ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN VĂN UY

SUY DIỄN TRÊN MÔ HÌNH BẢN THỂ HỌC VÀ ỨNG DỤNG

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

HÀ NỘI - 2016

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

NGUYỄN VĂN UY

SUY DIỄN TRÊN MÔ HÌNH BẢN THỂ HỌC VÀ ỨNG DỤNG

Ngành: Công nghệ thông tin

Chuyên ngành: Truyền dữ liệu và Mạng máy tính

LUẬN VĂN THẠC SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. LÊ ĐÌNH THANH

HÀ NỘI - 2016

LỜI CAM ĐOAN

Tôi xin cam đoan luận văn "*Suy diễn trên mô hình bản thể học và ứng dụng*" là do tôi thực hiện dưới sự hướng dẫn của TS. Lê Đình Thanh. Tất cả nội dung tham khảo đều có trích dẫn rõ ràng, trung thực.

Tôi xin hoàn toàn chịu trách nhiệm và chịu mọi hình thức kỷ luật theo quy định cho lời cam đoan của mình.

Hà nội, ngày 23 tháng 11 năm 2016

Người cam đoan

Nguyễn Văn Uy

LỜI CẢM ƠN

Lời đầu tiên tôi xin chân thành cảm ơn các thầy cô trong Khoa Công nghệ thông tin, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội, các thầy cô đã giảng dạy, chỉ dẫn và truyền đạt cho tôi những kiến thức quý báu trong suốt thời gian học tập và nghiên cứu tại trường.

Tôi xin bày tỏ sự cảm ơn đặc biệt tới thầy TS. Lê Đình Thanh đã định hướng cho tôi trong lựa chọn đề tài, đưa ra những nhận xét quý giá và trực tiếp hướng dẫn tôi trong suốt quá trình nghiên cứu và hoàn thành luận văn tốt nghiệp này.

Tôi xin chân trọng cảm ơn gia đình, bạn bè, đồng nghiệp đã động viên, hưởng ứng, giúp đỡ nhiệt tình đến tôi trong quá trình học tập và nghiên cứu.

Hà nội, ngày 23 tháng 11 năm 2016

Người thực hiện

Nguyễn Văn Uy

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Tên thuật ngữ	Tên tiếng Anh	Tên tiếng việt
API	Application Programming Interface	
BGP	Basic Graph Pattern	
DAML	Darpa Agent Markup Language	
GGP	Group Graph Pattern	
IRI	Internationnalized Resource Identifier	
OWL	Ontology Web Language	
Ontology		Bản thể học
RDF	Resource Description Framework	
RDFS	Resource Description Framework Schema	
SQL	Structured Query Language	
URI	Uniform Resource Locator	
WWW	World Wide Web	
XML	Extensible Markup Language	
RuleML	Rule Markup Language	
SWRL	Semantic Web Rule Language	
SPARQL		Ngôn ngữ truy vấn RDF
Class		Lớp
Property		Thuộc tính
Individuals		Các cá thể
HTML	HyperText Markup Language	Ngôn ngữ đánh dấu siêu văn bản

DANH MỤC CÁC HÌNH VẼ MINH HỌA

DANH MỤC CÁC BẢNG BIỂU

MỞ ĐẦU

Ngày nay, hầu như các thông tin cần thiết trong mọi mặt của đời sống xã hội như y tế, giáo dục, kinh tế, chính trị, pháp luật,… có thể dễ dàng tìm thấy trên môi trường mạng Internet. Người sử dụng Web có thể tìm ra thông tin này bằng cách chỉ ra địa chỉ URL và theo các liên kết để tìm ra các tài nguyên mong đợi.

Với nhu cầu thông tin ngày càng lớn của con người, khả năng đáp ứng thông tin càng trở lên bức thiết. Kỹ thuật Web hiện nay vẫn tạo khó khăn trong việc rút trích, bảo trì và phát triển thông tin. Máy tính chỉ được dùng như một thiết bị gửi và trả thông tin. Chúng không thể truy xuất những khả năng thực sự cần thiết, do đó chúng chỉ hỗ trợ ở mức giới hạn nào đó trong việc truy xuất và xử lý thông tin. Kết quả là người sử dụng phải phải gánh trên vai trách nhiệm không những truy cập và xử lý thông tin mà còn rút trích và thông dịch mọi thông tin.

Để khắc phục những yếu điểm của Web hiện tại khái niệm "Semantic Web" đã ra đời. Semantic Web là sự mở rộng của Web hiện tại mà trong đó thông tin được xử lý một cách tự động bằng máy tính, làm cho con người và máy tính có thể hợp tác với nhau.

Mô hình bản thể học về một lĩnh vực chuyên môn là một cấu trúc dữ liệu được xây dựng một cách đơn giản, cô đọng, nhưng cũng phải đầy đủ. Mục tiêu là mô tả rõ ràng nhất tri thức lĩnh vực chuyên môn này. Suy diễn trên mô hình bản thể học là thao tác giúp ta khai thác hiệu quả trên bản thể học này, bởi vì nếu không thực hiện quá trình suy diễn thì bản thể học chỉ có chức năng như kho chứa mà thôi. Suy diễn bằng các quy tắc có thể suy ra kiến thức mới, kiến thức tiềm ẩn cần thiết dựa trên những sự kiện được biết đến trước đó đã mang lại những sự hiệu quả to lớn cho thế hệ Web ngữ nghĩa.

Luận văn nghiên cứu các suy diễn trên mô hình bản thể học bằng cách xây dựng các tập quy tắc suy diễn, qua đó củng cố thêm sự mô tả mô hình bản thể học về một lĩnh vực, cuối cùng là việc xây dựng một ứng dụng trong đó sử dụng những nghiên cứu cho lý thuyết này, với tên: "Chương trình hỗ trợ cho người làm vườn".

Ngoài phần mở đầu và kết luận, nội dung của luận văn được chia làm 3 chương, trong đó:

Chương 1. Mô hình bản thể học: Trình bày các khái niệm chính trong đề tài, nhắc lại các khái niệm cơ bản về Semantic Web và Bản thể học. Chương này sẽ giải đáp các

câu hỏi như Semantic Web là gì, lợi ích của nó, thành phần của Semantic Web. Đi sâu nghiên cứu về Ontology, cấu trúc của nó, cách thức xây dựng Ontology.

Chương 2. Suy diễn trên mô hình bản thể học: Trong chương này chúng tôi trình bày các nội dung xây dựng các tập luật để suy diễn trên mô hình ontology với các ngôn ngữ RuleML, SWRL và SPARQL. Cấu trúc và sự hỗ trợ của máy suy diễn Jena Framework. Ví dụ về việc suy diễn trên mô hình bản thể học cụ thể.

Chương 3. Phát triển ứng dụng thử nghiệm: Xây dựng một hệ thống Semantic Web trong đó bao gồm việc xây dựng mô hình bản thể học (ontology model), Suy diễn trên mô hình bản thể học này bằng cách xây dựng các tập luật suy diễn cho mỗi chức năng của hệ thống.

Cuối cùng là việc tổng kết những kết quả đạt được và những mặt hạn chế của đề tài, đồng thời phát triển hệ thống trong tương lai.

