OWL Web Ontology Language

Contenido

- OWL2: Introducción
- Conceptos Básicos de OWL
- Sintaxis – DL, OWL, Manchester
- Lenguaje OWL
  - Clases
  - Propiedades: : Object Properties, Data Properties
  - Individuos
- Clases: Definidas y Primitivas
- Razonamiento

Clases Primitivas y Definidas en OWL

*Una clase que solo tiene las condiciones necesarias se conoce como clase **primitiva**.*

- $A \subseteq C$ (condiciones necesarias para A)

Las condiciones necesarias se pueden leer como: "Si algo es miembro de esta clase, entonces es necesario cumplir estas condiciones"

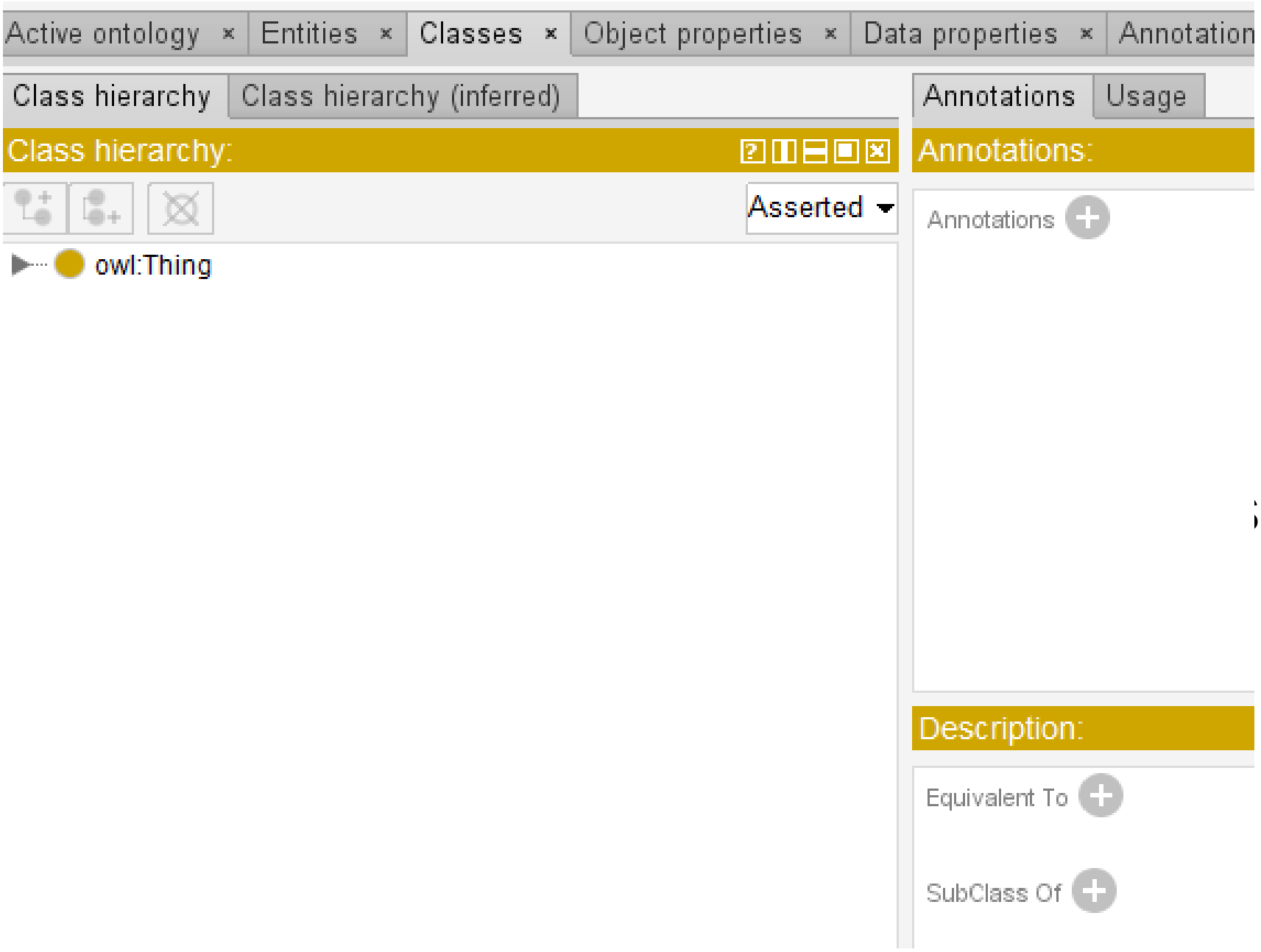
Clases Primitivas y Definidas en OWL

Una clase que tiene al menos un conjunto de condiciones necesarias y suficientes se conoce como clase ***definida****.*

- $A \equiv C$ (condiciones necesarias y suficientes para A – definición)

Las condiciones necesarias y suficientes se denominan clases equivalentes

Clases Primitivas y Definidas en OWL



Ejemplo: clases primitivas y definidas

Cuál es la diferencia entre usar:

- CosasCuenca ≡ localizadoEn.ZonaAustro

```
<owl:Class rdf:ID="CosasCuenca">
        <owl:equivalentClass>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#localizadoEn" />
                        <owl:someValuesFrom rdf:resource="#ZonaAustro" />
                </owl:Restriction>
        </owl:equivalentClass>
</owl:Class>
```

- CosasCuenca ⊆ localizadoEn.ZonaAustro

```
<owl:Class rdf:ID="CosasCuenca">
        <owl:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#localizadoEn" />
                        <owl:someValuesFrom rdf:resource="# ZonaAustro " />
                </owl:Restriction>
        </owl:subClassOf>
</owl:Class>
```

Ejemplo: clases primitivas y definidas

- Cuál es la diferencia entre usar:
  - `owl:subClassOf`
    - Las cosas que se encuentran localizadas en ZonaAustro no son necesariamente cosas de Cuenca
    - CosasCuenca ⊆ localizadoEn.ZonaAustro
      - Expresa condición necesaria
  - `owl:equivalentClass`
    - Si algo está localizado en ZonaAustro, entonces debe estar en la clase CosasCuenca
    - CosasCuenca ≡ localizadoEn.ZonaAustro
      - Expresa una condición necesaria y suficiente

Restricciones de Propiedad: Sintaxis

- Las restricciones de propiedad de OWL se utilizan para describir clases complejas mediante propiedades

- El axioma ***owl:Restriction*** permite describer este tipo de clases
- Tiene un elemento ***owl:onProperty*** y uno o más declaraciones de restricciones
- Ejemplo: Un padre debe tener al menos un hijo

```
:Padre rdfs:subClassOf
   [a owl:Restriction;
        owl:onProperty :tienehijo;
        owl:minCardinality "1"] .
```

Restricciones de Propiedad

- Tipos
  - Restriciones de Valor
    - owl:allValuesFrom: cuantificador universal
    - owl:someValuesFrom: cuantificador existencial
    - owl:hasValue
  - Restricciones de Cardinalidad
    - owl:cardinality
    - owl:minCardinality
    - owl:maxCardinality

Restricciones Propiedad: allValuesFrom

- Ejemplo: Expresar el conjunto de padres que solo tienen hijas

PadresConSoloHijas $\sqsubseteq$ Persona $\sqcap$ $\forall$tieneHijo.Mujer

PadresConSoloHijas subClassOf Persona and tieneHijo only Mujer

```
<owl:Class rdf:ID="PadresConSoloHijas">
 <rdfs:subClassOf rdf:resource="#Persona" />
   <rdfs:subClassOf>
     <owl:Restriction>
        <owl:onProperty rdf:resource="#tieneHijo"/>
        <owl:allValuesFrom rdf:resource="#Mujer"/>
     </owl:Restriction>
   </rdfs:subClassOf>
    ...
</owl:Class>
```

Restricciones de Propiedad: someValuesFrom

- Ejemplo: Expresar que una Madre es una Mujer que tiene un hijo (alguna Persona)

Madre $\sqsubseteq$ Mujer $\sqcap$ $\exists$tieneHijo.Persona

Madre subclassOf Mujer and tieneHijo some Persona

```
<owl:Class rdf:ID="Madre">
  <rdfs:subClassOf rdf:resource="#Mujer" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#tieneHijo"/>
      <owl:someValuesFrom rdf:resource="#Persona"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Restricciones de Propiedad: hasValue

- Permite definir clases basadas en la existencia de valores de propiedad particulares
- Ejemplo:

CursoFisica subClassOf esTomadoPor value "949352"

