KEMENTERIAN RISET, TEKNOLOGI PENDIDIKAN TINGGI
UNIVERSITAS UDAYANA
UPT PERPUSTAKAAN
Alamat : Kampus Unud Bukit Jimbaran Badung. Bali - 80364
Telepon (0361) 702772. Fax (0361) 701907
E-mail : perpustakaanudayana@yahoo.co.id Laman : www.e-lib.unud.ac.id

SURAT KETERANGAN

N● :0012/UN.14.I.2.1/Perpus/00.09/2017

Yang bertanda tangan dibawah ini Kepala UPT Perpustakaan Universitas Udayana menerangkan bahwa:

Nama	:Dr.Anak Agung Istri Ngurah Eka Karyawati.S.Si., M.Eng
NIP.	: 19740407 199802 2 001
Fakultas/ Program Studi	: MIPA Teknik Informatika

Memang benar telah menyerahkan 1 eksemplar **Modul** dan 1 keping CD di UPT Perpustakaan Universitas Udayana, dengan judul:

Logika Dan Penalaran, Description Logic, Protégé,Dan Spargl

Demikian surat pernyataan ini dibuat untuk dapat dipergunakan sebagaimana mestinya.

Bukit Jimbaran, 11 Juli 2017
Mengetahui,
Ka. Perpustakaan Universitas Udayana
a.n.
Bag.Pengembangan dan Pengolahan Koleksi

Anak Agung Sri Astuti, S.Sos.
NIP.197802212005012001

KEMENTERIAN RISET, TEKNOLOGI PENDIDIKAN TINGGI
UNIVERSITAS UDAYANA
UPT PERPUSTAKAAN
Alamat : Kampus Unud Bukit Jimbaran Badung, Bali - 80364
Telepon (0361) 702772, Fax (0361) 701907
E-mail : perpustakaanudayana@yahoo.co.id Laman : www.e-lib.unud.ac.id

SURAT PERNYATAAN PUBLIKASI

NO :0012/UN.14.I.2.1/Perpus/00.09/2017

Yang bertanda tangan dibawah ini:

Nama	:Dr.Anak Agung Istri Ngurah Eka Karyawati,S.Si., M.Eng
NIP.	: 19740407 199802 2 001
Fakultas/ Program Studi	: MIPA Teknik Informatika

Menyatakan bersedia menyerahkan hak publikasi kepada UPT Perpustakaan Universitas Udayana. Judul **Modul** yang akan dipublikasikan adalah:

Logika Dan Penalaran, Description Logic, Protégé,Dan Spargl

Demikian surat pernyataan ini dibuat untuk dapat dipergunakan sebagaimana mestinya.

Bukit Jimbaran, 11 Juli 2017
Ka. Perpustakaan Universitas Udayana
a.n.
Bag.Pengembangan dan Pengolahan Koleksi

Yang membuat pernyataan,

(Dr.Anak agung istri ngurah eka karyawati,S.Si.,M.Eng)

Anak Agung Sri Astuti, S.Sos.
NIP.197802212005012001

MODUL 2 SEMANTIC WEB

LOGIKA DAN PENALARAN, DESCRIPTION LOGIC, PROTÉGÉ, DAN SPARQL

Dr. Anak Agung Istri Ngurah Eka Karyawati, S.Si., M.Eng.
NIP. 19740407 199802 2 001

Program Studi Teknik Informatika
Jurusan Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Udayana
2017

Halaman Judul

MODUL 2 SEMANTIC WEB

LOGIKA DAN PENALARAN, DESCRIPTION LOGIC, PROTÉGÉ, AND SPARQL

Dr. Anak Agung Istri Ngurah Eka Karyawati, S.Si., M.Eng.
NIP. 19740407 199802 2 001

Program Studi Teknik Informatika
Jurusan Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Udayana
2017

KATA PENGANTAR

Puji dan syukur saya panjatkan kehadapan Tuhan Yang Maha Esa, Ida Sang Hyang Widhi Wasa, atas anugerahnya sehingga penyusunan Modul 1 mata kuliah Semantic Web ini bisa diselesaikan dengan baik.

Saya megucapkan terimakasih sebesar-besarnya kepada berbagai pihak yang membantu penyusunan Modul 2 Semantic Web ini.

Modul ini diperuntukkan untuk memperlancar proses pembelajaran mata kuliah Semantic Web yang diselenggarakan oleh Program Studi Teknik Informatika, FMIPA, UNUD. Materi pembelajaran mata kuliah Semantic Web terbagi menjadi 3 modul utama yaitu: modul 1 yang akan menyajikan empat pokok bahasan (konsep dasar dan teknologi yang mendasari semantic web; dasar-dasar XML untuk merepresetasikan RDF dokumen; RDF dan RDF Schema; dan OWL Language), modul 2 yang akan menyajikan tiga pokok bahasan (Logika dan penalaran; Protégé; Pengenalan SPARQL), dan modul 3 yang akan menyajikan tiga pokok bahasan (aplikasi-aplikasi semantic web; pemrograman dengan Jena; dan pengintegrasian informasi dari sumber data lain).

Semoga modul 2 Semantic Web ini bisa memberi manfaat yang besar bagi terselenggaranya proses pembelajaran, khususnya mata kuliah Semantic Web, dengan lebih baik.

Kritik dan saran yang membangun sangat saya harapkan agar Modul 2 Semantic Web ini bisa menjadi lebih baik ke depannya nanti.

Jimbaran, Juli 2017

Penulis

TINJAUAN MATA KULIAH

PRASYARAT MATA KULIAH

Web Programming

DESKRIPSI MATA KULIAH

Mata kuliah semantic web membahas tentang konsep dan aplikasi dari teknologi semantic web. Ini meliputi konsep RDF, RDFS, dan OWL, teknologi dari semantic web (yaitu, metadata, ontologi, dan bahasa query), pembangunan aplikasi menggunakan Protégé.

SASARAN BELAJAR

- Mahasiswa dapat menjelaskan konsep dasar dan teknologi yang mendasari semantic web
- Mahasiswa mampu memahami dan menjelaskan dasar-dasar XML sebagai salah satu bentuk bentuk representasi untuk RDF dan RDF schema
- Mahasiswa mampu dan memahami elemen-elemen dasar OWL dan bagaimana mengimplementasikannya
- Mahasiswa mampu memahami konsep dasar logika dan penalaran dan mesin penalaran
- Mahasiswa mengetahui tentang bahasa SPARQL dan mampu menggunakannya untuk meng-query web semantik.
- Mahasiswa mampu menerapkan konsep-konsep dasar semantic web menggunakan framework pendukungnya seperti protégé
- Mahasiswa mengetahui beberapa contoh aplikasi semantic web dan mampu memunculkan ide baru untuk aplikasi ini.
- Mahasiswa mampu mengembangkan aplikasi semantic web sederhana dengan menggunakan framework Jena
- Mahasiswa mampu mengembangkan aplikasi semantic web sederhana dengan menggabungkan informasi dari berbagai sumber

POKOK BAHASAN

1. Konsep dasar dan teknologi yang mendasari semantic web
2. Dasar-dasar XML
3. RDF dan RDF Schema
4. OWL Language, elemen-elemen dasar (Class, property, data type), dan implementasi OWL
5. Logika dan penalaran, Description Logic, dan mesin penalaran
6. Protégé
7. Pengenalan SPARQL dan dasar-dasar SPARQL
8. Aplikasi-aplikasi semantic web, semantic web services, dan semantic search
9. Pengenalan Jena dan pemrograman dengan Jena
10. Pengintegrasian informasi, mengekspos xml based web-service sebagai RDF, mengekspos data relasional sebagai RDF, dan mengekspos sumber data lainnya

URUTAN PENYAJIAN

Materi mata kuliah semantic web akan terbagi ke dalam 3 modul yaitu: modul 1 yang membahas tentang pengantar semantic web; modul 2 yang membahas tentang logika, penalaran, SPARQL, dan Protégé; dan modul 3 yang membahas aplikasi semantic web, Jena dan integrasi sumber informasi dengan memanfaatkan teknologi semantic web.

Modul 1 akan menyajikan empat pokok bahasan yang meliputi: 1) Konsep dasar dan teknologi yang mendasari semantic web; 2) Dasar-dasar XML untuk merepresetasikan RDF dokumen; 3) RDF dan RDF Schema; dan 4) OWL Language, elemen-elemen dasar (Class, property, data type), dan implementasi OWL. Modul 1 akan disajikan dalam 4 kali pertemuan kuliah tatap muka.

Selanjutnya, modul 2 akan menyajikan 3 pokok bahasan yang meliputi: 1) Logika dan penalaran, dan Description Logic; 2) Protégé dan 3) Pengenalan SPARQL dan dasar-dasar SPARQL. Seperti juga modul 1, modul 2 akan disajikan dalam 4 kali pertemuan kuliah tatap muka.

Modul 3 juga akan menyajikan 3 pokok bahasan yang meliputi: 1) Aplikasi-aplikasi semantic web, semantic web services, dan semantic search; 2) Pengenalan Jena dan pemrograman dengan Jena; dan 3) Menggabungkan informasi, mengekspos xml based web-service sebagai RDF, mengekspos data relasional sebagai RDF, dan mengekspos sumber data lainnya. Modul 3 akan disajikan dalam 5 kali pertemuan kuliah tatap muka.

DAFTAR PUSTAKA

[1] Grigoris Antoniou, Frank Van Harmelen, A Semantic Web Primer, Second Edition, MIT Press, 2008.

[2] Bob DuCharme, Learning SPARQL, Second Edition, O'Reilly Media, 2013

[3] Matthew Horridge, A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, Edition 1.3, The University Of Manchester, 2011

[4] Baader F., Calvanese D., McGuinness, D., Nardi, D. and Patel-Schneider P., 2007, *The Description Logic Handbook: Theory, Implementation and Applications*, Second Edition, Cambridge University Press, Cambridge.

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

I. PENDAHULUAN

1.1. Sasaran Pembelajaran yang Ingin Dicapai

Setelah mempelajari modul 2 Semantic Web ini, mahasiswa diharapkan dapat memahami dan menjelaskan konsep dasar logika dan penalaran, dan mesin penalaran, dapat mengetahui tentang bahasa SPARQL dan mampu menggunakannya untuk meng-query web semantik, dan dapat menerapkan konsep-konsep dasar semantic web menggunakan framework pendukungnya seperti protégé.

1.2. Ruang Lingkup Bahan Modul

Ruang lingkup bahan modul meliputi tiga pokok bahasan utama yaitu: Logika dan penalaran, Description Logic; Protégé, dan Pengenalan SPARQL dan dasar-dasar SPARQL.

1.3. Urutan Pembahasan

Modul 1 akan disajikan dalam 4 kali pertemuan kuliah tatap muka. Pertemuan pertama membahas pengertian logika dan penalaran, semantik RDF/RDFS dan entailment, dan penalaran pada ontologi.

Pertemuan kedua membahas tentang description logic, sintaks dan semantik description logic, dan keluarga description logic.

Pertemuan ketiga membahas tentang Protégé, ontologi, membangun ontologi OWL: membuat kelas, properti objek, properti datatype, properti invers, domain dan range, dan properti restriction.

Pertemuan keempat membahas tentang SPARQL, struktur basic query SPARQL, tipe query SPARQL, dan pola grafik SPARQL.

II. PERTEMUAN I : LOGIKA DAN PENALARAN

2.1. Pengertian Logika dan Penalaran

Logika adalah suatu cabang filsafat yang membahas tentang aturan-aturan, asas-asas, hukum-hukum dan metode atau prosedur dalam mencapai pengetahuan secara rasional dan benar, juga merupakan suatu cara untuk mendapatkan suatu pengetahuan dengan menggunakan akal pikiran, kata dan bahasa yang dilakukan secara sistematis. Penalaran merupakan suatu proses berpikir dalam menarik sesuatu kesimpulan yang berupa pengetahuan.

Pada bidang ilmu komputer (cabang ilmu kecerdasan buatan), penalaran diperlukan bila sebuah program harus menentukan beberapa informasi atau beberapa tindakan yang belum secara eksplisit diceritakan. Penalaran dilakukan dengan mencari tahu apa yang perlu diketahui dari apa yang sudah diketahui.

Contoh dari proses penalaran adalah sebagai berikut:

Fakta-fakta yang ada:

1. Robin adalah burung.
2. Semua burung punya sayap.

Pertanyaan yang ingin dijawab:

1. Apakah robin adalah burung?
2. Apakah semua burung punya sayap?
3. Apakah robin punya sayap?

Pertanyaan pertama dan kedua bisa dijawab berdasarkanfakta-fakta yang ada. Jadi jawaban yang pertama adalah: ‘YA’ (berdasarkan fakta pertama). Jawaban pertanyaan kedua adalah: ‘YA’ (berdasarkan fakta kedua). Sedangkan jawaban pertanyaan ketiga tidak bisa dijawab hanya berdasarkan fakta/informasi yang tersedia. Disini diperlukan penalaran.

2.1.1. Masalah pada Semantic Web

Knowledge based system (KBS) tradisional dulu dikembangkan dari atas ke bawah termasuk Web Semantic juga dikembangkan dari atas ke bawah. Ada beberapa masalah yang muncul antara lain:

- Skalabilitas: Semua orang yang memiliki halaman web adalah Knowledge Engineer yang potensial

- Distribusi: Potongan pengetahuan bisa berada di semua tempat, sehingga bisa menjadi sulit
- Heterogenitas: Orang menggunakan cara yang berbeda untuk menggambarkan informasi yang sama
- Kualitas: Pengetahuan tidak lengkap dan bahkan tidak konsisten di berbagai sumber pemetaan

2.1.2. Penalaran di Semantic Web

Metode dalam penalaran yang dibutuhkan adalah yang bisa beradaptasi ke sejumlah besar data dan pengetahuan. Metode penalaran harus bisa dikembangkan untuk menangani jumlah data dan pengetahuan yang besar. Selain itu, metode penalaran juga harus bisa toleran terhadap kesalahan, ketidaklengkapan, dan ketidakkonsistenan antara dan di dalam resource.

Selanjutnya, metode penalaran harus bisa bekerja pada representasi terdistribusi dan idealnya didistribusikan sendiri dan yang paling penting bisa kompatibel dengan standar web Semantic.

Metode penalaran yang diharapkan tergantung tingkat semantik representasi (RDFS, OWL), yang menyiratkan formalisasi matematika tertentu untuk basis pengetahuan, ukuran basis pengetahuan yang terlibat Kehadiran (atau tidak adanya) kejadian, dan kemampuan alat penalaran

Seringkali pengertian dari penalaran (reasoning) dan inferensi kurang diketahui. Reasoning seperti yang sudah dijelaskan adalah suatu tindakan atau proses untuk menentukan beberapa informasi atau beberapa tindakan yang belum secara eksplisit diceritakan. Sedangkan inferensi adalah suatu tindakan atau proses membangun ungkapan baru dari ungkapan yang ada, atau hasil dari tindakan atau proses semacam itu. Dalam RDF, kesimpulan yang sesuai dengan entailments dijelaskan sebagai benar atau valid.

Aturan inferensi adalah deskripsi formal dari inferensi. Sistem inferensi adalah sistem pengaturan inferensi yang terorganisir. Sistem inferensi juga adalah perangkat lunak yang menghasilkan kesimpulan atau memeriksa kesimpulan untuk validitas.