CHƯƠNG 1. MÔ HÌNH BẢN THỂ HỌC

Sau chương này sẽ trả lời được các câu hỏi sau:

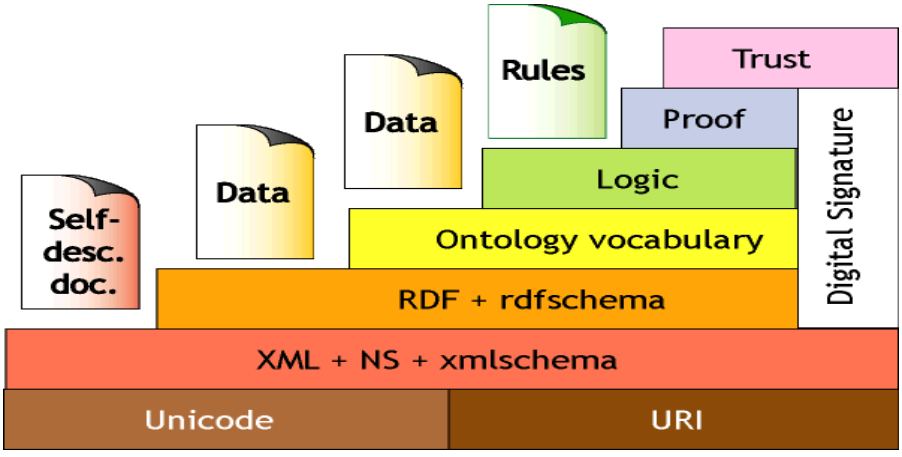
- *Bản thể học là gì, mô hình bản thể học là gì?*
- *Công nghệ Semantic Web là gì?*
- *Phương pháp xây dựng bản thể học?*
- *Ngôn ngữ OWL xây dựng bản thể học, và công cụ hỗ trợ xây dựng bản thể học như thế nào?*

1.1. Giới thiệu

Ngày nay, hầu như các thông tin cần thiết trong mọi mặt của đời sống xã hội như y tế, giáo dục, kinh tế, chính trị, pháp luật,… có thể dễ dàng tìm thấy trên môi trường mạng Internet. Người sử dụng Web có thể tìm ra thông tin này bằng cách chỉ ra địa chỉ URL và theo các liên kết để tìm ra các tài nguyên mong đợi.

Với nhu cầu thông tin ngày càng lớn của con người, khả năng đáp ứng thông tin càng trở lên bức thiết. Kỹ thuật Web hiện nay vẫn tạo khó khăn trong việc rút trích, bảo trì và phát triển thông tin. Máy tính chỉ được dùng như một thiết bị gửi và trả thông tin. Chúng không thể truy xuất những khả năng thực sự cần thiết, do đó chúng chỉ hỗ trợ ở mức giới hạn nào đó trong việc truy xuất và xử lý thông tin. Kết quả là người sử dụng phải phải gánh trên vai trách nhiệm không những truy cập và xử lý thông tin mà còn rút trích và thông dịch mọi thông tin.

Để khắc phục những yếu điểm của Web hiện tại khái niệm "Semantic Web" đã ra đời. Semantic Web là sự mở rộng của Web hiện tại mà trong đó thông tin được xử lý một cách tự động bằng máy tính, làm cho con người và máy tính có thể hợp tác với nhau.



Hình 1-1: Cấu trúc của Semantic Web [1]

Công nghệ "Semantic Web" gắn liền với nó là sự ra đời và phát triển của bản thể học (Ontology) ở đó người ta mô tả dữ liệu coi đó là tài nguyên và ý tưởng là gắn mỗi tài nguyên này với một mô tả gọi chung là URI, qua đó mà định nghĩa ra các siêu dữ liệu RDF (Resource Description Framework), sau đó cho phép truy vấn và suy diễn (inference), chính điều này đã tạo ra tính ngữ nghĩa, thông minh đáp ứng nhu cầu cần thiết mà thế hệ web mới yêu cầu.

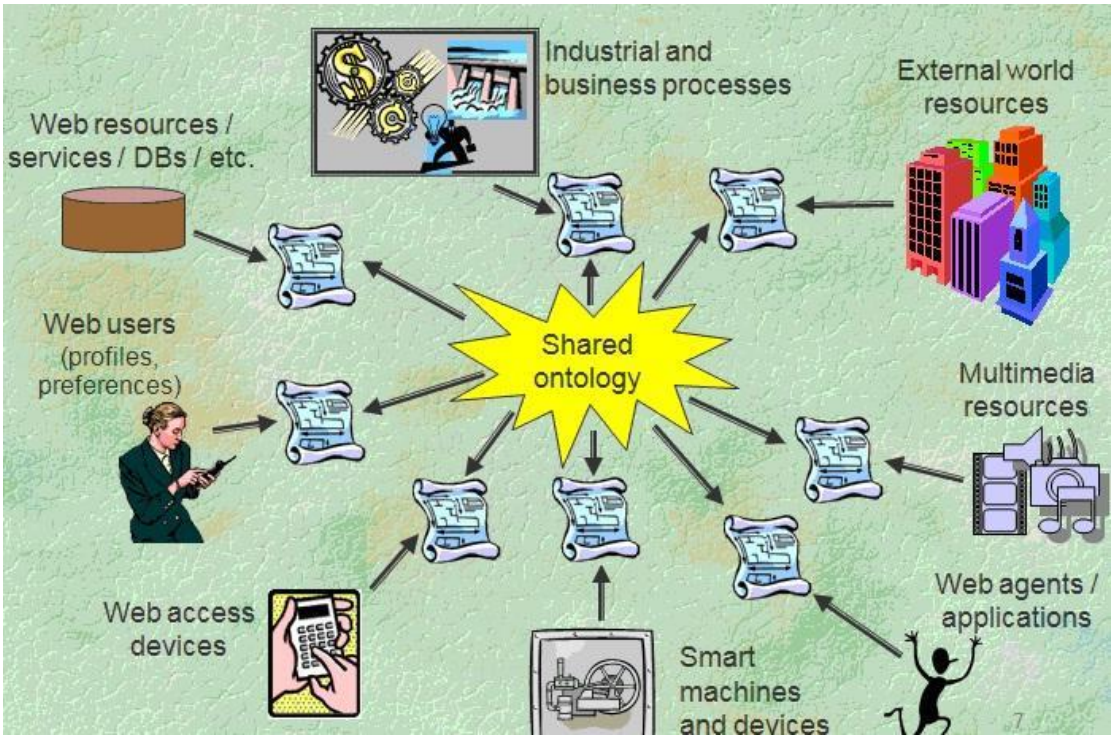
Việc xây dựng ra bản thể học về một lĩnh vực chuyên môn của cuộc sống yêu cầu phải thực sự đơn giản, cô đọng, nhưng cũng phải đầy đủ. Mục tiêu là mô tả rõ ràng nhất tri thức lĩnh vực, giúp người dùng dùng khai thác thông tin một cách dễ dàng và hiệu quả nhất. Suy diễn trên mô hình bản thể học là thao tác giúp ta khai thác hiệu quả và quan trọng nhất cho công việc này, bởi vì nếu không có nó thì bản thể học chỉ có chức năng như kho chứa mà thôi. Suy diễn với các quy tắc có thể suy ra kiến thức mới, kiến thức tiềm ẩn cần thiết dựa trên những sự kiện được biết đến trước đó đã mang lại những sự hiệu quả to lớn cho thế hệ Web ngữ nghĩa.

1.2. Bản thể học

Bản thể học (ontology) là tập từ vựng để mô hình hóa thế giới bên ngoài [11]. Nó đưa ra các khái niệm cơ bản và định nghĩa quan hệ giữa các khái niệm đó trong một miền lĩnh vực. Đồng thời ontology còn cung cấp các ràng buộc, là các giả định cơ sở về ý nghĩa mong muốn của bộ từ vựng.