```
<owl:Class>
<rdfs:subClassOf>rdf:resource="#CursoFisica"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource= "#esTomadoPor"/>
<owl:hasValue rdf:resource= "#949352"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Restricciones de Propiedad: cardinalidad

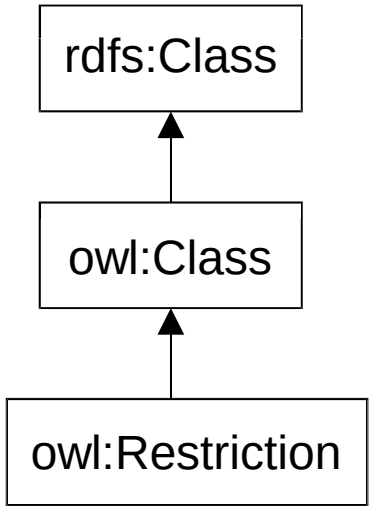
- Definición de **cardinalidad**:
  - el numero de ocurrencias, ya sea **maximo** (owl:maxCardinality) o **minimo** (owl:minCardinality) o **exacto** (owl:cardinality) basado en el contexto en la cual es usada

- Ejemplo. Cursos tomados al menos por dos personas

```
[a owl:Restriction;
   owl:onProperty :esTomadoPor;
   owl:minCardinality
   "2"^^xsd;nonNegativeInteger] .
```

Restriciones de Propiedad: Resumen

- Define una Clase usando restricciones LOCALES sobre una Propiedad especifica



- **Restricciones de Propiedad:**
  - allValuesFrom: rdfs:Class (lite/DL owl:Class)
  - hasValue: individuo especifico
  - someValuesFrom: rdfs:Class (lite/DL owl:Class)
  - cardinality: xsd:nonNegativeInteger (in lite {0,1})
  - minCardinality: xsd:nonNegativeInteger (in lite {0,1})
  - maxCardinality: xsd:nonNegativeInteger (in lite {0,1})

Inicio Tutorial – OWL
Creación de Clases Primitivas, Definidas y Restricciones de Propiedades

Ontología sobre Juegos de Video

Supongamos que queremos construir una ontología sobre videojuegos de la siguiente manera.

Clases	Clases	Propiedades	definiciones

- Juego
  - JuegoFamoso
  - LoL
  - Ajedrez
  - Sudoku
- Plataforma
  - Windows
  - MacOSX
  - Linux
	- TipoJuego
	  - UnSoloJugador
	  - MultiJugador
	  - DeRoles
	  - Enlinea
	- DificultadJuego
	  - Difícil
	  - Normal
	  - Fácil
		tieneDificultad
		tienePlataforma
		tieneTipo
			JuegoMultiPlataforma
			JuegoDificil
			JuegoNormal
			JuegoFacil
			JuegoParaLinux
			JuegoParaWindows
			JuegoParaMacOSX
			JuegoMultiJugador
			…

Agregando Clases Primitivas y Definidas

Cuáles clases?

- Solamente clases de las siguientes formas
  - $A \subseteq C$ (condiciones necesarias para A)
  - $A \equiv C$ (condiciones necesarias y suficientes para A – definición)
- Por cada subclase de JuegosFamosos se requiere insertar axiomas como:
  - Ajedrez puede ser instalado en cualquier plataforma
  - League of Legends es un juego en línea

Clases Primitivas – Ejemplo

- Especificación en lenguaje natural

  Ajedrez puede ser instalado en cualquier plataforma

- Replantear la especificación utilizando el vocabulario de ontología.

  Ajedrez tiene plataforma Windows, tiene plataforma MacOSX, y tiene plataforma Linux

- Escríbalo usando lógica descriptiva (opcional)

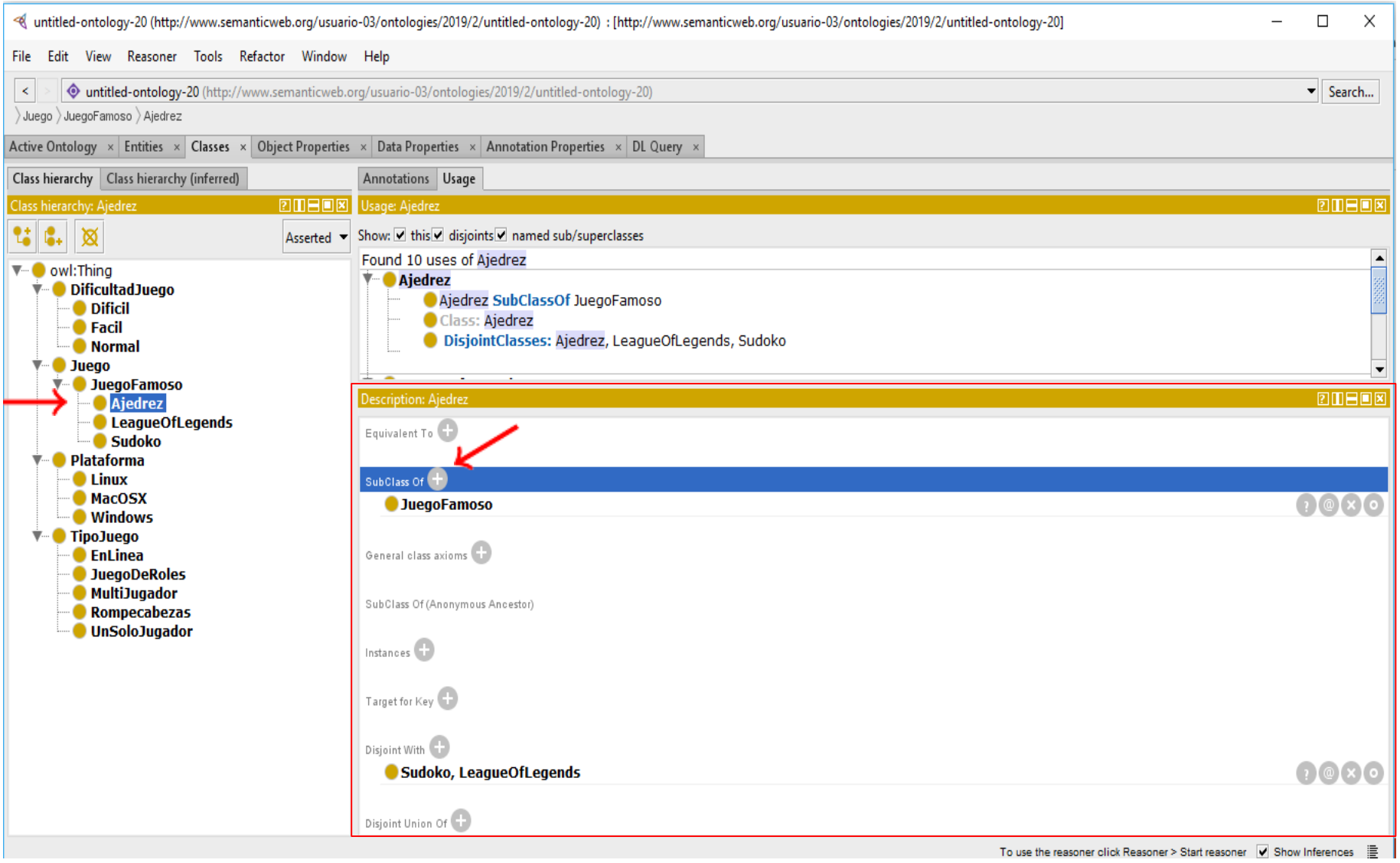
  $Ajedrez \subseteq \exists tienePlataforma.Windows$
  $Ajedrez \subseteq \exists tienePlataforma.MacOSX$
  $Ajedrez \subseteq \exists tienePlataforma.Linux$

- Escríbalo usando sintaxis Manchester (el lado derecho es suficiente)

