Tesina de Licenciatura

Uso de ontologías en tareas de recupero de información

Marcelo Tallarico

Docentes a cargo

Ing. Cristina Bender

Lic. Claudia Deco

Abstract

El presente trabajo pretende mostrar la ayuda que puede prestar el uso de ontologías en tareas de recupero de información. Para ello, se presentan varias definiciones de ontologías y se ve cómo las mismas van evolucionando a través del tiempo. Aparte del estudio de las ontologías en cuanto a definición, estructura, metodologías de desarrollo, lenguajes y herramientas disponibles se intenta obtener un *mecanismo sencillo y flexible para la utilización de ontologías que sea de ayuda concreta para el recupero de información*. Por tal motivo, se expone una ontología tomada del dominio musical, se la estudia estructuralmente y se plantea una situación real de búsqueda. La resolución de la búsqueda planteada pretende ser un ejemplo particular cuya metodología sea aplicable a cualquier ontología. El mecanismo sencillo y flexible que se busca es encontrado en una interfase de programación (API) de una herramienta de edición de ontologías (Protégé2000), de fácil utilización y amplia generalidad.

El trabajo está organizado de la siguiente manera: en el capítulo 1 se introduce el tema; en el capítulo 2 se define el término *ontología* de diversas maneras y se muestran sus componentes, distintos tipos, lenguajes de implementación y principios para construirlas; en el capítulo 3 se puntualizan históricamente algunas de las herramientas disponibles para el desarrollo de ontologías; en el capítulo 4 se expone un ejemplo de ontología tomado del dominio musical; en el capítulo 5 se muestra la ayuda que introduce el uso de ontologías para búsqueda de información a través de un ejemplo concreto; en el capítulo 6 se presentan las conclusiones y trabajos futuros.

Contenido

1. Introducción

La palabra ontología fue tomada de la filosofía, campo en el cual se define como la *explicación sistemática del ser*. Según el diccionario de la Real Academia Española, ontología es la *parte de la metafísica que trata del ser en general y de sus propiedades trascendentales.*

En las últimas décadas, la palabra ontología se ha tornado relevante y popular en la comunidad científica relacionada al estudio del conocimiento (Knowledge Engineering) y otras áreas cercanas. En el siguiente capítulo se detallarán distintas definiciones, pero analizadas desde el punto de vista de estas comunidades científicas, y no ya desde la filosofía o la lingüística. Una buena manera de captar su significado es recorrer las distintas definiciones teniendo en cuenta que las ontologías apuntan a obtener *conocimiento consensuado* de un dominio en cuestión con el objetivo de poder compartirlo.

Una vez definido el concepto de ontología se lo analiza estructuralmente: una ontología está compuesta por conceptos, relaciones, funciones, axiomas e instancias. En función de esa estructura se verá que una ontología puede clasificarse en ontología liviana u ontología pesada. También se la puede tipificar desde el punto de vista de su objetivo y dominio.

Aunque dista mucho de ser una disciplina ingenieril, para construir una ontología es necesario aplicar una metodología antes que llevar a cabo un proceso anárquico. Varias metodologías son enumeradas y suscintamente analizadas en este trabajo.

Al tener construida una ontología se procede a su implementación o representación. Para ello se cuenta con distintos lenguajes disponibles. En los últimos años se ha avanzado mucho en este área a partir del surgimiento de los lenguajes de markup (XML y derivados). Se brindará una lista de algunos lenguajes y su evolución histórica como también sus principales características.

Afortunadamente hay una amplia gama de herramientas disponibles para desarrollar y mantener ontologías y varias de ellas son enumeradas y comentadas. En particular y a los efectos del presente trabajo se utilizará Protégé2000. Los motivos de la elección serán justificados en lo sucesivo y a modo de introducción a esta herramienta se describe su funcionamiento básico en el Apéndice A.

Como ejemplo se muestra una ontología tomada del dominio musical. La misma es descripta y analizada en cuanto a su estructura. Al contar con varias clases y muchas instancias servirá como base para realizar, a modo de ejemplo, una búsqueda dentro de la ontología con el objetivo de hacer sencilla y flexible esta tarea.

2. Ontologías

Definiciones de ontología.

En la introducción se expuso la definición de ontología tomada de la filosofía y del diccionario de la Real Academia Española. A continuación se presentan varias definiciones de ontología, y se observa cómo esas definiciones cambian y evolucionan a través del tiempo.

- Según Gruber define en [1], ontología es una especificación de una conceptualización
  Es una **descripción** (como la descripción formal de un programa) de **conceptos** y **relaciones** que pueden existir para un agente o una comunidad de agentes
  Un cuerpo de conocimiento representado formalmente está basado en una conceptualización: los objetos, conceptos y otras entidades que se presumen existentes en un área de interés y las relaciones que las contienen. La conceptualización es una forma abstracta y simplificada de ver el mundo que se quiere representar con algún propósito.
- La definición precedente es extendida por Borst en [2]: ontología es una especificación **explícita** de una conceptualización **compartida**.
- Studer la extiende nuevamente en [3] diciendo que ontología es una especificación explícita, **formal** de una conceptualización compartida. La **conceptualización** se refiere a un modelo abstracto de algún fenómeno del mundo teniendo identificados los conceptos relevantes del fenómeno. **Explícito** significa que el tipo de conceptos usados, y las restricciones en su uso son explícitamente definidas. **Formal** se refiere al hecho de que la ontología debiera ser *machine-readable*. **Compartido** refleja la noción de que una ontología captura conocimiento no privado de algún individuo, sino aceptado por un grupo.
- Corcho, Fernandez-Lopez y Gomez-Perez, autores de [4], van más allá y la definen como una **teoría lógica** que da una explícita y parcial cuenta de una conceptualización.
- Una definición más constructiva es dada por Noy y McGuinness en [5]: ontología es una descripción formal y explícita de **conceptos** en un dominio de discurso (*clases* o *conceptos*), **propiedades** de cada concepto que describen varias características y atributos del concepto (*slots*, *roles* o *prepiedades*) y **restricciones** en los slots (*facetas* o *restricciones de rol*)

- Los autores de [6] ponen de manifiesto que una ontología define los términos básicos y relaciones que comprenden el vocabulario de un tema, como así también, define las reglas para combinar términos y relaciones y así introducir extensiones en el vocabulario. Cabe mencionar que una ontología no sólo incluye los términos que son expresamente definidos en ella, sino también los que pueden ser inferidos de ella.
- Uschold y Jasper comentan en [7] que una ontología puede tomar varias formas, pero que necesariamente deberá incluir un vocabulario de términos y alguna especificación de su significado. Esto incluye definiciones y una indicación de cómo los conceptos son **interrelacionados**; esta interrelación impone colectivamente una estructura en el dominio y restringe la posible interpretación de términos

Como se ve claramente, existen múltiples definiciones de ontología. Cabe mencionarse que el término suele diluirse en el sentido de que las **taxonomías** (o meras clasificaciones) son a menudo consideradas como ontologías. Las ontologías difieren de las taxonomías en dos aspectos: las ontologías tienen una **estructura interna** más rica y reflejan cierto **consenso**. El consenso depende del contexto en el cual la ontología es generada. Por ejemplo, si un hospital decide armar una ontología, el consenso está dado por los médicos que intervienen en su desarrollo [3]. Por tal razón, el desarrollo de una ontología es un proceso cooperativo en el cual intervienen diferentes personas.

La comunidad ontológica distingue ontologías, que son principalmente taxonomías, de las que modelan un dominio en una forma más ajustada y proveen más restricciones al dominio semántico. Se las suele distinguir como ontologías livianas y pesadas respectivamente.

- Ontología liviana: incluye conceptos, taxonomía de conceptos, relaciones entre conceptos y propiedades que describen conceptos.
- Ontología pesada: se agregan axiomas y restricciones a la anterior.

A pesar de las diferentes definiciones, existe un consenso generalizado de la comunidad ontológica y no existe confusión sobre su uso [4].

Para estructurar y enfocar más específicamente el presente trabajo, la definición elegida es la siguiente:

- *descripción formal y explícita de **conceptos** en un dominio de discurso (clases o conceptos), **propiedades** de cada concepto que describen varias características y atributos del concepto (slots, roles o prepiedades) y **restricciones** en los slots (facetas o restricciones de rol.)*

La elección se debe a la claridad de la definición para estructurar e identificar los componentes de una ontología. De esa manera se facilita la presentación y explicación de los ejemplos que se muestran más adelante.

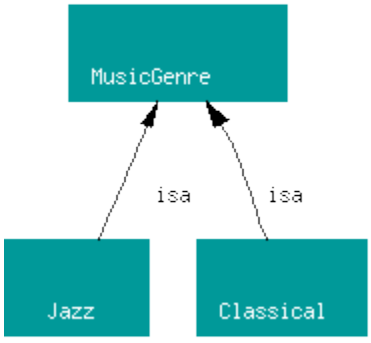
Finalmente, puede decirse que las ontologías ayudan a capturar conocimiento consensuado de una manera genérica y formal, y que puede ser reusada y compartida a través de aplicaciones (software) y por grupos de personas [4].

Componentes de una ontología

Una ontología está compuesta por cinco tipos de componentes: conceptos, relaciones, funciones, axiomas e instancias [1].

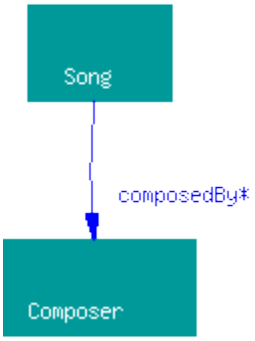
- Conceptos: son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.
  Ejemplo: El jazz y la música clásica (*Jazz* y *Classical*) pueden verse como conceptos dentro de los géneros musicales.

- Relaciones: representan la interacción y enlace entre los conceptos del dominio. Ejemplos de relaciones pueden ser *subclase-de*, *conectado-a*, *parte-de*, etc.
  Ejemplo de relación *subclase-de*: El jazz y la música clásica (*Jazz* y *Classical*) son subclases de géneros musicales (*MusicGenre*).