2.2. Semantik RDF / RDFS dan Entailment

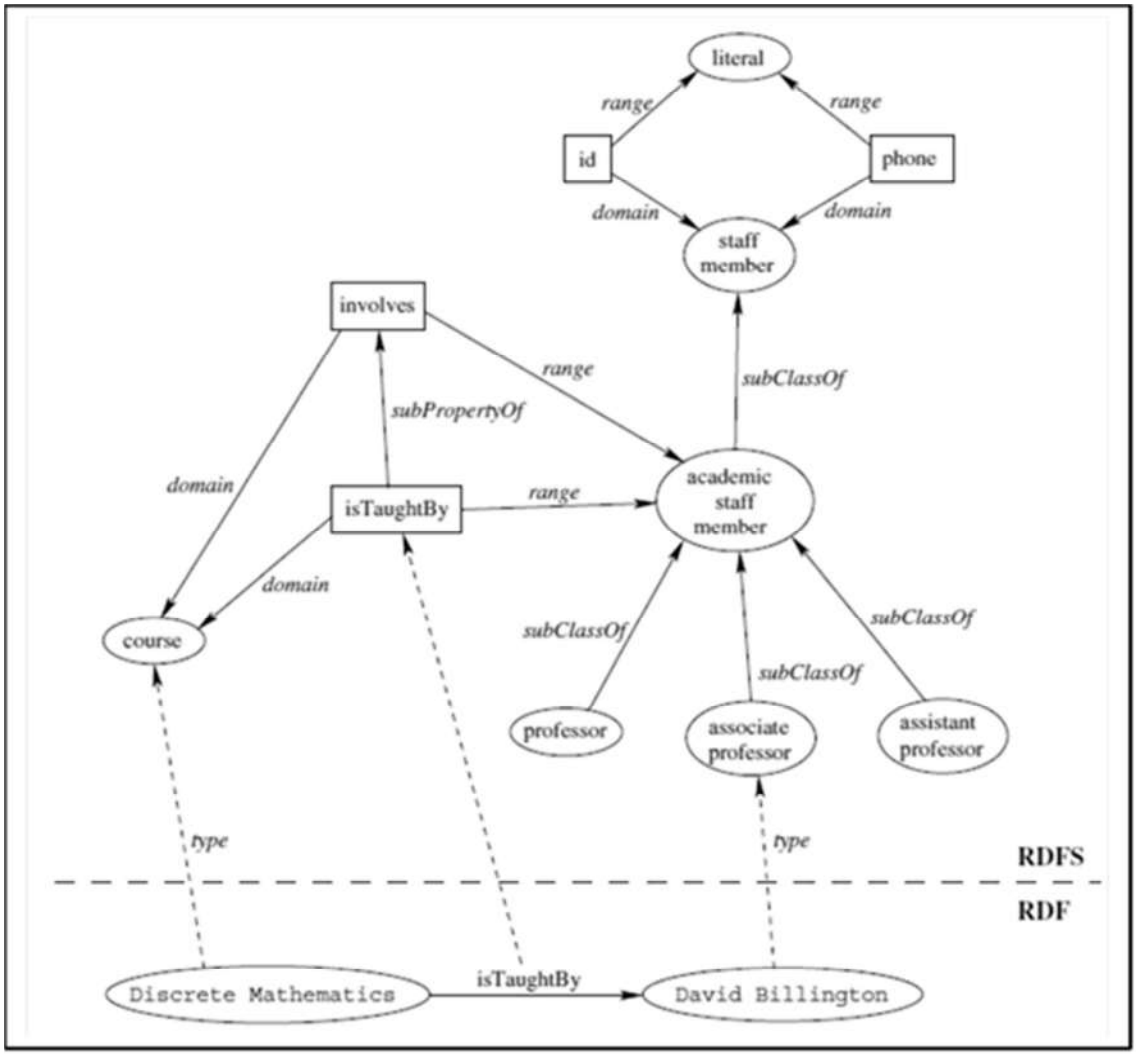
RDF adalah bahasa universal yang memungkinkan pengguna mendeskripsikan resource dengan menggunakan kosakata mereka sendiri. RDF tidak membuat asumsi tentang domain aplikasi tertentu, juga tidak mendefinisikan semantik dari domain apa pun. Sedangkan skema RDF (RDFS) menyediakan model primitif untuk mengekspresikan informasi. Pemodelan primitif dari skema RDF didefinisikan dengan menggunakan resource dan properti.

Berbagai layer yang terlibat dalam RDF dan RDFS digambarkan dengan menggunakan contoh sederhana. Perhatikan statemen RDF “Discere Mathematics is taught by David Billington”. Skema untuk pernyataan ini berisi kelas seperti lecturers, academic staff members, staff members, first-year courses, and properti seperti, is taught by, involves, phone, employee ID. Gambar 2.1 mengilustrasikan layer RDF dan skema RDF untuk contoh ini. Dalam gambar ini, persegi panjang adalah properti, elips di atas garis putus-putus adalah kelas, dan elips di bawah garis putus-putus adalah contoh.

Skema pada Gambar 2.1 dapat ditulis dalam bahasa formal (RDFS) yang dapat mengekspresikan komponen-komponennya: subClassOf, Class, Property, subPropertyOf, Resource, dan sebagainya.

Beberapa contoh dari aturan inferensi dalam RDFS adalah:

1. (X R Y), (R subPropertyOf Q) => (X Q Y)
2. (X R Y), (R domain C) => (X type C)
3. (X type C), (C subClassOf D) => (X type D)



Gambar 2.1 RDF dan RDFS layers

2.2.1. Semantik dari RDF

Semantik bukan tentang "makna" dari sebuah pernyataan. 'Arti' sebuah pernyataan dalam RDF atau RDFS dapat bergantung pada banyak faktor, termasuk konvensi sosial, komentar dalam bahasa alami atau tautan ke dokumen konten lainnya. Secara umum adalah tergantung pada informasi yang tidak dapat diproses oleh mesin.

Semantik membatasi dirinya pada gagasan formal tentang makna yang dapat dicirikan sebagai bagian yang umum bagi semua catatan makna lainnya, dan dapat ditangkap dalam peraturan inferensi mekanis.

2.2.2. Model semantik

Rekomendasi RDF Semantics W3C menggunakan teori model untuk menentukan semantik bahasa RDF. Teori model mengasumsikan bahwa bahasa mengacu pada 'dunia', dan menggambarkan kondisi minimal yang harus dipenuhi untuk menetapkan makna yang sesuai untuk setiap ekspresi dalam bahasa tertentu atau disebut dengan penafsiran. Idenya adalah untuk menyediakan sebuah catatan matematis abstrak tentang sifat-sifat yang harus dimiliki oleh interpretasi semacam itu, dengan membuat asumsi sesedikit mungkin tentang sifat sebenarnya atau struktur intrinsiknya, sehingga mempertahankan generalitas sebanyak mungkin.

2.2.3. Tujuan proses inferensi

Untuk memberikan cara teknis untuk menentukan kapan proses inferensi valid, yaitu, ketika mereka mempertahankan kebenaran. Mulai dari seperangkat pernyataan yang dianggap benar dalam model RDF, dapatkan apakah model RDF baru berisi pernyataan benar yang valid. Hal ini didasarkan pada kenyataan bahwa kebenaran "nyata" dari pernyataan apapun di dunia "sebenarnya" tidak pernah diketahui.

2.2.4. Definisi iterpretasi vs entailment

Interpretasi (Normatif) adalah pemetaan pernyataan RDF menjadi model abstrak, berdasarkan teori himpunan. Dengan "operator interpretasi" I(), memetakan grafik RDF menjadi rangkaian himpunan kardinalitas yang sangat abstrak. Model dari iterpretasi sangat teoritis, berguna untuk membuktikan sifat matematis.

Entailments (Informatif) adalah aturan transformasi untuk mendapatkan pernyataan baru dari yang sudah ada. Entailment bisa terbukti lengkap dan konsisten dengan menggunakan interpretasi formal.

2.2.5. Entailment

Secara formal dikatakan bahwa interpretasi *I* memenuhi *E* jika I(E) = benar. Satu himpunan *S* dari grafik RDF entail grafik *E* jika setiap interpretasi yang memenuhi setiap anggota *S* juga memenuhi *E*. Dalam bahasa alami manusia bisa dikatakan bahwa assertion adalah klaim bahwa dunia adalah interpretasi yang memberikan nilai yang sesuai dengan pernyataan. Jika *A* mengandung *B*, maka setiap interpretasi yang membuat *A* benar juga

membuat *B* benar. Pernyataan *A* sudah mengandung "makna" yang sama seperti pernyataan *B*. Dan makna *B* di dalamnya, atau diikutsertakan, dalam A.

2.3. Penalaran pada Ontologi

2.3.1. Pengertian Ontologi

Ontologi adalah teori konten tentang jenis objek, sifat objek, dan hubungan antara objek yang mungkin ada dalam domain pengetahuan tertentu. Ontologies memberikan istilah/term yang potensial untuk menggambarkan pengetahuan tentang domain tertentu.

Dalam bidang kecerdasan buatan ontologi adalah kosa kata representasi untuk beberapa domain spesifik. Namun, kosa kata yang memenuhi syarat sebagai ontologi adalah konseptualisasi bahwa istilah dalam kosa kata dimaksudkan untuk dipahami. Mengidentifikasi kosakata semacam itu - dan konseptualisasi yang mendasarinya - umumnya memerlukan analisis yang mendalam terhadap jenis objek dan relasi yang dapat ada dalam domain.

Selain itu, istilah ontologi terkadang digunakan untuk merujuk pada kumpulan pengetahuan yang menjelaskan beberapa domain, biasanya domain pengetahuan yang masuk akal, dengan menggunakan kosa kata representasi. Kosa kata representasi menyediakan satu set istilah untuk menggambarkan fakta di beberapa domain, sedangkan kumpulan pengetahuan yang menggunakan kosa kata tersebut adalah kumpulan fakta tentang sebuah domain.

Idealnya, sebuah ontologi seharusnya bisa menangkap sebuah pemahaman bersama tentang domain tertentu dan bisa memberikan model domain yang bisa dimanipulasi mesin/tool (yaitu reasoner/inference engine).

Analisis ontologis memperjelas struktur pengetahuan. Ontologi dari spesifik domain membentuk inti dari setiap sistem representasi pengetahuan untuk domain tersebut. Tanpa ontologi, atau konseptualisasi yang mendasari pengetahuan, tidak ada kosa kata untuk mewakili pengetahuan. Dengan demikian, langkah pertama dalam merancang sistem representasi pengetahuan yang efektif, dan kosa kata, adalah melakukan analisis ontologis yang efektif terhadap domain. Analisis yang lemah menghasilkan basis pengetahuan yang tidak koheren.

2.3.2. Mengapa Penalaran Diperlukan pada Ontologi

Peran penting ontologi dalam banyak aplikasi, mendorong tersedianya tool dan service untuk membantu membantu user menangani beberapa hal antara lain:

1. Mendesain dan mempertahankan ontologi yang berkualitas tinggi. Ontologi yang baik memiliki karakteristik sebagai berikut:
   - **Meaningful.** Semua kelas (named classes) dari ontologi memiliki instance/individu.
   - **Correct.** Ontologi mampu menangkap intuisi/pemikiran dari ahli (expert) pada domain tertentu.
   - **Minimally redundant**. Ontologi tidak/ sesedikit mungkin mempunyai sinonim yang tidak penting.
   - **Richly axiomatised**. Ontologi mempunyai deskripsi pengetahuan dalam bentuk aksioma/assertion cukup (sufficiently) lengkap.
2. Menjawab query tentang kelas ontology dan instance. Contohnya:
   - Menemukan kelas yang lebih general/spesifik.
   - Menemukan individu/tuple yang sesuai/cocok dengan query yang diberikan.
3. Mengintegrasikan dan menyelaraskan beberapa ontologi.

Semua proses-proses itu memerlukan metode penalaran yang efektif.

2.4. Bagaimana Melakukan Penalaran pada Ontologi

Semantic reasoner mesin penalaran, mesin inferensi atau reasoner, adalah perangkat lunak yang dapat menyimpulkan konsekuensi logis dari sekumpulan fakta atau aksioma. Reasoner menggeneralisasi mesin inferensi, dengan menyediakan seperangkat mekanisme yang lebih kaya untuk digunakan. Aturan inferensi (inference rules) biasanya ditentukan dengan menggunakan bahasa ontologi, dan seringkali merupakan bahasa description logic.

Reasoner seharusnya memainkan peran penting dalam mengembangkan dan menggunakan ontologi yang ditulis dalam OWL. Reasoner otomatis seperti Pelet, FaCT++, HerMiT, ELK dan seterusnya mengambil kumpulan aksioma yang ditulis dalam OWL dan menawarkan serangkaian operasi pada aksioma ontologi. Operasi yang paling sering digunakan adalah kesimpulan hirarki subsumption untuk Kelas yang dijelaskan dalam ontologi.

2.5. Contoh

Contoh 1:

- Driver is a person that drives a vehicle.
- A bus driver is a person that drives a bus.
- A bus is a vehicle.

Class Inference:

A bus driver drives a vehicle, so **a bus driver is a driver**.

Contoh 2:

- Cat owners have cats as pets.
- has pet is a subproperty of likes, so anything that has a pet must like that pet.

Class Inference:

- Cat owners must like a cat.

Contoh 3:

- Sheep only eat grass.
- Grass is a plant.
- Plants and parts of plants are disjoint from animals and parts of animals.
- Vegetarians only eat things which are not animals or parts of animals.

Class Inference:

- Sheep are Vegetarians.

2.6. Latihan

Why RDF model is different from the XML model?

2.7. Rangkuman

Analisis ontologis yang efektif terhadap domain diperlukan untuk bisa merancang sistem representasi pengetahuan yang efektif. Sebuah ontologi seharusnya bisa menangkap sebuah pemahaman bersama tentang domain tertentu dan bisa memberikan model domain yang bisa dimanipulasi mesin/tool (yaitu reasoner/inference engine).

Daftar Pustaka

[1] Grigoris Antoniou, Frank Van Harmelen, A Semantic Web Primer, Second Edition, MIT Press, 2008.
[2] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins, 1999, *What Are Ontologies, and Why Do We Need Them?* IEEE INTELLIGENT SYSTEMS, pp. 20-16.

III. PERTEMUAN II: DESCRIPTION LOGIC

3.1. Mengapa Perlu Description Logic

Description logic (DL) adalah salah satu keluarga bahasa representasi pengetahuan yang dapat digunakan untuk mewakili definisi konsep domain aplikasi (dikenal sebagai pengetahuan terminologi) dalam cara yang terstruktur dan formal dipahami dengan baik.

Representasi dari informasi berguna dalam banyak kegiatan. Di setiap domain, objek secara alami masuk dalam sebuah kategori dan sering masuk ke dalam beberapa kategori. Kategori yang berbeda bisa saling terkait satu sama lain. Misalnya, beberapa kategori bisa lebih umum (spesifik) daripada yang lain. Oject bisa disusun atau mengandung bagian, yaitu objek lain. Selain itu hubungan antar objek dan kategori sangat penting untuk bisa memahami informasi secara mendalam.

Untuk merepresentasikan informasi atau pengetahuan, maka komponen-komponen informasi atau pengetahuan itu harus bisa direpresentasikan dengan baik. Komponen tersebut antara lain:

- Objek yang juga dikenal sebagai individu (yaitu objek dari domain), misalnya: Andrea, Semantic Web, Rolex, etc.
- Kategori juga dikenal sebagai konsep (yaitu menggambarkan kelas kategori dasar), contohnya: pemain sepak bola, anjing, dan mata kuliah.
- Relasi juga dikenal sebagai property atau dikenal juga sebagai role (yaitu menggambarkan objek yang merupakan bagian atau atribut atau properti objek lainnya), contohnya: isPlayedFor, age, dan isHeldBy

Sehingga ada beberapa hal yang perlu diperhatikan antara lain:

- Bagaimana mendefinisikan beberapa hubungan generalisasi?
- Bagaimana merepresentasikan konsep kompleks sebagai hasil beberapa komposisi konsep yang lebih sederhana?
- Apakah satu individu termasuk dalam kategori tertentu atau tidak?

Sehingga memunculkan keinginan untuk menentukan bahasa yang memungkinkan untuk merepresentasikan semua informasi ini dan sekaligus juga untuk secara otomatis menyimpulkan hierarki generalisasi sebagai konsekuensi dari deskripsi yang telah dibuat tentang konsep.

3.2. Description Logic

Description logic adalah keluarga logika. Setiap logika berbeda tergantung operator mana yang masuk dalam logika. Tentu saja, lebih banyak operator berarti:

- Ekspresivitas lebih tinggi
- Biaya komputasi yang lebih tinggi
- Logika itu mungkin bisa diputuskan
- ...

Decidability adalah sebuah teori dapat dipecahkan jika ada metode yang efektif untuk menentukan apakah sembarang formula termasuk dalam teori. Dengan kata lain, DL adalah decidable jika, dengan menggunakan KB generik, ada sebuah metode atau algoritma yang (dengan diberikan suatu aksioma subsumption generik untuk dibuktikan) dapat menentukan apakah aksioma tersebut berlaku atau tidak.

DL adalah fragmen dari first order logic (FOL) klasik, yang memiliki empat elemen dasar yang melibatkan, nama konsep yang setara dengan predikat tak tertulis, nama role yang setara dengan predikat biner, nama individu yang setara dengan konstanta, dan konstruktor (yaitu, untuk konsep dan role).

3.3. Sintaks Description Logic

Sintaks dari description logic adalah sebagai berikut:

- Nama konsep (kelas), misalnya Cat, Animal, Doctor, adalah setara dengan predikat FOL unary.
- Nama role (properti), misalnya, sitsDown, hasParent, like, adalah setara dengan predikat FOL biner
- Nama individu, misalnya, Felix, John, Mary, Boston, Italia, adalah setara dengan konstanta FOL

Operator yang digunakan pada description logic banyak jenis tersedia, misalnya,

- Operator FOL Boolean standar (u, t, ¬)
- Bentuk terbatas dari quantifiers (9, 8)
- Menghitung (¸, ·, =), etc.