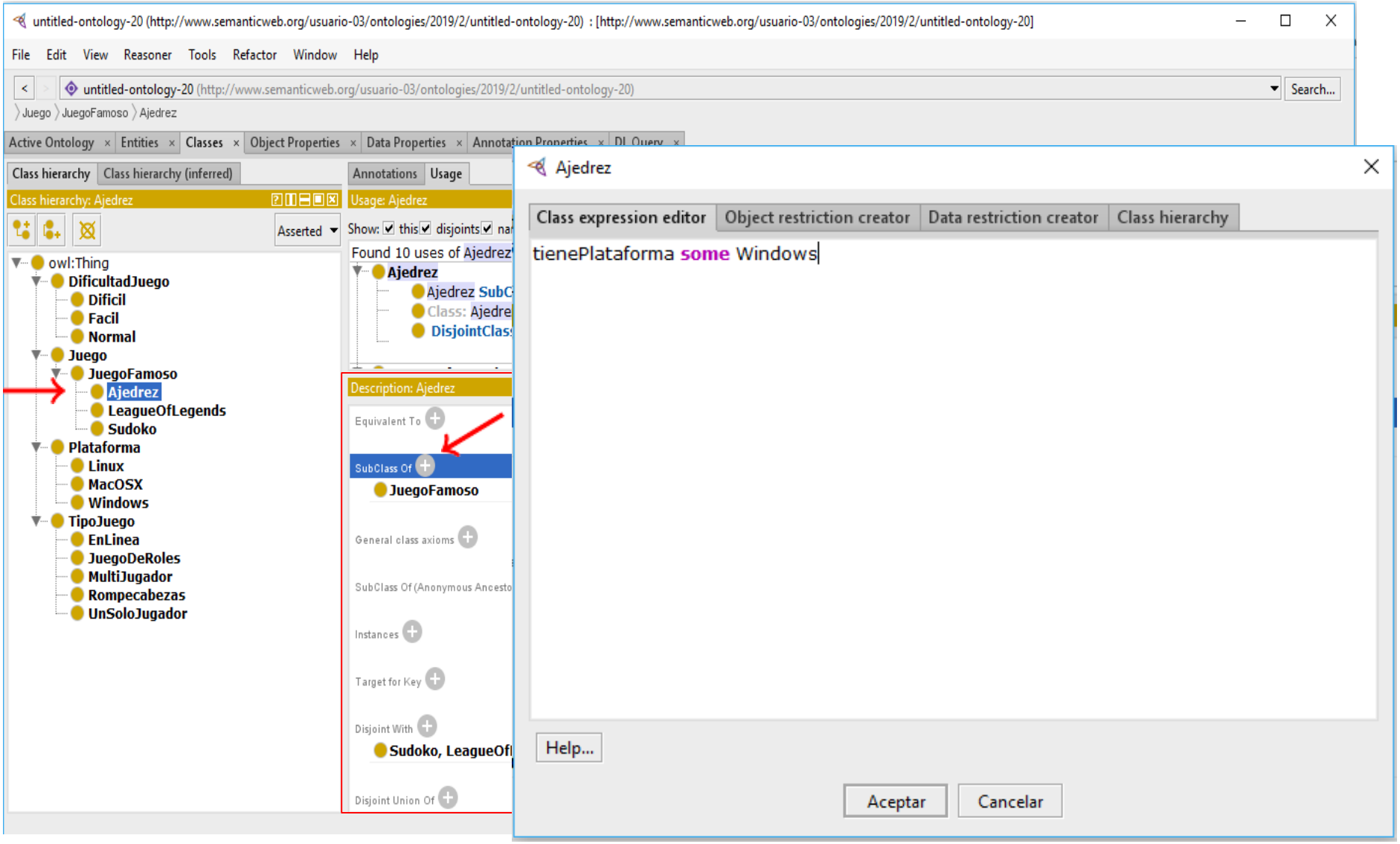
  tienePlataforma some Windows

  tienePlataforma some MacOSX
  tienePlataforma some Linux

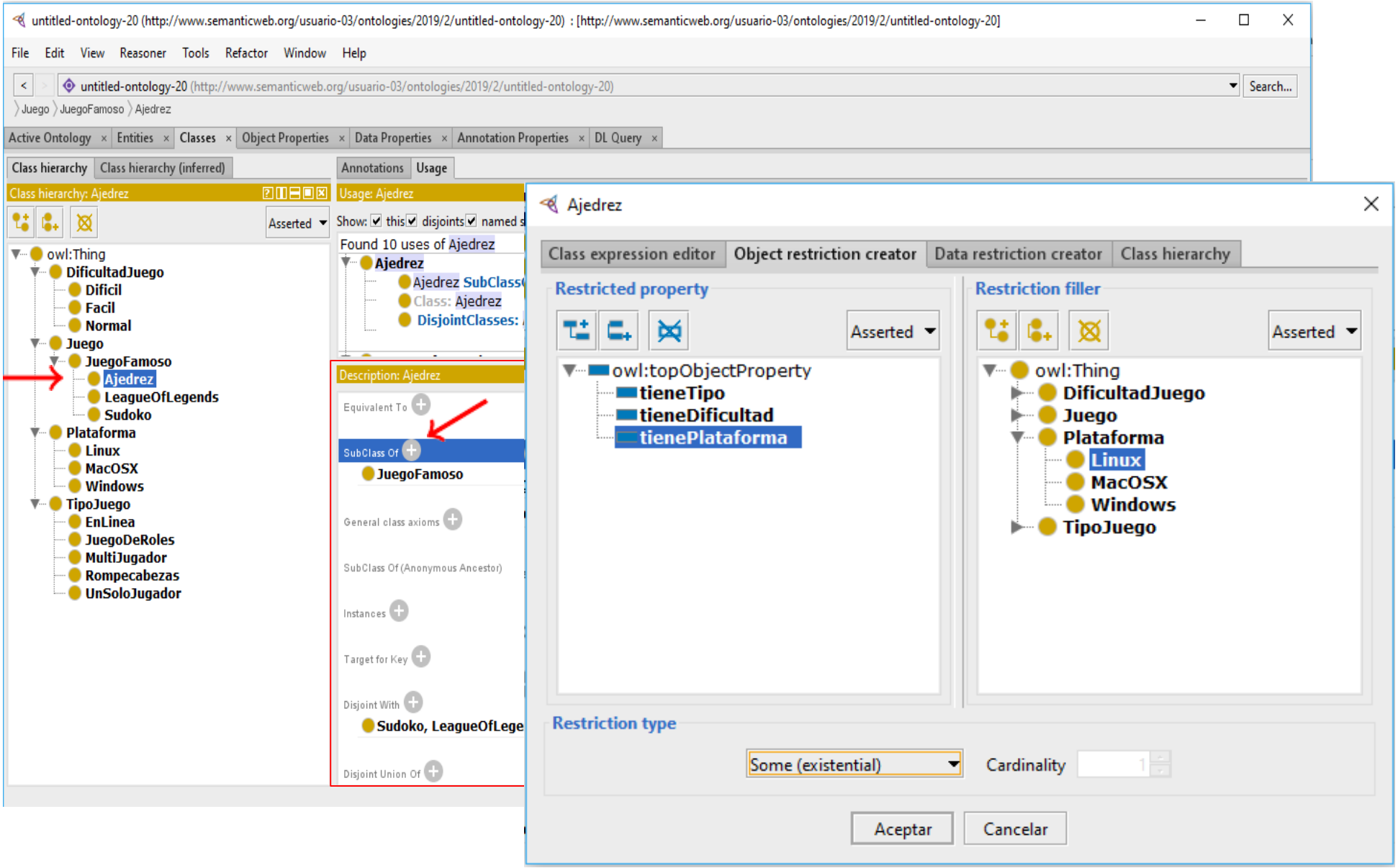
Agregando axiomas a la Clase “Ajedrez”