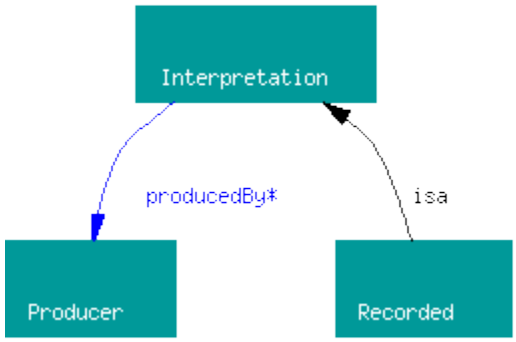


Ejemplo de relación *compuesto-por*: una canción (*Song*) está compuesta

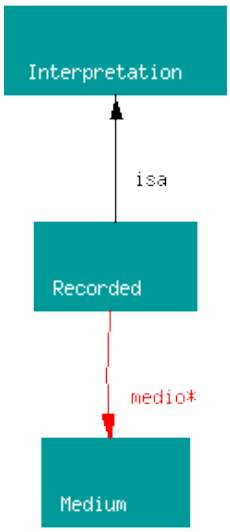
(*composedBy*) por un compositor (*Composer*).



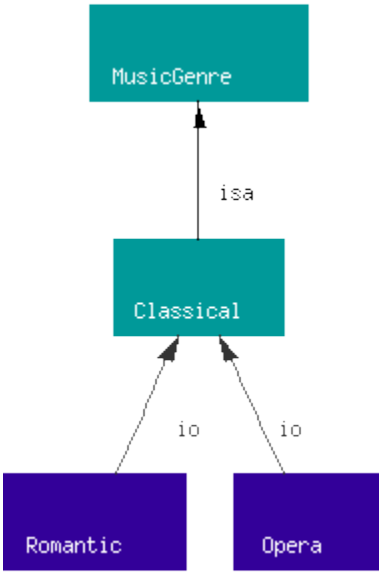
- Funciones: son un tipo especial de relación donde se identifica un elemento mediante el cálculo de una función.
  Ejemplo de función: *cant-productores-grabacion* devuelve la cantidad de productores (*Producer*) que tiene una interpretación (*Interpretation*).



- Axiomas: son teoremas sobre relaciones que deben cumplir los elementos de la ontología.
  Ejemplo de axioma: todas las grabaciones (*Recorded*) deben tener un medio asociado (*Medium*)



- Instancias: son usadas para representar objetos determinados de un concepto.
  Ejemplo: Los géneros musicales ópera y romántico pertenecen a música clásica, es decir, *Romantic* y *Opera* son instancias de la clase (o concepto) *Classical*.



Tipos de ontologías

En función del objetivo y dominio de las ontologías podemos tipificarlas de la siguiente manera:

- Ontologías para representación de conocimiento: capturan las primitivas de representación usadas para formalizar conocimiento en paradigmas de representación de conocimiento.
  Ejemplo de este tipo es Frame-Ontology [1], que captura la representación de primitivas usadas en lenguajes basados en marcos (*frame-based)*.
- Ontologías generales (o comunes): incluyen vocabulario relacionado a cosas, eventos, tiempo, espacio, comportamiento, etc.
  Ejemplo de este tipo es CYC [8] que posee un monto considerable de conocimiento humano fundamental
- Meta-ontologías: también llamadas *Generic Ontologies* o *Core Ontologies* son reusables a través de distintos dominios.
  Ejemplo de este tipo es Mereology [2]. En ella, Borst define la relación *parte-de*, estudia sus propiedades, y permite expresar, por ejemplo, que un dispositivo está ensamblado por componentes, cada uno de los cuales puede ser descompuesto en subcomponentes.
- Ontologías de dominio: son reusables en un dominio dado. Proveen vocabularios sobre los conceptos de un dominio y sus relaciones, sobre las actividades que toman lugar en el dominio y sobre las teorías y principios elementales que gobiernan tal dominio.
  Ejemplos de este tipo son Eng-Math [9] y PhysSys [2]. Eng-Math formaliza modelos matemáticos usados en ingeniería. PhysSys es una librería de ontologías y describe el dominio de conocimiento requerido para hacer modelos de simulaciones de dispositivos, como sistemas de calefacción, sistemas automotores y máquinas.
- Ontologías lingüísticas: tienen el objetivo de modelar el lenguaje natural. Ejemplo de este tipo de ontología es Sensus [10], que provee una amplia estructura conceptual para trabajar en traducciones. Se desarrolló para mezclar y extraer información de recursos electrónicos existentes.
- Ontologías de tareas: proveen vocabulario sistemático de términos usados para resolver problemas asociados con tareas que pueden o no pertenecer al mismo dominio. Incluyen nombres genéricos, verbos genéricos y adjetivos genéricos.
- Ontologías de tareas de dominios: son ontologías reusables en un dominio dado pero no a través de otros dominios.
- Ontologías de métodos: proveen definiciones de conceptos relevantes y

relaciones usadas para especificar un proceso de razonamiento para lograr una tarea particular.

- Ontologías de aplicación: contienen el conocimiento necesario para modelar una aplicación particular.

La ontología presentada en el punto anterior puede encuadrarse dentro de las ontologías de dominio. En su definición, se proveen los conceptos y relaciones de los instrumentos y estilos musicales (entre otras), y dentro de ese dominio, es reutilizable.

Principios y metodologías para construir ontologías

En este apartado se describen un conjunto de principios y criterios de diseño que suelen ser útiles en el desarrollo de ontologías [1]

- Claridad y objetividad: la ontología debe proveer el significado de los términos definidos proveyendo definiciones objetivas y documentación en lenguaje natural.
- Completitud: una definición expresada en términos de condición necesaria y suficiente es preferible sobre una definición parcial (definida sólo sobre condición necesaria o suficiente)
- Coherencia: permite hacer inferencias válidas consistentes con las definiciones
- Máxima extensibilidad monotónica: los nuevos términos (generales o especializados) debieran ser incluidos en la ontología de manera que no requiera la revisión de definiciones existentes
- Mínimo compromiso ontológico: minimizar como sea posible el mundo que se está modelando

Otros principios son: estandarización de nombres (tanto como sea posible), modularidad (para minimizar el acoplamiento entre módulos) y el hecho de que las clases en una ontología debieran ser disjuntas.

El proceso de construcción de ontologías es más un arte que una actividad ingenieril. Cada equipo de desarrollo usualmente sigue su propio conjunto de principios, criterios de diseño y fases en el proceso del desarrollo de la ontolgía. La ausencia de líneas de acción comunes entorpece el desarrollo de ontologías consensuadas y compartidas entre distintos equipos de desarrollo, como también su reuso y extensión.

Si las ontologías son desarrolladas a menor escala, algunas actividades pueden evitarse. Pero si la intención es construir ontologías a gran escala con garantía de corrección y completitud, es conveniente evitar construcciones

anárquicas y seguir un enfoque metódico.

En [11] se proponen los siguientes pasos:

1. identificar el propósito y alcance de la ontología
2. construir la ontología capturando, codificando e integrando el conocimiento con ontologías existentes
3. evaluar la ontología
4. documentarla
5. dar pautas claras para cada etapa

En [12] se propone construir un modelo lógico del conocimiento que se va a especificar en la ontología. Este modelo no se construye directamente. Primero, las especificaciones que deberán ser encontradas en la ontología, son descriptas informalmente identificando un conjunto de preguntas competentes, y su descripción luego es formalizada en un lenguaje basado en cálculo de predicados de primer orden. Las preguntas son las bases para una caracterización rigurosa del conocimiento que la ontología pretende alcanzar, y ellas especifican el problema y qué constituye una buena solución. A través de un mecanismo de composición y descomposición, las preguntas y respuestas pueden ser usadas para responder preguntas relacionadas más complejas en otras ontologías, permitiendo su integración.

Lenguajes de ontologías

Una vez que los componentes de la ontología están definidos, la ontología puede ser representada en varios lenguajes. Los mismos comenzaron a surgir a comienzos de 1990 y se basan principalmente en lógicas de primer orden, en marcos (*frames*) combinados con lógicas de primer orden y lógicas descriptivas (DL). A continuación se listan algunos de ellos siguiendo un orden cronológico:

- KIF: es un lenguaje basado en lógica de primer orden y se creó como formato de intercambio para diversos sistemas de representación de conocimiento. Es el más expresivo de los lenguajes usados para representar ontologías, permitiendo representar conceptos, taxonomías de conceptos, relaciones n-arias, funciones, axiomas, instancias y procedimientos. Sin embargo el lenguaje en sí, no provee soporte para razonamiento automático. [13]
- Ontolingua: es un sistema para describir ontologías de manera compatible con múltiples sistemas de representación. Provee formas para definir clases, relaciones, funciones, objetos y teorías. Las ontologías escritas en Ontolingua pueden ser compartidas por varios grupos de usuarios e

investigadores que usan su lenguaje de representación favorito. La sintaxis y semántica de las definiciones de Ontolingua están basados en KIF y traduce las definiciones a distintos sistemas de representación implementados [1]

- Loom: inicialmente no fue pensado para implementación de ontologías. Está basado en lógicas descriptivas y reglas de producción y provee una clasificación automática de conceptos. Puede representar conceptos, taxonomías de conceptos, relaciones n-arias, funciones, axiomas, y reglas de producción. [14]
- OCML: fue construido para desarrollar ontologías ejecutables y modelos en métodos de resolución de problemas. A las posibilidades de KIF se le agregan reglas de producción y deducción, y definiciones operacionales de funciones. [15]
- FLogic (*Frame Logic*): combina marcos y lógica de primer orden. Permite representar conceptos, taxonomías de conceptos, relaciones binarias, funciones, axiomas y reglas deductivas. Se diferencia de los anteriores en que es el único que no tiene una sintaxis similar al Lisp. Provee un mecanismo de inferencia (Ontobroker) que puede ser usado para verificación de restricciones y deducción de información nueva. [16]

El boom de Internet de los años sucesivos provocó el surgimiento de otros lenguajes que aprovechan las características de la web. Se los suele llamar como *web-based* o*ntology languages* u *ontology markup languages.*