Ontology được xây dựng nhằm các mục đích sau:

- Chia sẻ hiểu biết chung về cấu trúc thông tin giữa con người và phần mềm agent
- Sử dụng lại tri thức về một miền lĩnh vực đã được xây dựng từ trước

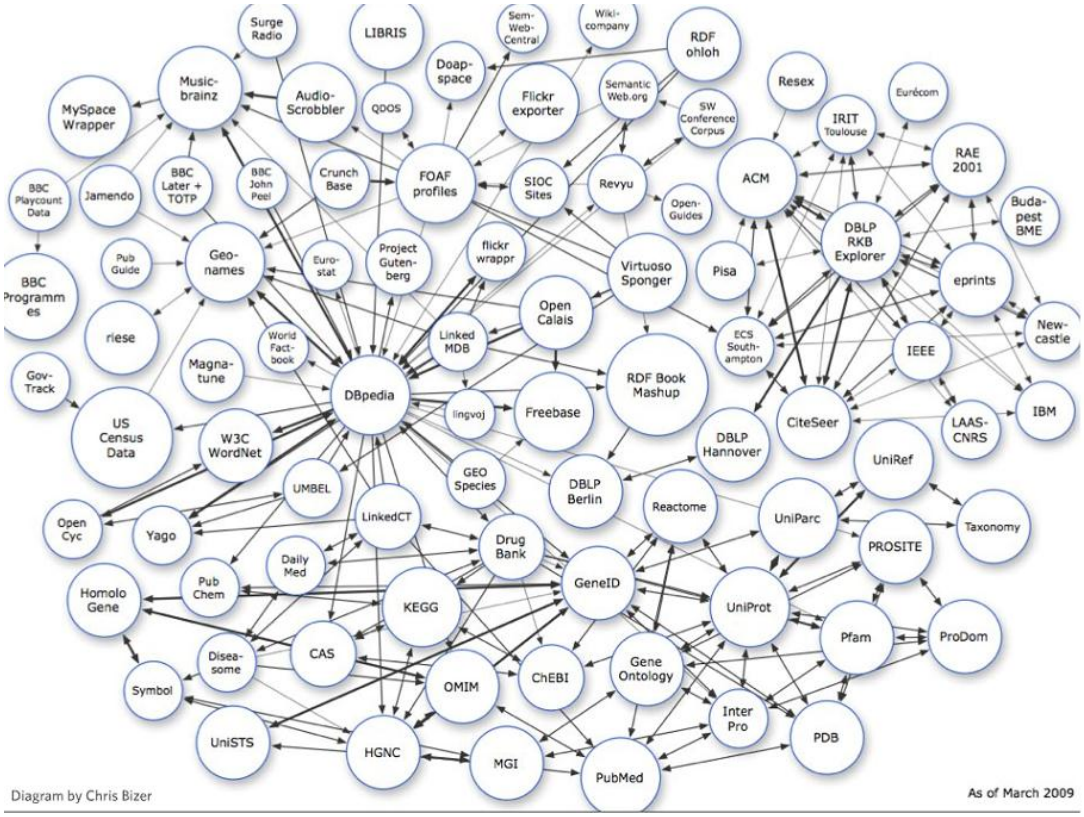


Hình 1-2: Chia sẻ bản thể học cho các ứng dụng [2].

Trong hình trên ta thấy các ứng dụng khác nhau, muốn trao đổi thông tin với nhau thì cần phải có một tri thức chung, vì vậy các ứng dụng này đều sử dụng một ontology để có thể chia sẻ tri thức cho nhau.

Ontology được sử dụng rộng rãi trong công nghệ tri thức, trí tuệ nhân tạo, và khoa học máy tính trong các ứng dụng liên quan đến quản lý tri thức, xử lý ngôn ngữ tự nhiên, thương mại điện tử, tích hợp thông tin, tìm kiếm thông tin, thiết kế cơ sở dữ liệu…

Một hình ảnh về mô hình Ontology:



Hình 1-3:Một mô hình bản thể học [3].

Ngôn ngữ ontology cho phép người sử dụng viết rõ ràng, các khái niệm hình thức của mô hình miền. Các yêu cầu chính:

- *Cấu trúc rõ ràng:* Đây là điều kiện cần cho máy có thể xử lý thông tin
- *Ngữ nghĩa hình thức miêu tả ý nghĩa tri thức một cách chính xác:*Ý nghĩa của ngữ nghĩa hình thức tồn tại trong một thời gian dài trong miền toán logic. Việc sử dụng ngữ nghĩa hình thức cho phép con người suy diễn tri thức. Với tri thức trong ontology chúng ta có thể suy diễn về:

+ *Thành viên của lớp:* Nếu x là một thể hiện của lớp C và C là lớp con của lớp D thì chúng ta suy ra x là thể hiện của lớp D.

+ *Các lớp tương đương:* Nếu lớp A tương đương với lớp B và lớp B tương đương với lớp C, thì lớp A cũng tương đương với lớp C.

+ *Tinh nhất quán:* Giả sử chúng ta khai báo x là thể hiện của lớp A và A là lớp con của B ∩ C, A là lớp con của lớp D, Lớp B và lớp D không có quan hệ với nhau (disjoint). Thì chúng ta không nhất quán bởi vì A nên là rỗng nhưng lại có thể hiện là x. Đây là một dấu hiệu của một lỗi trong ontology.

+ *Phân loại:* Nếu chúng ta khai báo các cặp thuộc tính giá trị đã biết là điều kiện đủ cho thành viên trong một lớp A, thì nếu một cá thể x thỏa mãn các điều kiện, chúng ta có thể kết luận x phải là một thể hiện của A.

+ *Ngữ nghĩa là điều kiện tiên quyết cho việc hỗ trợ suy diễn:* Hỗ trợ suy diễn rất quan trọng bởi vì nó cho phép kiểm tra tính nhất quán của ontology và tri thức, kiểm tra các quan hệ thừa giữa các lớp, tự động phân loại các thể hiện trong lớp.

Ngữ nghĩa hình thức và hỗ trợ suy diễn thường được cung cấp bởi việc ánh xạ một ngôn ngữ ontology đến hình thức logic và sử dụng suy diễn tự động bởi các hình thức luôn tồn tại. OWL được ánh xạ logic miêu tả và sử dụng các suy diễn đang tồn tại như FaCT và RACER. Các logic mô tả là tập con của logic vị từ nhằm hỗ trợ suy diễn hiệu quả.

1.3. Các thành phần của bản thể học

Ontology được sử dụng như là một biểu mẫu trình bày tri thức về thế giới hay một phần của nó. Ontology thường miêu tả:

- Cá thể (Individual): Các đối tượng cơ bản của Ontology, nền tảng; Các cá thể trong một Ontology có thể bao gồm các đối tượng rời rạc như: tên của một người, tên một cái cây, một chiếc xe ô tô,…
- Lớp (Class): Các tập hợp, hay kiểu của các đối tượng. Lớp là những nhóm, bộ hoặc một tập hợp các đối tượng. Một lớp có thể gộp nhiều lớp hoặc được gộp vào lớp khác. Một lớp gộp vào lớp khác được gọi là lớp con (subclass) của lớp gộp. Điều quan trọng của quan hệ xếp gộp là tính kế thừa.
- Thuộc tính (Property): Đó là các tính năng, đặc điểm, tính cách, hay các thông số mà các đối tượng có và có thể đem ra chia sẻ.
- Mối liên hệ (Relationships): Cách mà các đối tượng có thể liên hệ tới một đối tượng khác. Tập hợp các quan hệ cùng nhau mô tả ngữ nghĩa của lĩnh vực (domain). Tập các dạng quan hệ được sử dụng và cây phân loại thứ bậc của chúng thể hiện sức mạnh diễn đạt của ngôn ngữ dùng để biểu diễn ontology.