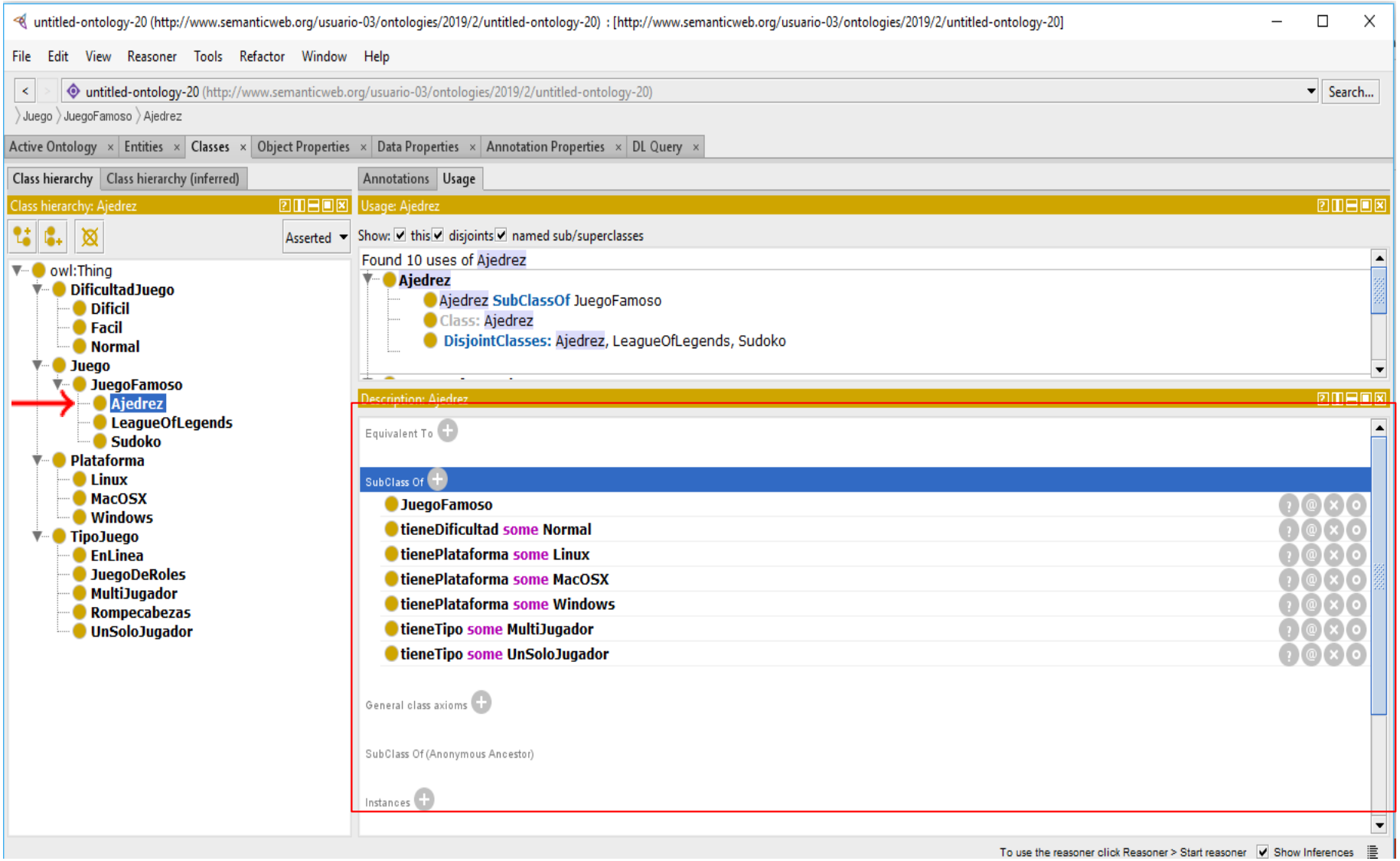
Agregando axiomas a la Clase “Ajedrez”



Agregando Axiomas a la Clase “Ajedrez”



Agregando Axiomas a la Clase “Ajedrez”



Mejorando la definición de la Clase “DificultadJuego”

Qué se requiere hacer?

- agregar *DificultadJuego* $\equiv$ *Dificil* $\sqcup$ *Normal* $\sqcup$ *Facil*

  Note que las clases *Dificil*, *Normal* y *Facil* son ya disjuntas

- agregar dominio y rango de la propiedad *tieneDificultad*

Mejorando la definición de la Clase “Ajedrez”

Qué se requiere hacer?

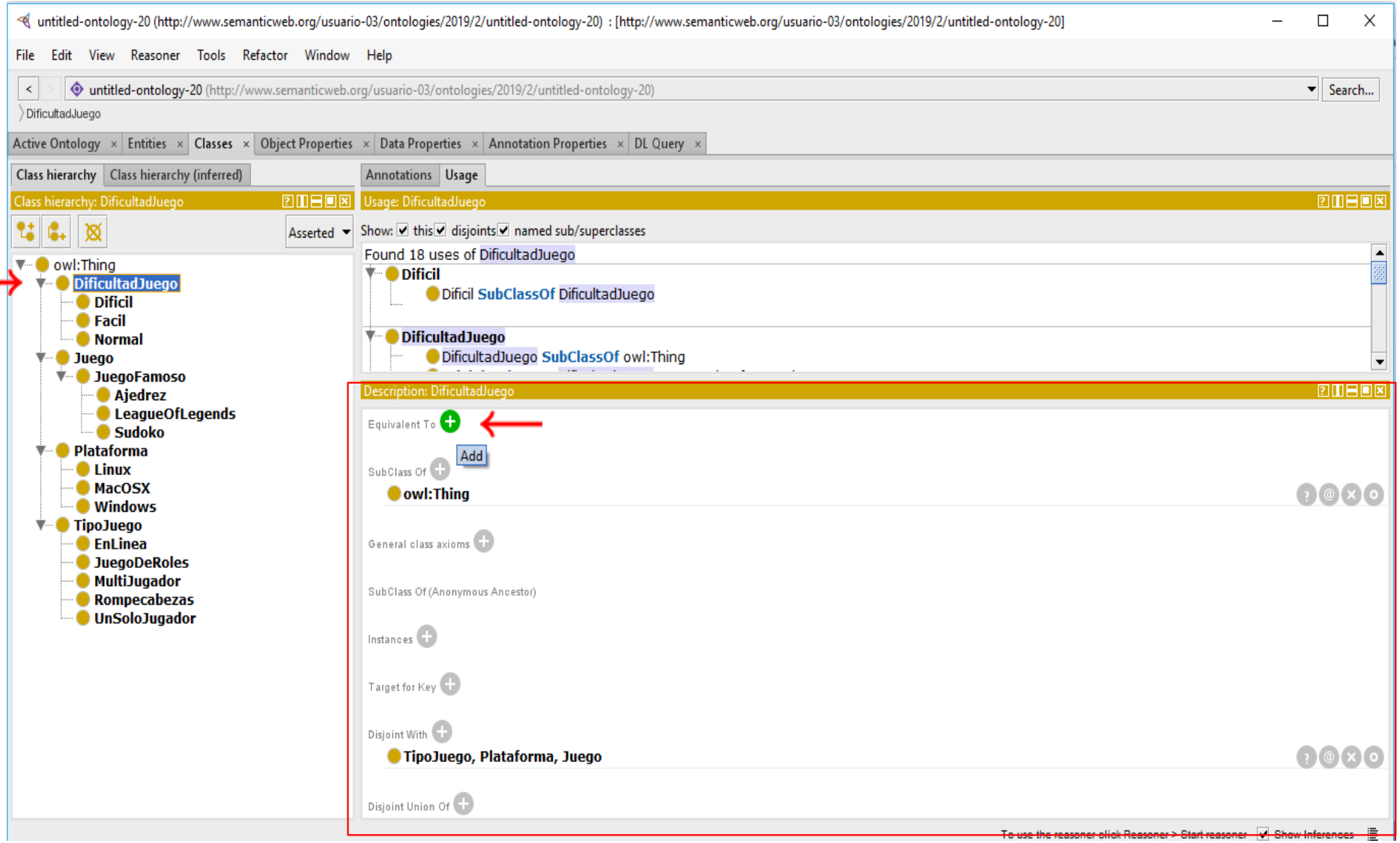
- **Ajedrez tiene dificultad normal de juego**
  - Replantear la especificación utilizando el vocabulario de ontología.

    Ajedrez tiene dificultad Normal
  - Escríbalo usando lógica descriptiva (opcional)