- SHOE: se creó como extensión del HTML. Usa diferentes *tags* que permite la inserción de ontologías en documentos HTML. Combina marcos y reglas y puede representar conceptos, taxonomías de conceptos, relaciones n-arias, instancias y reglas de deducción, que son usadas por su motor de inferencias para obtener nuevo conocimiento. [17]
- XML: luego surge este lenguaje y es ampliamente aceptado como lenguaje estandar para intercambio de información en la web. Como consecuencia, la sintaxis de SHOE fue modificada para usar XML y luego otros lenguajes de ontología se construyeron sobre XML. [18]
- XOL: surge a partir de la conversión a XML de un subconjunto pequeño de primitivas del protocolo OKBC (*Open Knowledge Base Connectivity*) [24]. Es muy restrictivo y representa conceptos, taxonomía de conceptos y relaciones binarias. No provee un mecanismo de inferencia. Su objetivo es el intercambio de ontologías en dominios biomédicos. [19]
- RDF: fue desarrollado en base a lenguajes sobre redes semánticas con el fin de describir recursos de la web. [20]
- RDF Schema: se desarrolló como una extensión de RDF con primitivas basadas en marcos. La combinación de ambas suele denotarse como RDF

(S). No es muy expresivo, sólo permite la representación de conceptos, taxonomías de conceptos y relaciones binarias. Algunos motores de inferencia fueron creados para este lenguaje, en especial para verificación de restricciones. [21]

- OIL: agrega al RDF(S) primitivas basadas en marcos y su semántica formal está basada en lógicas descriptivas. Se provee una clasificación automática de conceptos a través de FaCT.
- DAML + OIL: agrega primitivas de representación de conocimiento basados en lógica descriptiva a RDF(S). Permite representar conceptos, taxonomías, relaciones binarias, funciones e instancias. Se está invirtiendo un considerable esfuerzo en proveer mecanismos de razonamiento para este lenguaje. [22]
- OWL: recientemente surgido. Está basado en las principales características de DAML + OIL. [23]

Al implementar una ontología es importante decidir primero las necesidades en términos de expresividad y servicios de inferencia, porque no todos los lenguajes permiten representar los mismos componentes de la misma forma. La representación y razonamiento con información básica, como conceptos, taxonomías, y relaciones binarias, no siempre es suficiente si se quiere crear una ontología pesada y hacer razonamientos complejos. Es frecuente que las traducciones entre lenguajes no sea lo sufientemente precisa por lo que puede perderse información en el proceso de traducción. Por eso, la decisión del uso de un lenguaje específico para representar una ontología, juega un papel crucial [4].

3. Herramientas

En los últimos años, el número de entornos de desarrollo para ontologías y herramientas relacionadas ha crecido considerablemente. Las herramientas ayudan a proveer soporte para el proceso de desarrollo y uso de las ontologías. A continuación se presentan las características de las más relevantes.

- Ontolingua Server fue la primera herramienta desarrollada. Surgió a principios de 1990 en el Knowledge Systems Laboratory (KSL) de la universidad de Stanford. Es una aplicación basada en formularios web para que distintos editores remotos puedan ver y editar ontologías. También permite que aplicaciones locales o remotas puedan acceder a alguna ontología disponible en la librería de ontologías a través del protocolo OKBC [24].
- Al mismo tiempo surge Ontosaurus y fue desarrollado en el Information Sciences Institute (ISI) de la universidad de South California. Tiene características similares en cuanto a modalidades de visualización y edición [25].
- En 1997 aparece WebOnto [26], desarrollado en el Knowledge Media Institute de la Open University.

La principal similaridad de los entornos anteriores es que todos tienen una fuerte relación con un lenguaje específico de ontología: Ontolingua, LOOM y OCML respectivamente. Fueron creadas para permitir visualizar y editar fácilmente ontologías en esos lenguajes. Su objetivo primordial era emplearlas en tareas de investigación y se concibieron como herramientas aisladas, por lo cual, no proveen facilidades de extensibilidad.

En los últimos años se desarrollaron entornos de desarrollo un tanto más ambiciosos, con el objetivo de integrar ontologías a sistemas de información existentes. Por lo general son extensibles, basados en arquitecturas de componentes y lo más importante: por lo general son independientes del lenguaje de implementación de la ontología. Entre los más populares pueden citarse los siguientes:

- Protégé 2000 Protégé-2000 fue desarrollado por el Stanford Medical Informatics (SMI) de la universidad de Stanford. Es un proyecto Java de fuente libre Java que provee una arquitectura extensible para la creación de herramientas de bases de conocimiento. Es una herramienta porque permite al usuario construir el dominio de una ontología, configurar formularios para relevamiento de datos e ingresar el dominio de conocimiento. Es una plataforma que puede ser extendida con gráficos, diagramas, componentes animados para acceder a aplicaciones embebidas en sistemas de bases de

conocimientos. Es también una librería que otras aplicaciones pueden acceder [27].

- WebOde es el sucesor de ODE (*Ontology Design Environment*) y fue desarrollada en el Laboratorio de Inteligencia Artificial de la Universidad Técnica de Madrid (UPM). No es usada como aplicación Stand-Alone sino a través de la web. El corazón de este entorno es el servicio de acceso a ontologías, que es usado por todos los servicios y aplicaciones que se agregan al servidor. Los servicios que presta son edición de ontologías, importación/exportación, edición de axiomas, documentación evaluación y combinación de ontologías [28].
- KAON (Karlsruhe Ontology) es un entorno flexible y extensible en la misma medida que los anteriores. KAON también es un proyecto JAVA de fuente libre con infraestructura para editar y mantener ontologías. Es el sucesor de OntoEdit y fue desarrollado por AIFB en la universidad de Karlsruhe. Brinda una interfase de programación JAVA para acceder a ontologías e instancias, independientemente del formato de implementación de la ontología. Provee de una simple herramienta para generar portales web con soporte multilingual basados en ontologías [29]

Con el crecimiento de los antes mencionados lenguajes de ontologías basados en web, surgen herramientas que puedan trabajar con ellos también. De hecho, los tres anteriores, permiten exportar e importar ontologías DAML + OIL y RDF(S). También pueden mencionarse OILEd y DUET.

- OILEd surgió como editor de ontologías OIL en el contexto del Eruopean IST On-To-Knowledge project y actualmente es un editor de ontologías DAML + OIL. Tiene un motor de inferencias que provee chequeo de consistencia y clasificación automática de conceptos [30].
- DUET (DAML UML Enhanced Tool) fue desarrollado por AT&T Government Solutions Advanced Systems Group. Ofrece integración con UML como componenente en la suite Rational Rose. [31]

A los efectos del presente trabajo se eligió usar **Protégé2000** por su claridad para visualizar y editar ontologías Stand-Alone, como así también por la versatilidad de su interfase de programación JAVA para acceder, consultar y obtener información de la ontología en cuestión.

Esta herramienta brinda la posibilidad de editar y mantener ontologías en varios formatos estandarizados y en un formato propio también. Por ejemplo, Protégé2000 permite acceder a una ontología a través del protocolo OKBC [24]. De esta manera, la ontología se almacena en una base de datos relacional. Otros formatos conocidos en los que se permite trabajar son RDF Schema, XML y DAM-L + OIL (entre otros). Además se puede importar y exportar entre todos estos formatos distintos.

4. Ontología: Dominio musical

En los últimos años el intercambio de música en formato electrónico se vio beneficiado por los avances tecnológicos en redes, proceso de señales, compresión de audio y protección de datos digitales. Estos avances, brindan a los usuarios finales de estas tecnologías el acceso a vastos catálogos de música (en el orden de los 4 a 8 millones de títulos disponibles).

Esto hace tremendamente necesario tener disponible metadatos que describan el contenido musical de los catálogos, especialmente en el contexto de servicios de búsqueda de música en formato electrónico. Los metadatos son usados como *capa de conocimiento* para servicios de envío de música electrónica (EMD: *Electronic Music Delivery*) como ser sistemas de envío de música por demanda o Internet Radio.

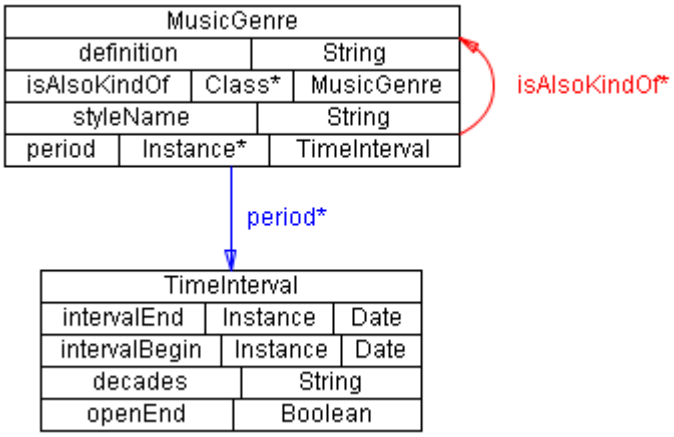
Cada item del catálogo está definido por un **conjunto de descriptores**, que toman su valor de una ontología (o mera taxonomía) predefinida. Algunos ejemplos de descriptores pueden ser el nombre del título (ej. *Footprints*), el nombre del autor (ej. *Wayne Shorter*), el género musical (ej. *Jazz*), principales instrumentos (ej. *piano, trompeta*), etc.

En este trabajo se utiliza una ontología que caracteriza esos metadatos. La misma se encuentra disponible en formato de Protégé2000 y al momento de su confección se generó a partir del sitio www.allmusic.com. En esa oportunidad se usó ese sitio como punto de partida, dado que ahí se dispone de una amplia categorización de los distintos tipos de géneros musicales. La ontología hallada fue tenida en cuenta por la categorización de conceptos musicales y por el buen número de instancias que dispone. De esta manera se la puede utilizar para distintas pruebas.

De acuerdo a consideraciones anteriores, puede decirse que esta ontología es una ontología liviana, ya que prácticamente no posee restricciones ni axiomas.

A continuación se muestra la estructura de las clases más importantes.