Ontology thường phân biệt các nhóm quan hệ khác nhau. Ví dụ:

+ Quan hệ giữa các lớp
+ Quan hệ giữa các thực thể
+ Quan hệ giữa một thực thể và một lớp
+ Quan hệ giữa một đối tượng đơn và một tập hợp
+ Quan hệ giữa các tập hợp

Bộ từ vựng Ontology được xây dựng trên cơ sở tầng RDF, RDFS, cung cấp khả năng biểu diễn ngữ nghĩa mềm dẻo cho tài nguyên Web và có khả năng hỗ trợ lập luận.

1.4. Phương pháp xây dựng bản thể học

Trong những năm gần đây, một loạt các phương pháp luận khác nhau được thiết kế để trợ giúp cho việc tiến hành phát triển các nhiệm vụ được báo cáo trong tài liệu trí tuệ nhân tạo. Các phương pháp truyền thống gồm Cyc (Lenat & Guha 1990) [4], Uschold và King (Uschold $ King 1995) [5], Gruninger và Fox (Gruninger $ Fox 1994) [6],… Các phương pháp luận cung cấp các chỉ dẫn chung và có cấu trúc. Nếu làm theo có thể tăng quy trình phát triển và cải tiến chất lượng cho kết quả cuối cùng. Theo như đánh giá phương pháp luận "Methontology" là phương pháp luận thiết kế ontology phổ biến nhất (được hỗ trợ môi trường WebODE).

Quy trình phát triển Ontology là một quy trình gồm nhiều bước, tuy nhiên vẫn chưa có một phương pháp chuẩn hóa nào để phát triển các ontology. Quy trình phát triển gồm bảy bước do Stanford Center for Biomedical Informatics Research đưa ra (đây là nhóm phát triển phần mềm Protégé để trình diễn và soạn thảo Ontology).

- *Bước 1: Xác định lĩnh vực và phạm vi của Ontology:*Trong giai đoạn này cần xác định mục đích của việc xây dựng ontology là gì? Phục vụ đối tượng nào? Ontology sắp xây dựng cần có đặc điểm gì, liên quan đến lĩnh vực, phạm vi nào. Quá trình khai thác, quản lý và bảo trì ontology được thực hiện ra sao?

- *Bước 2: Xem xét việc sử dụng lại các Ontology có sẵn:* Cấu trúc của một Ontology bao gồm 3 tầng: tầng trừu tượng (Abstract), tầng miền xác định (Domain) và tầng mở rộng (Extension). Trong đó tầng trừu tượng có tính tái sử dụng rất cao, tầng miền xác định có thể tái sử dụng trong một lĩnh vực nhất định. Cộng đồng Ontology cũng đang lớn mạnh và có rất nhiều Ontology đã được tạo ra, với tâm huyết của nhiều chuyên gia. Do đó trước khi bắt đầu xây dựng ontology, cần xét đến khả năng sử dụng lại các ontology đã có. Nếu có thể sử dụng lại một phần các ontology đã có, chi phí bỏ ra cho quá trình xây dựng ontology sẽ giảm đi rất nhiều.

- *Bước 3: Liệt kê các thuật ngữ quan trọng:* Ontology được xây dựng trên cơ sở các khái niệm trong một lĩnh vực cụ thể, vì vậy khi xây dựng ontology cần bắt đầu từ các thuật ngữ chuyên ngành để xây dựng thành các lớp trong ontology tương ứng. Tất nhiên không phải thuật ngữ nào cũng đưa vào ontology, vì chưa chắc đã định vị được cho thuật ngữ đó. Do đó cần phải liệt kê các thuật ngữ, để xác định ngữ nghĩa cho các thuật ngữ đó, cũng như cân nhắc về phạm vi của ontology. Việc liệt kê các thuật ngữ còn cho thấy được

phần nào tổng quan về các khái niệm trong lĩnh vực đó, giúp cho các bước tiếp theo được thuận lợi.

- *Bước 4: Xác định các lớp và phân cấp của các lớp:* Công việc xác định các lớp không chỉ đơn giản là tiến hành tìm hiểu về ngữ nghĩa của các thuật ngữ đã có để có được các mô tả cho thuật ngữ đó, mà còn phải định vị cho các lớp mới, loại bỏ ra khỏi ontology nếu nằm ngoài phạm vi của ontology hay hợp nhất với các lớp đã có nếu có nhiều thuật ngữ có ngữ nghĩa như nhau (đồng nghĩa, hay đa ngôn ngữ). Ngoài ra không phải thuật ngữ nào cũng mang tính chất như một lớp. Một công việc cần phải tiến hành song song với việc xác định các lớp là xác định phân cấp của các lớp đó. Việc này giúp định vị các lớp dễ dàng hơn.

Có một số phương pháp tiếp cận trong việc xác định phân cấp của các lớp:

Phương pháp từ trên xuống (top-down): bắt đầu với định nghĩa của các lớp tổng quát nhất trong lĩnh vực và sau đó chuyên biệt hóa các khái niệm đó. Ví dụ: Trong Ontology về quản lý nhân sự, ta bắt đầu với lớp Người, sau đó chuyên biệt hóa lớp Người đó bằng cách tạo ra các lớp con của lớp Người như : Kỹ sư, Công nhân, Bác sỹ,… Lớp Kỹ sư cũng có thể chuyên biệt hóa bằng cách tạo ra các lớp con như Kỹ sư CNTT, Kỹ sư điện, Kỹ sư cơ khí, …

Phương pháp từ dưới lên (bottom-up): bắt đầu với định nghĩa của các lớp cụ thể nhất, như các lá trong cây phân cấp. Sau đó gộp các lớp đó lại thành các khái tổng quát hơn. Ví dụ: ta bắt đầu với việc định nghĩa các lớp như: nhân viên lễ tân, nhân viên vệ sinh, nhân viên kỹ thuật. Sau đó tạo ra một lớp chung hơn cho các lớp đó là lớp nhân viên.

Phương pháp kết hợp: kết hợp giữa phương pháp từ trên xuống và từ dưới lên: bắt đầu từ định nghĩa các lớp dễ thấy trước và sau đó tổng quát hóa và chuyên biệt hóa các lớp đó một cách thích hợp. Ví dụ ta bắt đầu với lớp nhân viên trước, là thuật ngữ hay gặp nhất trong quản lý nhân sự. Sau đó chúng ta có thể chuyên biệt hóa thành các lớp con: nhân viên lễ tân, nhân viên vệ sinh,… hoặc tổng quát hóa lên thành lớp Người.

- *Bước 5: Xác định các thuộc tính:* Để xác định thuộc tính cho các lớp, ta quay trở lại danh sách các thuật ngữ đã liệt kê được. Hầu hết các thuật ngữ còn lại (sau khi đã xác định lớp) là thuộc tính của các lớp đó. Với mỗi thuộc tính tìm được, ta phải xác định xem nó mô tả cho lớp nào. Các thuộc tính đó sẽ trở thành thuộc tính của các lớp xác định. Ví dụ lớp Người có các thuộc tính sau: Họ, Tên, Ngày sinh, Giới tính, Nghề nghiệp, Địa chỉ, Điện thoại,…

- *Bước 6: Xác định ràng buộc của các thuộc tính:* Các thuộc tính có thể có nhiều khía cạnh khác nhau: như kiểu giá trị, các giá trị cho phép, số các thuộc tính (lực lượng), và các đặc trưng khác mà giá trị của thuộc tính có thể nhận. Ví dụ: "Năm sinh" của một "nhân viên" chỉ có duy nhất và là số nguyên, có thể nhận giá trị từ 1948 đến 1990. Cần phải xác định các ràng buộc cho một thuộc tính càng chặt chẽ càng tốt, để tránh trường hợp nhập dữ liệu sai, dẫn đến đổ vỡ của các ứng dụng sử dụng Ontology này.

