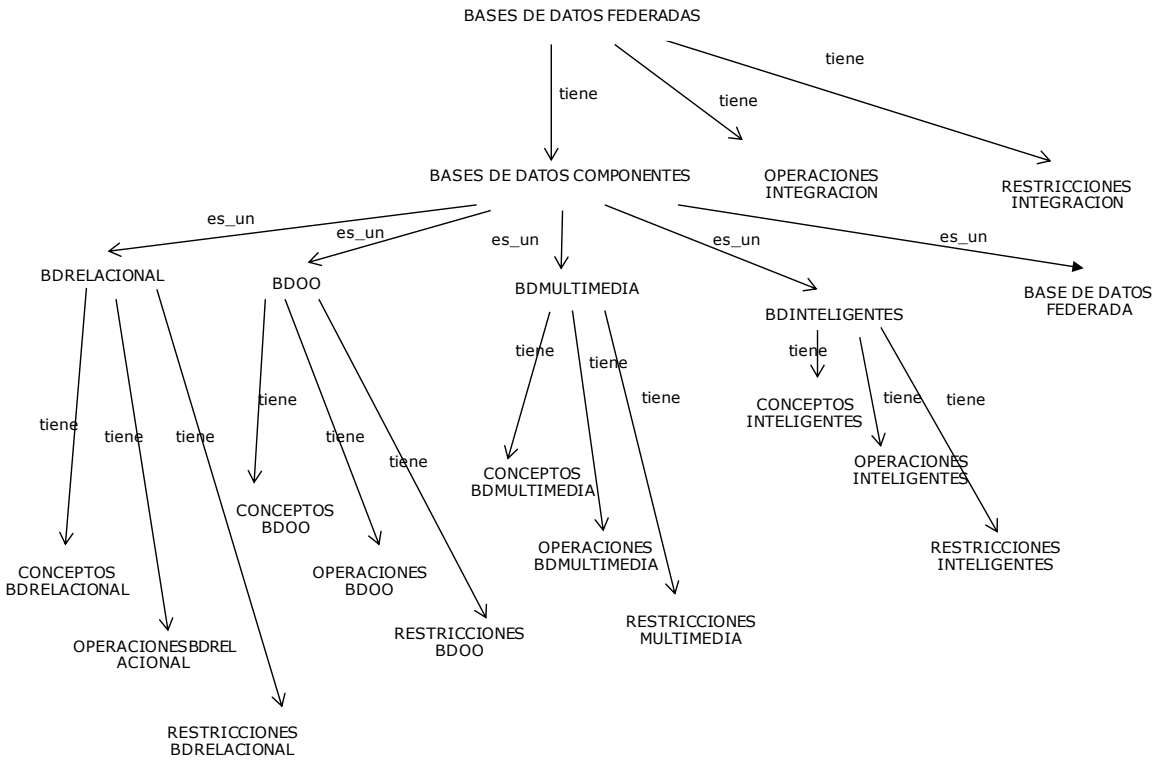
- Bases de Datos Inteligentes
  - Hechos y reglas
- Bases de Datos Multimedia
  - Objetos Multimedia: Texto, Audio, Video, Imagen
- Bases de Datos Orientadas a Objeto
  - Objetos, Clases, Métodos
- Bases de Datos Relacionales
  - Tablas, Relaciones

Modelo Ontológico

- Conceptos
  - Descripción de los elementos que conforman el modelo
- Operaciones
  - Descripción de las actividades que pueden realizar los elementos del modelo
- Restricciones
  - Describen el comportamiento del modelo

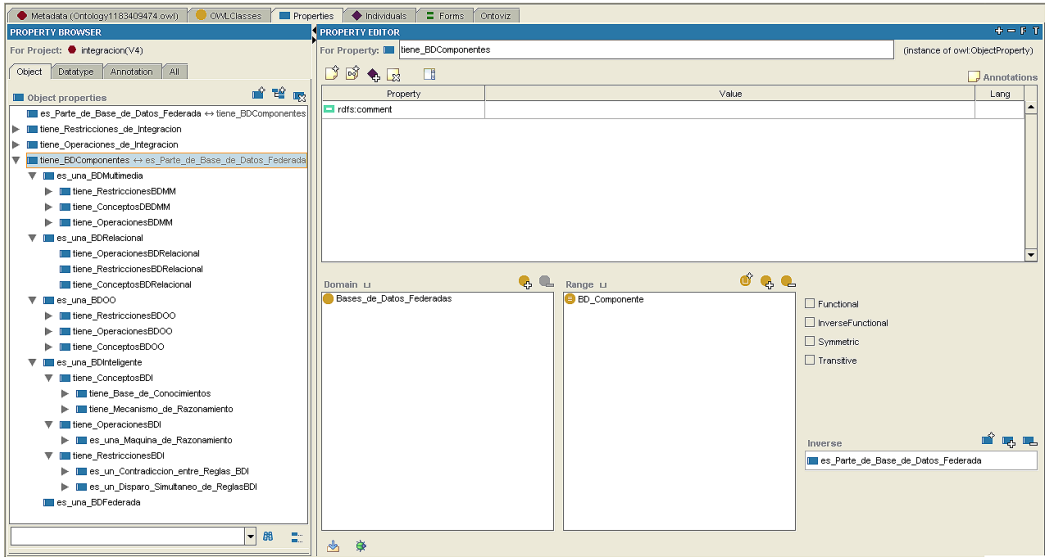
Integración

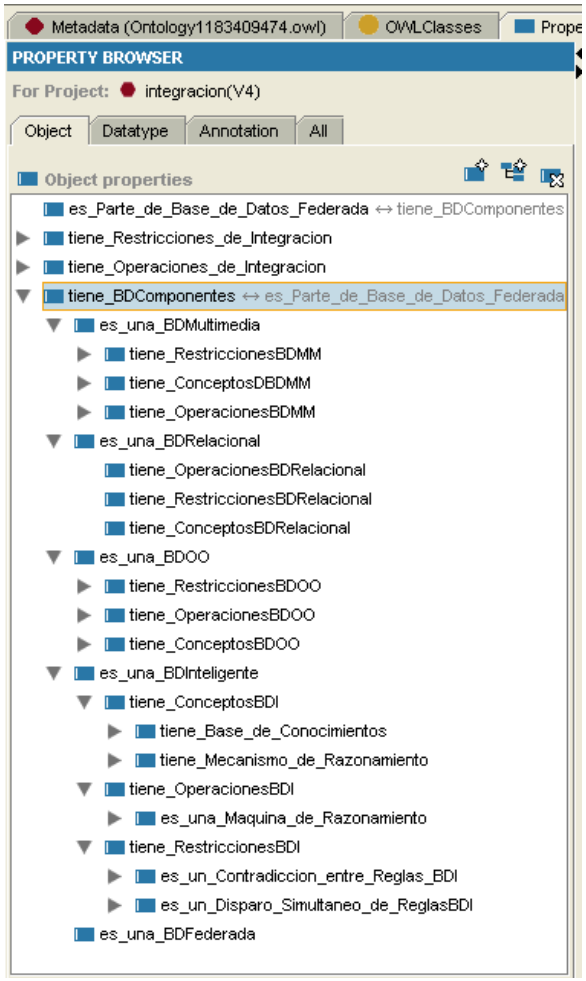
Taxonomía de Conceptos para Bases de Datos Federadas