Clase *MusicGenre*



La clase *MusicGenre* tiene los siguientes slots:

- Nombre del estilo (s*tyleName*): *String* que contiene el nombre del estilo
- Definición (*definition*): *String* que contiene la descripción del estilo. Entre otras cosas se encuentran datos como ser época donde surge el estilo, características técnico-musicales, historia y músicos que intervienen.
- Período (*period*): instancia de la clase *TimeInterval* que describe el período en que surge el estilo
- También es un tipo de (i*sAlsoKindOf*): relación que indica que un estilo de música (clase *MusicGenre*) puede ser considerado también como otro estilo de música.

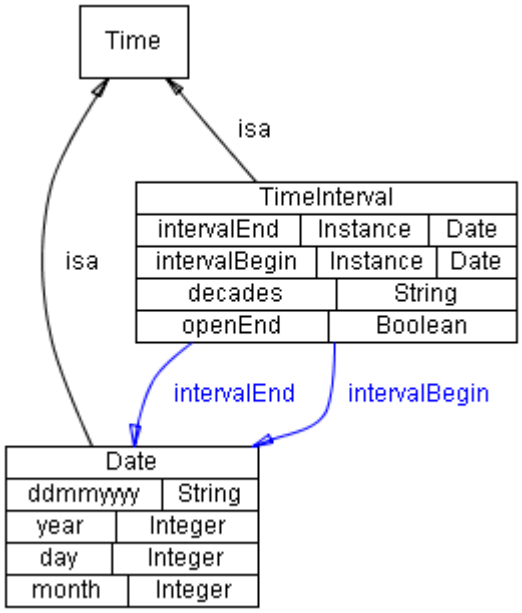
  Ejemplo de instancia de esta clase: *Smooth Jazz*

  Clase: *MusicGenre* / *Jazz* / *Fusion*

- *styleName*: Smooth Jazz
- *definition*: Smooth Jazz is an outgrowth of fusion, one that emphasizes its polished side. Generally, smooth jazz relies on rhythms and grooves instead of improvisation. There are layers of synthesizers, lite-funk rhythms, lite-funk bass, elastic guitars, and either trumpets, alto, or soprano saxophones. The music isn't cerebral, like hard bop, nor is it gritty and funky like soul-jazz or groove -- it is unobtrusive slick, and highly polished, where the overall sound matters more than the individual parts.
- *period*: 80's – today (instancia de clase *TimeInterval*)

Para crear esta instancia, primero fue necesario extender la clase *MusicGenre* a *Jazz* y esta última a la clase *Fusion*.

Clases *Time*, *Date* y *TimeInterval*



La clase *Date* extiende la clase *Time.* Se utiliza para señalar una fecha en particular y tiene los siguientes slots:

- Día (*day*): *Integer* que contiene el día de la fecha. Está restringido a un número <= 31
- Mes (*month*): *Integer* que contiene el mes de la fecha. Está restringido a un número <= 12
- Año (*year*): *Integer* que contiene el año de la fecha
- Día/Mes/Año (*ddmmyyyy*): *String* con que se quiere señalar la fecha

Ejemplo de instancia de esta clase: *1/10/1940*

Clase: *Date*

- *day*: 1
- month: 10
- year: 1940
- *ddmmyyyy:* 1/10/1940

La clase *TimeInterval* también extiende la clase *Time.* Se utiliza para señalar un intervalo de fechas y tiene los siguientes slots:

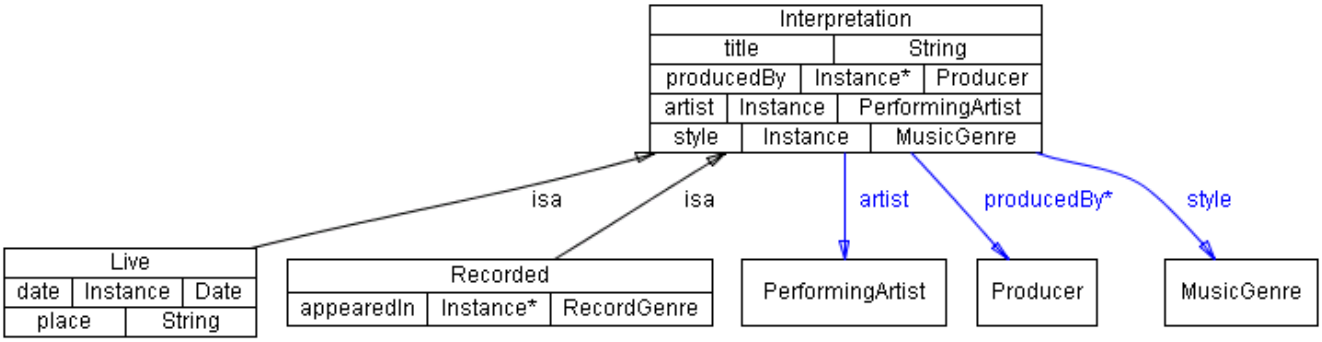
- Décadas (*decades*): *String* que contiene una descripción de las décadas que contiene de intervalo
- Intervalo abierto (*openEnd*): *Boolean* que indica que el intervalo es abierto y se extiende hasta la actualidad
- Inicio del intervalo (*intervalBegin*): instancia de la clase *Date* que contiene la fecha de inicio del intervalo
- Fin del intervalo (*intervalEnd*): instancia de la clase *Date* que contiene la fecha de fin del intervalo

Ejemplo de instancia de esta clase: *20's - 60's*

Clase: *TimeInterval*

- *decades*: 20's - 60's
- *openEnd*: false
- *intervalBegin*: 1/1/1920 (instancia de la clase *Date)*
- *intervalEnd*: 31/12/1969 (instancia de la clase *Date)*

Clases *Interpretation*, *Live* y *Recorded*



La clase *Interpretation* se utiliza para señalar una "interpretación", dado que una canción puede ser interpretada de distintas maneras por distintos artistas. Dos clases heredan de ella: *Live* y *Recorded*, para señalar que una interpretación puede existir como grabación de estudio o en vivo respectivamente. Tiene los siguientes slots:

- Título (*title*): *String* que contiene el título
- Producida por (*producedBy*): instancia de la clase *Producer* que contiene el productor de la interpretación
- Artista (*artist*): instancia de la clase *PerformingArtist* que contiene el artista que la interpreta
- Estilo (*style*): instancia de la clase *MusicGenre* que continene el género musical al que pertenece

Ejemplo de instancia de esta clase: *As Time Goes by*

Clase: *Interpretation* / *Recorded*

- *title*: As Time Goes by
- *producedBy*: Nathan Morris (instancia de la clase *Producer)*
- *artist*: Richard Clayderman (instancia de la clase *PerformingArtist)*
- *style*: Romantic (instancia de la clase *MusicGenre)*
- *apperedIn*: The Very Best Of Richard Clayderman (instancia de la clase *Recorded)*

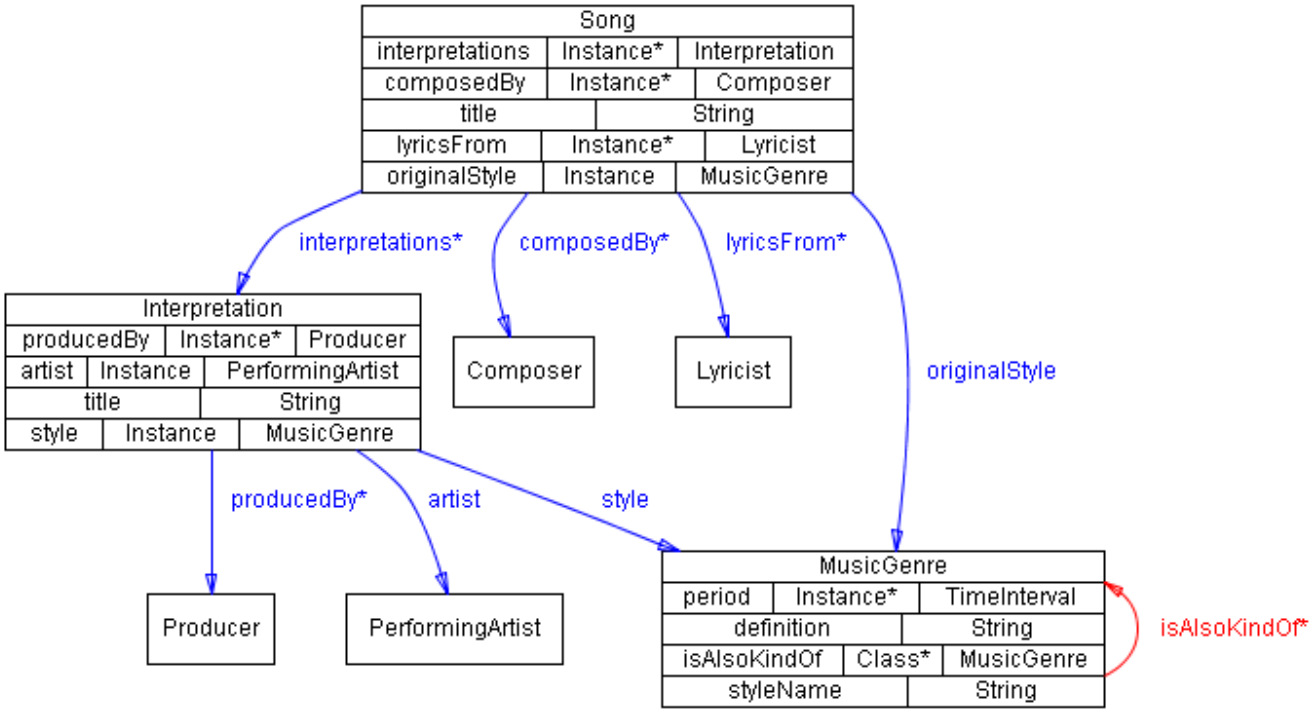
La clase *Live* agrega los siguientes slots:

- Lugar (*place*): *String* que contiene el lugar donde se hizo la grabación en vivo
- Fecha (*date*): instancia de la clase *Date* que contiene la fecha de grabación

La clase *Recorded* agrega el siguiente slot:

- Grabado en (*appearedIn*): instancia de la clase *RecordGenre* que contiene el tipo de grabación. Los tipos de grabación pueden ser álbum, compilación, simple, etc.