- *Bước 7: Tạo các thể hiện / thực thể:* Bước cuối cùng là tạo ra các thể hiện của các lớp trong sự phân cấp. Việc tạo thể hiện cho một lớp là quá trình điền các thông tin vào các thuộc tính của lớp đó.

1.5. Thể hiện bản thể học bằng OWL

1.5.1. Khái quát

OWL (The Web Ontology Language) [8] là một ngôn ngữ gần như XML dùng để mô tả các hệ cơ sở tri thức. OWL là một ngôn ngữ đánh dấu dùng để xuất bản và chia sẻ dữ liệu trên Internet thông qua những mô hình dữ liệu gọi là "ontology". Ontology mô tả một lĩnh vực (domain) và diễn tả những đối tượng trong lĩnh vực đó cùng những mối quan hệ giữa các đối tượng này. OWL là phần mở rộng về từ vựng của RDF và được kế thừa từ ngôn ngữ DAML+OIL Web ontology – một dự án được hỗ trợ bởi W3C. OWL biểu diễn ý nghĩa của các thuật ngữ trong các từ vựng và mối liên hệ giữa các thuật ngữ này để đảm bảo phù hợp với quá trình xử lý bởi các phần mềm.

OWL được xem như là một kỹ thuật trọng yếu để cài đặt cho Semantic Web trong tương lai. OWL được thiết kế đặc biệt để cung cấp một cách thức thông dụng trong việc xử lý nội dung thông tin của Web. Ngôn ngữ này được kỳ vọng rằng sẽ cho phép các hệ thống máy tính có thể đọc được thay thế cho con người. Vì OWL được viết bởi XML, các thông tin OWL có thể dễ dàng trao đổi giữa các kiểu hệ thống máy tính khác nhau, sử dụng các hệ điều hành và các ngôn ngữ ứng dụng khác nhau. Mục đích chính của OWL là sẽ cung cấp các chuẩn để tạo ra một nền tảng để quản lý các thuộc tính và để chia sẻ cũng như tái sử dụng dữ liệu trên Web. OWL được phát triển bởi nó có nhiều tiện lợi để biểu diễn ý nghĩa và ngữ nghĩa hơn so với XML, RDF và RDFS, và vì OWL ra đời sau các ngôn ngữ này, nó có khắc phục được những thiếu sót và tận dụng được nhiều tính ưu việt, khả năng biểu diễn các nội dung mà máy có thể biểu diễn được trên Web.

Các phiên bản của OWL: Hiện nay có ba loại OWL: OWL Lite, OWL DL (description logic), và OWL Full.

OWL Lite: Hỗ trợ cho những người dùng chủ yếu cần sự phân lớp theo thứ bậc và các ràng buộc đơn giản. Ví dụ: Trong khi nó hỗ trợ các ràng buộc về tập hợp, nó chỉ cho

phép tập hợp giá trị của 0 hay 1. Điều này cho phép cung cấp các công cụ hỗ trợ OWL Lite dễ dàng hơn so với các bản khác.

OWL DL (OWL Description Logic): Hỗ trợ cho những người dùng cần sự diễn cảm tối đa trong khi cần duy trì tính tính toán toàn vẹn (tất cả các kết luận phải được đảm bảo để tính toán) và tính quyết định (tất cả các tính toán sẽ kết thúc trong khoảng thời gian hạn chế). OWL DL bao gồm tất cả các cấu trúc của ngôn ngữ OWL, nhưng chúng chỉ có thể được sử dụng với những hạn chế nào đó (Ví dụ: Trong khi một lớp có thể là một lớp con của rất nhiều lớp, một lớp không thể là một thể hiện của một lớp khác).OWL DL cũng được chỉ định theo sự tương ứng với logic mô tả, một lĩnh vực nghiên cứu trong logic đã tạo nên sự thiết lập chính thức của OWL.

OWL Full: Muốn đề cập tới những người dùng cần sự diễn cảm tối đa và sự tự do của RDF mà không cần đảm bảo sự tính toán của các biểu thức. Ví dụ, trong OWL Full, một lớp có thể được xem xét đồng thời như là một tập của các cá thể và như là một cá thể trong chính bản thân nó. OWL Full cho phép một ontology gia cố thêm ý nghiã của các từ vựng được định nghĩa trước (RDF hoặc OWL).

Các phiên bản này tách biệt về các tiện ích khác nhau, OWL Lite là phiên bản dễ hiểu nhất và phức tạp nhất là OWL Full.

Mối liên hệ giữa các ngôn ngữ con của OWL:

- Mọi Ontology hợp lệ dựa trên OWL Lite đều là Ontology hợp lệ trên OWL DL
- Mọi ontology hợp lệ dựa trên OWL DL đều là ontology hợp lệ trên OWL Full
- Mọi kết luận hợp lệ dựa trên OWL Lite đều là kết luận hợp lệ trên OWL DL Mọi kết luận hợp lệ dựa trên OWL DL đều là kết luận hợp lệ trên OWL Full

1.5.2. Các thành phần của tài liệu OWL

Các tiêu đề (titles):

Tài liệu OWL thường được gọi là OWL ontologies nó là một tài liệu RDF. Vì thế, yếu tố gốc của Ontology là một thẻ rdf:RDF – chứa một số lượng lớn các không gian tên (namespace)

<rdf:RDF

xmlns:owl =http://www.w3.org/2002/07/owl#

xmlns:rdf =http://www.w3.org/1999/02/22-rdf-syntax-ns#

xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#

xmlns:xsd =http://www.w3.org/2001/XMLSchema#

/>

Một Ontology còn có thể bắt đầu với một tập hợp các mô tả về phiên bản, tài nguyên được thêm vào dưới cặp thẻ:

<owl:Ontology >

<owl:Ontology rdf:about="">

<rdfs:comment>Ví dụ về owl Ontology </rdfs:comment>

<owl:priorVersion rdf:resource="http://www.mydomain.org/uni-ns-old"/>

<owl:imports rdf:resource="http://www.mydomain.org/persons"/>

<rdfs:label>University Ontology</rdfs:label>

</owl:Ontology>

Thẻ owl:imports dùng để thêm vào Ontology hiện tại một Ontology khác được chỉ định bên trong thẻ rdf:resource. Owl:imports chính là một thuộc tính bắc cầu (transitive) mà ta sẽ nói đến ở phần sau.

Lớp (Class)

Lớp là một nhân tố quan trọng và cơ bản của một Ontology. Lớp là tập hợp các cá thể thuộc về nhau bởi vì chúng cùng chia sẻ những thuộc tính chung. Ngoài ra, nó còn được tổ chức thành các cấp bậc sử dụng subClassOf. Chú ý rằng: lớp Thing là một lớp tổng quát và có thể coi là cha của mọi lớp trong OWL còn Nothing thì ngược lại.

Để định nghĩa một lớp ta sử dụng thẻ owl:Class như trong ví dụ sau:

<owl:Class rdf:ID= "sinh_vien" />

Ở trên ta đã định nghĩa ra một lớp có tên (id) là sinh_vien. Từ bây giờ, lớp sinh_vien này có thể được tham chiếu tới bằng cách dùng ký hiệu #sinh_vien.

Một lớp có thể được chỉ ra là hoàn toàn phân biệt với lớp khác (disjointWith) hay bằng với lớp khác (equivalentClass).