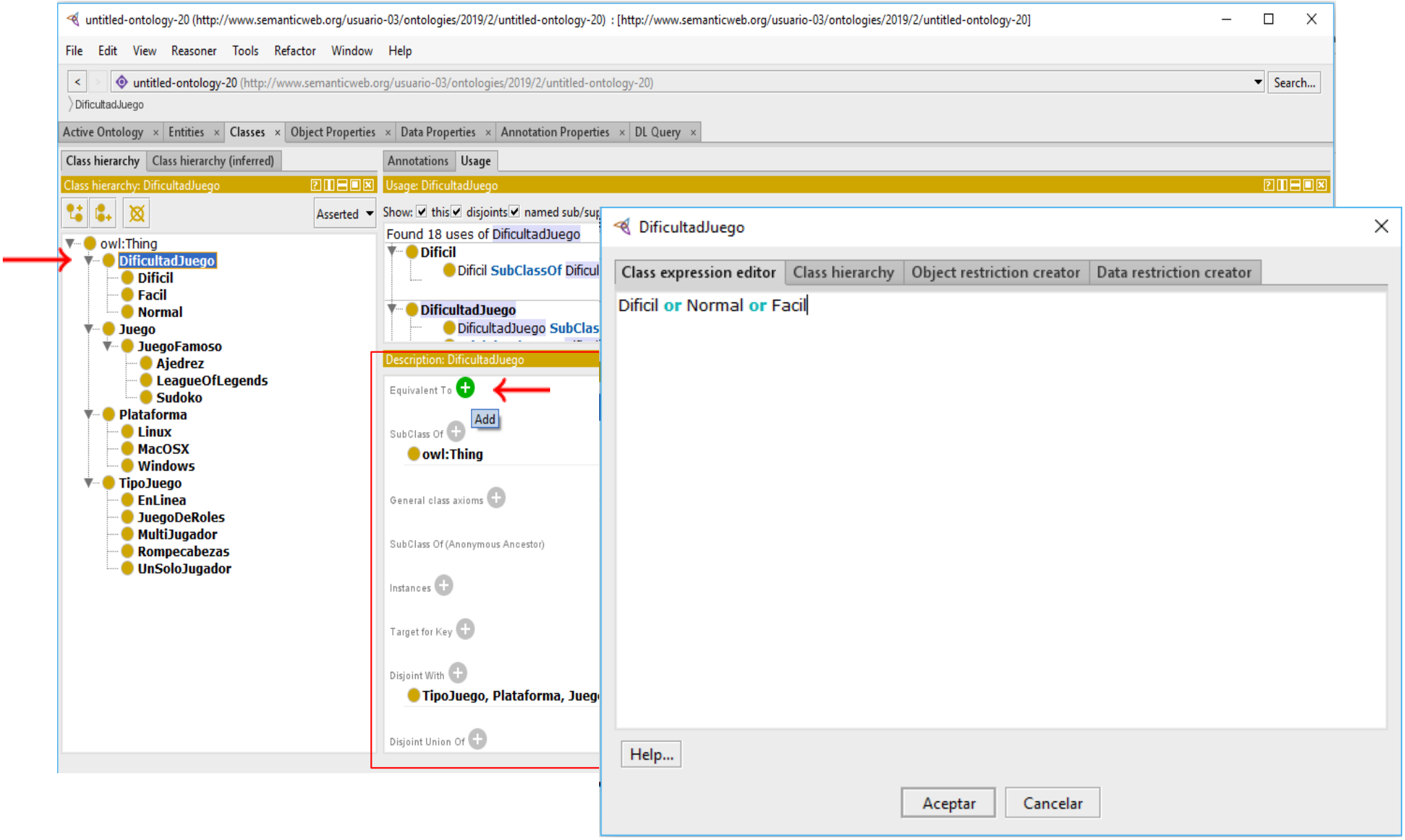
    Ajedrez $\subseteq \exists$*tieneDificultad.Normal*
  - Escríbalo usando sintaxis Manchester (el lado derecho es suficiente)

    tieneDificultad some Normal
- **Ajedrez es juego multijugador**
- **Ajedrez es un juego de un solo jugador**

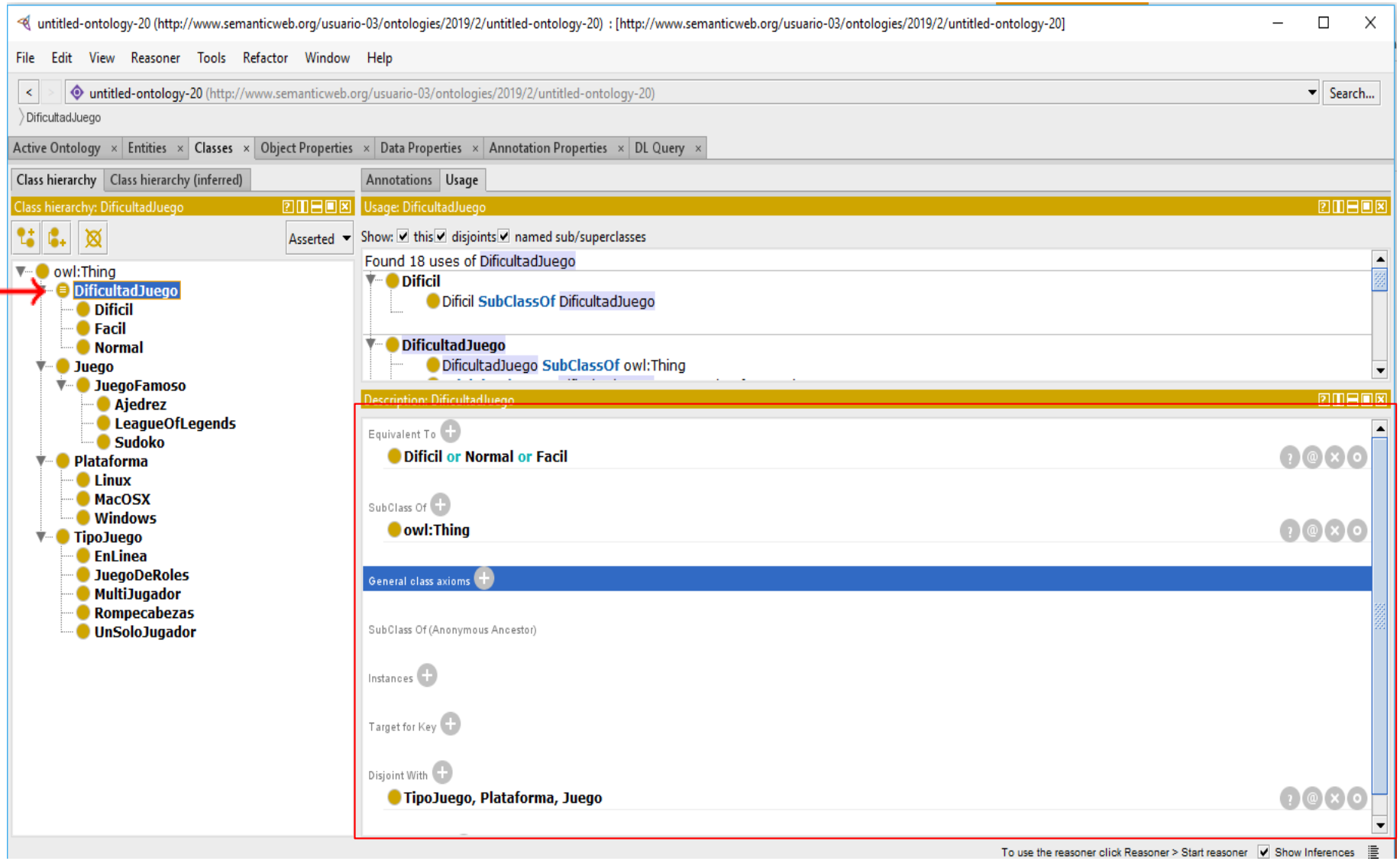
Mejorando la definición de la Clase “DificultadJuego”



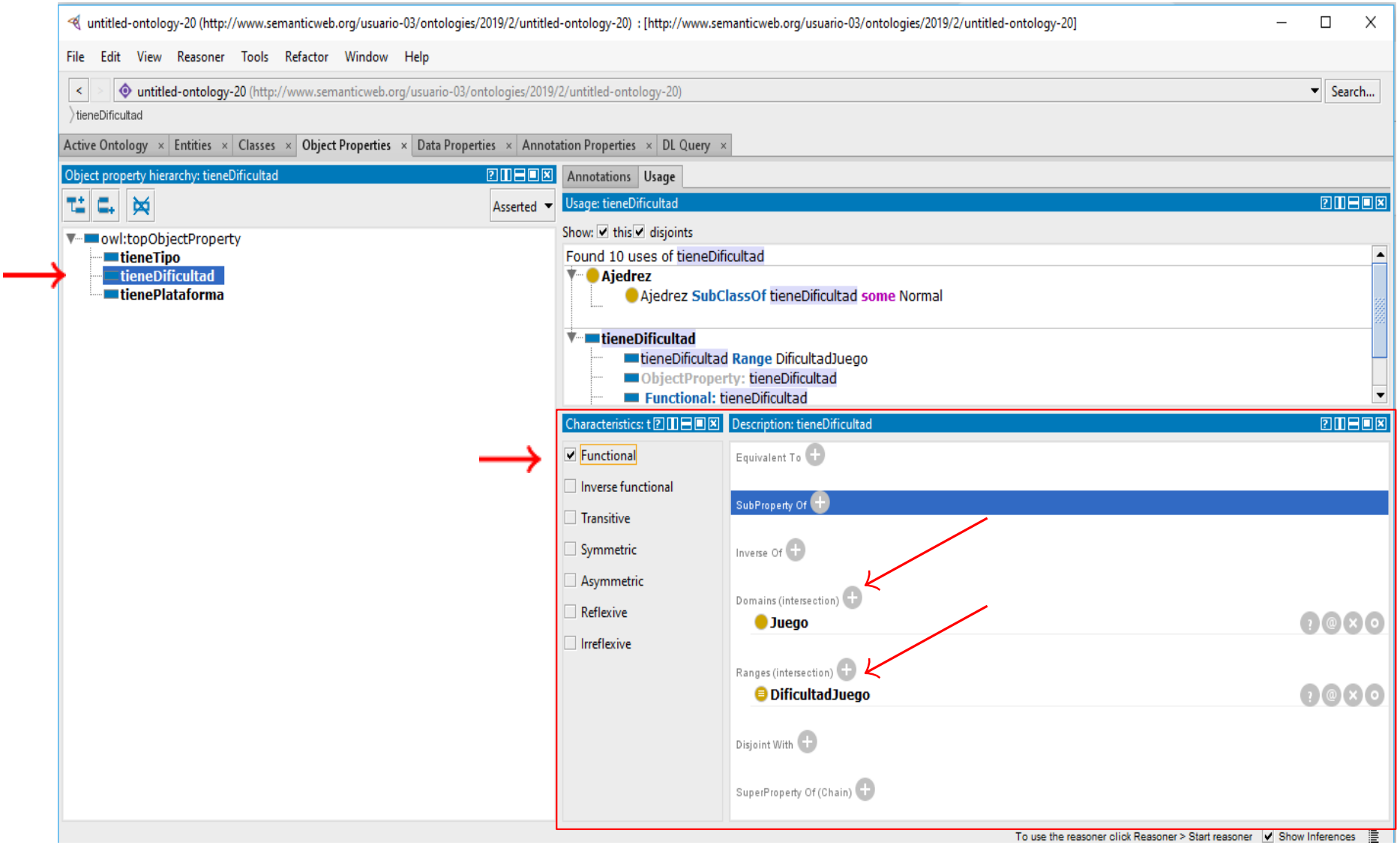
Mejorando la definición de la Clase “DificultadJuego”