Contoh ekspresi konsep:

- Doctor ⊔ Lawyer
- Rich ⊓ Happy
- Cat ⊓ ∃sits-on.Mat

Setara dengan formula FOL dengan satu variabel bebas:

- $\mathrm{Doctor}(x) \vee \mathrm{Lawyer}(x)$
- $\mathrm{Rich}(x) \wedge \mathrm{Happy}(x)$
- $\exists y.(\mathrm{Cat}(x) \wedge \mathrm{sits\text{-}on}(x, y))$

Konsep khusus:

- ⊤ (aka top, Thing, most general concept)
- ⊥ (aka bottom, Nothing, inconsistent concept)

Digunakan sebagai kependekan dari:

- (A ⊔ ¬ A) for any concept A
- (A ⊓ ¬ A) for any concept A

Contoh ekspresi role:

- loves
- hasParent
- hasBrother

Setara dengan formula FOL dengan dua peubah bebas:

- $\mathrm{loves}(y, x)$
- $\exists z.(\mathrm{hasParent}(x, z) \wedge \mathrm{hasBrother}(z, y))$

Contoh ekspresi untuk skema aksioma adalah sebagai berikut:

- "Schema" Axioms, e.g.,
  - Rich ⊑ ¬Poor (concept inclusion)
  - Cat ⊓ ∃sits-on.Mat ⊑ Happy (concept inclusion)
  - BlackCat ≡ Cat ⊓ ∃hasColour.Black (concept equivalence)
  - sits-on ⊑ touches (role inclusion)
  - Trans(part-of) (transitivity)
- Equivalent to (particular form of) FOL sentence, e.g.,
  - ∀x.(Rich(x) → ¬Poor(x))
  - ∀x.(Cat(x) ∧ ∃y.(sits-on(x,y) ∧ Mat(y)) → Happy(x))
  - ∀x.(BlackCat(x) ↔ (Cat(x) ∧ ∃y.(hasColour(x,y) ∧ Black(y)))
  - ∀x,y.(sits-on(x,y) → touches(x,y))
  - ∀x,y,z.((sits-on(x,y) ∧ sits-on(y,z)) → sits-on(x,z))

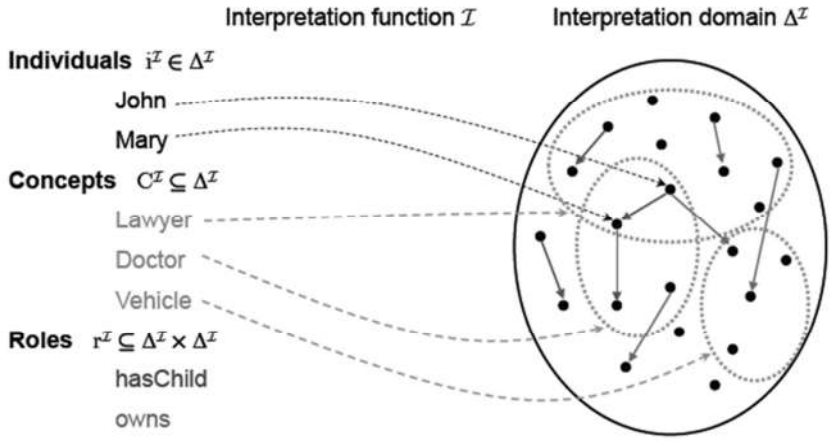
3.4. Semantik dari description logic

Sematik dari description logic adalah sebuah interpretasi I yaitu pasangan (Δ^I, $.I$), di mana Δ^I adalah domain dan I adalah fungsi yang memetakan,

- Konsep (kelas) nama A → subset A^I dari Δ^I
- Peran (properti) nama R → hubungan biner R^I di atas Δ^I
- Nama individu a → elemen I dari Δ^I

Aksioma terminologis inklusi (mengacu pada relasi subkelas) memiliki bentuk $C \sqsubseteq D$ ($R \sqsubseteq S$), dan aksioma persamaan memiliki bentuk $C \equiv D$ ($R \equiv S$), di mana C, D adalah konsep (dan R, S adalah peran). Sebuah interpretasi I memenuhi formula $C \sqsubseteq D$ jika $C^I \subseteq D^I$.

Gambar 3.1 adalah ilustrasi dari sematik atau interpretasi dari individu, konsep, dan role dalam description logic.



Gambar 3.1 Semantik dari Description Logic

Fungsi interpretasi diperluas ke arah ekspresi konsep sebagai contoh:

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$\{x\}^{\mathcal{I}} = \{x^{\mathcal{I}}\}$$
$$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$
$$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.(x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$$
$$(\leqslant nR)^{\mathcal{I}} = \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \leqslant n\}$$
$$(\geqslant nR)^{\mathcal{I}} = \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \geqslant n\}$$

3.5. Keluarga Description Logic

Banyak DL yang berbeda sering dengan nama yang aneh, misalnya: $\mathcal{EL}$, $\mathcal{ALC}$, $\mathcal{SHIQ}$. DL tertentu didefinisikan oleh:

- Operator konsep (contohnya: ⊓,⊔, ¬, ∃, ∀)
- Operator role (¯, o, etc.)
- Aksioma konsep (⊑, ≡, etc.)
- Aksioma role (⊑, Trans, etc.)

Pembentukan konsep kompleks

Setiap konsep atomic adalah sebuah konsep. Contohnya: SoccerPlayer, UniversityCourse, Lecturer, etc. Untuk mendefinisikan konsep yang kompleks ada beberapa hal harus diperhatikan antara lain:

- Jika C adalah sebuah konsep, maka ¬C adalah sebuah konsep.
- If C1, C2, …, Cn adalah konsep, maka (:and C1, C2, …, Cn) and (:or C1, C2, …, Cn) adalah konsep.
- Jika R adalah sebuah role dan C adalah sebuah konsep, maka (:some R, C) adalah sebuah konsep.
- Jika R adalah sebuah role dan C adalah sebuah konsep, maka (:all R, C) is a concept.

Aksioma

Sebuah aksioma adalah proposisi yang dapat memodelkan informasi dari domain. Terdapat beberapa aksioma yang berbeda antara lain:

- Jika C1 dan C2 adalah konsep, maka C1 ⊑ C2 adalah sebuah aksioma.
- Jika C1 dan C2 adalah konsep, maka C1 ≡ C2 adalah sebuah aksioma.
- Jika a adalah sebuah individu dan C adalah sebuah konsep, maka a : C adalah sebuah aksioma.
- Jika a dan b adalah sebuah individu dan R adalah sebuah role, maka (a, b) : R adalah sebuah aksioma.

Operator Pembentuk Konsep

Fitur yang diinginkan dalam description adalah mendefinisikan konsep kompleks dalam konsep yang lebih sederhana. Operator pembentuk konsep memungkinkan definisi konsep yang kompleks.

Terdapat banyak operator pembentuk konsep kompleks, empat diantaranya adalah:

1. Kostruksi :and merupakan konsep conjoined. Setiap individu adalah anggota dari semua konsep yang ditentukan oleh konsep kompleks
   - (:and Adult Man Person) akan mewakili konsep dari sesuatu yang pada saat bersamaan adalah Adult, Man dan Person.
   - Ini bisa dilihat sebagai irisan dari konsep-konsep yang ditentukan.
2. Konstruksi :or merupakan konsep disjoined. Setiap individu adalah anggota dari setidaknya satu dari konsep yang ditentukan oleh konsep-konsep kompleks
   - (:or Car Truck Van) akan mewakili konsep dari sesuatu yang dapat berupa Car, Truck, Van atau kombinasi keduanya.
   - Ini bisa dilihat sebagai union dari konsep-konsep yang ditentukan.
3. :some menjamin bahwa untuk setiap individu yang termasuk dalam konsep ini, setidaknya ada satu individu yang terkait dengan role R yang dimiliki oleh C.
   - (:some Child Thing) mewakili konsep orang tua (yang memiliki setidaknya satu anak)
   - (:some HasPet Cat) mewakili konsep pemilik kucing (yang memiliki setidaknya satu kucing)
4. :all menjamin bahwa untuk setiap individu yang termasuk dalam konsep ini terkait dengan role R hanya untuk individu yang termasuk dalam konsep C.
   - (:all Child Doctor) mewakili konsep tentang semua anaknya adalah dokter
   - (:all Child Male) mewakili konsep tentang sesuatu yang memiliki nol atau lebih anak-anak yang semuanya laki-laki

Kowledge base dari DL dinotasikan dengan K, adalah sepasang (T, A). T (yaitu singkatan dari TBox) adalah sekumpulan aksioma terminologis, yang melibatkan:

- Konsep umum inklusi (GCI), definisi konsep;
- Aksioma konstruktor konsep, misalnya irisan (intersection) konsep, gabuangan (union) konsep, konsep negasi, konsep eksistensial esensial dan konsep universal eksistensial);
- Aksioma pembentuk role (misalnya, intersection, union, inklusi, inverse, transitif, dan komposisi).

A (yaitu, singkatan dari ABox) adalah serangkaian aksioma assertional, yang melibatkan assertion keanggotaan konsep dan keanggotaan role.

Contoh dari TBox adalah: $\text{HappyFather} \sqcap \text{Man} \subseteq \exists \text{hasChild.Female};\ \text{Elephant} \subseteq \text{Animal}$

Contoh dari ABox adalah: John:HappyFather (i.e., HappyFather(John)); <John,Mary>:hasChild (i.e., hasChild(john, Mary)

Dalam ABox, individu (yaitu instance) diperkenalkan, dengan memberi mereka nama, dan properti dari individu-individu ini dinyatakan. Asersi dapat dibuat dengan menggunakan konsep *C* dan role *R*, seperti *C(a)* dan *R(b, c)*. Pernyataan pertama disebut asersi konsep, *a* adalah milik (interpretasi dari) *C*. Jenis kedua, yang disebut asertifikasi role, c adalah pengisi/nilai role *R* untuk *b*.

3.5.1 Description Logic $\mathcal{EL}$

Description logic $\mathcal{EL}$ dikenal sebagai sub-Boolean DL karena operator dan aksioma yang digunakan mengambil dari logika Boolen. DL $\mathcal{EL}$ didefinisikan oleh:

- Operator konsep: ⊓, ¬, dan ∃)
- Tidak ada operator role (hanya role atomic)
- Aksioma konsep (⊑ dan ≡)
- Tidak ada aksioma role

Contoh dari ekspresi DL $\mathcal{EL}$ adalah:

Parent ≡ Person ⊓ ∃ hasChild.Person

3.5.2 Description Logic $\mathcal{ALC}$

$\mathcal{ALC}$ adalah description logic yang paling sederhana, yang didefinisikan oleh

- Operator konsep (⊓, ⊔, ¬, ∃, ∀)
- Tidak ada operator role (hanya role atomic)
- Aksioma konsep (⊑, ≡)
- Tidak ada aksioma role

Contoh dari ekspresi DL $\mathcal{ALC}$ adalah:

ProudParent ≡ Person ⊓ ∀hasChild.(Doctor ⊔ ∃hasChild.Doctor)

$\mathcal{S}$ digunakan untuk menotasikan $\mathcal{ALC}$ yang diperluas dengan aksioma role transitif.

Karakter tambahan digunakan untuk menandakan variasi perluasan yang dilakukan, cebagai contoh:

- $\mathcal{H}$ for role hierarchy (e.g., hasDaughter $\sqsubseteq$ hasChild)
- $\mathcal{R}$ for role box (e.g., hasParent $\circ$ hasBrother $\sqsubseteq$ hasUncle)
- $\mathcal{O}$ for nominals/singleton classes (e.g., {Italy})
- $\mathcal{I}$ for inverse roles (e.g., isChildOf $\equiv$ hasChild$^-$)
- $\mathcal{N}$ for number restrictions (e.g., $\geqslant$2hasChild, $\leqslant$3hasChild)
- $\mathcal{Q}$ for qualified number restrictions (e.g., $\geqslant$2hasChild.Doctor)
- $\mathcal{F}$ for functional number restrictions (e.g., $\leqslant$1hasMother)

Sehingga,

$\mathcal{SHIQ} = \mathcal{S}$ + role hierarchy ($\mathcal{H}$) + inverse role ($\mathcal{I}$) + qualified number restriction($\mathcal{Q}$).

Dimana qualified number restriction mencakup: $\mathcal{Q}$, $\mathcal{N}$, $\mathcal{R}$.

Contoh ekspansi dari description logic:

ALC:	$\sqsubseteq$, $\equiv$ **for classes** $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$ $\top$, $\bot$
SR:	**+** property chains, property characteristics, **role hierarchies** $\sqsubseteq$
SRO:	**+ nominals {o}**
SROI:	**+ inverse properties**
SROIQ:	**+** qualified **cardinality constraints**
SROIQ(D):	**+ datatypes (including** facets**)**

3.5.3 Description Logic $\mathcal{SHOIN}$ dan ($\mathcal{D}$)

$\mathcal{SHOIN}$ diperluas dengan operator role: reflexive, antisymmetric, and irreflexive roles, disjoint roles, a universal role, and constructs $\exists$R.Self.

$\mathcal{SHOIN}$ juga memperhatikan egasi dalam role assertion di A Box dan qualified number restrictions.

Sedangkan ($\mathcal{D}$) melibatkan datatype property.

DL $\mathcal{SHOIN}$ adalah basis dari OWL 1, sehingga ada dua kowledge base yaitu TBox dan ABox.

DL ($\mathcal{D}$) adalah basis dari OWL 2, sehingga ada tambahan aksioma role, dimana aksioma-aksioma ini kadang didefinisikan tersendiri sebagai RBox. Sehigga DL ($\mathcal{D}$) memiliki tiga kowledge base yaitu: TBox, RBox, dan ABox. Dimana RBox adalah sekumpulan aksioma role, yang melibatkan:

- Role inklusi, pendefinisian role;
- Aksioma konstruktor role, misalnya inverse dan komposisi;

3.5.4 Karakteristik dari Properti

Properti bisa dinyatakan menjadi:

- **Transitive** **hasAncestor** R(a,b) and R(b,c) → R(a,c)
- **Symmetric** **hasSpouse** R(a,b) → R(b,a)
- **Asymmetric** **hasChild** R(a,b) → not R(b,a)
- **Reflexive** **hasRelative** R(a,a) for all a
- **Irreflexive** **parentOf** not R(a,a) for any a
- **Functional** **hasHusband** R(a,b) and R(a,c) → b=c
- **InverseFunctional** **hasHusband** R(a,b) and R(c,b) → a=c

3.5.5 Datatypes ($\mathcal{D}$)

Argumen kedua dari formula description logic (nilai dari properti) bukan hanya individu/instance atau disebut dengan object property, tetapi juga datatype literal bisa sebagai argumen kedua. Property dengan datatype literal disebut dengan datatype property.

OWL menyediakan banyak datatype skema XML antara lain: xsd:integer; xsd:string; xsd:float; xsd:boolean; xsd:anyURI; xsd:dateTime.

Contoh ekspresi description logic dengan datatypes adalah: hasAge(John, “51”^^xsd:integer).

Contoh lain adalah penambahan batasan pada property. Sebagai catatan, batasan properti untuk kardinaliti ini diperoleh dari skema XML sehingga notasi yang digunakan bukan standar notasi dari ekspresi DL.

Contoh: Teenager ≡ Person ⊓ ∃hasAge.(xsd:integer:≥ 12 and ≤19)

Contoh ekspresi description logic yang lain adalah:

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg\text{Woman} \\
\text{Mother} &\equiv \text{Woman} \sqcap \exists\text{hasChild.Person} \\
\text{Father} &\equiv \text{Man} \sqcap \exists\text{hasChild.Person} \\
\text{Parent} &\equiv \text{Father} \sqcup \text{Mother} \\
\text{Grandmother} &\equiv \text{Mother} \sqcap \exists\text{hasChild.Parent} \\
\text{MotherWithManyChildren} &\equiv \text{Mother} \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall\text{hasChild.}\neg\text{Woman} \\
\text{Wife} &\equiv \text{Woman} \sqcap \exists\text{hasHusband.Man}
\end{aligned}
$$

Ekspresi DL selanjutnya adalah:

Self
- **PersonCommittingSuicide ≡ ∃kills.Self**

Keys (not really in SROIQ(D), but in OWL)
- **set of (object or data) properties whose values uniquely identify an object**

disjoint properties
- **Disjoint(hasParent,hasChild)**

explicit anonymous individuals
- **as in RDF: can be used instead of named individuals**

3.6. Contoh

Contoh 1: Contoh TBox dari hubungan anggota keluarga:

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg\text{Woman} \\
\text{Mother} &\equiv \text{Woman} \sqcap \exists\text{hasChild.Person} \\
\text{Father} &\equiv \text{Man} \sqcap \exists\text{hasChild.Person} \\
\text{Parent} &\equiv \text{Father} \sqcup \text{Mother} \\
\text{Grandmother} &\equiv \text{Mother} \sqcap \exists\text{hasChild.Parent} \\
\text{MotherWithManyChildren} &\equiv \text{Mother} \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall\text{hasChild.}\neg\text{Woman} \\
\text{Wife} &\equiv \text{Woman} \sqcap \exists\text{hasHusband.Man}
\end{aligned}
$$

Ekspansi TBox dari hubungan anggota keluarga (lihat contoh 1) ke konsep yang lebih sederhana:

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg(\text{Person} \sqcap \text{Female}) \\
\text{Mother} &\equiv (\text{Person} \sqcap \text{Female}) \sqcap \exists\text{hasChild.Person} \\
\text{Father} &\equiv (\text{Person} \sqcap \neg(\text{Person} \sqcap \text{Female})) \sqcap \exists\text{hasChild.Person} \\
\text{Parent} &\equiv ((\text{Person} \sqcap \neg(\text{Person} \sqcap \text{Female})) \sqcap \exists\text{hasChild.Person}) \\
&\quad \sqcup ((\text{Person} \sqcap \text{Female}) \sqcap \exists\text{hasChild.Person}) \\
\text{Grandmother} &\equiv ((\text{Person} \sqcap \text{Female}) \sqcap \exists\text{hasChild.Person}) \\
&\quad \sqcap \exists\text{hasChild.}(((\text{Person} \sqcap \neg(\text{Person} \sqcap \text{Female})) \\
&\qquad \sqcap \exists\text{hasChild.Person}) \\
&\qquad \sqcup ((\text{Person} \sqcap \text{Female}) \\
&\qquad\quad \sqcap \exists\text{hasChild.Person})) \\
\text{MotherWithManyChildren} &\equiv ((\text{Person} \sqcap \text{Female}) \sqcap \exists\text{hasChild.Person}) \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv ((\text{Person} \sqcap \text{Female}) \sqcap \exists\text{hasChild.Person}) \\
&\quad \sqcap \forall\text{hasChild.}(\neg(\text{Person} \sqcap \text{Female})) \\
\text{Wife} &\equiv (\text{Person} \sqcap \text{Female}) \\
&\quad \sqcap \exists\text{hasHusband.}(\text{Person} \sqcap \neg(\text{Person} \sqcap \text{Female}))
\end{aligned}
$$

Contoh 2:

Andaikan ada sejumlah objek dan sebuah relasi biner has-branch antara objek yang mengarah dari sebuah tree ke subtree-nya. Maka binary tree adalah tree dengan paling banyak dua subtree yang adalah binary tree.