Clases *Song*, *Composer* y *Lyricist*



La clase *Song* modela una canción, y *Composer* y *Lyricist* modelan al compositor y escritor, respectivamente. La clase *Song* y tiene los siguientes slots:

- Interpretaciones (*interpretations*): instancias de la clase *Interpretation*, que contiene las distintas interpretaciones que pueda tener una canción.
- Compuesta por (*composedBy*): instancias de la clase *Composer* que contiene los compositores que pueda tener la canción
- Título (*title*): *String* que contiene el título
- Letras de (*lyricsFrom*): instancias de la clase *Lyrycist*, que contiene los escritores que pueda tener la canción.
- Estilo original (*originalStyle*): instancia de la clase *MusicGenre* que continene el género musical al que pertenece la canción

Ejemplo de instancia de esta clase: *I shot the sherif*

Clase: *Song*

- *interpretations*: I shot the sherif#1, I shot the sherif#2 (dos interpretaciones distintas, ambas instancias de la clase *Interpretation)*
- *composedBy*: Bob Marley (instancia de la clase *Composer)*
- *title*: I shot the sherif
- *lyricsFrom*: Bob Marley (instancia de la clase *Lyrycist)*
- *originalStyle*: Reggae Pop (instancia de la clase *MusicGenre)*

5. Caso de estudio

Introducción

En el punto anterior se mostró la necesidad de disponer metadatos que describan la diversidad de contenidos musicales, de manera de poder ser utilizados, entre otras cosas, para servicios de búsqueda de música.

Un caso usual que se presenta en la comunidad de músicos es la necesidad de búsqueda de distintas interpretaciones de una misma canción en diversos estilos musicales. Esto se hace con fines musicales, de manera que el músico pueda acceder a muchas interpretaciones de una misma canción y así nutrirse y embeberse de la canción (en sus distintas interpretaciones) con el objetivo de hacer su propia interpretación.

Por ejemplo, si un músico deseara hacer su propia interpretación de la canción *Footprints* (cuyo autor es *Wayne Shorter*), puede hacer una búsqueda de música y así acceder a las distintas interpretaciones. Si esta búsqueda se hiciera en base a la ontología que se dispone, hay 4 interpretaciones disponibles:

Interpretación	*Intérprete*	*Estilo*	*Año*
Footprints #1	Miles Davis	Hard Bop	1966
Footprints #2	Liquid Soul	Free Funk	1996
Footprints #3	Wayne Shorter	Standards	1966
Footprints Live	Wayne Shorter	Free Jazz	2002

De esta manera se obtienen distintas interpretaciones en distintos estilos (Hard bop, Free Funk, Standards y Free Jazz) y pertenecientes a distintas épocas.

Objetivo

El objetivo de este apartado es demostrar la sencillez y flexibilidad para acceder a una ontología usando la API Java de Protégé2000. En este caso, se pretende realizar la búsqueda planteada en la introducción del presente capítulo: buscar una canción y así obtener distintas interpretaciones de la misma. Para ello se mostrarán los lineamientos necesarios y las técnicas específicas utilizadas para llevar a cabo el proceso de búsqueda. Como se

mencionara anteriormente, Protégé2000 cuenta con una interfase de programación Java (API Java) que permite acceder a la ontología a través del lenguaje Java y manipularla como se desee.

Desarrollo

El primer paso a realizar es acceder a la ontología. Se hace de la siguiente manera:

```
String PROJECT_FILE_NAME = "/ontologia/musicW.pprj";
Collection errors = new ArrayList();
Project project = new Project(PROJECT_FILE_NAME, errors);
if (errors.size() > 0) {
      Iterator i = errors.iterator();
      while (i.hasNext()) {
            System.out.println("Error: " + i.next());
      }
}
else {
      ejecutarBusqueda(project.getKnowledgeBase());
}
```

Una vez que se tiene disponible la ontología, ya se está en condiciones de manipularla como sea necesario.

Por ejemplo, para obtener todas las instancias de la clase *Song* (en la cual se pretende hacer la búsqueda), se procede la de siguiente manera:

```
Cls songClass = kb.getCls("Song");
Collection intanciasSong=songClass.getInstances();
```

Así, la variable `songClass` representa la clase *Song* y en la variable `intanciasSong` se encuentran todas las instancias de la clase *Song*.

Supongamos ahora que se desea buscar entre las canciones disponibles en la ontología. Por ejemplo, si se pretende buscar por la canción **Footprints**, hay que iterar por las instancias y comparando el título de cada instancia (usando el algoritmo de similitud o proximidad que se desee) hasta encontrar la canción buscada.

```
String busqueda="Footprints";
Iterator iter=intanciasSong.iterator();
Instance insSong=null;
while (iter.hasNext()){
```

```
      insSong= (Instance) iter.next();
      String titulo=insSong.getOwnSlotValue(kb.getSlot("title")).
toString();
      if (titulo.toUpperCase().indexOf(busqueda.toUpperCase())>=0)
           break;
}
```

Una vez ubicada la canción, se pueden buscar las interpretaciones.

```
Collection interpretaciones=insSong.getOwnSlotValues(kb.getSlot
("interpretations"));
```

Teniendo las interpretaciones puede mostrarse cualquier dato disponible de la ontología y que brinde información relativa a la búsqueda (por ejemplo, estilo, descripción del estilo, intérprete, año, etc).

Por ejemplo, un resultado útil para quien está buscando información sobre la canción **Footprints**, en base a canciones y distintas interpretaciones, sería el siguiente:

Canción encontrada: Footprints
Estilo original: Standards
Escrita por: Wayne Shorter
Definición de estilo original: During the golden age of the American popular song (around 1915-60), several dozen very talented composers wrote a countless number of flexible songs that were adopted (and often transformed)by creative jazz musicians and singers...
==========
Interpretación encontrada: Footprints Live!
Estilo: Free Jazz
Definición de estilo: Dixieland and swing stylists improvise melodically, and bop, cool, and hard bop players follow chord structures in their solos. Free Jazz was a radical departure from past styles, for typically after playing a quick theme, the soloist does not have to follow any progression or structure and can go in any unpredictable direction...
Artista: Wayne Shorter
==========
Interpretación encontrada: Footprints
Artista: Miles Davis
Estilo: Hard Bop
Definición de estilo: Although some history books claim that Hard Bop arose as a reaction to the softer sounds featured in

```
cool jazz, it was actually an extension of bop that largely
ignored West Coast jazz. The main differences between hard
bop and bop are that the melodies tend to be simpler and
often more "soulful"; the rhythm section is usually looser,
with the bassist not as tightly confined to playing four-
beats-to-the-bar as in bop; a gospel influence is felt in
some of the music...
============================================================
```

Interpretación encontrada: Footprints

```
Artista: Wayne Shorter
Estilo: Standards
Definición de estilo: During the golden age of the American
popularsong (around 1915-60), several dozen very talented
composers wrote a countless number of flexible songs that
were adopted (and often transformed) by creative jazz
musicians and singers. Often originally written for Broadway
shows and Hollywood films, many of these works (generally 32
bars in length) have been performed and recorded a seemingly
infinite number of times, including "Body and Soul,"
"Stardust," and "All the Things You Are...
============================================================
```

Interpretación encontrada: Footprints

```
Artista: Liquid Soul
Estilo: Free Funk
Definición de estilo: Free Funk is a mixture of avant-garde
jazz with funky rhythms. When Ornette Coleman formed Prime
Time in the early '70s, he had a "double quartet" (comprised
of two guitars, two electric bassists, and two drummers, plus
his alto) performing with freedom tonally but over eccentric
funk rhythms. Three of Coleman's sidemen (guitarist James
"Blood" Ulmer, bassist Jamaaladeen Tacuma, and drummer Ronald
Shannon Jackson) have since led free funk groups ...
============================================================
```

Conclusión

Si bien la búsqueda planteada es particular, la amplia flexibilidad para utilizar la ontología y hacer búsquedas dentro de ella, permite obtener resultados cualquiera sea la situación y con un mínimo nivel de complejidad para realizar la tarea.

Esto lo hace potencialmente poderoso para realizar búsquedas en una ontología de considerable magnitud.

6. Conclusiones

En este trabajo se pusieron de manifiesto varios aspectos de las ontologías: definiciones, componentes, metodologías para su desarrollo, tipos, lenguajes y distintas herramientas disponibles para su manipulación.

Si bien se presentaron diversas definiciones de ontologías, existe un consenso generalizado de lo que es una ontología y por lo general no se advierten confusiones en su uso. El objetivo de las ontologías es el de obtener conocimiento consensuado de una manera genérica y formal para ser reusado y compartido.

Como se explicara oportunamente, aunque hay muchas metodologías planteadas para la construcción de ontologías, por el momento se está lejos de ser una actividad ingenieril. Pero siempre se recomienda tener un enfoque metódico. También es importante destacar que no hay una única ontología correcta para un dominio dado. La corrección de una ontología va a estar dado en la medida que cumpla con el objetivo de obtener conocimiento consensuado.

Para determinar qué lenguaje es conveniente utilizar para implementar una ontología primero hay que definir las necesidades en términos de expresividad y servicios de inferencia que deberán estar disponibles. Los primeros lenguajes, que estaban basados en lógicas de primer orden y lógicas descriptivas, son los más expresivos. Los lenguajes más modernos están basados en lenguajes tipo *markup* (como ser XML) y no son tan expresivos como los anteriores.

Las herramientas disponibles para la edición y mantenimiento de ontologías juegan un papel preponderante, dado que es una actividad un tanto tediosa. La elección de la herramienta adecuada depende de los objetivos que se persigan. En la realización de este trabajo se ha encontrado que la API Java de programación que brinda Protégé2000 posee características de versatilidad, flexibilidad y sencillez por lo que la convierte en una herramienta potencialmente útil para cualquier necesidad.

El ejemplo presentado, persigue demostrar la utilidad de las ontologías para servir como ayuda en una búsqueda de información en un dominio dado. Si bien se trata de un caso particular y un problema puntual (la búsqueda de distintas versiones de una canción) se lo puede ver desde un punto de vista más general, de manera de poner de manifiesto la utilidad que puede cobrar una ontología desarrollada para una dominio dado.