Mejorando la definición de la Clase “DificultadJuego”



Mejorando la definición de la Clase “DificultadJuego”



Ontología sobre Juegos de Video

Supongamos que queremos construir una ontología sobre videojuegos de la siguiente manera.

Clases	Clases	Propiedades	definiciones

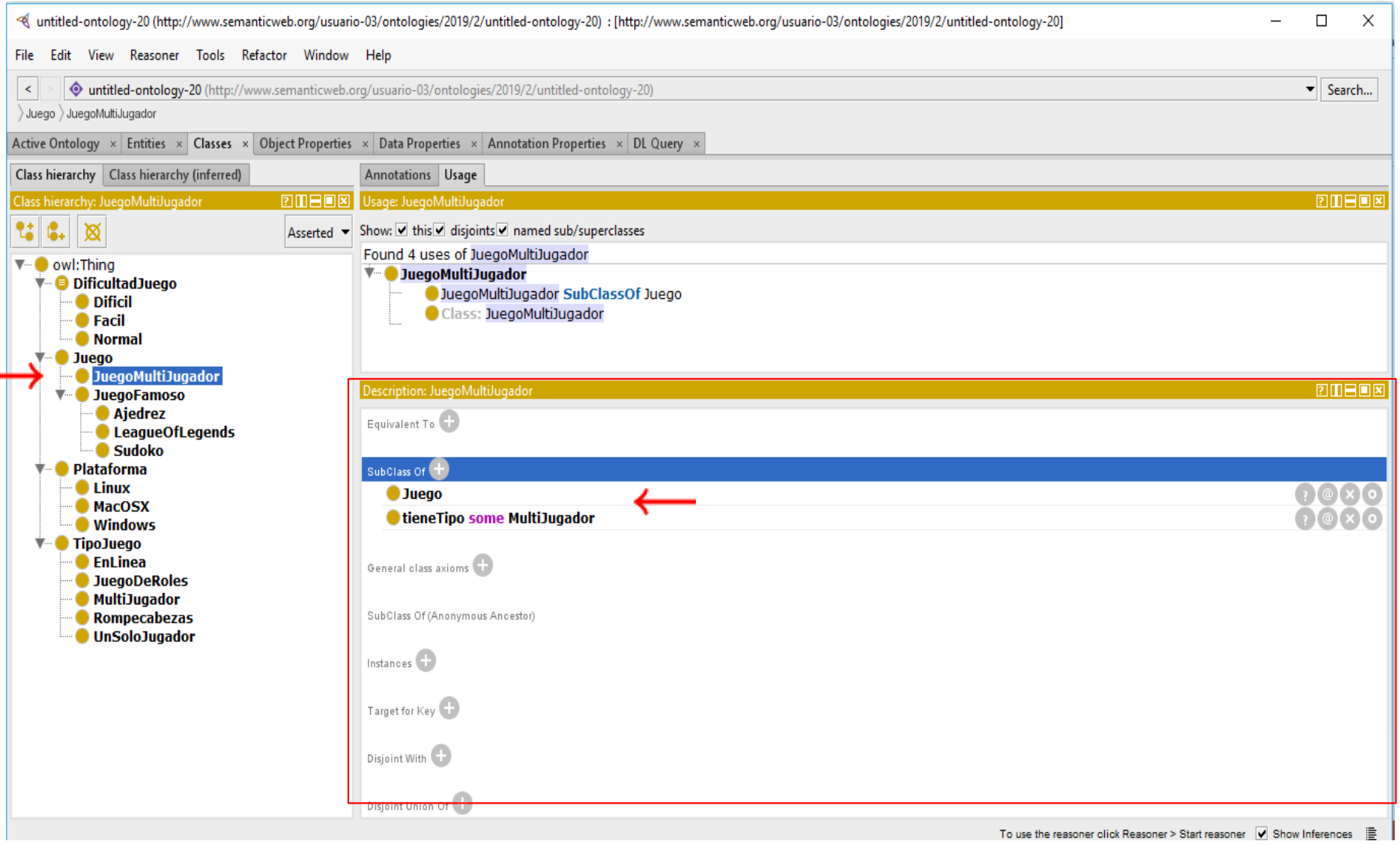
- Juego
  - JuegoFamoso
  - LoL
  - Ajedrez
  - Sudoku
- Plataforma
  - Windows
  - MacOSX
  - Linux
	- TipoJuego
	  - UnSoloJugador
	  - MultiJugador
	  - DeRoles
	  - Enlinea
	- DificultadJuego
	  - Difícil
	  - Normal
	  - Fácil
		tieneDificultad
		tienePlataforma
		tieneTipo
			JuegoMultiPlataforma
			JuegoDificil
			JuegoNormal
			JuegoFacil
			JuegoParaLinux
			JuegoParaWindows
			JuegoParaMacOSX
			JuegoMultiJugador
			…

Agregando la definición de la Clase “JuegoMultiJugador”