Maka, ekspresi description logic untuk binary tree adalah:

$$\text{BinaryTree} \equiv \text{Tree} \sqcap \leqslant 2\,\text{has-branch} \sqcap \forall \text{has-branch}.\text{BinaryTree}.$$

3.7. Latihan

Express the statements below in Description Logic Formula.

a. VegetarianDish is a Dish that consists only of Vegetarian Ingredients. Note: use Ex:Dish, Ex:VegetarianDish, and Ex: VegetarianIngredient class, and ex:hasIngredient object property.
b. Bedroom is a Room that has at least one Bed. Note: use Ex:Room, Ex:Bedroom, and Ex:Bed class, and Ex:furnishedWith object property.
c. RedWine is a Wine that has the red color. Note: use Ex:Wine, and Ex:RedWine class, Ex:color data type property, and "red" individu.
d. Grandfather is both a man and a parent. Note: use Ex:Parent, Ex:Man, and Ex:Grandfather class.

Answer:

a. Dish ⊓ ∀hasIngredient.VegetarianIngredient ⊑ VegetarianDish
b. Room ⊓ ∃furnishedWith.Bed ⊑ BedRoom
c. Wine ⊓ ∍color. "red" ⊑ RedWine
d. Man ⊓ Parent ⊑ Grandfather

3.8. Rangkuman

DL adalah fragmen dari first order logic (FOL) klasik, yang memiliki empat elemen dasar yang melibatkan, nama konsep, nama role, nama individu, dan konstruktor (yaitu, untuk konsep dan role).

Sematik dari description logic adalah sebuah interpretasi I yaitu pasangan (Δ^I, $.I$), di mana Δ^I adalah domain dan I adalah fungsi.

Kowledge base dari DL dinotasikan dengan K, adalah sepasang (T, A). T (yaitu singkatan dari TBox) adalah sekumpulan aksioma terminologis dan A (yaitu, singkatan dari ABox) adalah serangkaian aksioma assertional, yang melibatkan assertion keanggotaan konsep dan keanggotaan role.

Daftar Pustaka

[5] Grigoris Antoniou, Frank Van Harmelen, A Semantic Web Primer, Second Edition, MIT Press, 2008.

[6] Baader F., Calvanese D., McGuinness, D., Nardi, D. and Patel-Schneider P., 2007, *The Description Logic Handbook: Theory, Implementation and Applications*, Second Edition, Cambridge University Press, Cambridge.

IV. PERTEMUAN III: PROTÉGÉ

4.1. Ontologi

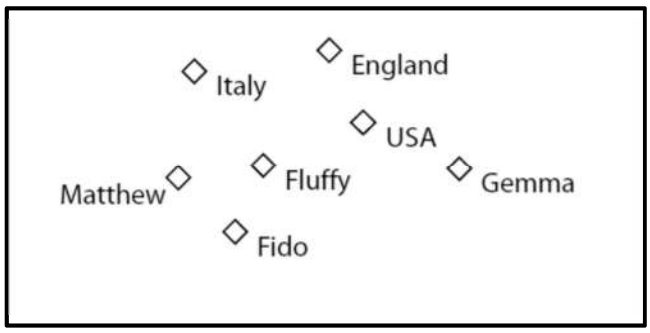
Ontologi digunakan untuk menangkap pengetahuan tentang beberapa domain yang diminati. Ontologi menggambarkan konsep dalam domain dan juga hubungan yang ada di antara konsep-konsep tersebut. Bahasa ontologi yang berbeda menyediakan fasilitas yang berbeda. Perkembangan terbaru dalam bahasa ontologi standar adalah OWL dari World Wide Web Consortium (W3C). Seperti Protege, OWL memungkinkan untuk mendeskripsikan konsep tapi juga menyediakan fasilitas baru. Ini memiliki seperangkat operator yang lebih kaya - mis. Persimpangan, persatuan dan negasi. Hal ini didasarkan pada model logis yang berbeda yang memungkinkan konsep-konsep yang akan dideskripsikan dan juga dijelaskan. Oleh karena itu, konsep kompleks dapat dibangun dalam konsep yang lebih sederhana.

Selanjutnya, model logika memungkinkan penggunaan reasoner yang dapat memeriksa apakah semua pernyataan dan definisi dalam ontologi saling konsisten dan juga dapat mengenali konsep mana di mana nuansa tersebut. Oleh karena itu, reasoner dapat membantu mempertahankan hierarki dengan benar. Ini sangat berguna saat menangani kasus di mana kelas dapat memiliki lebih dari satu parent.

Ontologi OWL memiliki komponen yang mirip dengan ontologi berbasis Protege. Namun, terminologi yang digunakan untuk menggambarkan komponen ini sedikit berbeda dengan yang digunakan di Protege. Ontologi OWL terdiri dari Individu, Properti, dan Kelas, yang kira-kira sesuai dengan contoh Protege Frame, Slots and Classes.

4.1.1. Individu

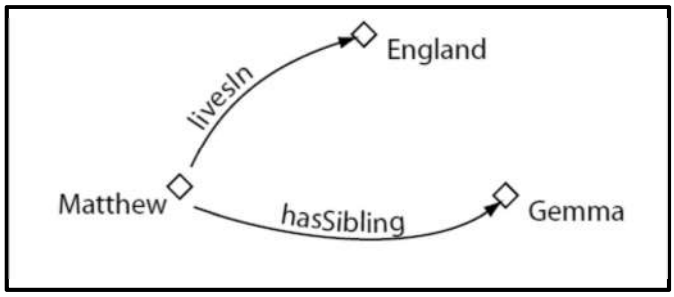
Individu, merepresentasikan objek dalam sebuah domain. Perbedaan penting antara Protege dan OWL adalah OWL tidak menggunakan Unique Name Assumption (UNA). Ini berarti bahwa dua nama yang berbeda sebenarnya bisa merujuk pada individu yang sama. Misalnya, “Queen Elizabeth”, “The Queen” dan “Elizabeth Windsor” mungkin semua mengacu pada individu yang sama. Di dalam OWL, harus dinyatakan secara eksplisit bahwa individu itu sama satu sama lain, atau berbeda satu sama lain. Mereka mungkin sama satu sama lain, atau mereka mungkin saling berbeda satu sama lain. Gambar 4.1 menunjukkan representasi beberapa individu dalam beberapa domain. Pada gambar tersebut, individu direpresentasikan sebagai diamond.



Gambar 4.1 Representasi dari individu

4.1.2. Properti

Properti adalah hubungan biner pada individu - yaitu properti menghubungkan dua individu secara bersamaan. Sebagai contoh, properti hasSibling mungkin menghubungkan individu Matius dengan individu Gemma, atau properti hasChild yang menghubungkan individu Petrus dengan individu Matius. Properti bisa memiliki invers. Misalnya, kebalikan dari hasOwner adalah isOwnedBy. Properti dapat dibatasi untuk memiliki nilai tunggal (i.e. Untuk menjadi fungsional). Mereka juga bisa berupa transitif atau simetris. Gambar 4.2 menunjukkan representasi beberapa properti yang menghubungkan beberapa individu secara bersamaan.

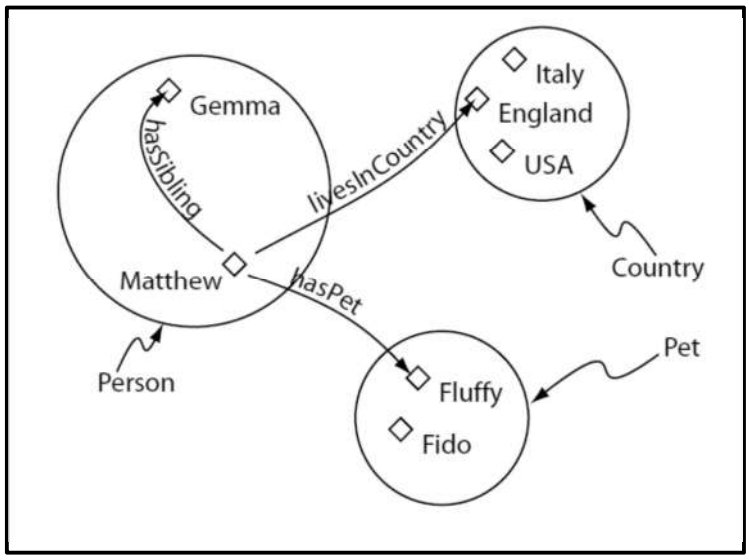


Gambar 4.2 Representasi dari properti

4.1.3. Kelas

Kelas OWL ditafsirkan sebagai kumpulan yang berisi individu. Mereka dideskripsikan menggunakan deskripsi formal (matematis) yang menyatakan secara persis persyaratan keanggotaan kelas. Misalnya, kelas Cat akan berisi semua individu yang merupakan kucing. Kelas dapat diatur menjadi hirarki kelas superkelas, yang juga dikenal sebagai taksonomi. Subclass mengkhususkan diri (“are subsumed by”) superclasses mereka. Misalnya perhatikan

kelas Animal and Cat - Cat mungkin subclass Animal (jadi Animal adalah superclass Cat). Ini mengatakan bahwa"All cats are animals", "All members of the class Cat are members of the class Animal", "Being a Cat implies that you're an Animal", dan "Cat is subsumed by Animal". Salah satu fitur kunci dari OWL-DL adalah bahwa hubungan superclass-subclass ini (hubungan subsumption) dapat dihitung secara otomatis oleh reasoner. Gambar 4.3 menunjukkan representasi dari beberapa kelas yang berisi individu. Kelas direpresentasikan sebagai lingkaran atau oval, seperti himpunan dalam diagram Venn.



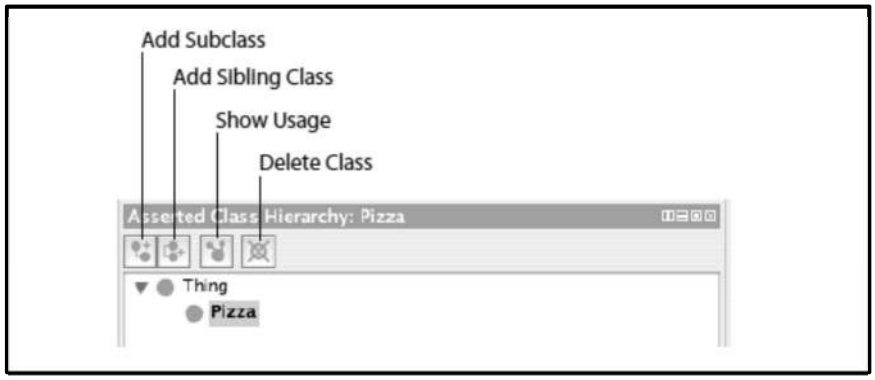
Gambar 4.3 Representasi dari kelas

4.2. Membangun ontologi OWL

4.2.1. Membuat Kelas

Seperti disebutkan sebelumnya, ontologi mengandung kelas. Di Protege 5, editing kelas dilakukan dengan menggunakan tab 'Entities' atau tab 'Classes' yang ditunjukkan pada Gambar 4.4. Tampilan hierarki kelas awal harus menyerupai gambar yang ditunjukkan pada Gambar 4.5. Ontologi kosong berisi satu kelas bernama Thing. Seperti disebutkan sebelumnya, kelas OWL diinterpretasikan sebagai kumpulan individu (atau kumpulan objek). Kelas Thing adalah kelas yang mewakili kumpulan yang berisi semua individu. Karena semua kelas ini adalah subclass dari Thing.

Gambar 4.4 Tab dari kelas



Gambar 4.5 Panel hirarki kelas

Latihan 1: membuat kelas Pizza, PizzaTopping dan PizzaBase

Langkah-langkahnya:

1. Pastikan bahwa tab Classes atau Entities sudah dipilih.
2. Tekan icon “Add” bisa dilihat pada Gambar 4.5. Tombol ini membuat kelas baru dari subkelas dari kelas yang dipilih (pada kasus ini sub kelas dari kelas Thing akan dibuat).
3. Sebuah dialog akan muncul untuk memberikan nama kelas. Berikan nama berturut-turut sampai ketiga kelas berhasil dibuat.
4. Hasilnya bisa dilihat pada Gambar 4.6.

Gambar 4.6 Hirarki kelas awal

4.2.2. Kelas Disjoint

Setelah menambahkan kelas Pizza, PizzaTopping dan PizzaBase ke ontologi, sekarang perlu didefinisikan bahwa kelas-kelas ini terpisah, sehingga individu (atau objek) tidak dapat menjadi instance dari lebih satu kelas dari ketiga kelas ini. Untuk menentukan kelas yang dipisahkan dari kelas yang dipilih klik tombol “Disjoints classes” yang terletak di bagian bawah tampilan “Class Description”.

Latihan 2: Definisikan kelas Pizza, PizzaBase, dan PizzaTopping adalah disjoint.

Langkah-langkahnya:

1. Pilih kelas Pizza dalam hirarki kelas.
2. Tekan tombol `Disjoint With' pada `class description', akan muncul kotak dialog dimana multiple kelas yang akan dibuat disjoint bisa dipilih. Ini akan membuat kelas PizzaBase dan PizzaTopping (sibling dari kelas Pizza) disjoint dari kelas Pizza.
3. Hasilnya bisa dilihat pada Gambar 4.7.

Gambar 4.7 Hirarki kelas awal: disjoint

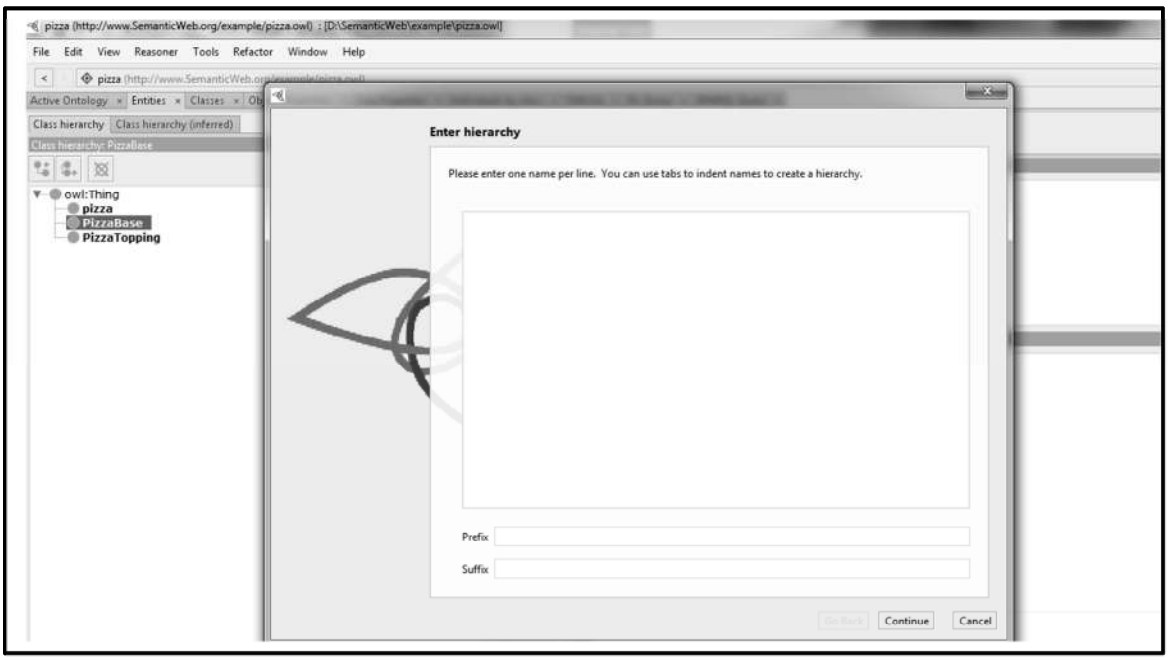
4.2.3. Meggunakan 'Create Class Hierarchy' untuk Membuat Kelas

Tool 'Create Class Hierarchy' bisa digunakan untuk menambahkan beberapa subkelas dari sebuah kelas.