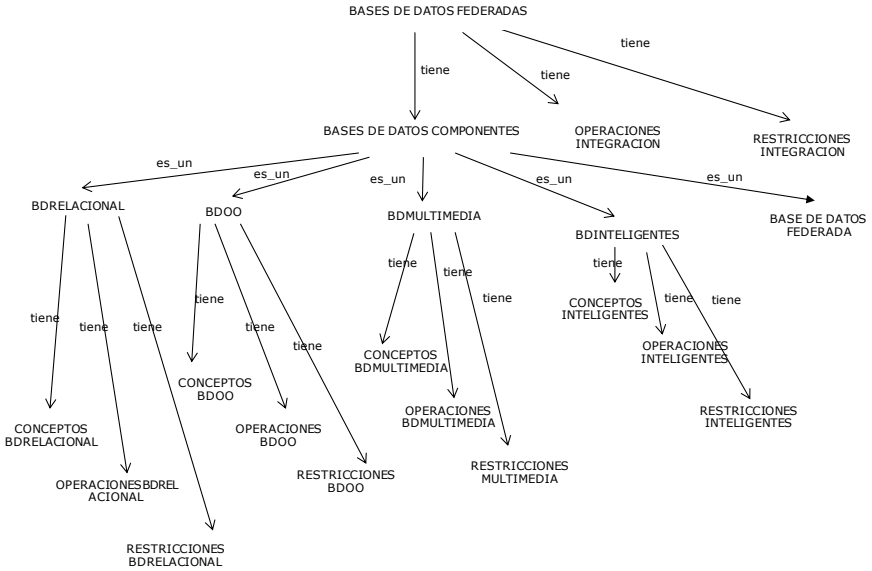


Integración

Propiedades de las Bases de Datos Federadas

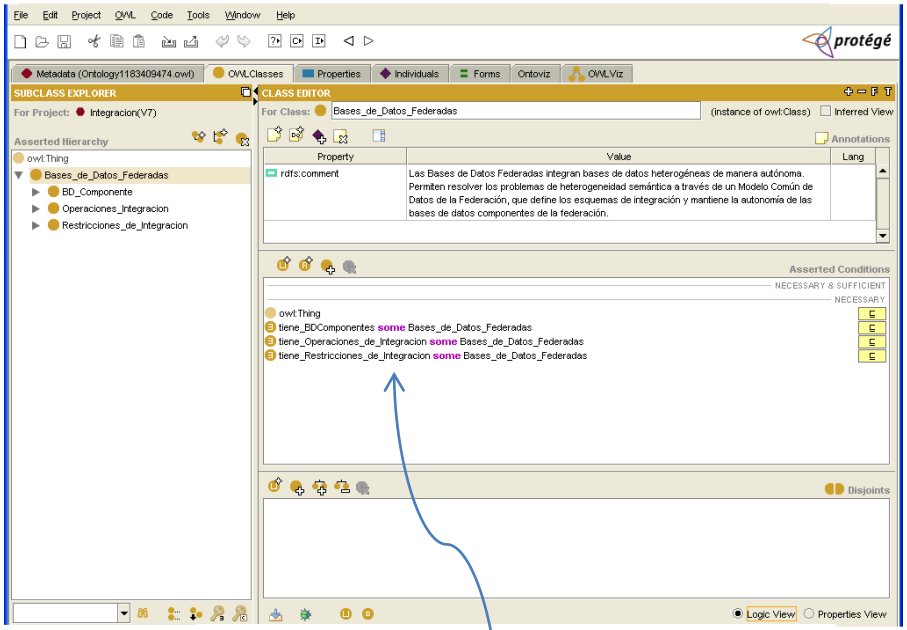


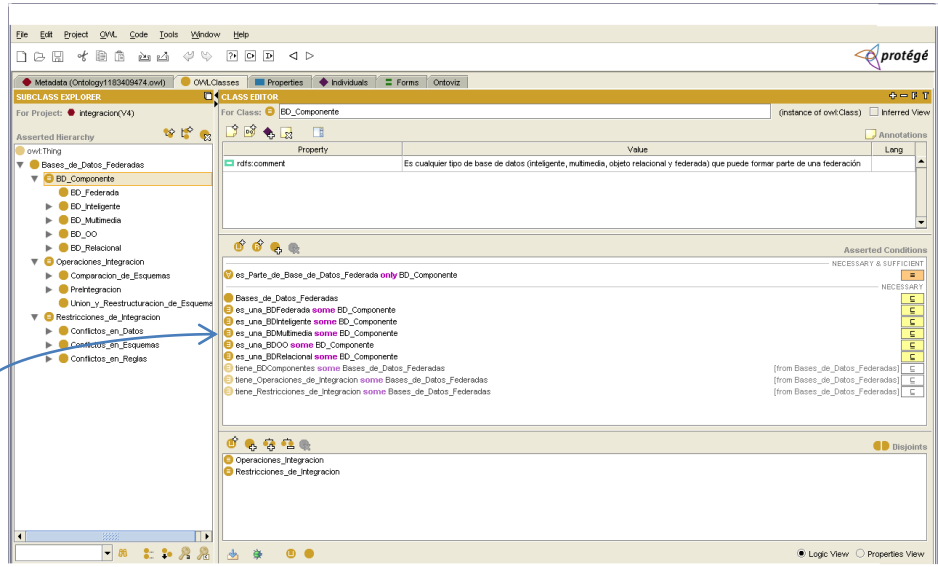




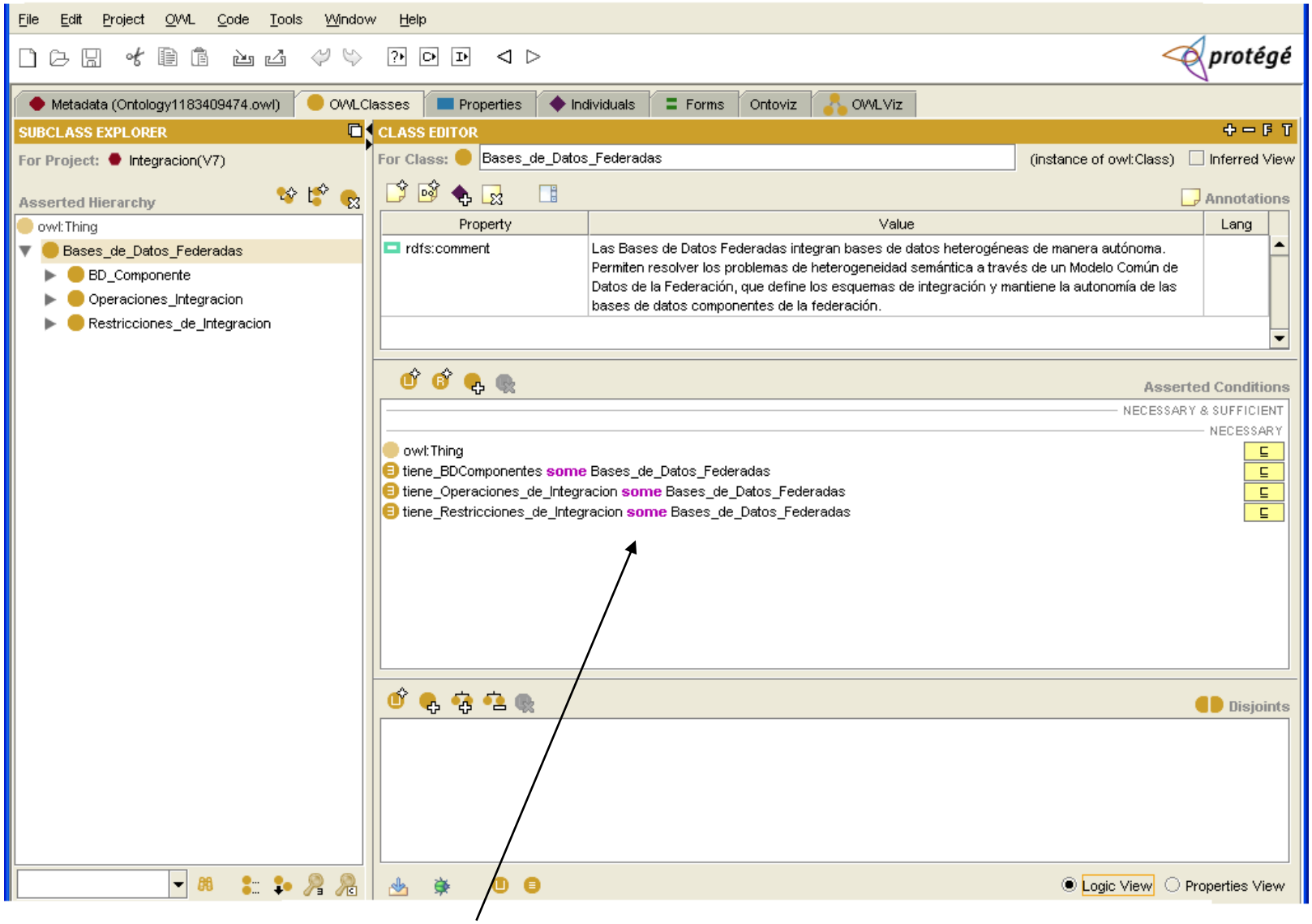
Integración

Axiomas de Conceptos Bases de Datos Federadas

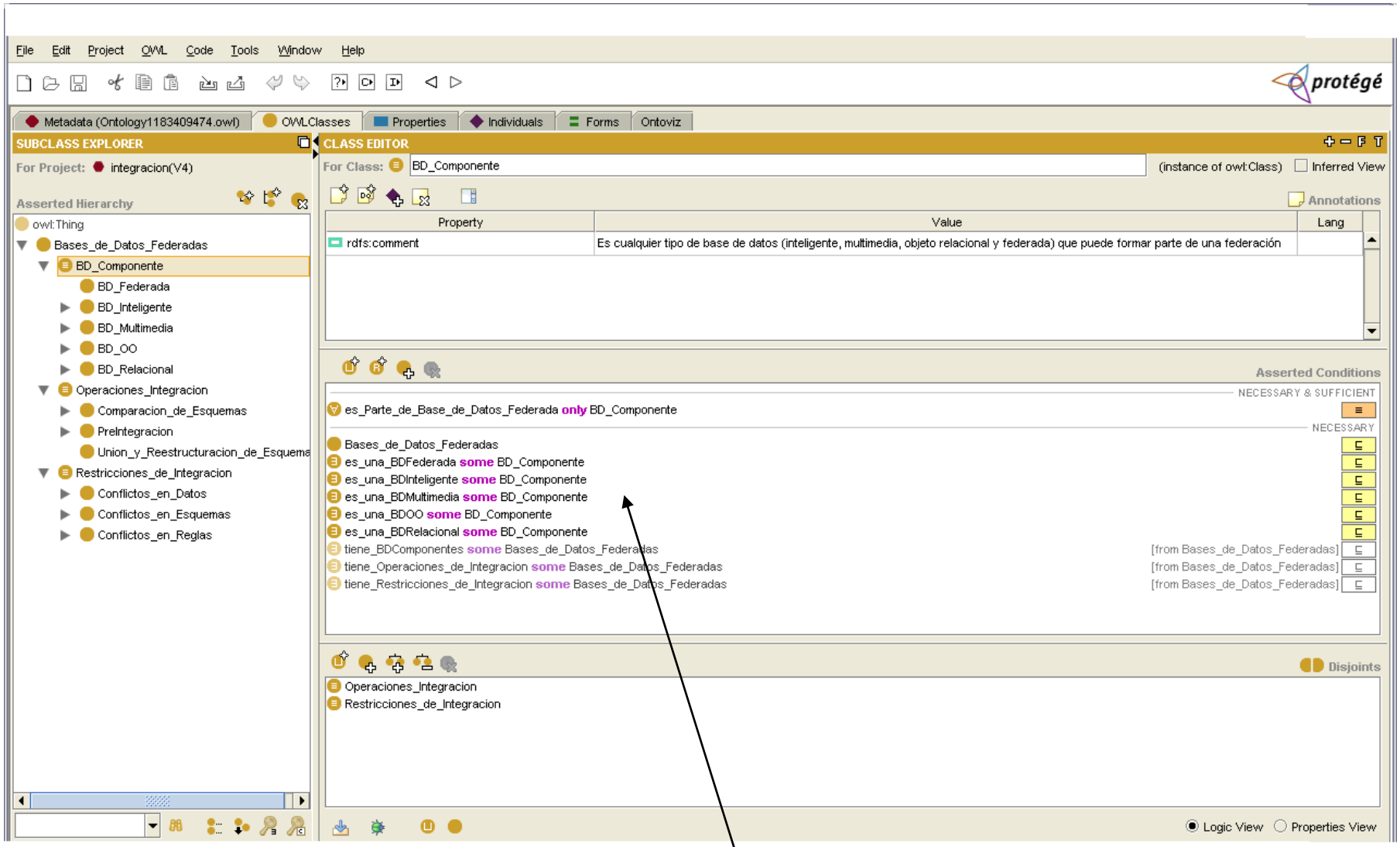




Sentencia	LPO
Una base de datos federada tiene base de datos componentes y tiene operaciones de integración y restricciones de integración	∀ x BDFederada(x) => tiene (x,BDComponente) ∧ tiene (x,OperacionIntegracion) ∧ tiene (x,RestriccionesIntegracion)