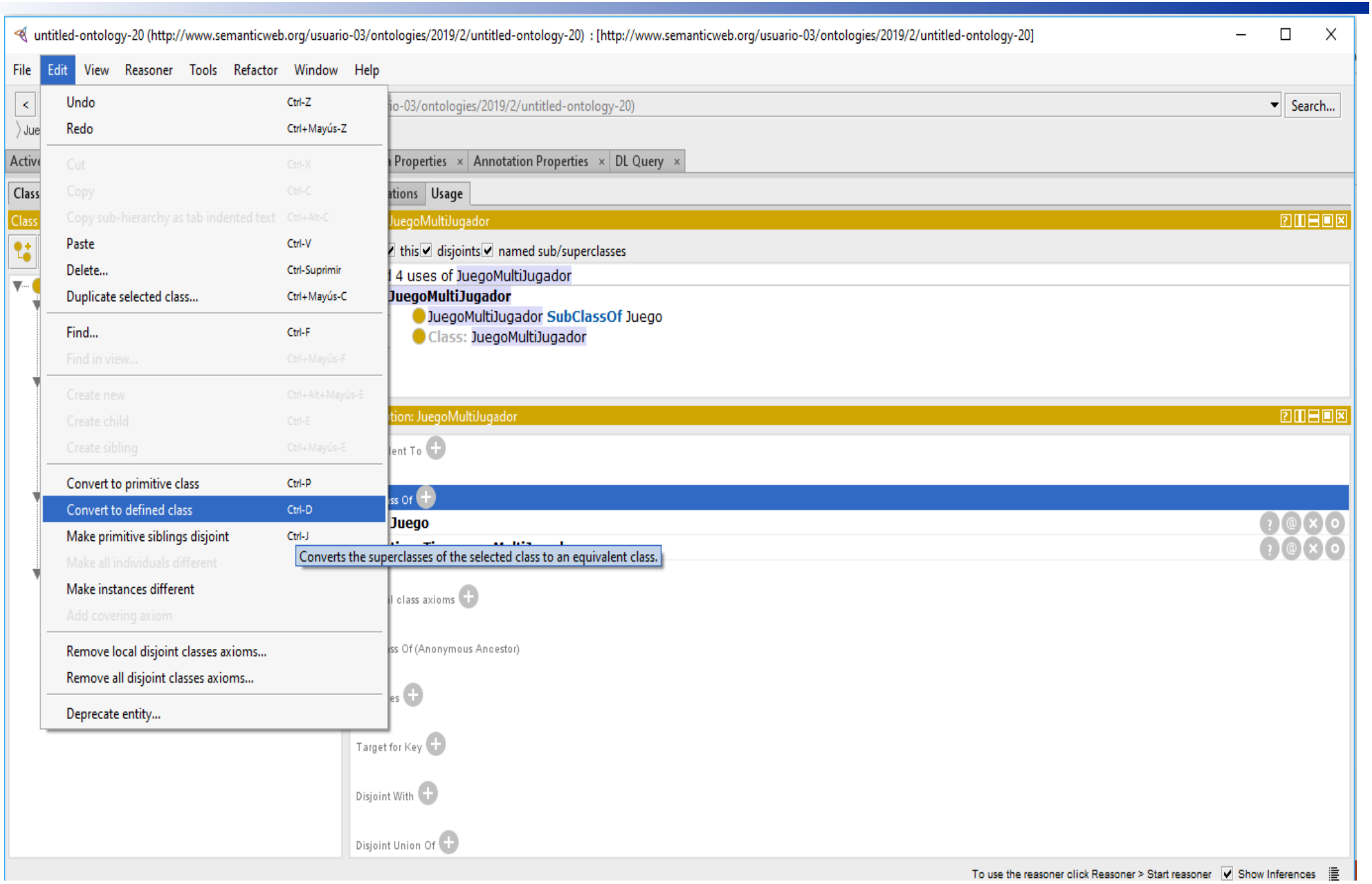
Qué se requiere hacer?

- agregar la clase *JuegoMultijugador*
- *la cual es una subclase de Juego y que tiene como tipo de juego múltiples jugadores*
- $JuegoMultiJugador \equiv Juego \sqcap \exists tieneTipo.MultiJugador$

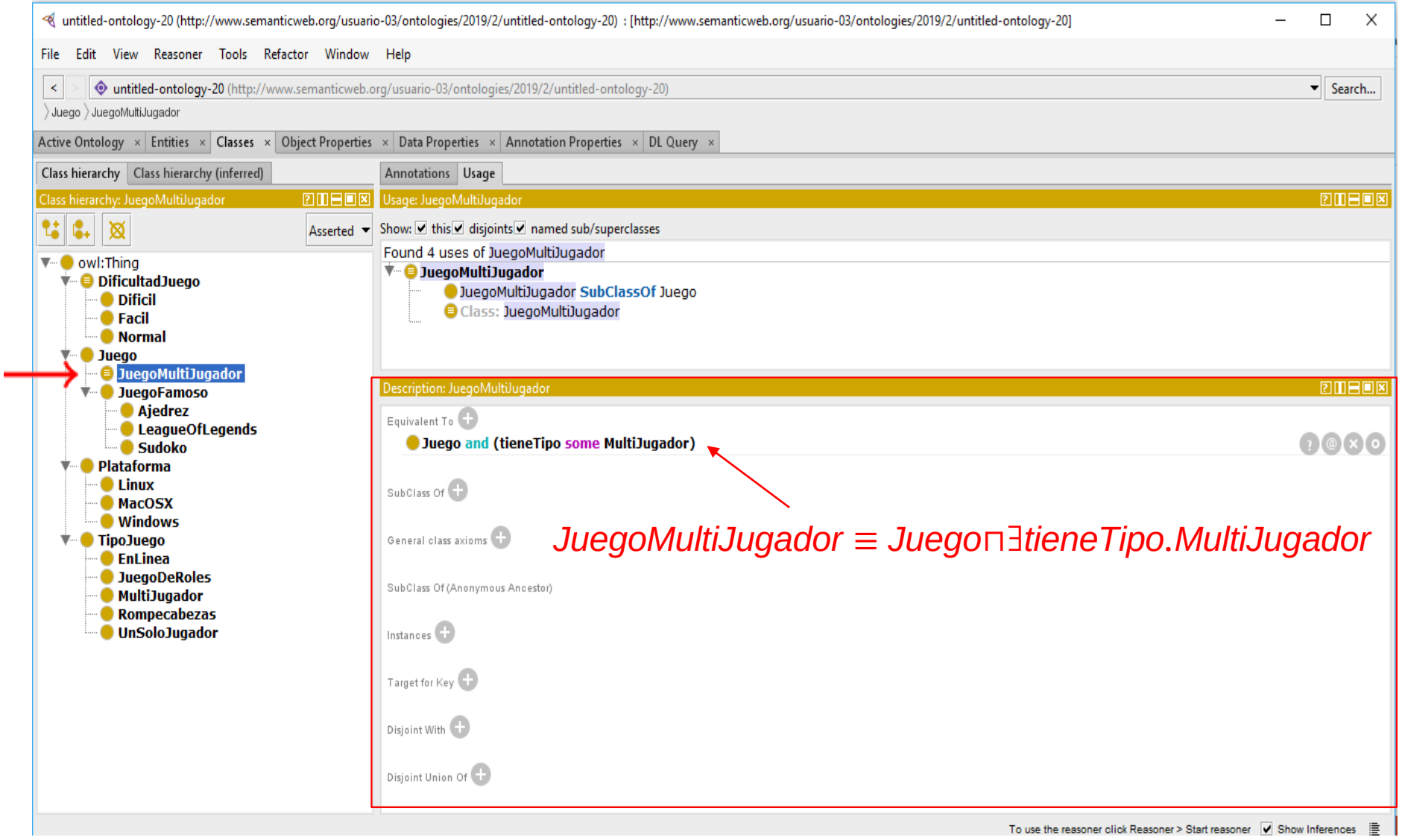
Agregando la Clase definible “JuegoMultiJugador”



Agregando la Clase definible “JuegoMultiJugador”



Agregando la Clase definible “JuegoMultiJugador”



Agregar más definiciones

- Agregue las definiciones de las siguientes clases:

- JuegoParaLinux
- JuegoParaMacOSx
- JuegoParaWindows
- JuegoMultiPlataforma
- JuegoDeRoles
- JuegoUnSoloJugador
- JuegoEnLinea
- JuegoDificil
- JuegoFacil
- JuegoNormal

Fin Tutorial – OWL
Creación de Clases Primitivas, Definidas y Restricciones de Propiedades

Inicio Tutorial – OWL Inferencia

Ontología sobre Juegos de Video

Supongamos que queremos construir una ontología sobre videojuegos de la siguiente manera.

Clases	Clases	Propiedades	definiciones

- Juego
  - JuegoFamoso
  - LoL
  - Ajedrez
  - Sudoku
- Plataforma
  - Windows
  - MacOSX
  - Linux
	- TipoJuego
	  - UnSoloJugador
	  - MultiJugador
	  - DeRoles
	  - Enlinea
	- DificultadJuego
	  - Difícil
	  - Normal
	  - Fácil
		tieneDificultad
		tienePlataforma
		tieneTipo
			JuegoMultiPlataforma
			JuegoDificil
			JuegoNormal
			JuegoFacil
			JuegoParaLinux
			JuegoParaWindows
			JuegoParaMacOSX
			. . .