Latihan 3: Buat beberapa subkelas dari ketiga kelas (Pizza, PizzaBase, dan PizzaToping) yang telah dibuat sebelumnya.

Langkah-langkahnya:

1. Pilih kelas PizzaBase pada 'class hierarchy'.
2. Dari menu 'Tools' pilih 'Create Class Hierarchy...'
3. Akan muncul kotak dialog seperti pada Gambar 4.8. Karena kelas PizzaBase sudah dipilih, maka subkelas dari kelas PizzaBase yang akan dibuat. Tuliskan nama kelas satu nama per baris.
4. Tekan tombol 'Continue'.
5. Tandai box `Do you want to maka sibling classes disjoint?' untuk memastikan semua kelas disjoint. Kemudian tekan tombol 'Finish'.



Gambar 4.8 Kotak dialog dari tool 'Create Class Hierarchy...'

Membuat beberapa topping pizza

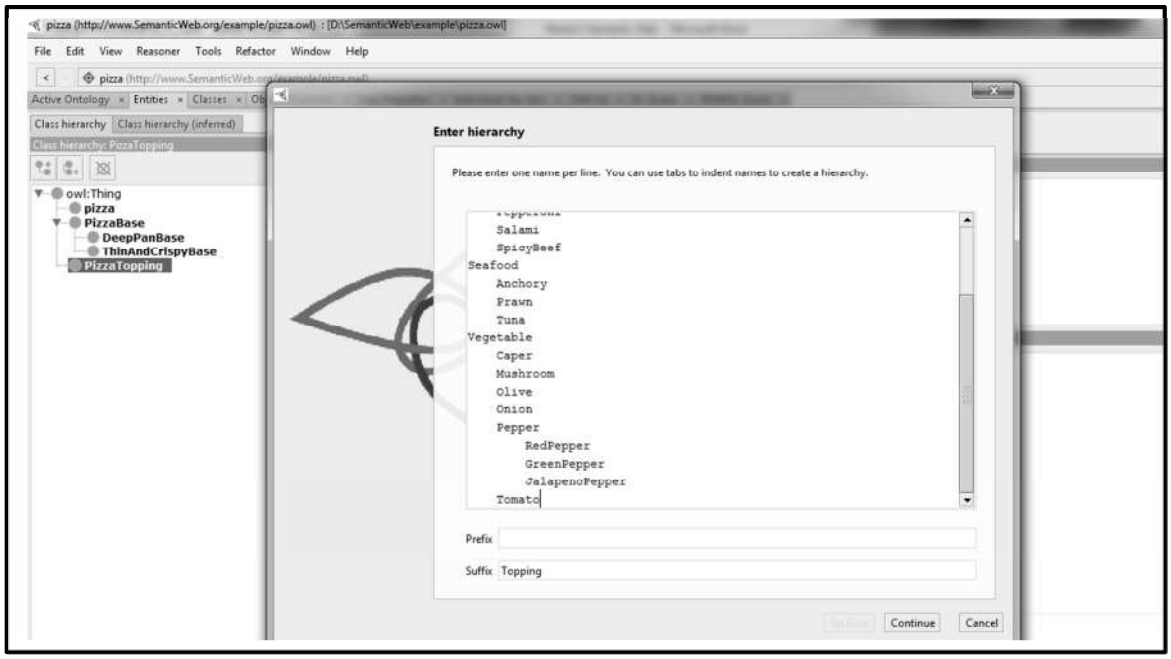
Beberapa kelas basic aka dibuat untuk dimaa beberapa topping akan dikelompokkan ke beberapa kategori: topping meat, vegetable, cheese, dan seafood.

Latihan 4:Buat beberapa subkelas dari kelas PizzaTopping dan kelompokkan subkelas tersebut berdasarkan kategori (yaitu: meat, vegetable, cheese, dan seafood).

Langkah-lankahnya:

1. Pilih kelas PizzaTopping pada 'class hierarchy'.
2. Dari menu 'Tools' pilih 'Create Class Hierarchy...'
3. Akan muncul kotak dialog seperti pada Gambar 4.8. Karena kelas PizzaTopping sudah dipilih, maka subkelas dari kelas PizzaTopping yang akan dibuat. Karena semua nama berakhiran dengan kata Topping, maka pada kotak 'Suffix' tuliskan 'Topping'.
4. Tool ini memungkinkan hirarki kelas untuk dimasukkan menggunakan tab pohon indentasi. Dengan menggunakan area teks pada tool, masukkan nama kelas seperti yang ditunjukkan pada Gambar 4.9. Perhatikan bahwa nama kelas harus diindasikan menggunakan tab, jadi misalnya SpicyBeef, yang ingin kita beri subkelas Meat dimasukkan di bawah Meat dan diberi label dengan tab. Begitupun, Pepperonic juga masuk di bawah Meat di bawah SpicyBeef dan juga menjorok dengan tab.
5. Tekan tombol 'Continue'.

6. Tandai box `Do you want to maka sibling classes disjoint?' untuk memastikan semua kelas disjoint. Kemudian tekan tombol ‘Finish’.
7. Hasilnya bisa dilihat pada Gambar 4.10.



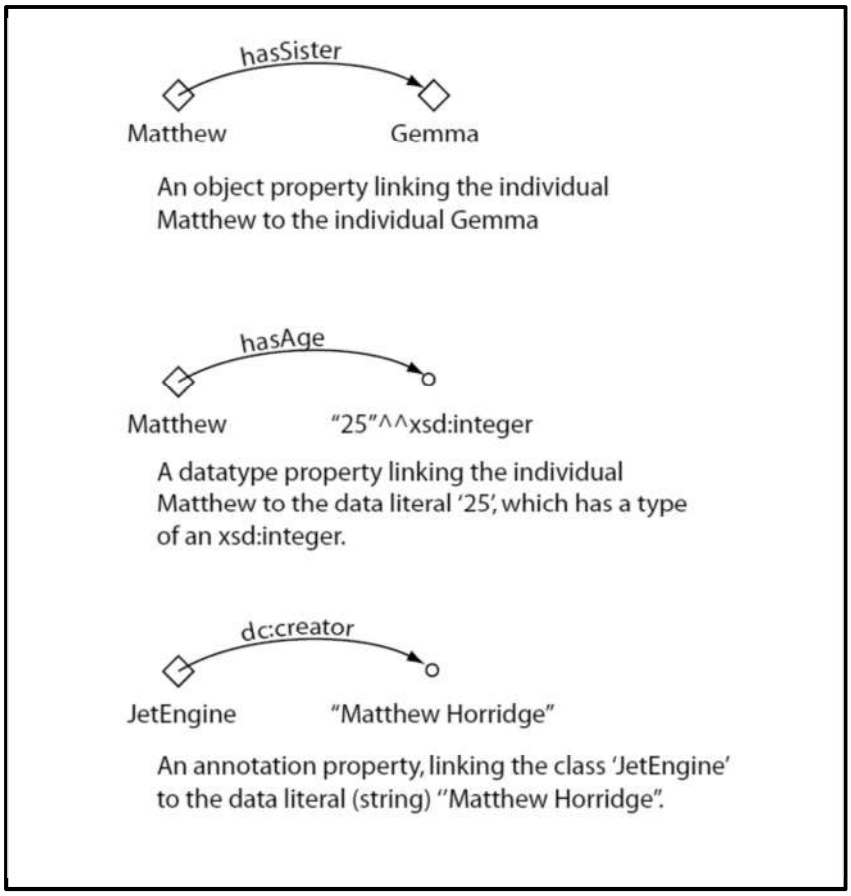
Gambar 4.9 Hirarki kelas Topping

Gambar 4.10 Hirarki kelas

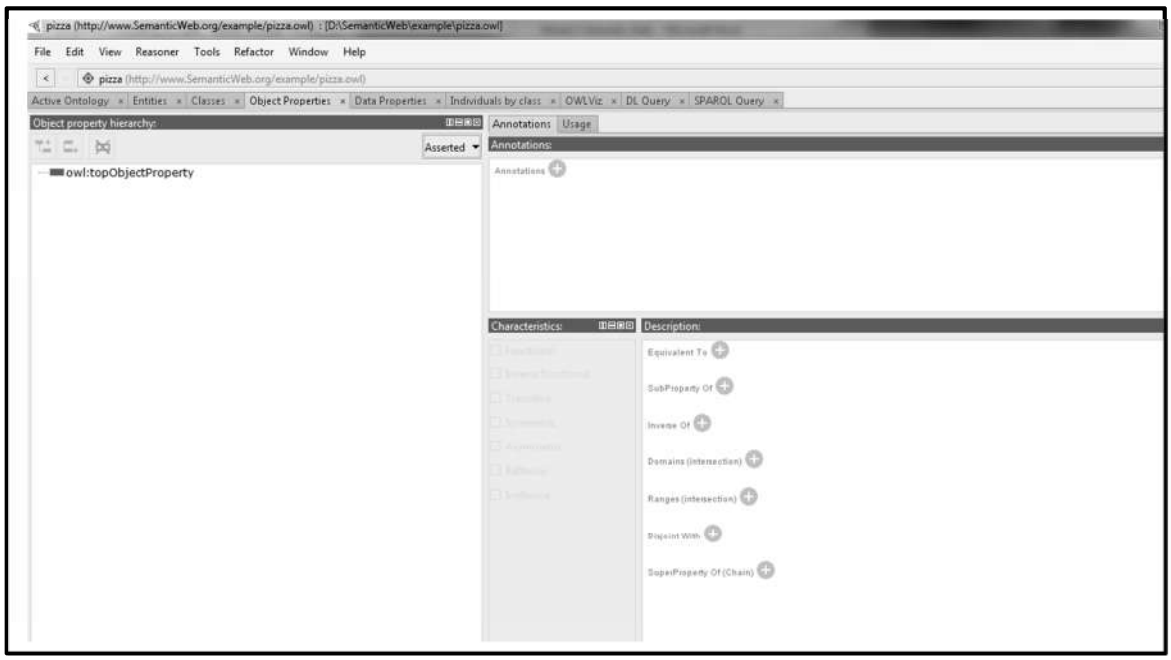
4.2.4. Properti pada OWL

Properti pada OWL mewakili relasi. Secara umum, ada dua jenis properti yaitu: properti objek dan properti datatype. Properti objek adalah relasi antara dua individu. Properti objek menghubungkan individu ke individu. OWL juga memiliki tipe properti ketiga, properti anotasi. Properti anotasi dapat digunakan untuk menambahkan informasi (metadata, data tentang data) ke kelas, individu dan properti objek/datatype. Gambar 4.11 menggambarkan contoh dari setiap jenis properti pada OWL yang ada.

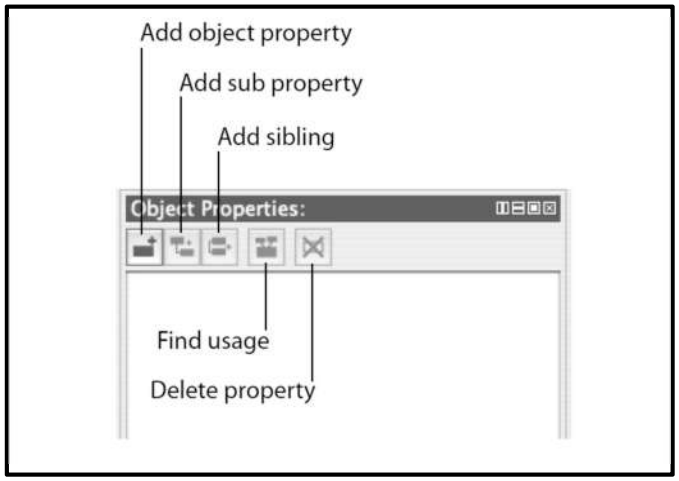
Properti dapat dibuat dengan menggunakan tab 'Object Properties' yang ditunjukkan pada Gambar 4.12. Gambar 4.13 menunjukkan tombol yang terletak di pojok kiri atas tab 'Objek Properties' yang digunakan untuk membuat properti OWL. Seperti dapat dilihat dari Gambar 4.13, ada tombol untuk membuat properti Datatype, properti Objek dan properti Anotasi.



Gambar 4.11 Tipe berbeda dari properti OWL



Gambar 4.12 Tab properti

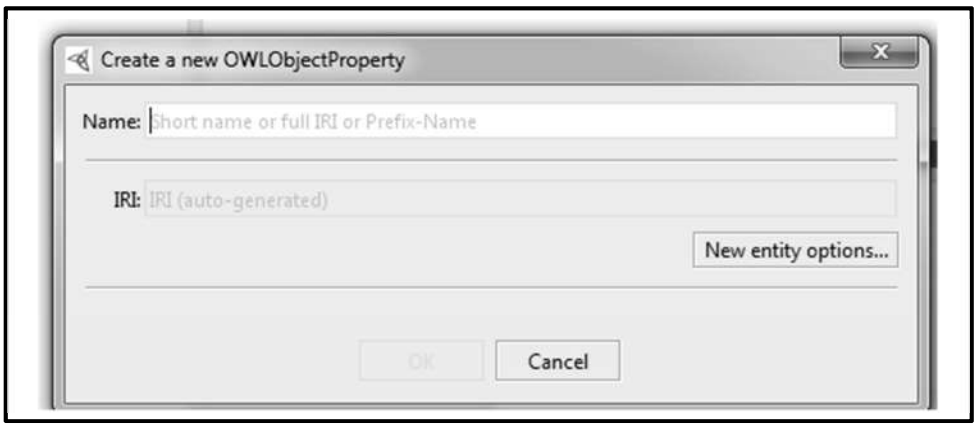


Gambar 4.13 Tombol pembuatan property, letaknya pada tab properti diatas list/pohon properti

Latihan 5: Buat properti objek dengan nama hasIngredient.

Langkah-langkahnya:

1. Pilih tab `Object Properties'. Gunakan tombol `Add Object Property' (lihat Gambar 4.13) untuk membuat properti Object baru.
2. Beri nama properti itu hasIngredient menggunakan pops up kotak dialog ‘Create a new OWLObjectProperty’, seperti bisa dilihat pada Gambar 4.14.



Gambar 4.14 Kotak dialog untuk membuat properti objek baru

Setelah menambahkan properti hasIngredient, sekarang akan ditambahkan dua properti lagi, hasTopping, dan hasBase. Dalam OWL, properti bisa memiliki sub properti, sehingga memungkinkan untuk membentuk hierarki properti. Sub properti mengkhususkan sifat super mereka (dengan cara yang sama bahwa subclass mengkhususkan super kelas mereka). Misalnya, properti hasMother bisa mengkhususkan properti yang lebih umum dari hasParent. Dalam kasus ontologi pizza, sifat-sifat hasTopping dan hasBase harus dibuat sebagai subproperti hasIngredient. Jika properti hasTopping (atau properti hasBase) menghubungkan dua individu, ini berarti bahwa kedua individu tersebut terkait oleh properti hasIngredient.

Latihan 6: Buat properti hasTopping dan hasBase sebagai subproperti dari properti hasIngredient.

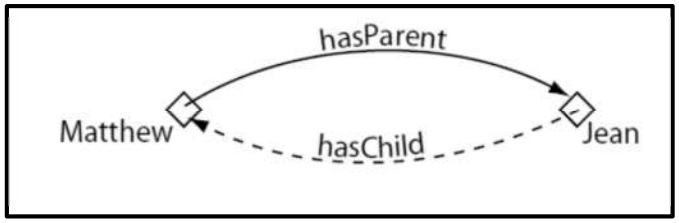
Langkah-langkahnya:

1. Untuk membuat hasTopping properti sebagai subproperti dari properti hasIngredient, pilih properti hasIngredient dalam hirarki properti pada tab ‘Object Properties’.
2. Tekan tombol `Add subproperty '. Properti objek baru akan dibuat sebagai subproperti properti hasIngredient.
3. Beri nama properti baru dengan hasTopping.
4. Ulangi langkah di atas tapi beri nama property hasBase.

4.2.5. Inverse Properties

Setiap properti objek bisa memiliki properti invers yang sesuai. Jika beberapa properti menghubungkan individu *a* ke individu *b* maka properti inversnya akan menghubungkan individu *b* ke individu *a*. Sebagai contoh, Gambar 4.15 menunjukkan properti hasParent dan inversnya hasChild. Jika Matthew hasParent Jean, maka karena properti invers dapat disimpulkan bahwa Jean hasChild Matthew.

Properti invers dapat dibuat/ditentukan dengan menggunakan tampilan properti invers seperti yang ditunjukkan pada Gambar 4.16. Untuk kelengkapan, akan ditentukan properti invers untuk properti yang ada di ontologi Pizza.



