Introducción a la Inteligencia Artificial
Trabajo Práctico 2: Ontologías

Agustín Fernández Bergé y Ramiro Gatto

14/04/2025

1. Introducción

La idea de este trabajo es modelar una ontología (en Protege) sobre lenguajes de programación, en la cual se represente las características que estos poseen. El principal uso de la misma es permitir ayudar a un programador a elegir un lenguaje apropiado para un proyecto según sus necesidades.

2. Pasos a Seguir

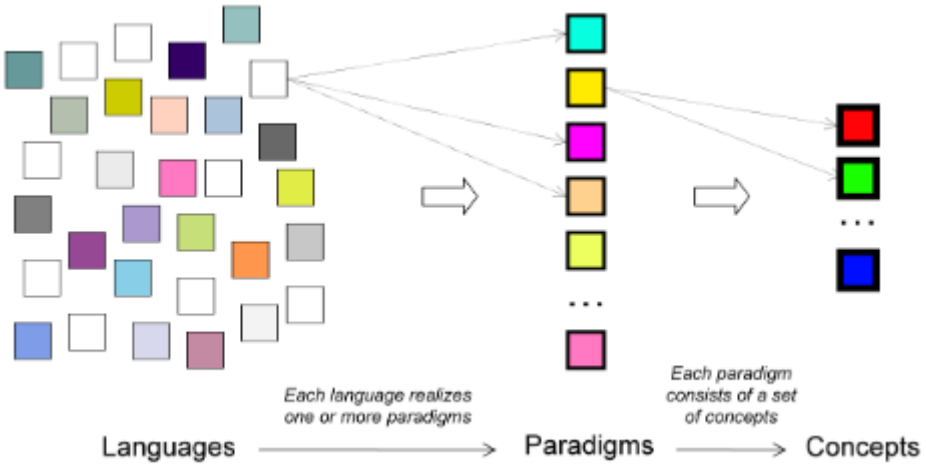
Para guiarnos en la creación de la ontología seguimos el "Pipeline" que vimos en clase, el cual consiste en lo siguientes pasos:

1. **Determinar dominio y alcance**
   En esta primera instancia nos planteamos preguntas que deben ser respondidas por la ontología. Estas deben estar relacionadas con: el dominio que cubre, el propósito de la misma y para que consultas da solución. Nosotros propusimos las siguientes:

   - "¿Qué lenguajes funcionales tienen tipado estático y fuerte?"
   - "¿Qué lenguajes tienen recolector de basura?"
   - "¿Qué lenguajes son multiparadigmas?"
   - "¿Qué lenguajes son aptos para programar de forma concurrente?"

2. **Analizar Reuso**
   Para poder facilitar la creación de más clases y/o tener mas idea de que características usar en la ontología, una forma seria viendo otras ontología ya creadas.

   En nuestro caso intentamos seguir el enfoque de Peter Van Roy[1] que asocia lenguajes de programación con uno o más paradigmas de programación y a su vez asocia paradigmas de programación con una o más conceptos de programación.



Sin embargo definir un paradigma puramente por los conceptos es complicado y puede terminar en muchos paradigmas diferentes que difieren solo en un concepto. Separar demasiado los paradigmas siguiendo este enfoque puede complicar mucho el diseño de la ontología y puede hacerla demasiado general como para los objetivos que nos propusimos.

Aun asi no abandonamos este enfoque del todo, simplemente decidimos separar los conceptos (nosotros lo llamamos características) de los paradigmas de programación.

3. **Enumerar términos**
   En base al lo pensado en los apartados anteriores se comenzó a enumerar los elementos que van a aparecer en la ontología. Pensando:

   - ¿De qué términos necesitamos hablar?
   - ¿Propiedades que poseen esos términos?
   - ¿Qué queremos decir acerca de esos términos?

   Para esta ontología surgieron los siguientes (entre muchos mas):

   - Lenguajes: C, C++, Python, Haskell, ...
   - Características: paradigma, gestor de memoria, forma de ejecución, ...