Razonador

Protégé puede ser usado para tareas de razonamiento como la clasificación

- configure el razonador

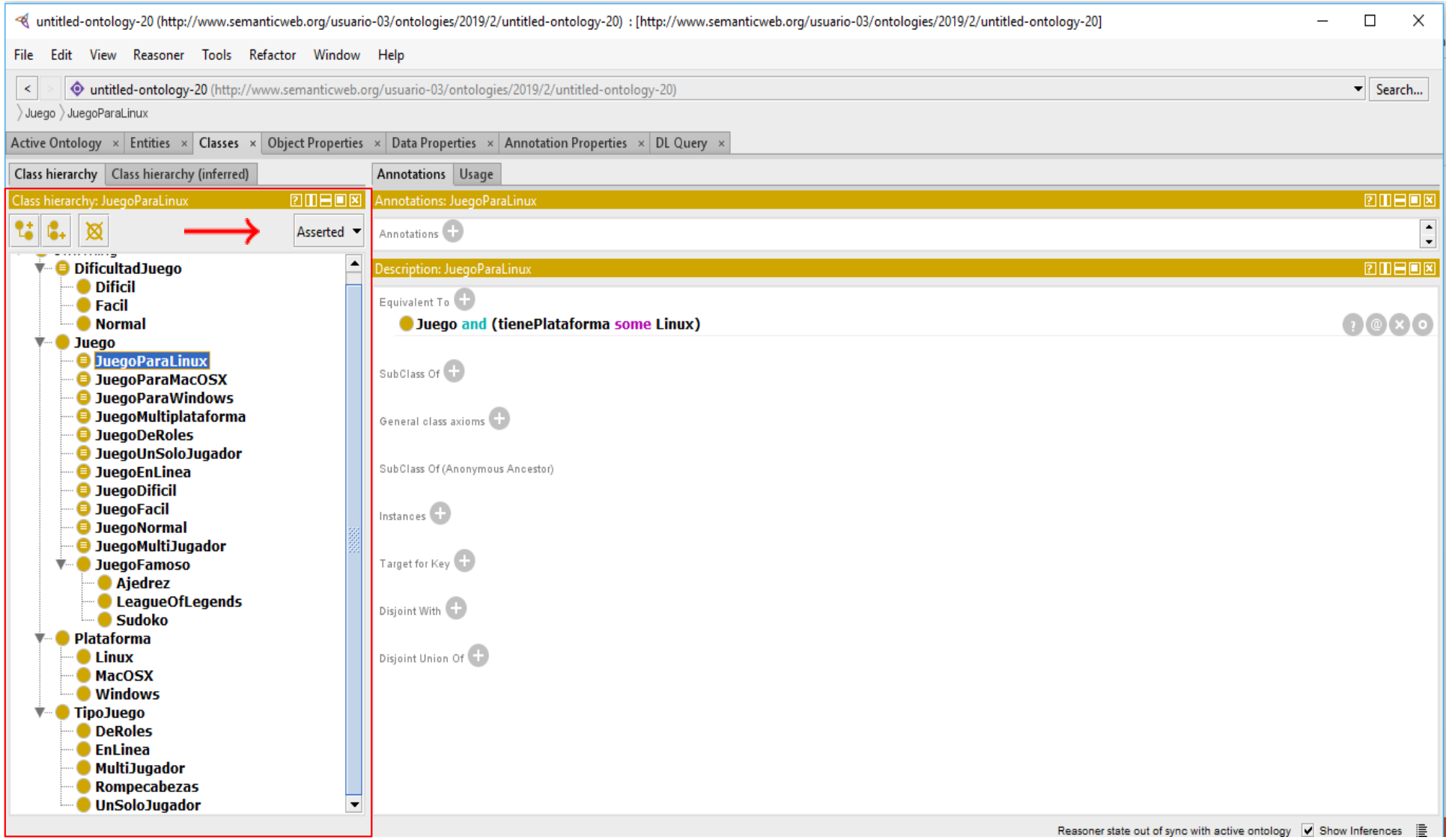
  Reasoner → Configure. . . (para este tutorial, seleccione todo bajo *Class inferences* y *Object property inferences*)

- seleccione un razonador

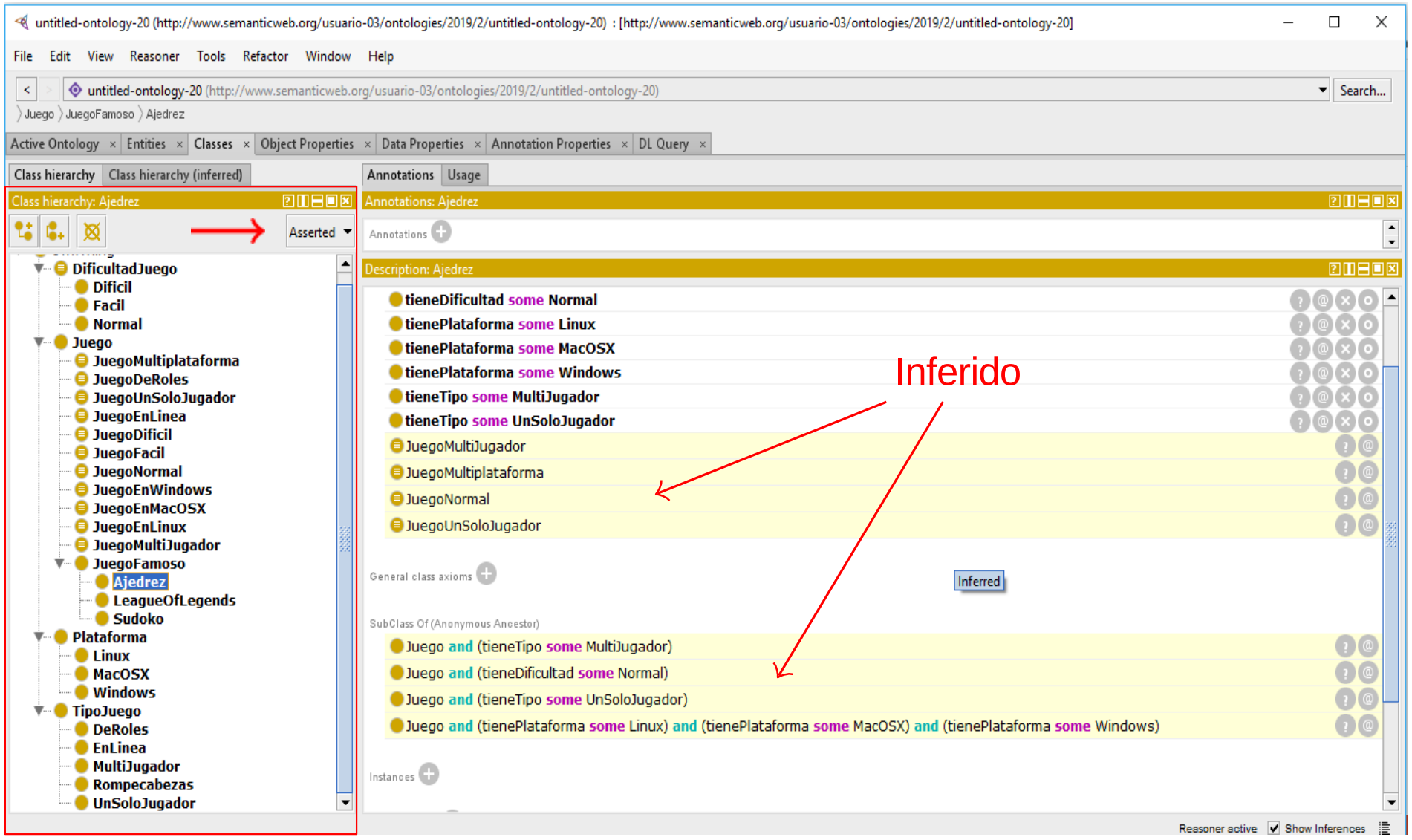
  por ejemplo, Reasoner → HermiT (otros razonadores pueden ser agregados, cuál utilizar depende de varios factores, como la expresividad de la ontología)

- finalmente, Reasoner → Start reasoner

Ejemplo Razonamiento



Ejemplo Razonamiento



Explicar las clases inferidas

Revise y verifique que las clases hayan sido clasificadas correctamente

Qué fue inferido para la Clase Ajedrez?

Fin Tutorial – OWL Inferencia

OWL 2 RESUMEN

Class Expressions

- Class names A, B
- Conjunction $C \sqcap D$
- Disjunction $C \sqcup D$
- Negation $\neg C$
- Exist. property restriction $\exists R.C$
- Univ. property restriction $\forall R.C$
- Self $\exists S.Self$
- Greater-than $\geq n\ S.C$
- Less-than $\leq n\ S.C$
- Enumerated classes $\{a\}$

Properties

- Property names R, S, T
- Simple properties S, T
- Inverse properties R^-
- Universal property U

Tbox (Class axioms)

- Inclusion $C \sqsubseteq D$
- Equivalence $C \equiv D$

Rbox (Property Axioms)

- Inclusion $R_1 \sqsubseteq R_2$
- General Inclusion $R^{(-)}_1 \circ R^{(-)}_2 \circ \ldots \circ R^{(-)}_n \sqsubseteq R$
- Transitivity
- Symmetry
- Reflexivity
- Irreflexivity
- Disjunctiveness

Abox (Facts)

- Class membership $C(a)$
- Property relation $R(a, b)$
- Negated property relation $\neg S(a, b)$
- Equality $a=b$
- Inequality $a \neq b$

Práctica OWL
Creación ontología en Protege