Las bases de datos componentes pueden ser bases de datos relacionales, bases de datos orientadas a objeto, bases de datos multimedia, bases de datos inteligentes o bases de datos federadas	∀ x BDComponente(x) => es_un(x,BDRelacional) V es_un (x,BDOO) V es_un (x, BDMultimedia) V es_un (x, BDInteligente) V es_un (x, BDFederada)
Las BDRelacionales tienen Conceptos, Operaciones, Restricciones	∀ x BDRelacional(x) => tiene(x,ConceptosR) ∧ tiene(x, OperacionesR) ∧ tiene(x, RestriccionesR)
Las BDOO tienen Conceptos, Operaciones y Restricciones	∀ x BDOO(x) => tiene(x,ConceptosOO) ∧ tiene(x, OperacionesOO) ∧ tiene(x, RestriccionesOO)
Las BDMultimedia tienen Conceptos, Operaciones y Restricciones	∀ x BDMultimedia(x) => tiene(x,ConceptosMM) ∧ tiene(x, OperacionesMM) ∧ tiene(x, RestriccionesMM)
Las BDInteligentes tienen Conceptos, Operaciones y Restricciones	∀ x BDInteligentes(x) => tiene(x,ConceptosInt) ∧ tiene(x, OperacionesInt) ∧ tiene(x, RestriccionesInt)



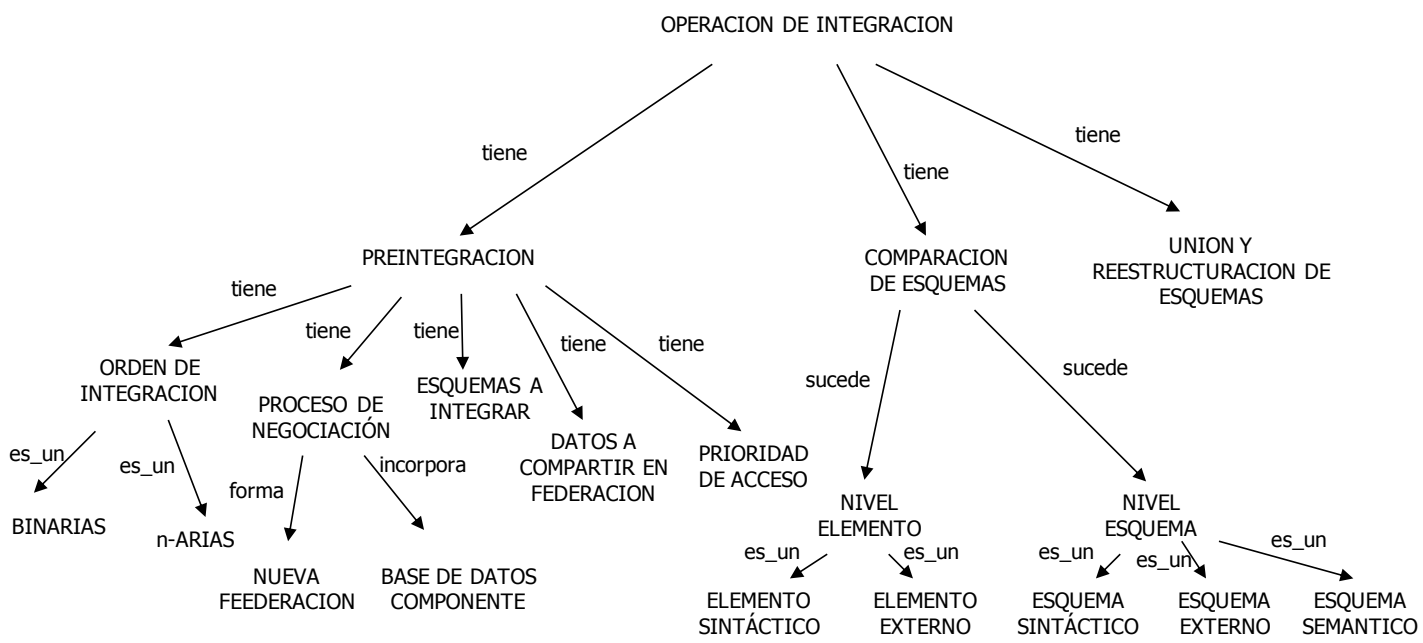
V x BDFederada(x) => tiene (x,BDComponente) Λ tiene (x,OperacionIntegracion) Λ tiene (x,RestriccionesIntegracion)

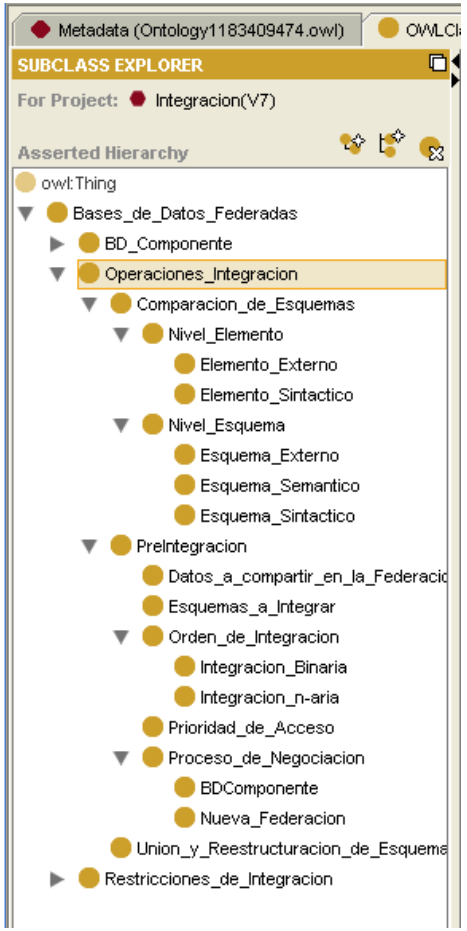


V x BDComponente(x) => es_un(x,BDRelacional) V es_un (x,BDOO) V es_un (x, BDMultimedia)
V es_un (x, BDInteligente) V es_un (x, BDFederada)