Ví dụ:

<owl:Class rdf:ID= " associateProfessor " >

<owl:disjointWith rdf:resource= " #Professor " />

<owl:disjointWith rdf:resource= " #assitantProfessor " />

</owl:Class>

DisjointWith đảm bảo rằng một thành viên của lớp này không thể đồng thời là thành viên của lớp khác.

EquivalentClass thì để chỉ ra hai lớp có chính xác những thể hiện giống nhau:

<owl:Class rdf:ID= "faculty" >

<owl:equivalentClass rdf:resource= "academicStaffMember" />

</owl:Class>

Lớp con (subClass)

Để tạo sự phân cấp trong Ontology, ta có thể định nghĩa một lớp là lớp con của lớp khác. Sự định nghĩa này cũng mang tính bắc cầu: nếu A là lớp con của B và B là lớp con của C thì A là lớp con của C.

<owl:Class rdf:ID= "sinh_vien" >

<rdfs:subClassOf rdf:resource= "#con_nguoi" />

</owl:Class>

Ở đây thì lớp sinh_vien được định nghĩa là lớp con của lớp con_nguoi. Cần phân biệt một chút giữa lớp bằng nhau (equivalentClass) và lớp con (subClassOf). Với lớp bằng nhau thì những thể hiện của lớp này cũng chính là những thể hiện của lớp kia, còn với lớp con thì những thể hiện của lớp cha không nhất thiết phải là những thể hiện của lớp con.

Nhãn (lable)

Ta có thể sử dụng nhãn để chú thích cho các tên gọi. Nhãn sẽ không giúp ích được gì cho quá trình suy diễn của Ontology mà chỉ là một sự lựa chọn khác cho các tên gọi.

<owl:class rdf:id="wine">

<rdfs:subClassOf rdf:resource="&food;PortalbeLiQuid">

<rdfs:lable xml:lang="en">wine</rdfs:label>

<rdfs:lable xml:lang="fr">vin</rdfs:label>

</owl:class>

Thuộc tính (properties)

- Ví dụ về ObjectProperty:
  <owl:ObjectProperty rdf:ID="madefromGrape" >

<rdfs:domain rdf:resource=”#Wine” />

<rdfs:range rdf:resource= “#WineGrape” />

</owl:ObjectProperty>

- Ví dụ về DataTypeProperty

<owl:DatatypeProperty rdf:ID="age">

<rdfs:rangerdf:resource="http://www.w3.org/2001/XLMSchema#nonNegativeInteger"/>

</owl:DatatypeProperty>

Các kiểu dữ liệu do người dùng định nghĩa sẽ được tập hợp trong một XML Schema và được sử dụng vào trong Ontology. Một Domain là một miền mà thuộc tính đó tồn tại còn Range là một sự phân loại thuộc tính trong miền đó. Ta có thể khai báo hơn một Domain hay một Range, khi đó Domain thực sự sẽ là giao của các Domain và Range là các Range tách biệt nhau.

<owl:Class rdf:about="#firstYearCourse">

<rdfs:subClassOf>

<owl:Restriction>

<owl:onProperty rdf:resource="#isTaughtBy"/>

<owl:allValuesFrom rdf:resource="#Professor"/>

</owl:Restriction>

</rdfs:subClassOf>

</owl:Class>

owl:allValuesFrom được dùng để chỉ những giá trị mà thuộc tính owl:onProperty có thể được lấy. Có nghĩa là giá trị của isTaughtBy chỉ có thể là những Professor.

Chúng ta còn có thể chỉ định một giá trị cụ thể mà thuộc tính phải có sử dụng thẻ: *owl:hasValue* (cú pháp tương tự *allValuesFrom*). Ngoài ra, ta còn dùng*owl:someValuesFrom* để chỉ rằng thuộc tính phải chứa ít nhất một giá trị trongrdf:resource.

<owl:Class rdf:about="#academicStaffMember">

<rdfs:subClassOf>

<owl:Restriction>

```
<owl:onProperty rdf:resource="#teaches"/>
    <owl:someValuesFrom
    rdf:resource="#undergraduateCourse"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Ví dụ trên nói rõ rằng, một *academicStaffMember* phải *teaches* ít nhất một *undergraduateCourse*.

Những cấu trúc *owl:allValuesFrom, owl:someValuesFrom* và *owl:hasValue* là những giới hạn về giá trị của thuộc tính. Một loại khác của giới hạn thuộc tính là giới hạn về lực lượng. Chẳng hạn chúng ta có thể chỉ ra rằng một course phải được dạy (*isTaughtBy*) bởi một người nào đó:

```
<owl:Class rdf:about="#course">
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#isTaughtBy"/>
            <owl:minCardinality
                rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:minCardinality>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

owl:mincardinality là một giới hạn dưới của một thuộc tính, ta còn có thể định ra giới hạn trên bằng cách sử dụng *owl:maxCardinality* và chính xác số lực lượng bằng: *owl:Cardinality* (cú pháp như trong ví dụ trên).

Thuộc tính đặc biệt

Ta có thể định nghĩa trực tiếp các đặc trưng của thuộc tính như sau:

owl:TransitiveProperty: hai thuộc tính gọi là bắc cầu nếu với mỗi cặp (x,y) là một thể hiện của thuộc tính P và cặp (y,z) là thể hiện của thuộc tính P thì (x,z) cũng là thể hiện của thuộc tính P.

owl:SymmetricProperty: dùng để chỉ ra rằng một thuộc tính là thuộc tính đối xứng. Với mỗi cặp (x,y) là thể hiện của thuộc tính P thì (y,x) cũng là thể hiện của thuộc tính P.

owl:FunctionalProperty: thuộc tính được chỉ ra là có giá trị duy nhất. Nó có nhiều nhất là một giá trị cho mỗi cá thể.

owl: InverseFunctionalProperty: ngược lại với *FunctionalProperty*.

Một ví dụ về cú pháp cho các đặc trưng trên:

```
<owl:ObjectProperty rdf:ID="hasSameGradeAs">
<rdf:type rdf:resource="&owl;TransitiveProperty" />
<rdf:type rdf:resource="&owl;SymmetricProperty" />
<rdfs:domain rdf:resource="#student" />
</owl:ObjectProperty>
```

Ngoài ra, một thuộc tính có thể định nghĩa là thuộc tính con của thuộc tính khác sử dụng:

```
<owl:ObjectProperty rdf:ID= " hasColor" >
<rdfs:subPropertyOf rdf:resource= "#hasWineDescriptor"/ >
<rdfs:range rdf:resource= "#WineColor" />
</owl: ObjectProperty>
```

Sự kết nối :

Các lớp có thể được kết hợp với nhau bằng phép giao (intersection), hợp (union) và lấy phần bù (complement). Ví dụ, ta có thể chỉ ra course và Staff member là phân biệt(disjoint) với nhau chỉ cần sử dụng:

```
<owl:Class rdf:about="#course">
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:complementOf rdf:resource="#staffMember"/>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Thay vì sử dụng owl:disjointWith.

Tương tự, sự hợp của các lớp thì sử dụng owl:unionOf:

<owl:Class rdf:ID="peopleAtUni">

<owl:unionOf rdf:parseType="Collection">

<owl:Class rdf:about="#staffMember"/>

<owl:Class rdf:about="#student"/>

</owl:unionOf>

</owl:Class>

rdf:parseType là một lối viết tắt của cấu trúc xây dựng danh sách trong RDF là <rdf:first>

và <rdf:rest> và là một yếu tố bắt buộc trong cú pháp.

Chú ý rằng, ví dụ trên không nói lớp được tạo ra là con của lớp Union mà nó chính bằng với Union.

Phép giao thì được bắt đầu với owl:intersectionOf:

<owl:Class rdf:ID="WhiteWine">

<owl:intersectionOf rdf:parseType="Collection">

<owl:Class rdf:about="#Wine" />

<owl:Restriction>

<owl:onProperty rdf:resource="#hasColor" />

<owl:hasValue rdf:resource="#White" />

</owl:Restriction>

</owl:intersectionOf>

</owl:Class>

Một vấn đề nữa cũng được đề cập trong phần kết nối này là các phép toán hợp, giao và lấy phần bù hoàn toàn có thể sử dụng lồng nhau:

<owl:Class rdf:ID owl:Class rdf ="NonFrenchWine">

<owl:intersectionOf rdf:parseType="Collection">

<owl:Class rdf:about="#Wine"/>

<owl:Class>