Gambar 4.15 Contoh properti invers

Gambar 4.16 Kotak dialog membuat properti invers

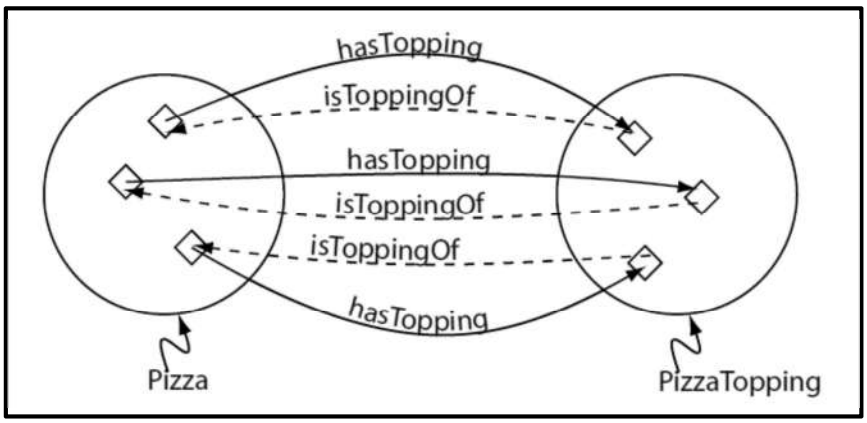
Latihan 7: Buat beberapa subproperti.

Langkah-langkahnya:

1. Gunakan tombol 'Add object property' pada tab 'Object Properties' untuk membuat properti Objek baru yang disebut isIngredientOf (ini akan menjadi properti invers dari hasIngredient).
2. Tekan tombol 'Add' di sebelah 'Inverse properties' pada tampilan 'Property Description' yang ditunjukkan pada Gambar 4.16. Ini akan menampilkan dialog dari mana properti dapat dipilih. Pilih properti isIngredient dan tekan `OK '. Properti tersebut ditampilkan dalam tampilan 'Inverse Property'.
3. Pilih properti hasBase.
4. Tekan ikon 'Add' di sebelah 'Inverse properties' pada tampilan `Property Description'. Buat properti baru yang disebut isBaseOf. Pilih properti ini dan klik `OK '. Perhatikan bahwa hasBase sekarang memiliki invers properti yang diberi nama isBaseOf. Bisa juga secara opsional properti baru isBaseOf ditempatkan sebagai subproperty dari isIngredientOf.
5. Pilih properti hasTopping.
6. Tekan ikon 'Add' di sebelah `Inverse properties 'pada tampilan` Property Description'. Gunakan dialog properti yang muncul untuk membuat properti isToppingOf dan tekan `OK '.

4.2.6. Domain dan Range dari Properti

Properti bisa memiliki domain dan range yang ditentukan. Properti menghubungkan individu dari domain ke individu dari range. Misalnya, dalam ontologi pizza, properti hasTopping akan menghubungkan individu-individu yang termasuk dalam kelas Pizza ke individu-individu yang masuk dalam kelas PizzaTopping. Dalam hal ini domain dari properti hasTopping adalah Pizza dan rangenya adalah PizzaTopping. Ini digambarkan pada Gambar 4.17.



Gambar 4.17 Domain dan range dari kelas hasTopping dan properti inversnya isToppingOf

Latihan 8: Buat range dari properti hasTopping

Langkah-langkahnya:

1. Pastikan properti hasTopping dipilih di hierarki properti pada tab 'Object Properties'.
2. Tekan ikon 'Add' di sebelah ‘Ranges’ pada tampilan` Property Description' (lihat Gambar 4.16). Dialog akan muncul yang memungkinkan sebuah kelas dipilih dari hierarki kelas ontologi.
3. Pilih PizzaTopping dan tekan tombol `OK '. PizzaTopping sekarang ditampilkan dalam daftar range.

Latihan 9: Buat domain dari properti hasTopping

Langkah-langkahnya:

1. Pastikan properti hasTopping dipilih di hierarki properti pada tab 'Object Properties'.
2. Tekan ikon 'Add' di sebelah ‘Domains’ pada tampilan` Property Description' (lihat Gambar 4.16). Dialog akan muncul yang memungkinkan sebuah kelas dipilih dari hierarki kelas ontologi.
3. Pilih Pizza dan tekan tombol `OK '. Pizza sekarang ditampilkan dalam daftar domain.

Latihan 10: Buat domain dan range dari properti hasBase dan properti inversnya, isBaseOf

Langkah-langkahnya:

1. Pilih properti hasBase.
2. Tentukan domain dari properti hasBase yaitu Pizza.
3. Tentukan range dari properti hasBase yaitu PizzaBase.
4. Pilih properti isBaseOf. Ingat bahwa domain dari isBaseOf adalah range dari property inversnya, hasBase, dan bahwa range dari isBaseOf adalah domain dari properti inversnya, hasBase.
5. Buat domain dari properti isBaseOf yaitu PizzaBase.
6. Buat range dari properti isBaseOf yaitu Pizza.

4.2.7. Property Restrictions

Di OWL, properti menggambarkan hubungan biner. Properti datatype menggambarkan hubungan antara individu dan nilai data. Properti objek menggambarkan hubungan antara dua individu. Misalnya, pada Gambar 4.2, individu Matius terkait dengan individu Gemma melalui properti hasSibling. Perhatikan semua individu yang memiliki hubungan hasSibling dengan beberapa individu lainnya. Individu-individu ini bisa dipikirkan sebagai anggota kelas individu yang memiliki hubungan hasSibling. Gagasan utamanya adalah bahwa sekelompok individu digambarkan atau diakibatkan oleh hubungan yang diikuti oleh individu-individu ini. Dalam OWL kita dapat mendefinisikan kelas-kelas semacam ini dengan menggunakan batasan-batasan.

Contoh Restriction

Untuk memperjelas jenis kelas di OWL yang bisa digambarkan berdasarkan sifat-sifat dari individunya, akan diberikan beberapa contoh.

- Kelas dari individu yang mempunyai relasi ‘at least one hasSibling’.
- Kelas dari individu yang mempunyai relasi ‘**at least one** hasSibling’ ke individu-individu dari kelas Man, i.e. sesuatu yang mempunyai ‘at least one sibling that is a man'.
- Kelas dari individu yang mempunyai relasi ‘**only** hasSibling’ ke individu dari kelas Women, i.e. sesuatu yang mempunyai hanya‘siblings that are women (sisters)’.
- Kelas dari individu yang mempunyai relasi ‘more than three hasSibling ‘.
- Kelas dari individu yang mempunyai relasi ‘at least one hasTopping’ ke individu-individu dari kelas MozzarellaTopping, i.e. kelas dari sesuatu yang mempunyai ‘at least one kind of mozzarella topping’.
- Kelas dari individu yang ‘only have hasTopping’ to members of VegetableTopping, i.e. kelas dari individu-individu yang hanya mempunyai topping yang adalah topping vegetable.

Pada OWL, semua kelas-kelas dari individu-individu menggunakan restriction. Restrictions OWL terbagi ke dalam tiga kategori utama:

1. Quantifier Restrictions
2. Cardinality Restrictions

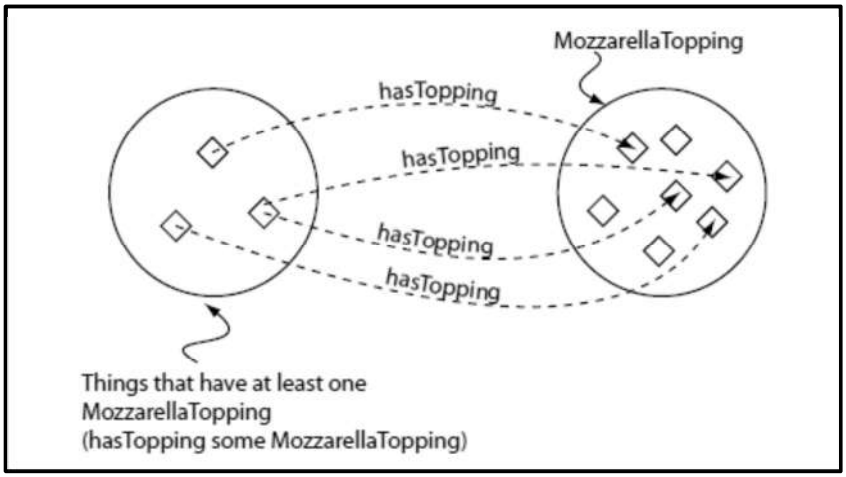
3. hasValue Restrictions.

Quantifie restriction dikategorikan menjadi dua kategori: existential restrictions dan universal restrictions.

Existential and Universal Restriction

- Restriction existential menggambarkan kelas dari individu yang mengikuti relasi ‘at least one’ pada properti tertentu ke individu-individu yang adalah anggota dari kelas tertentu. Sebagai contoh, kelas dari individu yang mempunyai relasi ‘ at least one (some) hasTopping’ ke individu-individu dari dari kelas MozzarellaTopping. Pada Protégé 5, keyword `some' untuk menotasikan restriction existential.
- Restriction universal menggambarkan kelas dari individu yang mengikuti relasi ‘only’ pada properti tertentu ke individu-individu yang adalah anggota dari kelas tertentu. Sebagai contoh, kelas dari individu yang mempunyai relasi ‘only hasTopping’ ke individu-individu dari dari kelas VegetableTopping. Pada Protégé 5, keyword ‘only' untuk menotasikan restriction universal.

Perhatikan contoh dari restriction eksistential. Restriction ‘hasTopping some MozzarellaTopping’ adalah sebuah restriction existential (ditandai dengan keyword ‘some’), pada properti hasTopping property, dan mempunyai nilai MozzarellaTopping. Restriction ini menggambarkan kelas individu yang mempunyai relasi ‘at least one hasTopping’ ke individu-individu dari kelas MozzarellaTopping. Restriction ini digambarkan pada Gambar 4.18. Seperti dapat dilihat pada Gambar 4.18, restrictionnya adalah sebuah kelas yang mengandung individu yang memenuhi restriction.

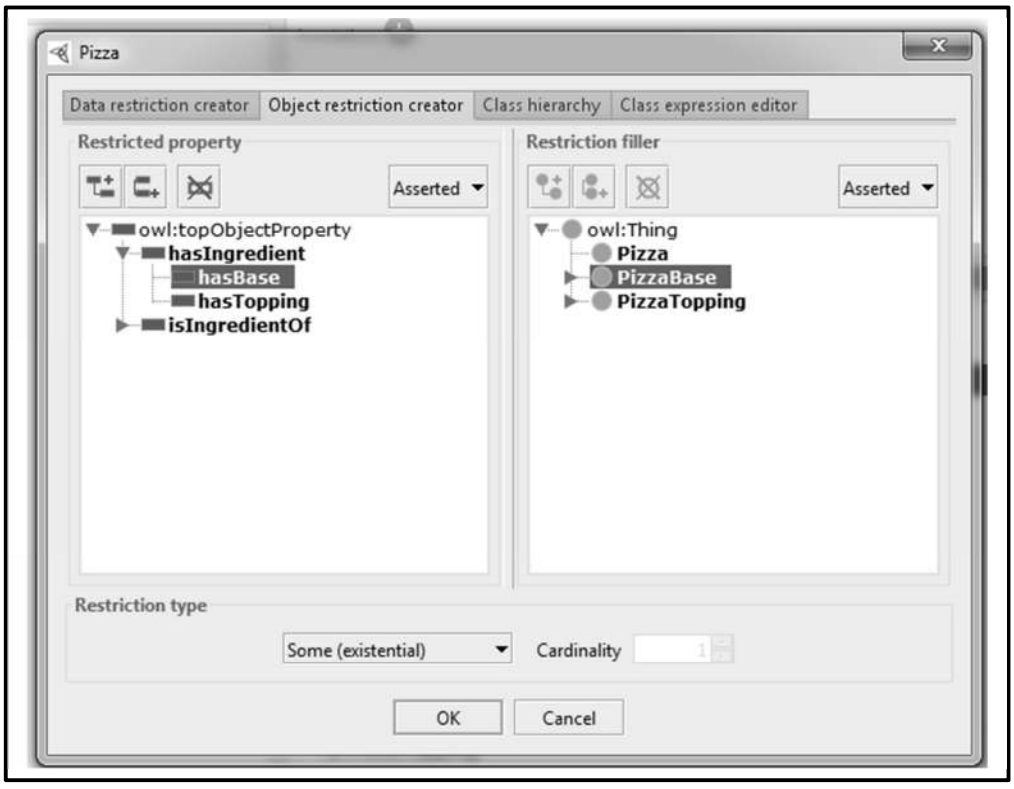


Gambar 4.18 Restriction ‘hasTopping some Mozzarella’

Latihan 11: Tambahkan restriction ke kelas yang menentukan sebuah Pizza harus mempunyai sebuah PizzaBase.

Langkah-langkahnya:

1. Pilih kelas Pizza dari hirarki kelas pada tab `Entities'.
2. Pilih ikon `Add' di sebelah `Subclass Of' pada `Class Description View' untuk membuat kondisi yang diperlukan. Ini akan membuka kotak dialog dimana restriction bisa dibuat.
3. Tekan tab 'Object restriction creator'.
4. Tambahkan tipe restriction, pilih restriction existential, maka pilih `some' pada 'Restriction type'.
5. Tentukan property yang akan dibatasi yaitu properti hasBase dengan memilih properti hasBase pada 'Restricted property'.
6. Tentukan nilai propertinya yaitu kelas PizzaBase dengan memilih kelas PizzaBase pada 'Restriction filler', seperti terlihat pada Gambar 4.19.
7. Tekan `Ok' untuk membuat restriction dan menutup kotak dialog.



Gambar 4.19 Kotak dialog untuk membuat restriction properti

.

4.3. Properti Datatype

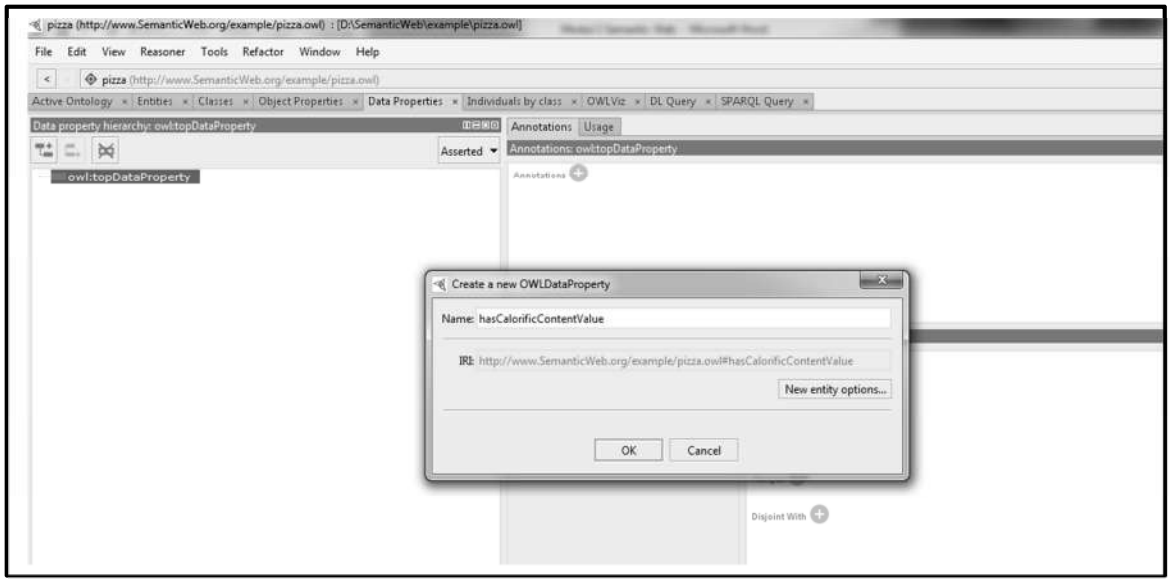
Properti datatype dapat dibuat dengan menggunakan tampilan tab `Data Properties' seperti yang ditunjukkan pada Gambar 4.20. Properti datatype akan digunakan untuk menggambarkan kandungan kalori dari pizza. Beberapa rentang numerik akan digunakan untuk secara luas mengklasifikasikan pizza tertentu sebagai kalori tinggi atau rendah. Untuk melakukan ini, kita perlu menyelesaikan langkah-langkah berikut:

- Buat properti datatype hasCalorificContentValue, yang akan digunakan untuk menyatakan kandungan kalori dari pizza tertentu.
- Buat beberapa contoh individu pizza dengan kandungan kalori tertentu.
- Buat dua kelas yang secara luas mengkategorikan pizza dengan kalori rendah atau tinggi.
- Properti datatype dapat digunakan untuk menghubungkan individu dengan nilai data konkret yang mungkin bertipe atau tidak bertipe.

Latihan 12: Membuat properti datatype yang disebut hasCalorificContentValue

Langkah-langkahnya:

1. Pilih tab `Data Properties'. Gunakan tombol `Add Data Property' untuk membuat properti Datatype baru.
2. Beri nama properti itu hasCalorificContentValue menggunakan pops up kotak dialog ‘Create a new OWLDatatypeProperty’, seperti bisa dilihat pada Gambar 4.20.