Trabajos futuros

Son innumerables las posibilidades de continuar desarrollando las ideas y tecnologías presentadas.

Una aplicación útil e inmediata, sería la implementación de una interfase de búsqueda más completa y rica para el caso de estudio del capítulo anterior. Podrían hacerse búsquedas por autores, intérpretes, discos, canciones, estilos de música, tipo de grabación, épocas, etc.; incluso contando con la posibilidad de combinar varios de estos criterios (característica muy útil para la comunidad musical).

Sería interesante también tener la posibilidad de integrar esta ontología con otras; por ejemplo, integrarla con ontologías de compañías discográficas, fábricas de instrumentos, etc. Un tema en estudio de los grupos de investigación es la integración automática de distintas ontologías y aquí se hace evidente tal necesidad.

De la misma forma, es posible extender la presente ontología para alcanzar necesidades aún no cubiertas por ella. Por ejemplo, sería útil contar con las letras de las canciones (ampliamente difundidas en diversos formatos y sistemas de búsquedas), como también los instrumentistas participantes en una grabación. Esto está ampliamente relacionado con la integración de ontologías, ya que habría que extender la ontología sólo en el caso de no poder integrarla con otras ontologías por el hecho de no existir aún o no cumplir con las características necesitadas. Pero este último caso estaría evidenciando que tal ontología no tiene consenso, una de las características básicas de una ontología.

Lo que se ha propuesto en este trabajo es realizar la búsqueda dentro de la ontología, sin embargo, las ontologías pueden servir para clasificar y/o expandir un elemento de búsqueda, de manera de utilizarlo como entrada en otro sistema de búsqueda (como ser un motor de búsqueda de páginas en Internet).

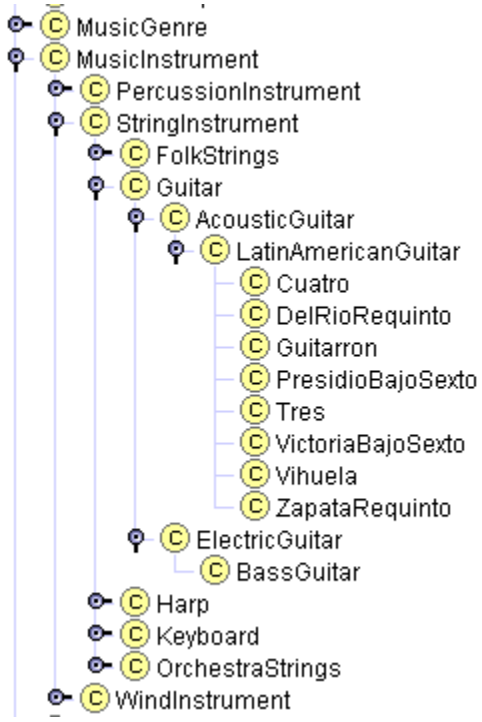
El uso de ontologías en tareas de recupero de información es una actividad emergente que va ganando terreno a medida que se van generando e integrando distintas ontologías.

Apéndice A – Edición de ontologías con Protégé2000

A continuación, se presenta una descripción práctica de los pasos a seguir para editar y mantener una ontología usando Protégé2000.

Como primera medida, se deben definir las clases y una jerarquía de clases. Para ello hay varias estrategias a seguir [32]:

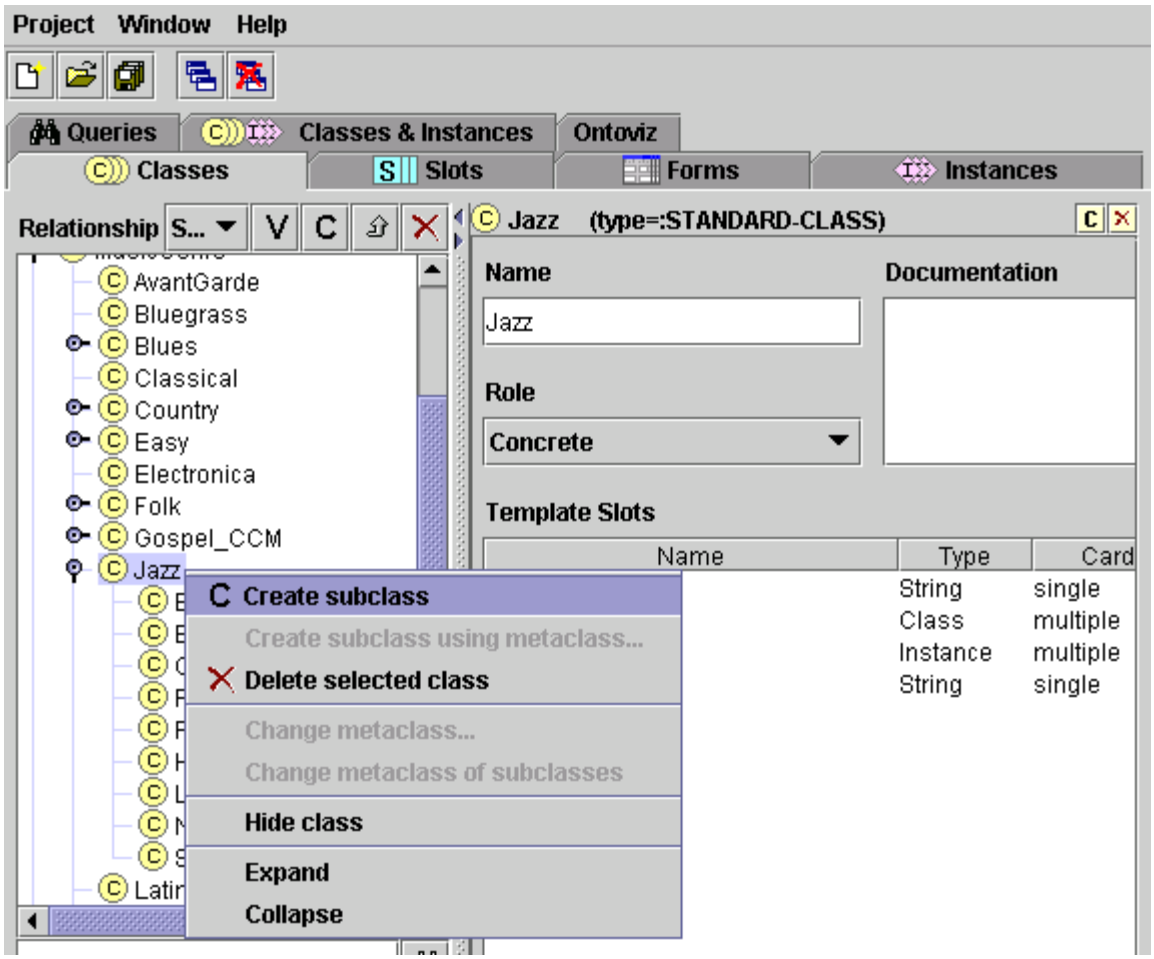
- *top-down*: el desarrollo comienza con la definición de los conceptos más generales del dominio y luego en la especialización de esos conceptos
- *bottom-up*: el desarrollo comienza con la definición de las clases más específicas y luego agrupándolas en conceptos más generales.
- *una combinación de las anteriores*: primero se definen los conceptos salientes y luego se generalizan y/o especializan apropiadamente.



Por ejemplo, una estrategia top-down, podría empezar definiendo la clase *MusicInstrument*, luego *PercussionInstrument*, *StringInstrument*, etc. y terminar la especialización en las clases inferiores, como *BassGuitar* y

Guitarron. Una estrategia bottom-up definiría primero *Guitarron*, *Vihuela*, etc y las agruparía en las clases *LatinAmericanGuitar*, *AcousticGuitar*, etc. hasta llegar a definir la clase superior *MusicInstrument*.

Una vez definida la jerarquía de clases, hay que posicionarse en el lugar deseado de la jerarquía para ir creando las nuevas clases.

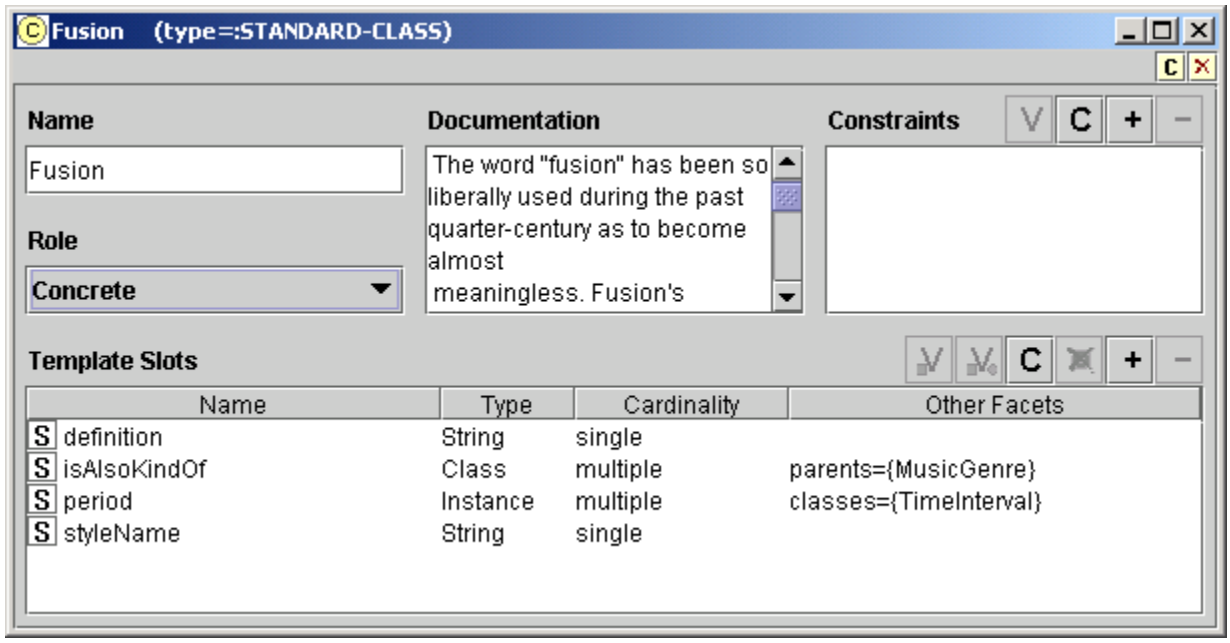


Para crear una clase, hay que ingresar los siguientes elementos:

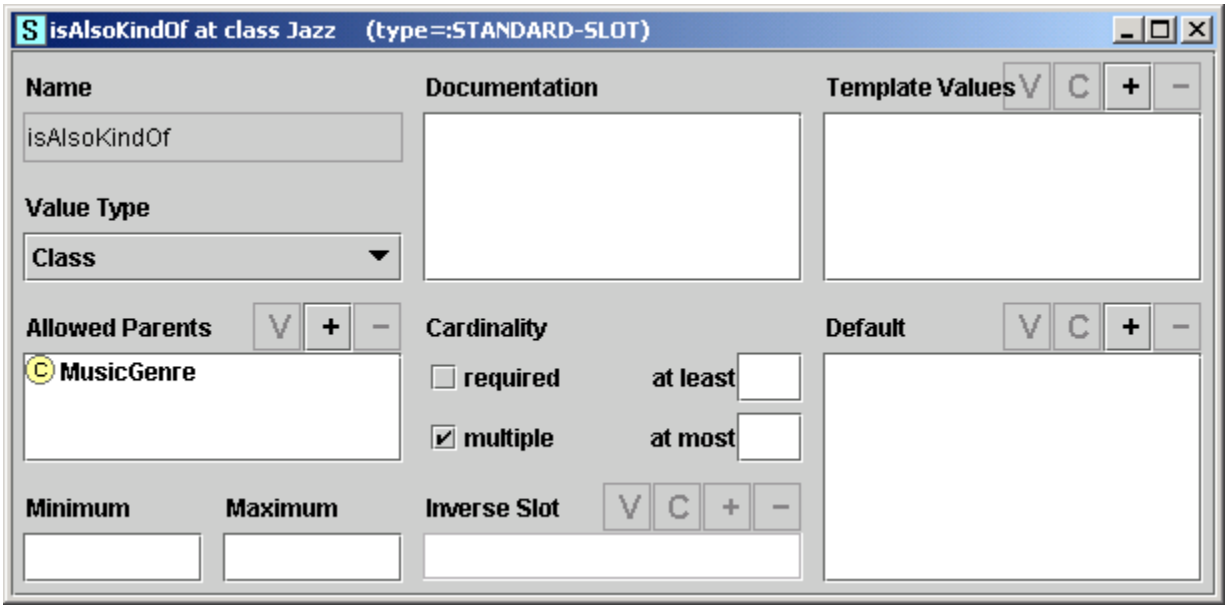
- Nombre (*Name*): nombre de la clase. Se recomienda comenzar con mayúscula y seguir con minúsculas; para separar palabras, usar guión bajo.
- Documentación (*Documentation*): es para ingresar un texto con el fin de describir y documentar la clase en cuestión. El campo es opcional, pero se recomienda utilizarlo siempre con el objetivo de ayudar al mantenimiento de

la ontología.

- Restricciones (*Constraints*):
- Rol (*Role*): el rol de la clase puede ser concreta o abstracta. Las clases concretas pueden tener instancias directas, mientras que las abstractas, no.



- Slots o propiedades (*Template Slots*): representan las propiedades de la clase y juegan un papel fundamentental en la definición de la ontología. Si la clase que se está editando es subclase de otra, va a haber dos tipos de slots: los propios de la clase y los heredados.



Una vez que se definieron las clases, hay que describir la estructura interna de los conceptos (propiedades o slots de las clases). Para definir una propiedad (slot), se deben ingresar los siguientes elementos:

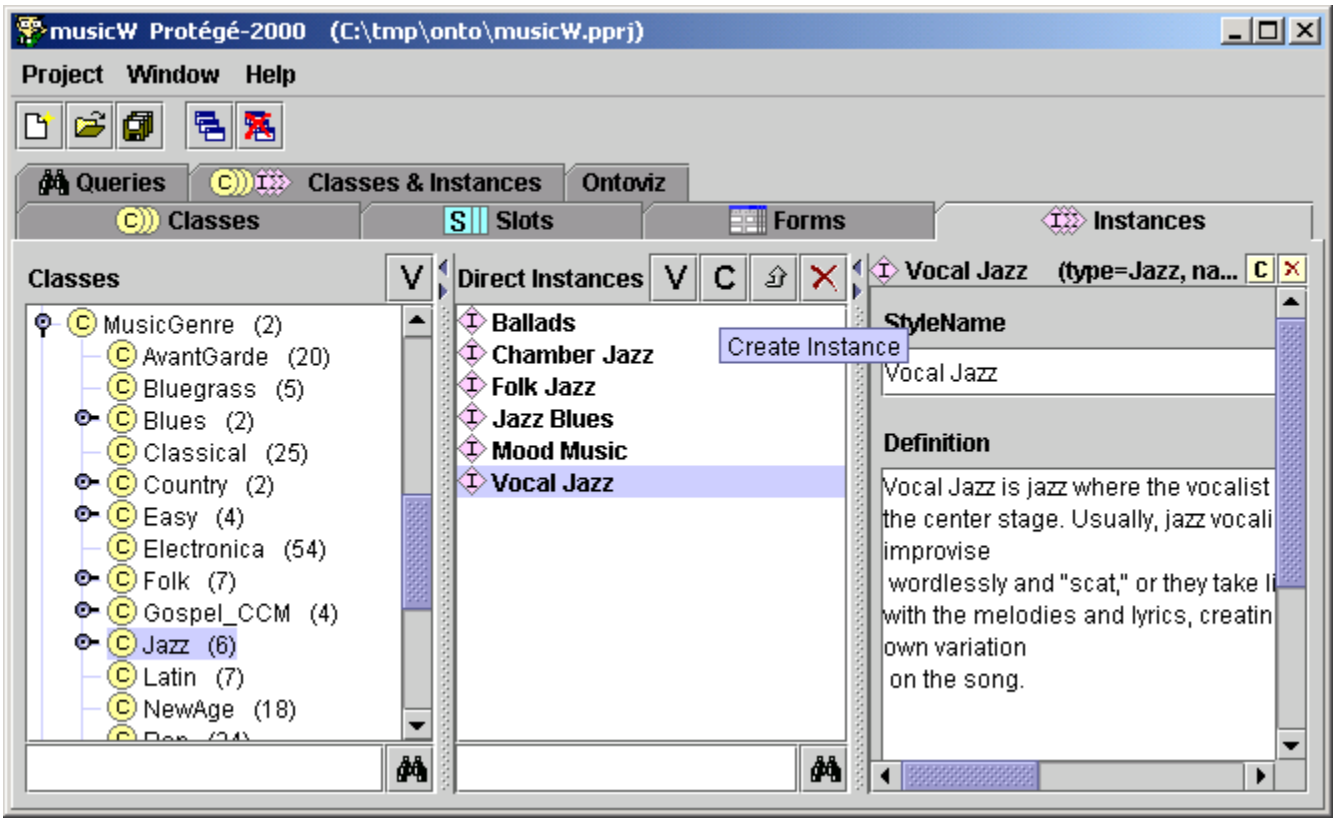
- Nombre (*Name*): nombre del slot.
- Tipo de valor (*Value Type*): el tipo de valor determina los valores que puede asumir un slot. Los tipos disponibles son:
    - *Boolean*: valor booleano
    - *Class*: clase definida en la ontología
    - *Float*: valor real
    - *Instance*: instancia de una clase de la ontología
    - *Integer*: valor entero
    - *String*: cadena de caracteres
    - *Symbol*: lista de valores enumerados
    - *Any*: cualquiera de los tipos anteriores
- Documentación (*Documentation*): es para ingresar un texto con el fin de describir y documentar el slot en cuestión. El campo es opcional, pero se recomienda utilizarlo siempre con el objetivo de ayudar al mantenimiento de la ontología.
- Valores (*Template Values*): permite especificar el o los valores para un slot a

nivel de clase. Este valor se llena en todas las clases y las instancias que usan o heredan el slot. Un *template value* es requerido y no puede ser cambiado o *sobreescrito* a nivel de instancia. Para un valor que pueda ser *sobreescrito*, hay que usar el elemento *Defaults*. El número de *template values* debe respetar la cardinalidad del slot.

- Valores por omisión (*Defaults*): permite especificar los valores por omisión del slot. Cuando se crea una instancia de una clase a la que pertenece el slot, se setea automáticamente el valor por omisión definido. Este valor puede ser cambiado. El número de valores por omisión debe respetar la cardinalidad del slot.
- Cardinalidad (*Cardinality*): permite especificar el número de valores permitidos o requeridos para un slot. Si no se ingresa cardinalidad se permite al slot tener a lo sumo un valor o ninguno.
- Mínimo (*Minumum*) / Máximo (*Maximum*): es para especificar los valores mínimos y máximos que puede tener un slot de tipo *Integer* o *Float*. Este campo es opcional. Cuando se crea una instancia de la clase a la que pertenece este slot, la instancia deberá respetar los valores de mínimo y máximo definidos.
- Slot inverso (*Inverse Slot*): también es opcional y sólo tiene validez cuando el slot es de tipo *Class* o *Instance*. Permite crear una relación recíproca entre dos slots. Si se define correctamente esta relación, asignando un valor (por ejemplo, una clase específica o una instancia) al slot de una instancia, automáticamente asigna la instancia como valor del slot inverso apropiado.