```
<owl:complementOf>
    <owl:Restriction>
    <owl:onProperty rdf:resource="#locatedIn" />
    <owl:hasValue rdf:resource="#FrenchRegion" />
    </owl:Restriction>
</owl:complementOf>
</owl:Class>
</owl:intersectionOf>
</owl:Class>
```

Trong ví dụ trên, ta đã định nghĩa lớp NonFrenchWine chính là giao của lớp Wine và lớp bù của lớp những cái ở French (tức là lớp những cái không ở French).

Liệt kê

Sự liệt kê được thê hiện bằng thẻ owl:oneOf. Nó được dùng để định nghĩa một lớp bằng cách liệt kê các phần tử của lớp:

```
<owl:oneOf rdf:parseType="Collection">
<owl:Thing rdf:about="#Monday"/>
<owl:Thing rdf:about="#Tuesday"/>
<owl:Thing rdf:about="#Wednesday"/>
<owl:Thing rdf:about="#Thursday"/>
<owl:Thing rdf:about="#Friday"/>
<owl:Thing rdf:about="#Saturday"/>
<owl:Thing rdf:about="#Sunday"/>
</owl:oneOf>
```

Thể hiện

Thể hiện của một lớp được khai báo bằng thẻ <rdf:Description>.

```
<rdf:Description rdf:ID="949352" >
<rdf:type rdf:resource="#academicStaffMember"/>
</rdf:Description>
```

Hay tương đương:

<academicStaffMember rdf:ID="949352"/>

Nghĩa là một thể hiện của lớp academicStaffMember là 949352.

Chúng ta còn có thể cung cấp thêm thông tin chi tiết như sau:

<academicStaffMember rdf:ID="949352">

<uni:age rdf:datatype="&xsd;integer">39<uni:age>

</academicStaffMember>

Không giống như những hệ thống cơ sở dữ liệu khác, OWL không cho phép giả định những cái tên (hay ID) phân biệt nhau. Vì thế, hai thể hiện có tên khác nhau thì không có nghĩa là chúng là những thể hiện khác nhau thật sự. Ví dụ, ta định nghĩa mỗi course chỉ được dạy bởi một staff member:

<owl:ObjectProperty rdf:ID="isTaughtBy">

<rdf:type rdf:resource="&owl;FunctionalProperty" />

</owl:ObjectProperty>

Rồi sau đó, chúng ta đưa ra một course được dạy bởi hai staff member :

<course rdf:about="CIT1111">

<isTaughtBy rdf:resource="949318">

<isTaughtBy rdf:resource="949352">

</course>

Điều này không là nguyên nhân gây lỗi cho nhà lập luận bởi vì sau đó hệ thống sẽ suy ra một cách hợp lý là 949318 và 949352 là bằng nhau. Do đó, để chắc chắn những cá nhân khác nhau là thật sự ta phải chỉ ra một cách rõ ràng bằng owl:differentFrom:

<lecturer rdf:about="949318">

<owl:differentFrom rdf:resource="949352">

</lecturer>

Bởi vì, sự khác nhau giữa các cá thể luôn được phân biệt thường xuyên, và số lượng thành viên sẽ bùng nổ nên OWL đã đưa ra một sự biểu diễn ngắn gọn giúp ta chỉ ra các cá nhân là khác nhau từng đôi một :

<owl:allDifferent>

<owl:distinctMembers rdf:parseType="Collection">

<lecturer rdf:about="949318">

<lecturer rdf:about="949352">

<lecturer rdf:about="949111">

</owl:distinctMembers>

</owl:allDifferent>

Chú ý rằng <owl:distinctMembers> chỉ có thể được sử dụng kết nối với <owl:allDifferent>

Kiểu dữ liệu

Mặc dù XML Schema cung cấp một kỹ thuật cho phép người dùng định nghĩa kiểu dữ liệu cho riêng mình. Nhưng thật sự, nhiều kiểu dữ liệu không thể sử dụng trong OWL.

Sau đây là liệt kê một số kiểu dữ liệu có thể được dùng :

xsd:string	xsd:normalizedString	xsd:boolean
xsd:decimal	xsd:float	xsd:double
xsd:integer	xsd:nonNegativeInteger	xsd:positiveInteger
xsd:nonPositiveInteger	xsd:negativeInteger	xsd:short
xsd:long	xsd:int	xsd:time
xsd:byte	xsd:unsignedShort	xsd:gDay
xsd:unsignedLong	xsd:unsignedByte	xsd:language
xsd:unsignedInt	xsd:hexBinary	xsd:NCName
xsd:date	xsd:base64Binary	xsd:gMonthDay
xsd:gMonth	xsd:dateTime	xsd:token
xsd:anyURI	xsd:gYearMonth	xsd:Name
xsd:NMTOKEN	xsd:gYear	

Bảng 1-1: Các kiểu dữ liệu trong OWL

Ví dụ:

<owl:Class rdf:ID="VintageYear" />

<owl:DatatypeProperty rdf:ID="yearValue">

<rdfs:domain rdf:resource="#VintageYear" />

<rdfs:range rdf:resource="&xsd;positiveInteger"/>

</owl:DatatypeProperty>

</owl:Class>

Phiên bản

Chúng ta đã thấy owl:priorVersion ở phần mở đầu của Ontology để chỉ ra phiên bản củ hơn của Ontology hiện tại. Thông tin này không nằm trong lý thuyết của semantic nhưng có thể được khai thác bởi người đọc và chương trình với mục đích quản lý Ontology.

Bên cạnh câu lệnh owl:priorVersion, OWL còn có thêm ba câu lệnh khác để có những thông tin thêm về phiên bản:

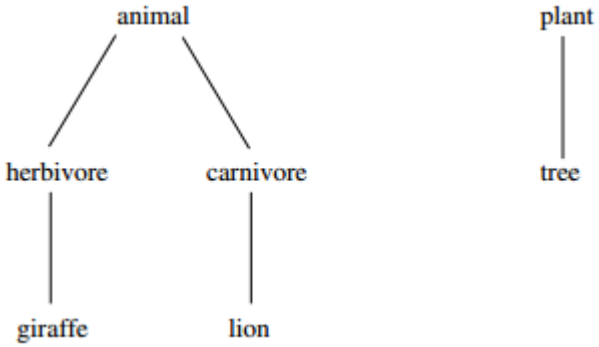
owl:versionInfo thường chứa một chuỗi để đưa thông tin về phiên bản củaOntology hiện tại.

owl:backwardCompatibleWith chỉ ra rằng một Ontology là phiên bản trước củaOntology hiện tại và nó tương thích với Ontology hiện tại.

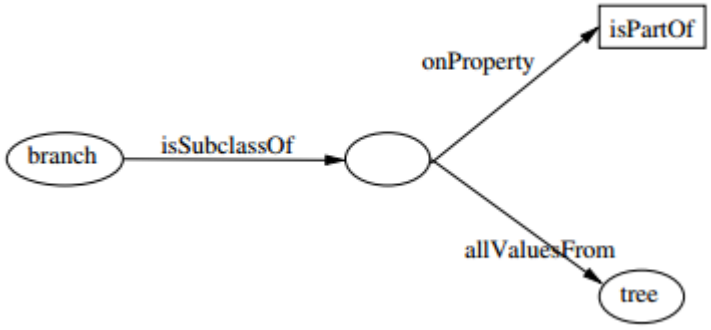
owl:incompatibleWith mang ý nghĩa ngược lại chỉ một Ontology là phiên bản sau của Ontology hiện tại và không tương thích với Ontology hiện tại.

1.5.3. Ví dụ

Trong phần này, chúng tôi trình bày một ví dụ bản thể học mô tả động vật hoang dã ở Châu Phi [7]. Hình 1 cho thấy các Class cơ bản và các mối quan hệ giữa subClass của chúng. Lưu ý rằng các thông tin về subclass chỉ là một phần của thông tin bao gồm trong ontology. Toàn bộ Graph là lớn hơn nhiều. Hình 2 cho thấy rằng các đại diện của graphic statement mà các nhánh (branches) là các bộ phận của trees.



Hình 1-5: Class và subClasses của Ontology African wildlife



Hình 1-6: Branches là thành phần của trees