4. **Definición de clases y jerarquía**
   Para poder definir las clases nos guiamos por la idea de que una clase es una colección de elementos con propiedades similares (Ejemplos de clases en esta ontología serian los Lenguajes y Programas).

   Ademas, también se tuvo en consideración la idea de subclases y superclases (Como puede ser en el caso de las Características)

5. **Definición propiedades de las clases (slots)**
   En esta sección es donde pensamos las propiedades (slots) que van a tener las instancia de una clase y como se van a relacionar con las instancias de otra clase. Un ejemplo seria:

   a) Cada Programa fue escrito en algún Lenguaje
   b) Cada Lenguaje tiene algún paradigma
   c) Cada Lenguaje tiene una forma de gestión de memoria
   d) Cada Lenguaje tiene una forma de ejecución
   e) Cada lenguaje tiene varias características de su sistema de tipo
      - El sistema de tipo puede ser fuerte o débil
      - El sistema de tipo puede tener tipado estático o dinámico
      - La Declaración de tipos puede ser implícita o explícita

6. **Restricción de Propiedades**
   En esta sección, se establecieron los posibles valores de los slots teniendo se:

   - Un lenguaje de programación es una instancia de Lenguaje

- Un programa puede ser escrito por multiples lenguajes
- Los Lenguaje tienen una sola forma de gestión de memoria
- Los Lenguajes tienen una sola forma de ejecución
- Los Lenguajes pueden tener multiples paradigmas
- El sistema de tipos puede ser fuerte o débil, pero no ambos a la vez
- El sistema de tipos puede ser estático o dinámico, pero no ambos a la vez
- La declaración de tipos puede ser implícita, explícita o ambas al mismo tiempo

7. **Crear instancia**
   Para esta sección simplemente creamos las instancia en las distintas clases y le asignamos los valores según las slots.
   Un ejemplo de esto seria, en Lenguaje agregamos la instancia **C**, en GestionMemoria **Manual**. Luego podemos relacionar **C** y **Manual** mediante la propiedad **tieneGestionMemoria**

3. Conceptos representados

En la ontología final representamos los siguientes conceptos

- Característica, cualidades que cumplen los lenguajes de programación, nosotros optamos pos usar:
  - Gestión de memoria, representa las forma en las que los lenguaje manejan el uso de la memoria
  - Forma de ejecución,
  - Paradigma, con paradigma nos referimos al enfoque para crear programs
  - Sistema de tipo, esta clasificación se divide en 3 mas Chequeo de tipo (verificar que los tipos de datos en un programa sean correctos), Declaración de tipo (especificar explícitamente el tipo de una variable), Seguridad de tipo (garantizar que las operaciones solo se realicen entre datos de tipos compatibles)
- Lenguaje, hace referencia a todos los lenguajes de programación que decidimos agregar
- LenguajeInteresnte, este es una clase definida (no primitiva) en la la cual fue creada para probar el razonador, en esta solo se encentran las instancias de lenguajes que cumplen una serie de restricciones (posee al menos 3 paradigmas, posee recolector de basura y tiene chequeo y seguridad de tipos)

Quedándonos la siguiente ontología:

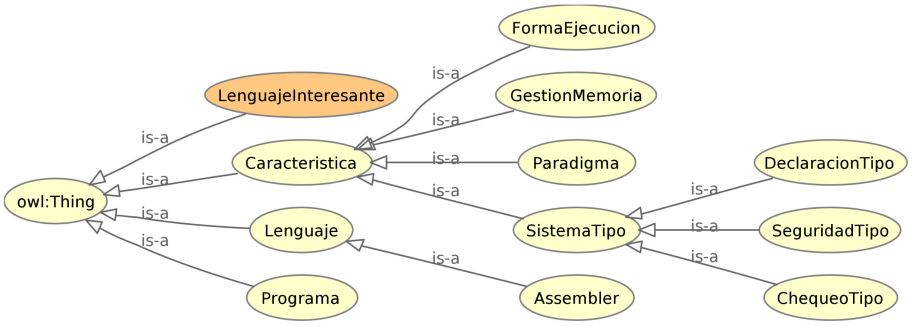


Figura 1: Ontología

4. Instancias propuesta

Las instancias nos quedaron de la siguiente forma:

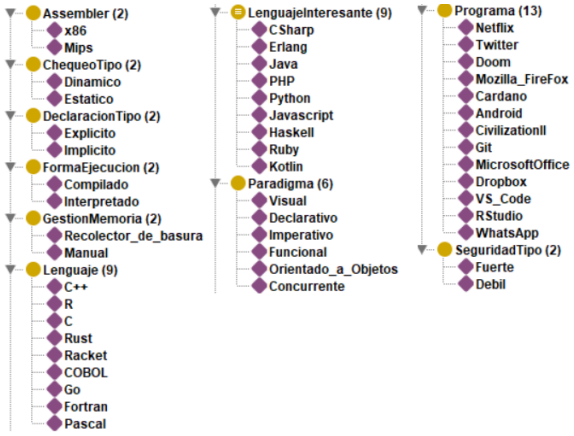


Figura 2: Instancia

En cuanto al temas de las relaciones tenemos las siguientes

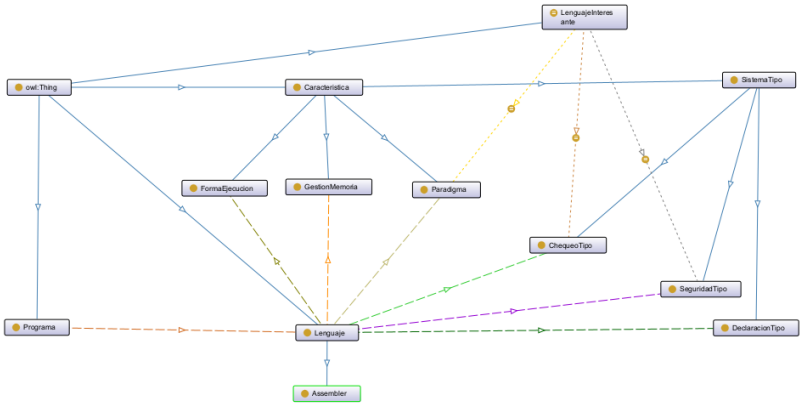


Figura 3: Relaciones, las inversas no aparecen

5. Resultados obtenidos con el razonador

Para ver como funciona el razonador lo que hacemos es lo siguiente, definimos cada relación y su inversa. Cuando "Instanciamos" solo lo hacemos con las normales, no con la inversa. De esta forma el razonador infiere el tipo de la inversa.

La otra que hacemos es dejar que intuya solo el valor de la inversa, es decir si las instancias A y B se relaciona mediante la realización f (A $\rightarrow^{f}$ B) dejamos que el razonador haga la inversa (es decir, B $\rightarrow^{f^{-1}}$ A)

6. Consultas realizadas

Para ver el resultado de las consultas vamos a probar con las preguntas que planteamos al momento de determinar el alcance y veamos si en efecto son correctas. Escribimos las consultas utilizando **DL Query**.

Pregunta: ¿Qué lenguajes funcionales tienen tipado estático y fuerte?

Consulta: (tieneParadigma **value** Funcional) **and** (tieneSeguridadTipo **value** Fuerte) **and** (tieneChequeoTipo **value** Estatico)

Respuesta: C++, CSharp, Fortran, Haskell, Java, Kotlin y Rust.

Pregunta: ¿Qué lenguajes tienen recolector de basura?

Consulta: tieneGestionMemoria **value** Recolector_de_basura

Respuesta: CSharp, Erlang, Go, Haskell, Java, Javascript, Kotlin, PHP, Python, Racket y Ruby.

Pregunta: ¿Qué lenguajes son multiparadigmas?

Consulta: tieneParadigma **min** 2

Respuesta: C++, COBOL, CSharp, Erlang, Fortran, Go, Haskell, Java, Javascript, Kotlin, PHP, Python, Racket, Ruby y Rust.

Pregunta: ¿Qué lenguajes son aptos para programar de forma concurrente?