Integración

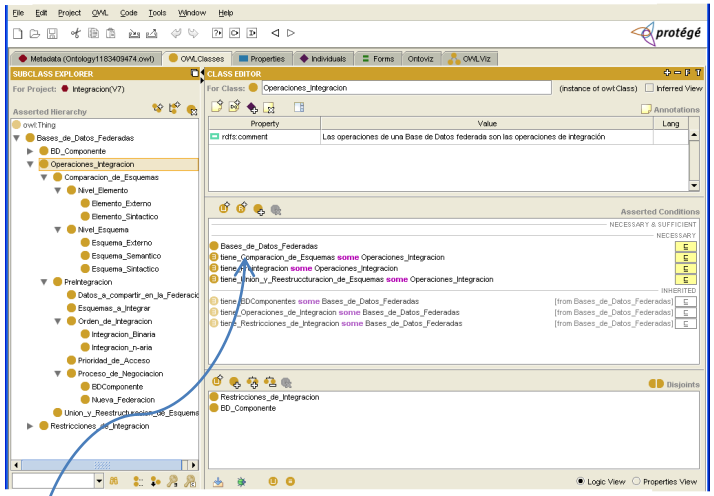
Taxonomía de Operaciones de las Bases de Datos Federadas

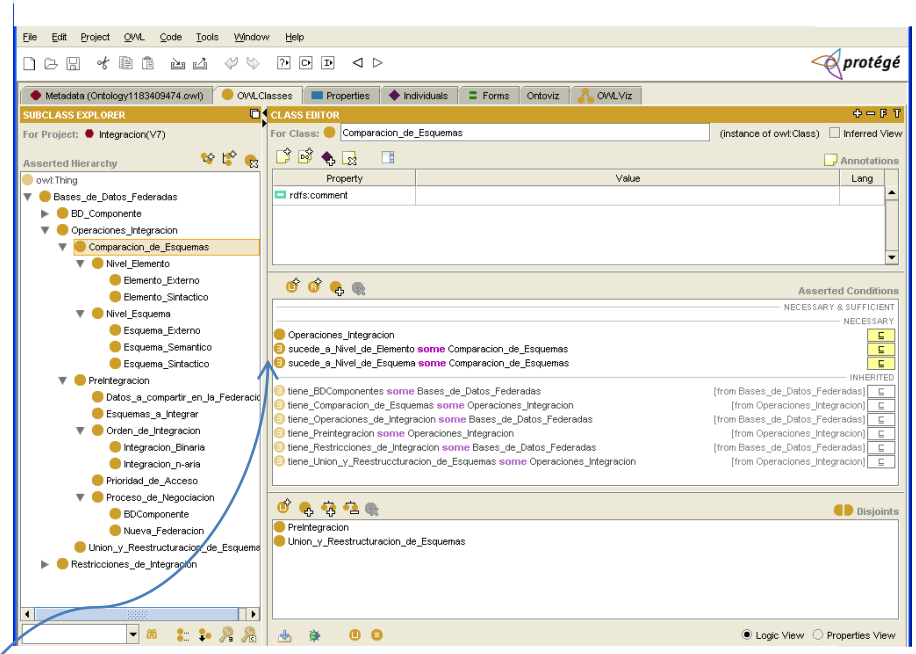


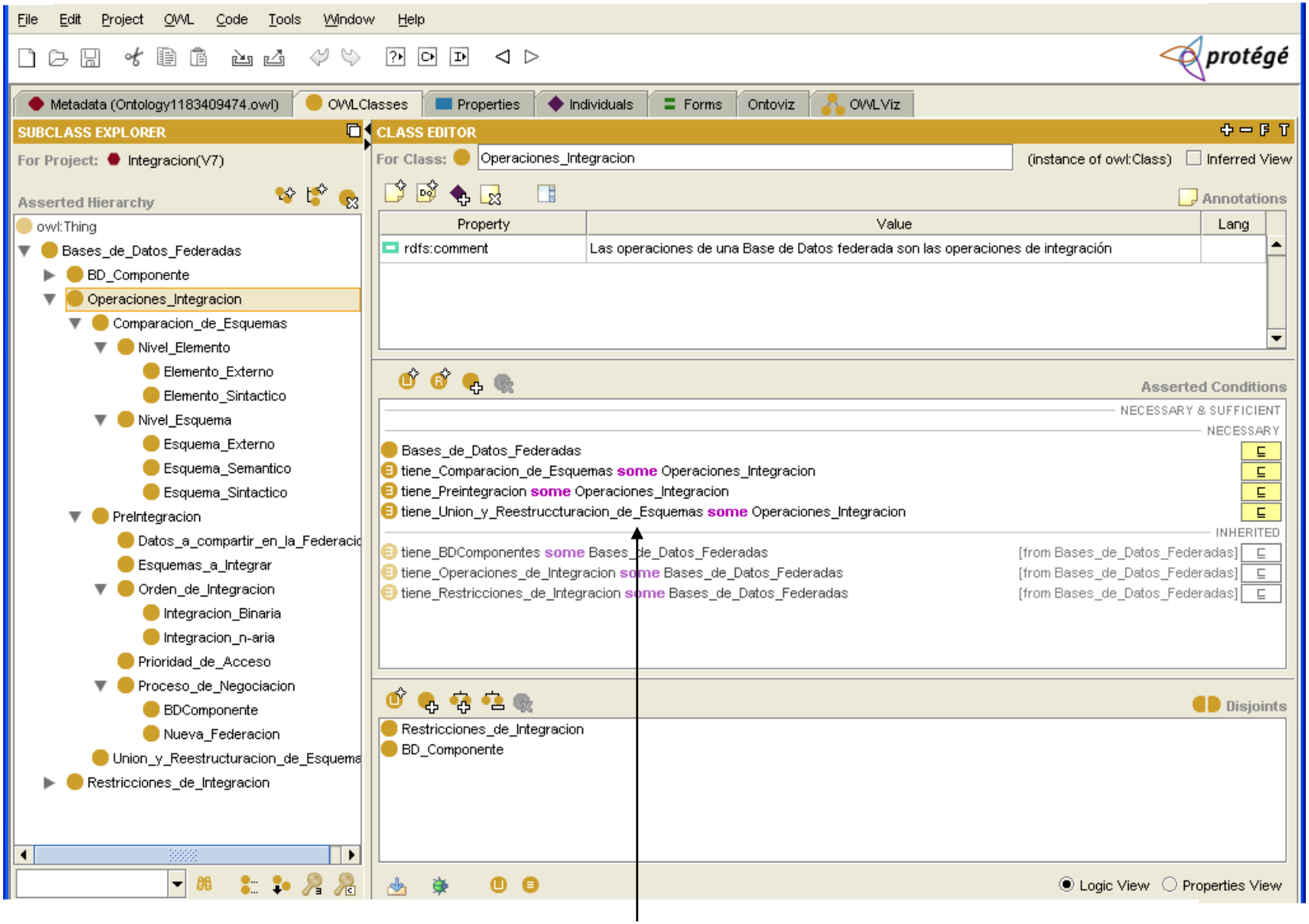


Integración

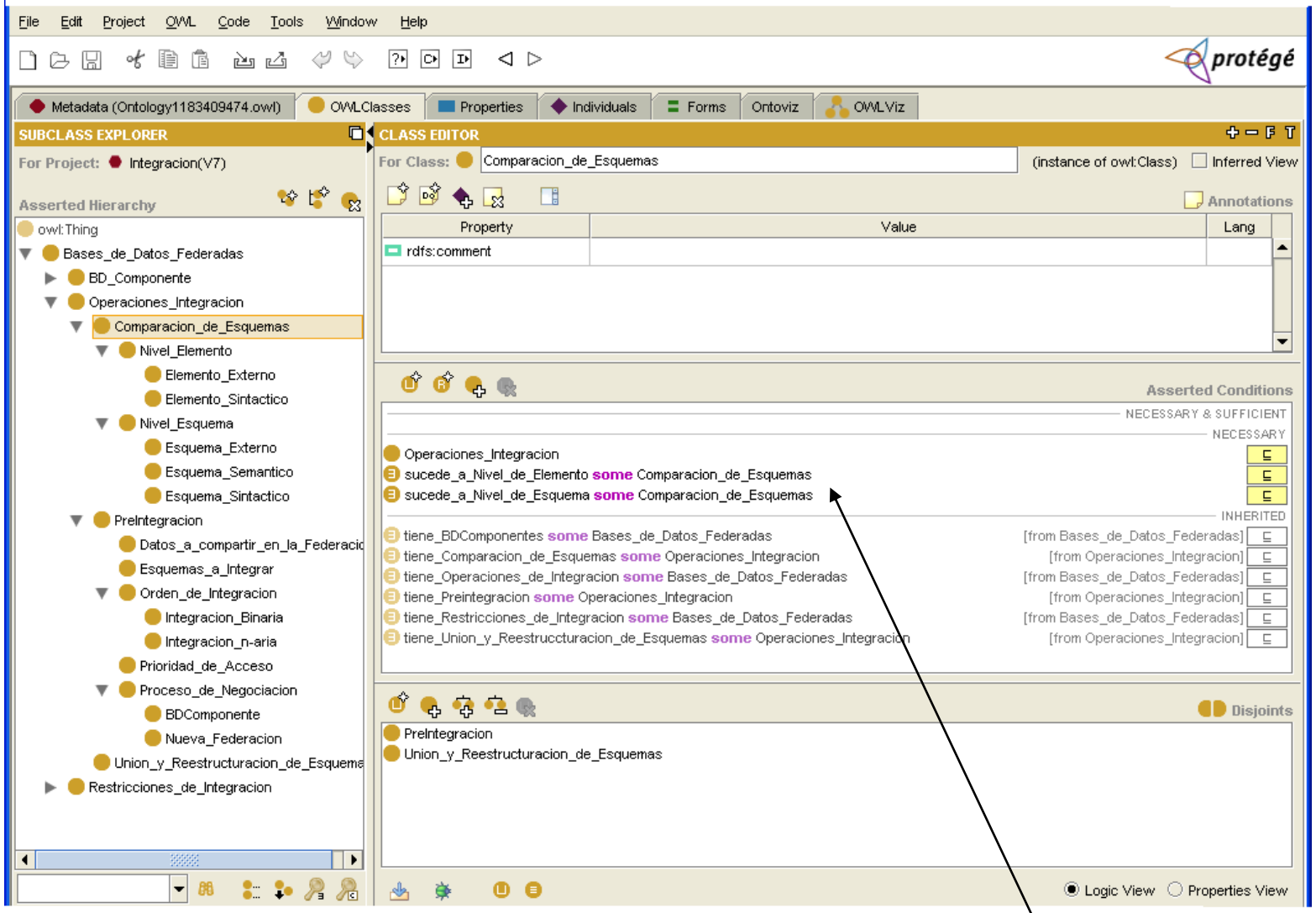
Axiomas de Operaciones de Bases de Datos Federadas







V x OperaciónIntegración(x) => tiene (x,Preintegración) Λ tiene(x,ComparacióndeEsquemas) Λ tiene (x,Union_ReestructuraciónEsquemas) Λ tiene (x, UnionyReestructuraciondeEsquemas)



V x,y ComparacionEsquemas(x,y) => Sucede ((x,y),NivelElemento) V Sucede((x,y),NivelEsquema)