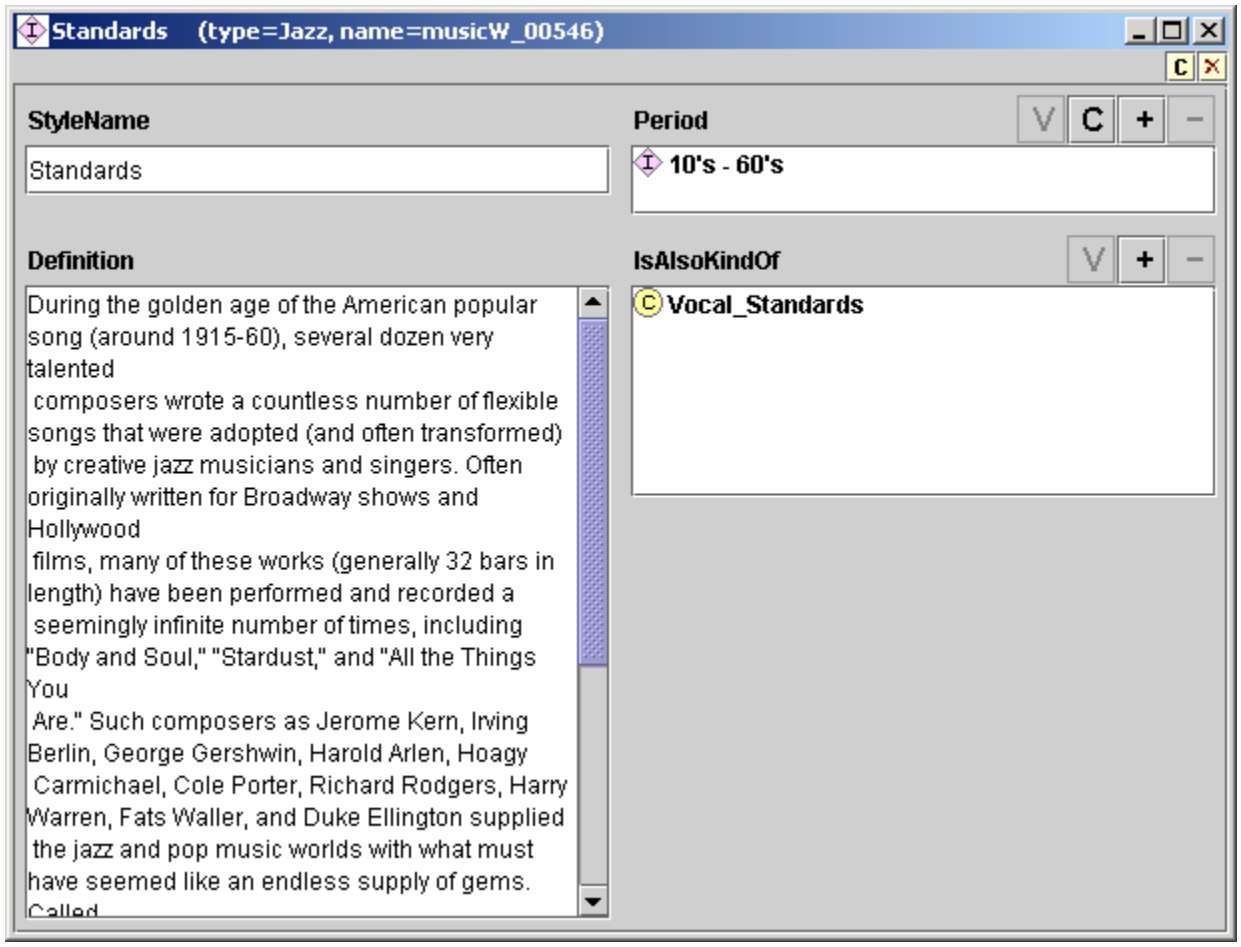
Teniendo definida la jerarquía de clases y la estructura interna de cada clase, se está en condiciones de crear las instancias de cada clase. Para ello, hay que posicionarse en la clase deseada, crear una instancia de la misma e ingresar los valores de los slots.

Por ejemplo, para crear la instancia *Standards*, que pertenece al género musical del Jazz, en la pestaña de Instancias (Instances) hay que posicionarse en la clase *MusicGenre / Jazz* y crear una instancia presionando en el botón *C* del cuadro *Direct Instances*.



Luego de esto, se ingresan los valores de los slots para una instancia de la clase *MusicGenre / Jazz*:

- Nombre de estilo (*StyleName*): Standards
- Definición: (*Definition*): Descripción de los elementos que definen el estilo Standards
- Período (*Period*): El período al que corresponde, es de 1915 a 1969 aproximadamente. Por eso se selecciona la instancia 10's-60's que es la que más se ajusta a ese período.
- También es un tipo de (*IsAlsoKindOf*): Los Standards pueden ser considerados también como standards vocales (clase *MusicGenre / Vocal_Standards)*


Standards (type=Jazz, name=musicW_00546)
StyleName
Standards
Period
10's - 60's
Definition
During the golden age of the American popular song (around 1915-60), several dozen very talented composers wrote a countless number of flexible songs that were adopted (and often transformed) by creative jazz musicians and singers. Often originally written for Broadway shows and Hollywood films, many of these works (generally 32 bars in length) have been performed and recorded a seemingly infinite number of times, including "Body and Soul," "Stardust," and "All the Things You Are." Such composers as Jerome Kern, Irving Berlin, George Gershwin, Harold Arlen, Hoagy Carmichael, Cole Porter, Richard Rodgers, Harry Warren, Fats Waller, and Duke Ellington supplied the jazz and pop music worlds with what must have seemed like an endless supply of gems. Called
IsAlsoKindOf
Vocal_Standards

Bibliografía

[1] – T. R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (1993) 199-220

[2] – W.N. Borst, Construction of Engineering Ontologies, PhD Thesis, University of Tweenty, Enschede, NL – Centre for Telematica and Information Technology, 1997

[3] – R. Studer, V. R. Benjamins, D. Fensel, Knowledge engineering: principles and methods, Data and Knowledge Engineering 25 (1998) 161-197

[4] – O. Corcho, M. Fernandez-Lopez, A. Gomez-Perez, Methodologies, tools and languages for building ontologies. Where is their meeting point ?, Data & Knowledge Engineering (2002)

[5] – N. F. Noy, D. McGuinness, Ontology development 101: A guide to creating your first ontology

[6] – R. Neches, R. E. Fikes, T. Finin, T. R. Gruber, T. Senator, W. R. Swarttout, Enabling technology for knowledge sharing, AI Magazine 12 (3) (1991) 36-56

[7] – M. Uschold, R. Jasper, A Framework for Understanding and Classifying Ontology Applications, Proc. IJCAI99 Workshop on Ontologies and Problem-Solving Methods, Stockholm, 1999

[8] – D. B. Lenat, R. V. Guha, Building large knowledge-based systems. Representation and inference in the Cyc project, Addison-Wesley, Reading, Massachusetts, 1990

[9] – T. Gruber, R. Olsen, An ontology for engineering mathematics, Technical report, Knowledge Systems Laboratory, Stanford University, CA, 1994

[10] – W. Swartout, R. Patil, K. Knight, T. Russ. Toward distributed use of large-scale ontologies. Spring Symposium Series on Ontological Engineering (33-40), 1997

[11] – M. Uschold. Building ontologies: towards a unified methodology. In Expert Systems 96, 1996

[12] – Gruninger, M., and Fox, M.S. (1995), Methodology for the Design and Evaluation of Ontologies, Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal.

[13] – M. Genesereth, R. Fikes, Knowledge interchange format, Technical

Report Logic-92-1, Computer Science Department, Stanford University, 1992

[14] – R. MacGregor, Inside the LOOM clasifier, SIGART bulletin 2 (3) (1991) 70-76

[15] – E. Motta, Reusable Components for Knowledge Modelling, IOS Press, Amsterdam, 1999.

[16] – M. Kifer, G. Lausen, J. Wu, Logical foundations of object-oriented and frame-based languages, Journal of the ACM 42 (4) (1995) 741-843.

[17] – S. Luke, J. Heflin, SHOE 1.01. Proposed Specification, SHOE Project technical report, University of Maryland, 2000. Disponible en <http://www.cs.umd.edu/projects/plus/SHOE/spec.html> (visitado en 03/2004).

[18] – T. Bray, J. Paoli, CM. Sperberg-McQueen, E. Maler, Extensible Markup Language (XML) 1.0, second ed., W3C Recommendation, 2000. Disponible en <http://www.w3.org/TR/REC-xml> (visitado en 09/2003).

[19] – R. Karp, V. Chaudhri, J. Thomere, XOL: An XML-Based Ontology Exchange Language, technical report, 1999. Disponible en <http://www.ai.sri.com/~pkarp/xol/xol.html> (visitado en 09/2003).

[20] – O. Lassila, R. Swick, Resource description framework (RDF) model and syntax specification, W3C Recommendation (1999), Disponible en <http://www.w3.org/TR/REC-rdf-syntax> (visitado en 09/2003).

[21] – I. Horrocks, D. Fensel, F. Harmelen, S. Decker, M. Erdmann, M. Klein, OIL in a Nutshell, in: ECAI'00 Workshop on Application of Ontologies and PSMs, Berlin, 2000.

[22] – I. Horrocks, F. van Harmelen, Reference Description of the DAML + OIL (March 2001) Ontology Markup Language, Technical report, 2001. Disponible en <http://www.daml.org/2001/03/reference.html> (visitado en 09/2003).

[23] – M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, OWL Web Ontology Language 1.0 Reference, W3C Working Draft, 2002. Disponible en <http://www.w3.org/TR/owl-ref/> (visitado en 09/2003).

[24] – V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, J.P. Rice, Open Knowledge Base Connectivity 2.0.3, Technical Report, 1998. Disponible en <http://www.ai.sri.com/~okbc/okbc-2-0-3.pdf> (visitado en 08/2003).

[25] – <http://www.isi.edu/isd/ontosaurus.htm> (visitado en 07/2003)

[26] – <http://kmi.open.ac.uk/projects/webonto> (visitado en 07/2003)

[27] – <http://protege.stanford.edu> (visitado en 07/2003)

[28] – <http://delicias.dia.fi.upm.es/webODE> (visitado en 07/2003)

[29] – <http://kaon.semanticweb.org> (visitado en 07/2003)

[30] – <http://oiled.man.ac.uk> (visitado en 07/2003)

[31] – <http://codip.grci.com/Tools/Tools.html> (visitado en 07/2003)

[32] – M. Uschold, M. Gruninger. Ontologies: Principles, Methods and Applications. Knowledge Engineering Review 11(2), 1996