Consulta: aptoParaProgramar **value** Concurrente

Respuesta: C++, CSharp, Erlang, Fortran, Go, Haskell, Java, Kotlin, Racket y Rust.

7. Análisis del razonamiento del razonador

Para esta sección veamoslo con algunas de las consultas de la seccion anterior anteriores.

Empecemos con: tieneGestionMemoria **value** Recolector_de_basura
Si vemos la respuesta que nos dan se ve que una de ellas es Haskell, entonces para responder porque da este resultado podemos ver la explicación, siendo esta.

Figura 4:

La cual nos dice que Recolector_de_basura se relacion con Haskell por medio de la relacion esGestionMemoriaDe y esta es inversa de tieneGestionMemoria por lo que el razonador puede inferir que Haskell se relaciona con Recolector_de_basura mediante tieneGestionMemoria (dando el resultado correcto)
Veamos ahora con: tieneParadigma **min** 2
De la respuesta vemos que Rust es una de ellas, veamos el porque.
El razonamiento fue el siguiente:

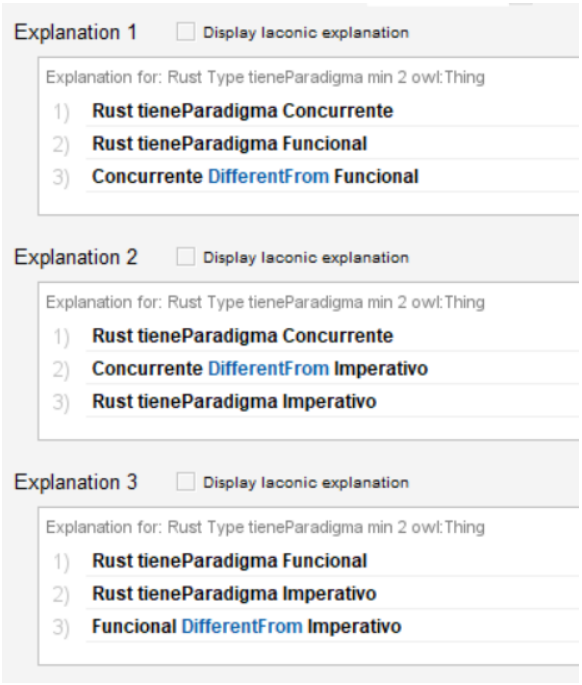


Figura 5:

Donde en un inicio se tiene que Rust tiene 3 tipos paradigmas (distintos entre si). Lo que hace el razonador es verificar que haya al menos 2 distintos, y como los tres son distintos entre si, el razonador encuentra 3 justificaciones validas para dar a Rust como respuesta.

8. Conclusion

Además de intentar el ya mencionado enfoque de Peter Van Roy, surgieron otras cuestiones durante la creación de la ontología.

- Intentamos declarar los lenguajes como clases y las versiones de los mismos con instancias. Puede haber diferencias significativas entre versiones de un mismo lenguaje (ejemplo, Python 3.5 en adelante incluye tipado explicito opcional y C++11 incluye tipado dinámico). Pero por lo general una persona no busca elegir una versión especifica de un lenguaje, sino que elige el lenguaje en si y despues una version estable. Además que la inclusión de las versiones de los lenguajes esta fuera del alcance que necesitamos para responder nuestras preguntas.
- Intentamos incluir como característica el nivel de abstracción del lenguaje (si es de bajo o alto nivel). El nivel de abstracción es un concepto relativo y no es fácil de definir sin conocer el lenguaje en si. C hasta hace no mucho era un lenguaje de alto nivel ya que proveía de conceptos structs y funciones recursivas parametrizadas. Sin embargo en la actualidad es considerado de bajo nivel debido a que no tiene características como un recolector de basura. Como el nivel de abstracción no es facilmente definible, decidimos no incluirlo como característica.

Referencias

[1] Peter Van Roy. *Programming Paradigms: What Every Programmer Should Know*(PDF).

[2] Wikipedia, *List of programming languages by type*.

[3] Wikipedia, *Comparison of programming languages by type system*.

[4] Wikipedia, *Comparison of multi-paradigm programming languages*.