ONTOLOGIA BDMM

CONCEPTO	LPO	COMENTARIO
El objeto de aprendizaje curso Gestión de Conocimiento tiene metadatos texto, metadatos audio, metadatos imagen	ConceptoBDMultimedia (ObjetoAprendizajeGC) => tiene(ObjetoAprendizajeGC,MetadatoTexto) V tiene (ObjetoAprendizajeGC,MetadatoAudio) V tiene (ObjetoAprendizajeGC,MetadatoImagen) V tiene (ObjetoAprendizajeGC,MetadatoVideo)	∀ x ConceptoBDMultimedia(x) => tiene(x,MetadatoTexto) V tiene(x,MetadatoImagen) V tiene(x,MetadatoAudio) V tiene(x,MetadatoVideo) (Axioma tabla 11)
El objeto de aprendizaje curso Gestión de Conocimiento es un objeto multimedia	ConceptoBDMultimedia (ObjetoAprendizajeGC) => es_un (ObjetoAprendizajeGC,ObjetoMM)	∀ x ConceptoBDMultimedia(x) => es_un(x,ObjetoMM) (Axioma tabla 11)
El metadato texto del curso GC tiene Descriptores de documento, histórico de documento, localización de documento y fuentes	MetadatoTexto (CursoGC) => tiene(CursoGC,Resumen) Λ tiene(CursoGC,Historico) Λ tiene(CursoGC,Localizacion) Λ tiene(CursoGC,Fuente)	∀ x MetadatoTexto(x) => tiene(x,DescriptorDocumento) Λ tiene(x,HistoricoDocumento) Λ tiene(x,LocalizacionDocumento) Λ tiene(x,Representación_Datos_Texto) (Axioma tabla 11)
Un objeto de aprendizaje del curso Gestión de Conocimiento puede contener metadatos imagen	MetadatoImagen(CursoGC) => es_un (CursoGC,ImagenSatelital) V es_un (CursoGC, ImagenDiseñoArquitectonico) V es_un (CursoGC, ImagenFacial)	∀ x MetadatoImagen(x) => es_un(x,ImagenSatelital) V es_un(x,ImagendeDiseñoArquitectonico) V es_un(x,ImagenFacial) (Axioma Tabla 11)
La materia GC tiene Metadatos audio	∀ x MetadatoAudio(CursoGC) => tiene (CursoGC,ReconocimientoAudio) Λ tiene (CursoGC, IdentificacionSignificado) Λ tiene (CursoGC, Identificaciondel Hablante)	∀ x MetadatoAudio(x) => tiene (x,Reconocimiento_del_Habla) Λ tiene (x,Identificacion_del_Significado) Λ tiene (x,Identificacion_del_Hablante) (Axioma Tabla 11)
El curso GC tiene Metadato Video	∀ x MetadatoVideo(CursoGC) => tiene (CursoGC,Contenido) Λ tiene (CursoGC, HerramientaSoporteUsuario) Λ tiene (CursoGC, HerramientaSoporteAplicacion)	∀ x MetadatoVideo(x) => tiene (x,Contenido_de_Video) Λ tiene (x,Herramientas_Sop_Usurio) Λ tiene (x,Herramientas_Sop_Aplicacion) (Axioma Tabla 11)
El objeto Video01 es un objeto multimedia	Objeto (Video01) => es_un (Video01, ObjetoMM)	∀ x Objeto (x) => es_un (x, ObjetoMM) (Axioma Tabla 7)
Video01 es un dato video como mpeg, mov, avi	ObjetoMM (Video01) => es_un (Video01, Dato_Video)	∀ x ObjetoMM(x) => es_un(x, Dato_Texto) V es_un(x, Dato_Audio) V es_un (x, Dato_Imagen) V es_un(x, Dato_Grafico) V es_un (x, Dato_Video) V es_un (x, Dato_MediaGenerado) (Axioma Tabla 7)