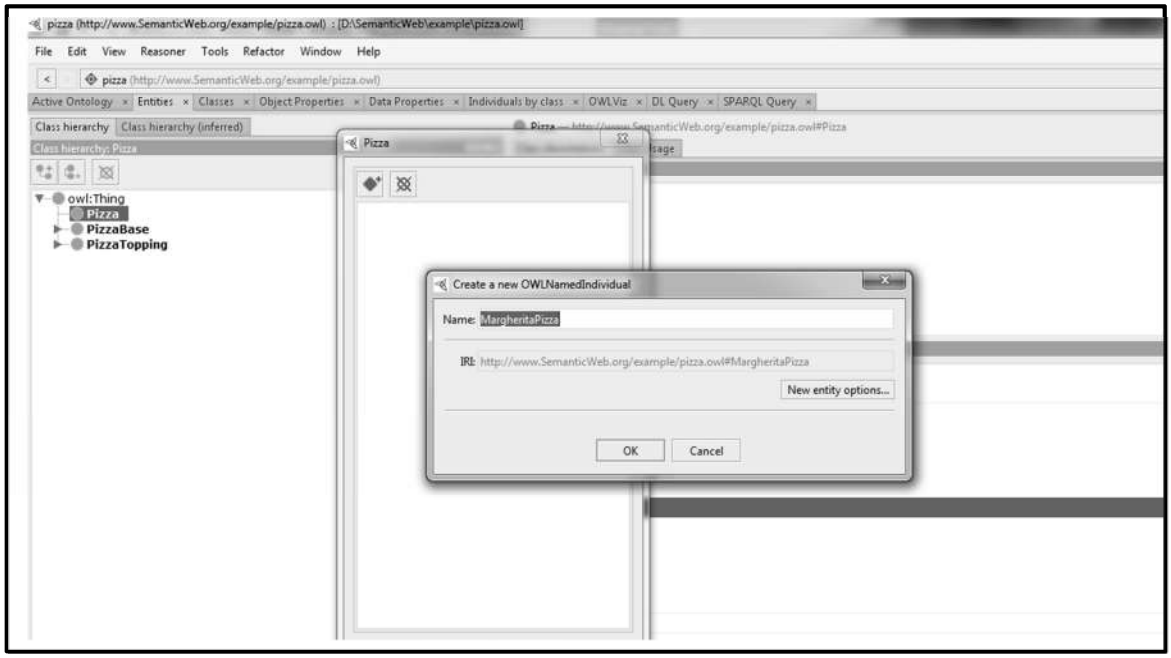


Gambar 4.20 Membuat properti datatype

Latihan 13: Membuat contoh individu pizza

Langkah-langkahnya:

1. Pilih tab `Entities Tab'. Pilih kelas Pizza pada hirarki kelas.
2. Tekan ikon ‘Add' di sebelah `Instances' pada `Class Description View' untuk membuat individu MargheritaPizza yang merupakan anggota dari kelas Pizza. Ini akan membuka kotak dialog dimana individu bisa dipilih/dibuat.
3. Pada kotak dialog, tekan tombol ‘Add individual’. Ini akan memunculkan text box untuk menuliskan nama individu baru.
4. Tuliskan nama individu, MargheritaPizza, seperti terlihat pada Gambar 4.21. Tekan tombol ‘OK’ untuk membuat individu baru, MargheritaPizza.
5. Tekan tombol ‘OK’ untuk membuat individu (MargheritaPizza) pada kelas Pizza dan menutup kotak dialog.



Gambar 4.21 Membuat contoh individu pada kelas Pizza

Exercise 48: Membuat restriction datatype untuk menyatakan bahwa semua Pizzas mempunyai nilai kalori.

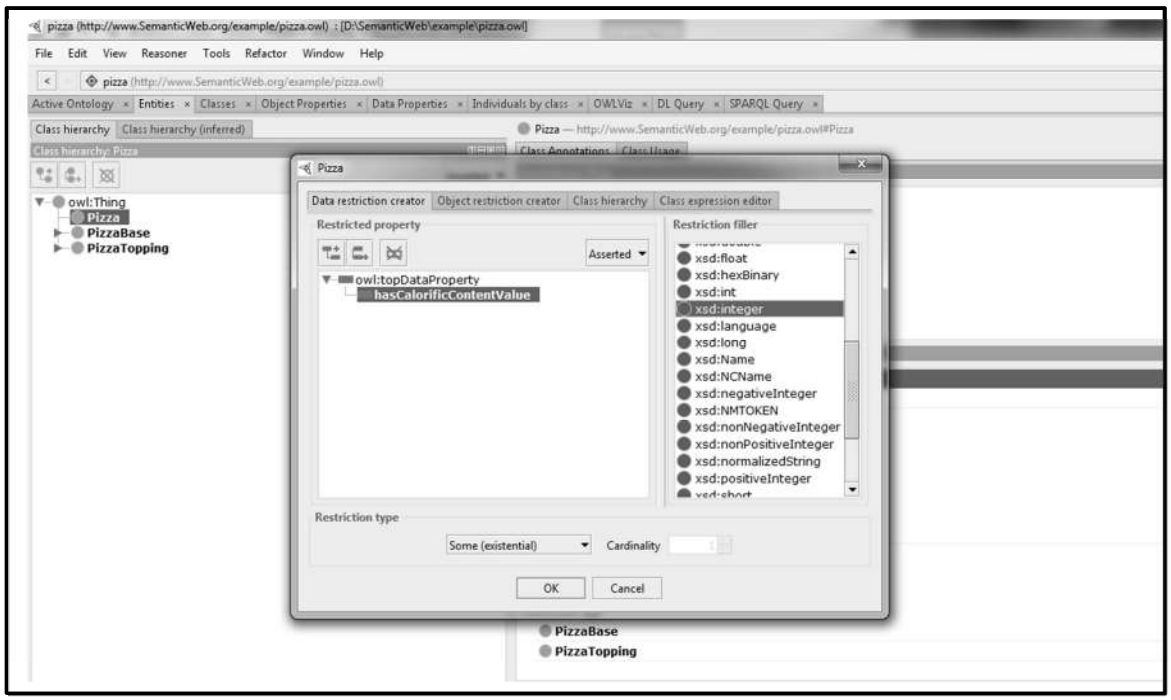
Langkah-langkahnya:

1. Pilih tab `Entities' atau `Classes'.
2. Pilih kelas Pizza. Pada `Class Description view' tekan ikon ‘Add’ di sebelah ‘Subclass Of’. Ini akan memunculkan kotak dialog untuk membuat property restriction.

3. Pada kotak dialog, pilih tab `Data restriction creator' seperti ditunjukkan pada Gambar 4.22.Ini sama seperti pada pembuatanrestrictio untuk properti object , tetapi nilai propertinya adalah datatype.
4. Pilih ‘Restriction type’-nya adalah `some'.
5. Pilih ‘Restricted property’-nya adalah hasCalorificContentValue.
6. Terakhir, pilih ‘Restriction filler’-nya adalah datatype integer.
7. Tekan `OK'. Restriction ‘hasCalorificContentValue some integer’ akan muncul pada subclass of dari kelas Pizza.

Selain menggunakan tipe datatype yang telah terdefinisi, datatype dengan menentukan batasan pada nilai yang mungkin dapat juga digunakan. Misalnya, mudah untuk menentukan berbagai nilai untuk sebuah angka.

Dengan menggunakan properti datatype yang telah dibuat, akan dibuat kelas yang didefinisikan yang menentukan berbagai nilai yang menarik. Akan dibuat definisi yang menggunakan aspek MinInclusive dan maxExclusive yang dapat diterapkan pada tipe data numerik. HighCaloriePizza akan akan didefinisikan sebagai pizza yang memiliki nilai kalori sama dengan atau lebih tinggi dari 400.

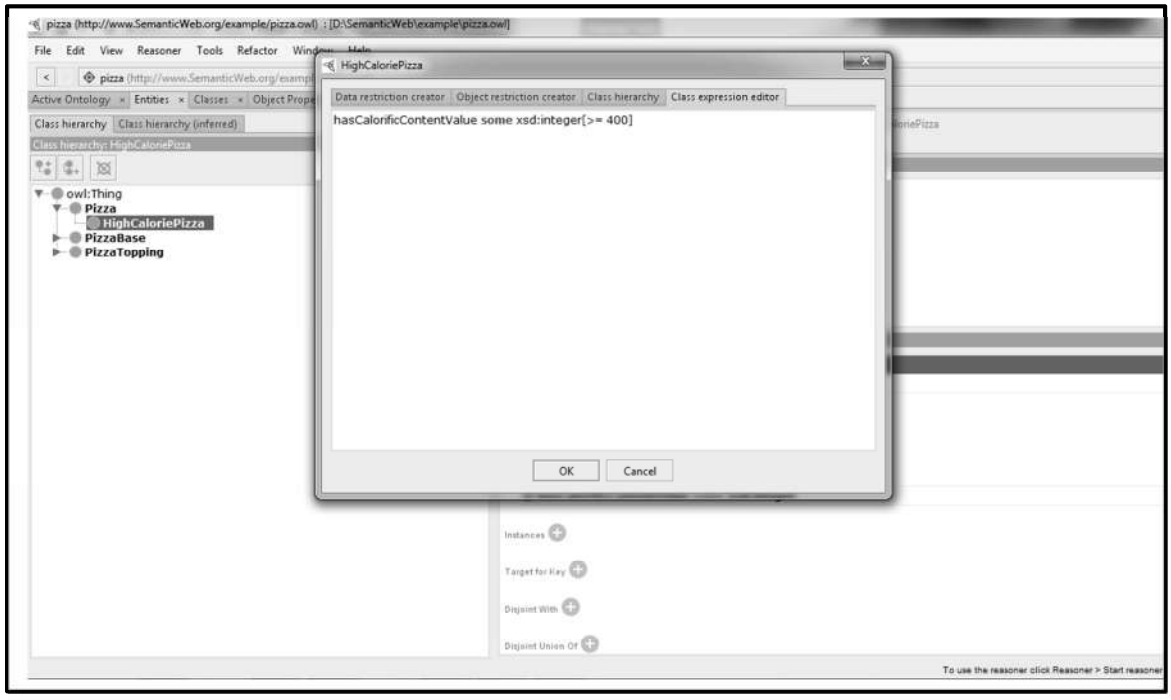


Gambar 4.22 Membuat restriction pada properti datatype

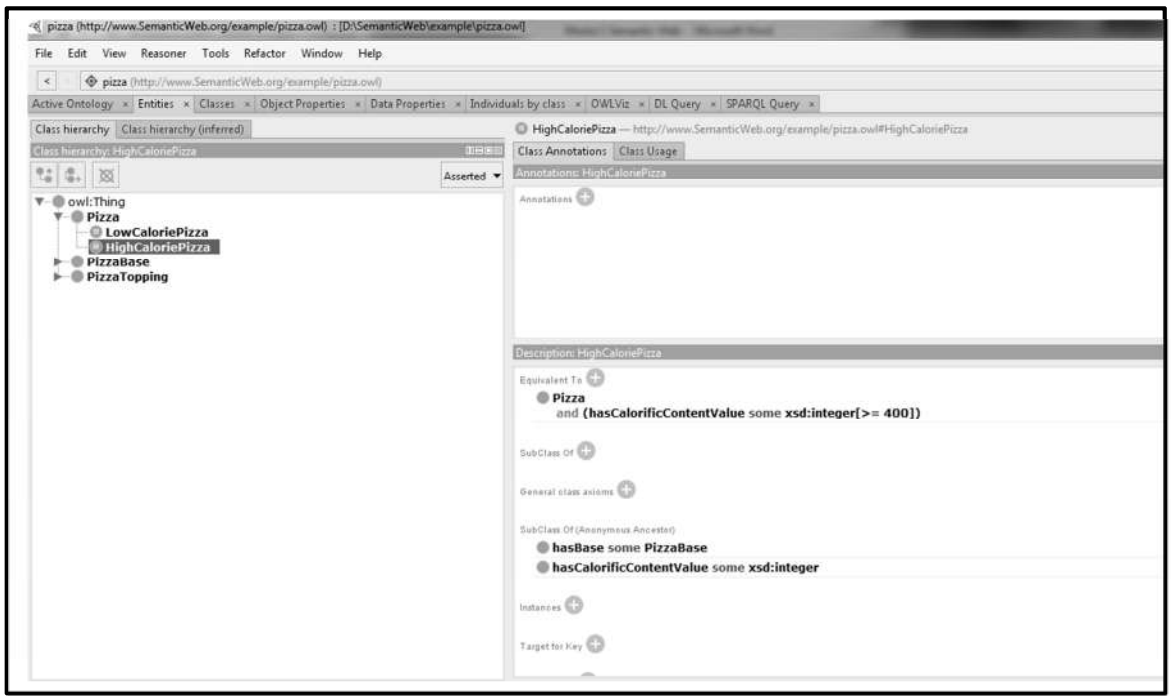
Exercise 49: Membuat kelas HighCaloriePizza yang mempunyai nilai kalori lebih besar atau sama dengan 400.

Langkah-langkahnya:

1. Pilih tab `Classes' atau `Entities'.
2. Buat subkelas dari kelas Pizza yang disebut HighCaloriePizza.
3. Pada `Class Description view' klik ikon `Add' pada `Subclass Of' untuk menambahkan restriction baru.
4. Pada `Class expression editor', ketik `hasCalorificContentValue some xsd:integer[>= 400]', lihat Gambar 4.23, and click `OK'.
5. Ubah kelas ke 'defined class' (`Ctrl-D'), seperti terlihat pada Gambar 4.24.
6. Buat kelas LowCaloriePizza dengan cara yang sama, tetapi didefinisikan equivalent to Pizza dan hasCalorificContentValue some xsd:integer[< 400].



Gambar 4.23 Membuat restriction property datatype: Class expression editor



Gambar 4.24 Mengkonversi kelas ke ‘defined class’

Daftar Pustaka

[1] Matthew Horridge, *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools*, Edition 1.3, The University Of Manchester, 2011

V. PERTEMUAN IV: SPARQL

5.1. Pendahuluan

SPARQL adalah bahasa query pencocokan grafik untuk RDF. SPARQL adalah akronim dari Protocol and RDF Query Language. Query SPARQL terdiri dari tiga bagian:

- Pattern matching: optional, union, nesting, filtering.
- Solution modifiers: projection, distinct, order, limit, offset.
- Output part: construction of new triples, …

Selain bahasa inti, W3C juga mendefinisikan:

- Protokol SPARQL untuk spesifikasi RDF: ia mendefinisikan protokol jarak jauh untuk mengeluarkan kueri SPARQL dan menerima hasilnya.
- Spesifikasi Hasil XML SPARQL Query Format: mendefinisikan format dokumen XML untuk mewakili hasil query SPARQL.
- Query Federate SPARQL: ini memperluas SPARQL untuk mengeksekusi query yang didistribusikan melalui endpoint SPARQL yang berbeda.
- Deskripsi Layanan SPARQL: ini adalah metode untuk menemukan dan kosa kata untuk mendeskripsikan layanan SPARQL.

Terdapat beberapa tantangan dalam mendesain query SPARQL. SPARQL harus mempertimbangkan fitur khas RDF. Selain itu, SPARQL harus bisa mengekstrak informasi dari grafik RDF yang saling berhubungan. SPARQL juga harus konsisten dengan semantik open-world RDF. Sebaiknya SPARQL menawarkan kemungkinan penambahan informasi pilihan jika ada, dan harus bisa menafsirkan grafik RDF dengan benar dengan kosakata dengan semantik yang telah ditentukan. Sebagai tambahan, SPARQL sebaiknya tawarkan beberapa fungsi untuk navigasi RDF grafik RDF.

5.2. Struktur Basic Query SPARQL

Terdapat empat bagian utama dari strutur query SPARQL yaitu: prefix, select, from, dan where. PREFIX adalah setara dengan pendeklarasian namespace XML. SELECT sangat mirip dengan sintaks SELECT pada bahasa SQL, yaitu digunakan untuk mendefinisikan data/variabel yang ingin di-querying. FROM menentukan data yang digunakan mengeksekusi query. WHERE mendefinisikan bagian dari grafik RDF yang diinginkan.

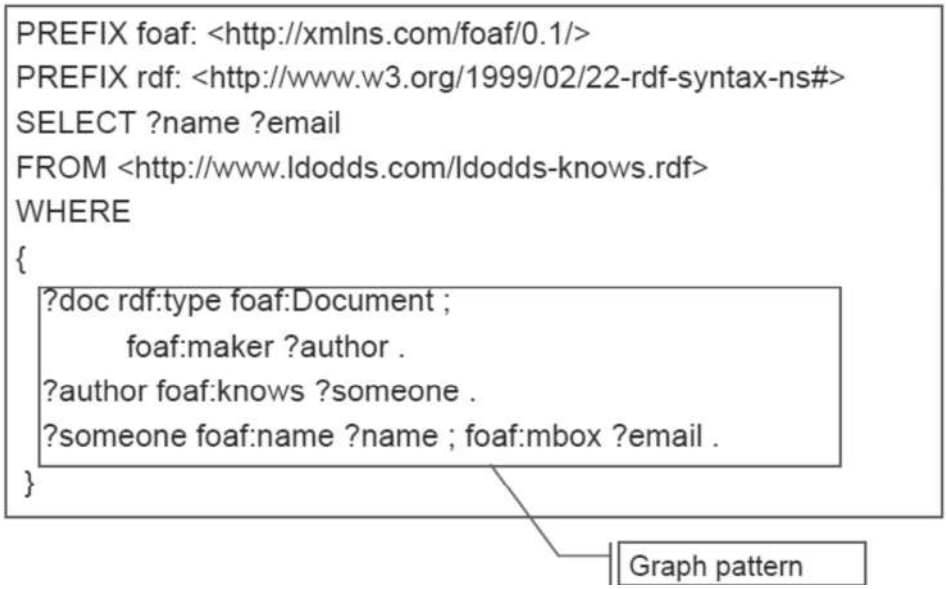
Beberapa catatan tentang sintaks SPARQL:

- Variabel diawali oleh salah satu: "? " atau "$“ keduanya bisa dipertukarkan.
- Blank nodes are ditandai dengan bentuk label, seperti: "_:abc", atau bentuk singkat "[]".
- Titik (.) memisahkan pola triple
- Titik koma (;) memisahkanpola triple dengan common subject (subjek yang digunakan bersama).