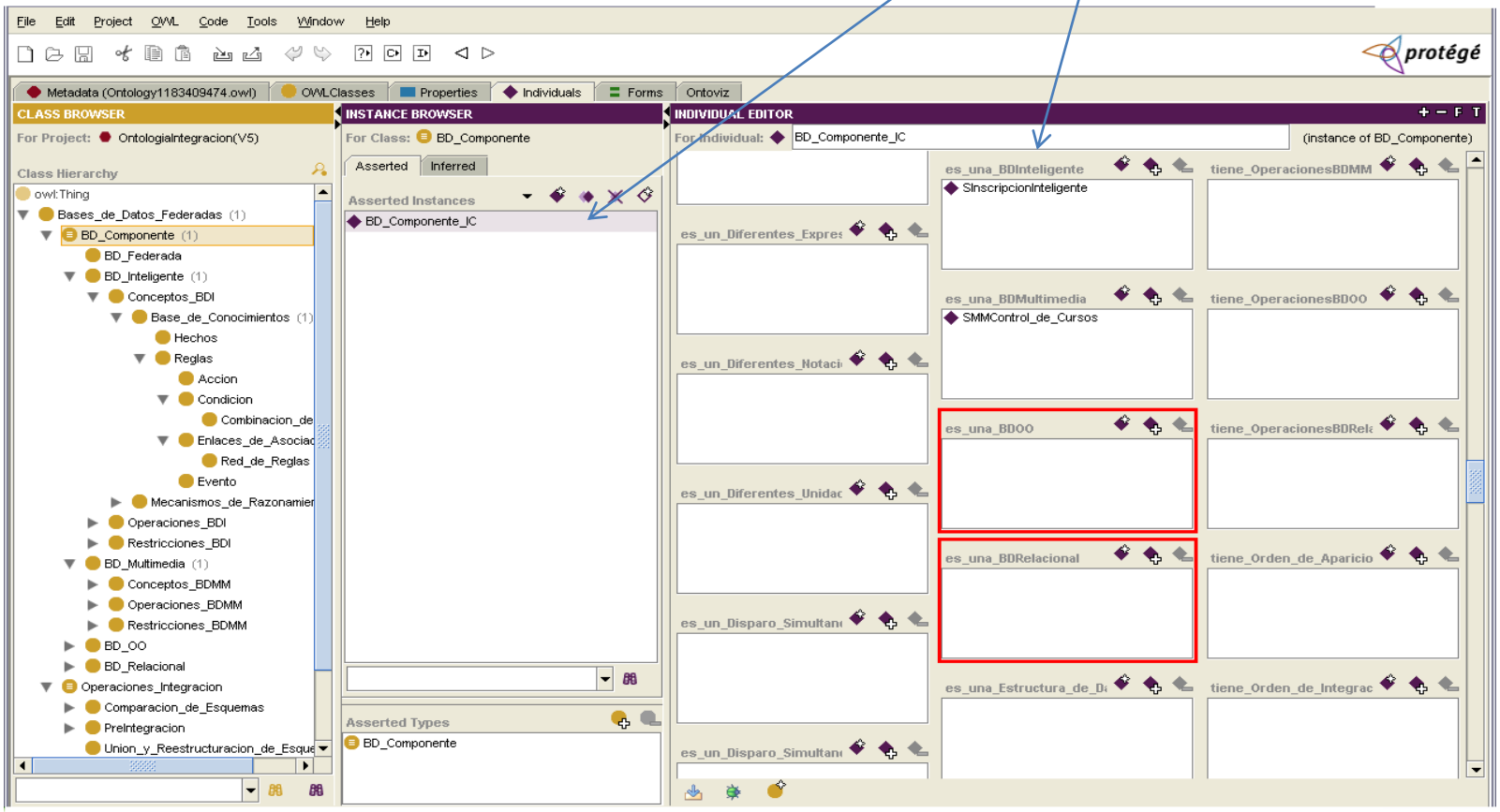
ONTOLOGIA INTEGRACION

Sentencia	LPO	Comentario
La base de datos federada Inscripción_Clase tiene bases de datos componentes, tiene operaciones de integración y tiene restricciones de integración	BDFederada(Inscripción_Clase) => tiene (Inscripción_Clase,BDComponentes) Λ tiene (Inscripción_Clase,OperacionesIntegracion) Λ tiene (Inscripción_Clase,RestriccionesIntegracion)	∀ x BDFederada(x) => tiene (x,BDComponente) Λ tiene (x,OperacionIntegracion) Λ tiene (x,RestriccionesIntegracion) (Axioma de Tabla 1)
Las bases de datos componentes de Inscripción_Clase son: "Sistema Multimedia de Control de Curso" que una BDMM, y el "Sistema Inteligente de Inscripción" que es una BDI	BDComponente(Inscripción_Clase) => es_un (Inscripción_Clase , SMMControldeCurso) V es_un (Inscripción_Clase , SInscripciónInteligente)	∀ x BDComponente(x) => es_un(x,BDRelacional) V es_un (x,BDOO) V es_un (x, BDMultimedia) V es_un (x, BDInteligente) V es_un (x, BDFederada) (Axioma Tabla 1)
El Sistema Multimedia de Control de Curso tiene conceptos, operaciones y restricciones de BDMM	BDMultimedia (SMMControldeCurso) => tiene(SMMControldeCurso,ConceptosMM) Λ tiene(SMMControldeCurso, OperacionesMM) Λ tiene(SMMControldeCurso, Restricciones MM)	∀ x BDMultimedia(x) => tiene(x,ConceptosMM) Λ tiene(x, OperacionesMM) Λ tiene(x, RestriccionesMM) (Axioma Tabla 1)
El Sistema de Inscripción Inteligente tiene conceptos, operaciones y restricciones de BDI	BDInteligentes (SInscripciónInteligente) => tiene(SInscripciónInteligente,ConceptosInt) Λ tiene(SInscripciónInteligente, OperacionesInt) Λ tiene(SInscripciónInteligente, RestriccionesInt)	∀ x BDInteligentes(x) => tiene(x,ConceptosInt) Λ tiene(x, OperacionesInt) Λ tiene(x, RestriccionesInt) (Axioma Tabla 1)

ONTOLOGIA BDI

CONCEPTO	LPO	Comentario
El Sistema Inteligente de Inscripción (SII) tiene una base de conocimiento y un mecanismo de razonamiento	ConceptoBDInteligente(SII) => tiene(SII,BasedeConocimiento) Λ tiene(SII,MecanismodeRazonamiento)	∀ x ConceptoBDInteligente(x) => tiene(x,BasedeConocimiento) Λ tiene(x,MecanismodeRazonamiento) (Axioma Tabla 15)
La base de conocimientos del SII *tiene reglas y tiene hechos*. La **base de Reglas** está conformada por Reglas que establecen el orden de inscripción de los estudiantes, Reglas que permiten inscribir a los alumnos de acuerdo a un estatus establecido, Reglas que permiten inscribir las materias de acuerdo a la carrera que esté cursando y las prelaciones de cada una de las materias a inscribir, Reglas de excepción al inscribirse un estudiante, Reglas que establecen las capacidades de cupo en las materias, entre otras. La **base de hechos** está conformada por los esquemas: alumnos, materias cursadas, materias a cursar, materias aprobadas, régimen de prelaciones, historial académico del alumno, materias a inscribir, etc.	BasedeConocimientos(SII) => [tiene(SII,ReglasdeOrdendeInscripcion) Λ tiene(SII,ReglasdeestatusAlumno) Λ tiene(SII,ReglasMateriaCarreraPrelacion) Λ tiene(SII,ReglasExcepcionInscripcion) Λ tiene(SII,ReglasCapacidadMaterias] Λ [tiene(SII, Datosdealumnos) Λ tiene(SII,MateriasCursadas) Λ tiene(SII, MateriasaCursar) Λ (tiene(SII, MateriasAprobadas) Λ tiene (SII , RegimendePrelaciones) Λ tiene(SII, HistorialAcadémicodelAlumno) Λ tiene (SII, MateriasaInscribir)]	∀ x BasedeConocimientos(x) => tiene(x,Reglas) Λ tiene(x,Hechos) (Axioma Tabla 15)
Las reglas tienen una condición, y una acción. La regla "OrdendeInscripcion": se activa con de Inscripción, y tiene verificar la selección de Estudiante por Promedio para realizar la ACCION Inscribir	Regla(OrdenInscripcion) => tiene(OrdenInscripcion, Solicitudde InscripcionANDSeleccionaEstudianteporPromedio) Λ tiene(OrdenInscripcion, EstablecefechasdeInscripción)	∀ x Regla(x) => tiene(x,Condición) Λ tiene(x,Acción) Λ tiene(x,EnlaceAsociación) (Axioma Tabla 15)
El mecanismo de razonamiento para el SII es deductivo. De la condición "SolicitudInscripcion" se activan, entre otras, las reglas para establecer el "orden de Inscripción por promedio del estudiante" y de las "Materias de ", para que con el hecho "materias a inscribir" determinar qué materias puede inscribir el estudiante	MecanismoRazonamiento(SII) => es_un(SII,Deductivo) Por ejemplo: ∀ x SolicitudInscripcion(x) => tiene (x, ReglasdeOrdenInscripcion) Λ tiene (x, ReglasMateriaCarreraPrelacion) Λ…]	∀ x MecanismoRazonamiento(x) => es_un(x,Deductivo) (Axioma Tabla 15)

ONTOLOGIA BDMM	ONTOLOGIA BDI	ONTOLOGIA INTEGRACION
ConceptoBDMultimedia (ObjetoAprendizajeGC) => tiene(ObjetoAprendizajeGC,MetadatoTexto) V tiene (ObjetoAprendizajeGC,MetadatoAudio) V tiene (ObjetoAprendizajeGC,MetadatoImagen) V tiene (ObjetoAprendizajeGC,MetadatoVideo)	ConceptoBDInteligente(SII) => tiene(SII,BasedeConocimiento) Λ tiene(SII,MecanismodeRazonamiento)	BDComponente(Inscripción_Clase) => es_un (Inscripción_Clase , SMMControldeCurso) V es_un (Inscripción_Clase , SInscripciónInteligente)



Tarea

Crear una ontología para el curso de IA usando protegé