Sebagai contoh query SPARQL adalah pencarian nama seseorang. Seperti bisa dilihat pada Gambar 5.1, (?x foaf: name ? name) adalah sebuah pola triple. Contoh yang lain adalah pencarian nama dan email seseorang (lihat Gambar 5.2). Pada Gambar 5.2, bisa dilihat bagian yang merupakan pola grafik.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
  ?x rdf:type foaf:Person.
  ?x foaf:name ?name.
}
```

Gambar 5.1 Pola triple



Gambar 5.2 Pola grafik

5.3. Tipe Query SPARQL

Terdapat empat tipe query SPARQL yaitu:

1. Query SELECT menghasilkan binding variabel;
2. Query CONSTRUCT menghasilkan grafik RDF yang ditentukan oleh template grafik.
3. Query ASK dapat digunakan untuk menguji apakah sebuah pola grafik punya penyelesaian atau tidak. Tidak ada informasi yang dihasilkan tentang penyelesaian query yang mungkin, hanya informasi apakah ada penyelesaian atau tidak;
4. Query DESCRIBE yang tidak penting dan SPARQL tidak merumuskan sematik untuk itu.

Urutan Solusi dan Modifier

Pola grafik dalam klausa WHERE menghasilkan koleksi solusi yang tidak berurutan, masing-masing solusi menjadi pemetaan, fungsi parsial dari variabel ke persyaratan RDF (URI, Literals, Bnodes), atau disebut juga variable binding. Solusi ini kemudian diperlakukan sebagai urutan awalnya tanpa urutan tertentu yang digunakan untuk menghasilkan hasil query SPARQL.

Modifier urutan penyelesain dapat diterapkan untuk membuat urutan lain, diantaranya:

- Order modifier: letakkan solusinya dalam beberapa urutan tertentu

- Projection modifier: pilih variabel tertentu. Hal ini dilakukan dengan menggunakan klausa SELECT
- Distinct modifier: memastikan solusi dalam urutan unik
- Reduced modifier : memungkinkan penghapusan beberapa solusi yang tidak unik
- Offset modifier: mengontrol dari mana solusi dimulai dari, dalam rangkaian keseluruhan solusi
- Limit modifier: membatasi jumlah dari solusi

5.4. Pola Grafik SPARQL

Untuk mendefinisikan pola grafik, pola triple harus didefinisikan dahulu. Pola triple adalah seperti triple RDF, tetapi dengan option variabel sebagai pengganti term RDF (i.e., IRIs, literal atau blank nodes) pada tempat subjek, predikat atau objek.

Pola grafik dapat dibedakan menjadi beberapa pola grafik:

- Pola grafik Basic
- Pola grafik Group
- Pola grafik Optional
- Pola grafik Alternative
- Kondisi Filter

Pola Grafik Basic

Pola grafik basic adalah sekumpulan pola triple yang ditulis sebagai sekuen dari pola triple (dipisah oleh tanda titik). Pola grafik basic adalah conjunctive dari pola-pola triple-nya.

Example:

```
?x foaf:name ?name . ?x foaf:mbox ?mbox
```

Pola Grafik Grup

Pola grafik grup adalah sekumpulan pola grafik yang dibatasi dengan kurung kurawal { }. Misalnya: { P1 P2 } dan {} (yaitu pola grafik grrup yang kosong).

Contoh: { ?x foaf:name ?name . ?x foaf:mbox ?mbox }

```
{ ?x foaf:name ?name . ?x foaf:mbox ?mbox . }
{ { ?x foaf:name ?name . } { ?x foaf:mbox ?mbox . } }
```

Ketiga pola grafik grup diatas adalah sama. Ketika pola grafik grup mengandung hanya pola triple atau hanya pola grafik basic. Pola ini ditafsirkan secara konjungtif, dan pola grafik grup setara dengan sekumpulan pola triple yang sesuai.

Pola Grafik Optional

Struktur yang regular dan komplit, tidak bisa diasumsikan pada semua grafik RDF sehingga sangat berguna untuk memformulasikan query dengan mengijinkan penambahan informasi (optional) ke jawaban. Jawaban tidak akan ditolak hanya karena beberapa bagian dari pola query tidak cocok. Pencocokan pola grafik optional menyediakan fasilitas ini: jika bagian optional tidak cocok, tidak akan dibuat binding tetapi jawaban tidak akan dihapus. Bagian optional dari sebuah pola grafik kompleks yang dihitung bisa ditentukan dengan mulai dengan pola grafik P1 dan kemudian gunakan keyword OPTIONAL ke pola grafik seperti berikut:

```
{P1 OPTIONAL { P2 }}
```

Pola grafik grup yang mengikuti keyword OPTIONAL bisa menjadi sekompleks mungkin, misalnya, dapat mengandung FILTER.

Pola grafik Alternative

SPARQL menyediakan metode untuk membentuk disjungsi dari pola grafik dengan keyword UNION sehingga satu dari beberapa pola grafik alternatif bisa cocok. Jika lebih dari satu dari grapfik alternatif tersebut cocok, semua penyelesaian yang memungkinkan ditemukan. Pola-pola grafik alternatif yang dikombinasikan dengan UNION diproses secara terpisah satu dengan yang lain dan hasilnya dikombinasikan menggunakan operator union. Misalnya:

```
{P1} UNION {P2}
```

Ekspresi FILTER

FILTER membangun batasan-batasan binding variabel yang mana ekspresi filter menghasilkan nilai TRUE dengan mempertimbangkan beberapa hal yaitu:

- equality = diantara variabel dan term RDF
- unary predicate bound
- Kombinasi Boolean (∧, ∨, ¬)

– Pola grafik grup yang digunakan membatasi cakupan kondisi FILTER

Kondisi FILTER adalah pembatasan dalam penyelesaian dari keseluruhan grafik grup dimana filter digunakan, misalnya:

P1 . P2 . FILTER (... boolean expression...)

5.5. Contoh

Contoh 1: Pola Grafik Grup

- **Data**:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

- **Query**:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox }
```

- **Result**:

?name	?mbox
"Peter Goodguy"	<mailto:peter@example.org>
"Johnny Lee Outlaw"	<mailto:jlow@example.com>

Contoh 2: Pola Grafik Optional

Data:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntaxns#>.
_:a rdf:type foaf:Person .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@example.com> .
_:a foaf:mbox <mailto:alice@work.example> .
_:b rdf:type foaf:Person .
_:b foaf:name "Bob" .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        OPTIONAL { ?x foaf:mbox ?mbox }
      }
```

Result:

?name	?mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

Contoh 3: Pola Grafik Alternatif

Data:

```
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .
_:a dc10:title "SPARQL Query Language Tutorial" .
_:a dc10:creator "Alice" .
_:b dc11:title "SPARQL Protocol Tutorial" .
_:b dc11:creator "Bob" .
_:c dc10:title "SPARQL" .
_:c dc11:title "SPARQL (updated)" .
```

Query:

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE {{?book dc10:title ?title
       UNION
       {?book dc11:title ?title
      }
```

Result:

?title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

Query:

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?author ?title
WHERE {{?book dc10:title ?title. ?book dc10:creator ?author.}
       UNION
       {?book dc11:title ?title. ?book dc11:creator ?author.}
      }
```

Result:

?author	?title
"Bob"	"SPARQL Protocol Tutorial"
"Alice"	"SPARQL Query Language Tutorial"

Contoh 4: Query dengan literal RDF

Data:

```
@prefix dt: <http://example.org/datatype#> .
@prefix ns: <http://example.org/ns#> .
@prefix : <http://example.org/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
:x ns:p "cat"@en .
:y ns:p "42"^^xsd:integer .
:z ns:p "abc"^^dt:specialDatatype .
```

- The queries `SELECT ?v WHERE { ?v ?p "cat" }` and `SELECT ?v WHERE { ?v ?p "cat"@en }` have different results
- Only the second one finds a matching triple and returns:

?v
<http://example.org/ns#x>

Contoh 5: Constraints on Variables

Data:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .
:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Query:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
FILTER (?price < 30.5)
?x dc:title ?title . }
```

Result:

?title	?price
"The Semantic Web"	23

5.6. Latihan

1. Assume you run the query on the given RDF data. Construct the RDF graph and list the results of the query below.

RDF Data:

```
@prefix ex: <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
_:a dc:title "Very good introductory Textbook" .
_:a ex:website <http://kti.tugraz.at/examples/intro-tb-website> .
_:b dc:title "Exceptionally even better intermediate Textbook" .
_:b ex:website <http://kti.tugraz.at/examples/intermediate-tb-website> .
_:c ex:website <http://kti.tugraz.at/examples/tb-website-no-title> .
```

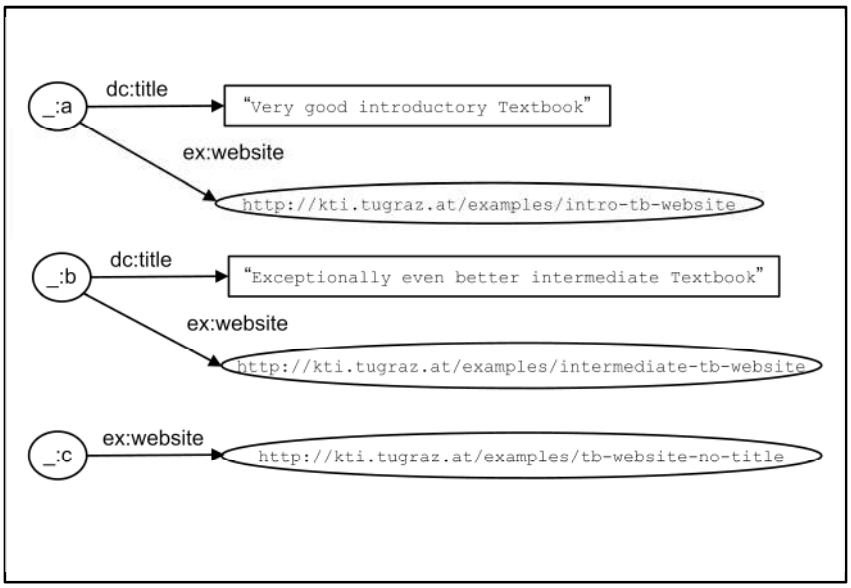
Query:

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT *
WHERE { ?book dc:title ?title .
        ?book ex:website ?website }
```

Answer:

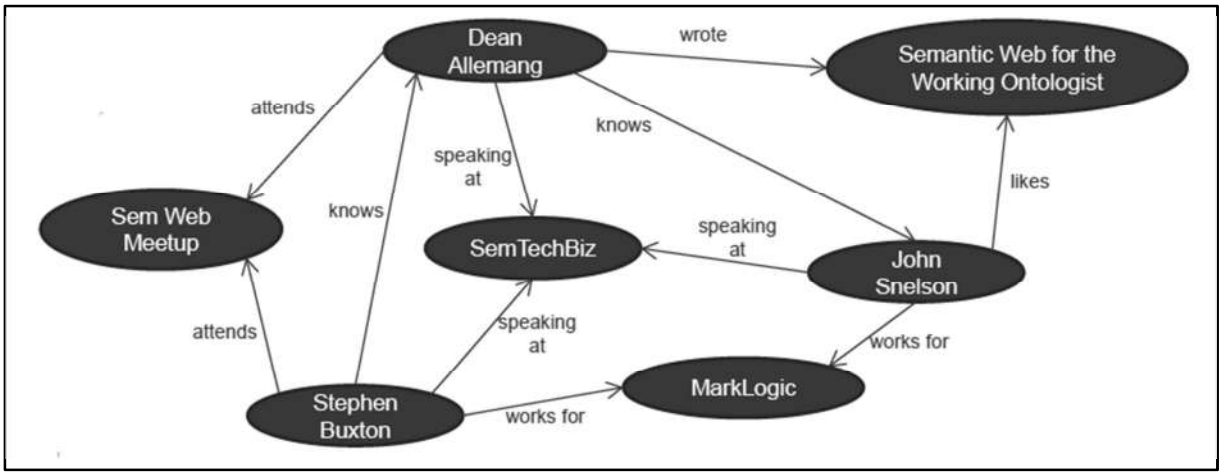
RDF Graph:



Query Results:

book	title	website
_:a	"Very good introductory Textbook"	http://kti.tugraz.at/examples/intro-tb-website
_:b	"Exceptionally even better intermediate Textbook"	http://kti.tugraz.at/examples/intermediate-tb-website

2. Construct a SPARQL query using the RDF graph below to answer the question, "Who works at Snelson's company and knows the author of a book Snelson likes?". Note: use the namespace Ex.



Answer:

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?person
WHERE { ?person Ex:works-for ?company ;
         Ex:knows ?author .
         JohnSnelson Ex:works-for ?company ;
         Ex:likes ?book .
         ?author Ex:wrote ?book .
       }
```

3. Use the classes, object properties, and datatype properties presented in the problem 2. Suppose the ontology constructed has URI: http://www.semanticweb.org/2017/UAS and some data (object property and datatype property assertions) available in the ontology are as follows.
   - A1 is an instance of Admin that has NIK = 11111, name = "Wayan Bagus", role = "Admin", and address = "Banjar Dangin Peken, Badung".
   - A1 posts Article1 (date = 07/03/2017; category = "news"; title = "the latest product offered"; content = "Our company offers some new products ….") and Article5 (date = 0=26/05/2017; category="event"; title="Grand Opening"; content = "The grand opening of ….").
   - C1 is an instance of Customer that has NIK = 22222, name = "Kadek Jegeg", role = "Customer", and address = "Banjar Dauh Puri, Karangasem".
   - C3 is an instance of Customer that has NIK = 34345, name = "Ngurah Rai", role = "Customer", and address = "Banjar Dangin Tukad, Tabanan".
   - C7 is an instance of Customer that has NIK = 76588, name = "Lanang Oka", role = "Customer", and address = "Banjar Jero Dauh, Negara".
   - C1 give some comments to Article5 and recorded in Comment15 (date=28/05/2017; content="It is great…."). On the other words, Comment15 is posted by C1 and Comment15 is posted to Article5.
   - C7 give some comments to Article5 and recorded in Comment20 (date=29/05/2017; content="The opening was so spectacular …."). On the other words, Comment20 is posted by C7 and Comment20 is posted to Article5.

   a. Assume you run the query on the given data. Construct the SPARQL query to list all customers, the information includes the name and the address. List the results.
   b. Assume you run the query on the given data. Construct the SPARQL query to list all comment posted by customers, the information includes the date and the content. List the results.
   c. Assume you run the query on the given data. Construct the SPARQL query to list the customer names who gives comment to article with title "Grand Opening". List the results.
   d. Assume you run the query on the given data. Construct the SPARQL query to list the customer names who give comments on 28/05/2017.

Answer:

a. SELECT (STR(?n) AS ?name) (STR(?a) AS ?address)
WHERE {
?x rdf:type UAS:Customer.
?x UAS:name ?n.
?x UAS:address ?a}

Output:

name	address
Kadek Jegeg	Banjar Dauh Puri, Karangasem
Ngurah Rai	Banjar Dangin Tukad, Tabanan
Lanang Oka	Banjar Jero Dauh, Negara

b. SELECT (STR(?d) AS ?date) (STR(?c) AS ?content)
WHERE {
?x rdf:type UAS:Comment.
?x UAS:date ?d.
?x UAS:content ?c}

Output:

date	content
28/05/2017	”It is great….”
29/05/2017	=”The opening was so spectacular ….”

c. SELECT (STR(?n) AS ?name)
WHERE {
?x rdf:type UAS:Comment.
?C1 rdf:type UAS:Customer.
?A1 rdf:type UAS: Article.
?x UAS:isPostedBy ?C1.
?x UAS:isPostedTo ?A1.
?A1 UAS:title ?title.
FILTER (STR(?title) = “Grand Opening”).
?C1 UAS:name ?n}

Output:

name
Lanang Oka
Kadek Jegeg

d. SELECT (STR(?n) AS ?name)
WHERE {
?x rdf:type UAS:Comment.
?C rdf:type UAS:Customer.
?x UAS:isPostedBy ?C.
?x UAS:date ?date.
FILTER(STR(?date) = “28/05/2017”).
?C UAS:name ?n}

Output:

Name
Kadek Jegeg

Daftar Pustaka

[1] Bob DuCharme, Learning SPARQL, Second Edition, O’Reilly Media, 2013