```
<rdf:RDF>
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl ="http://www.w3.org/2002/07/owl#">

<owl:Ontology rdf:about="xml:base"/>

<owl:Class rdf:ID="animal">
        rdfs:comment>Animals form a class.</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="plant">
<rdfs:comment>
        Plants form a class disjoint from animals.
</rdfs:comment>
        <owl:disjointWith rdf:resource="#animal"/>
</owl:Class>

<owl:Class rdf:ID="tree">
```

```
        <rdfs:comment>Trees are a type of plant.</rdfs:comment>
        <rdfs:subClassOf rdf:resource="#plant"/>
    </owl:Class>

    <owl:Class rdf:ID="branch">
        <rdfs:comment>Branches are parts of trees.</rdfs:comment>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#is_part_of"/>
                <owl:allValuesFrom rdf:resource="#tree"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:ID="leaf">
        <rdfs:comment>Leaves are parts of branches.</rdfs:comment>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#is_part_of"/>
                <owl:allValuesFrom rdf:resource="#branch"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>

    <owl:Class rdf:ID="herbivore">
        <rdfs:comment>
            Herbivores are exactly those animals that eat only plants or parts of
plants.
        </rdfs:comment>
```

```
        <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#animal"/>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#eats"/>
                    <owl:allValuesFrom>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="#plant"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#is_part_of"/>
                        <owl:allValuesFrom rdf:resource="#plant"/>
                    </owl:Restriction>
                </owl:unionOf>
            </owl:Class>
        </owl:allValuesFrom>
        </owl:Restriction>
        </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="carnivore">
        <rdfs:comment>
            Carnivores are exactly those animals that eat animals.
        </rdfs:comment>
        <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#animal"/>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#eats"/>
                <owl:someValuesFrom rdf:resource="#animal"/>
            </owl:Restriction>
```

```
            </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="giraffe">
        <rdfs:comment>
            Giraffes are herbivores, and they eat only leaves.
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="#herbivore"/>
            <rdfs:subClassOf>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#eats"/>
                    <owl:allValuesFrom rdf:resource="#leaf"/>
            </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="lion">
        <rdfs:comment>
            Lions are animals that eat only herbivores.
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="#carnivore"/>
            <rdfs:subClassOf>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#eats"/>
                    <owl:allValuesFrom rdf:resource="#herbivore"/>
                </owl:Restriction>
            </rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:ID="tasty_plant">
    <rdfs:comment>
        Tasty plants are plants that are eaten both by herbivores and carnivores.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#plant"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#eaten_by"/>
                    <owl:someValuesFrom>
                        <owl:Class rdf:about="#herbivore"/>
                    </owl:someValuesFrom>
            </owl:Restriction>
            </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#eaten_by"/>
                <owl:someValuesFrom>
                    <owl:Class rdf:about="#carnivore"/>
                </owl:someValuesFrom>
            </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
<owl:TransitiveProperty rdf:ID="is_part_of"/>
<owl:ObjectProperty rdf:ID="eats">
    <rdfs:domain rdf:resource="#animal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="eaten_by">
    <owl:inverseOf rdf:resource="#eats"/>
</owl:ObjectProperty>
```

</rdf:RDF>

1.6. Công cụ hỗ trợ phát triển bản thể học

Một số công cụ phát triển và hiệu chỉnh có giá trị trong việc làm giảm độ phức tạp và thời gian dùng cho nhiệm vụ xây dựng ontology. Các công cụ như Kaon, OileEd và Protégé [8][10] cung cấp các giao diện nhằm giúp đỡ người sử dụng thực hiện các hoạt động chính yêu trong trong quá trình phát triển một ontology. Việc lựa chọn một công cụ hiệu chỉnh phù hợp nhất có nhiều khó khăn vì mỗi kiểu ontology có các yêu cầu về kinh phí, thời gian, tài nguyên khác nhau. Để giúp cho việc giải quyết vướng mắc này, Singh & Murshed (2005) đưa ra các tiêu chuẩn đánh giá công cụ tạo ontology. Tiêu chuẩn bao gồm tính năng, khả năng sử dụng lại, lưu trữ dữ liệu, mức độ phức tạp, quan hệ, tính lâu bền, độ an toàn, độ chắc chắn, khả năng học, tính khả dụng, hiệu lực, và tính rõ ràng. Protégé và OntoEditFree [8] được phát triển bởi Singh & Murshed sử dụng các tiêu chuẩn này.

Protégé là một nền tảng miễn phí, mã nguồn mở cung cấp một cộng đồng người dùng phát triển với một bộ công cụ để xây dựng các mô hình lĩnh vực và các ứng dụng dựa trên tri thức với bản thể học.

Protégé hỗ trợ OWL, là công cụ được sử dụng rộng rãi và lâu nhất hiện nay. Nó cho phép người sử dụng định nghĩa và chỉnh sửa các lớp ontology, các thuộc tính và quan hệ và các thể hiện sử dụng cấu trúc cây. Các ontology có thể được đưa ra theo các định dạng RDF(S), XML Schema. Platform protégé cung cấp hai cách chính mô hình hóa ontology thông qua Protégé - Frame và Protégé – OWL, ngoài ra có thể có nhiều plugin. Chúng ta có thể quan sát một cách trực quan thông qua OWL Viz, nó cho phép quan sát ontology dưới dạng đồ họa và đưa file ảnh JPEG. Ngoài ra còn hỗ trợ truy vấn SPARQL.

CHƯƠNG 2. SUY DIỄN TRÊN MÔ HÌNH BẢN THỂ HỌC

Sau chương này sẽ trả lời được những câu hỏi như sau:

- Suy diễn trên bản thể học với các quy tắc suy diễn như thế nào?

- Các ngôn ngữ biểu diễn quy tắc suy diễn trên mô hình bản thể học?

- Máy suy diễn và sự hỗ trợ của Jena Framework?

- Phân tích sự suy diễn trên bản thể học với ví dụ cụ thể?

2.1. Quy tắc suy diễn

Hiện tại ngôn ngữ OWL không hỗ trợ cho việc kết hợp các thuộc tính (Property Composition). Lấy một ví dụ về một Ontology lĩnh vực về mối quan hệ trong một gia đình của con người. Các logic mô tả (Description Logic) không thể xác định một cá thể A có quan hệ "có chú (hasUncle)" với cá thể B vì nó cần đến hai thành phần thông tin. Thứ nhất, liệu A có quan hệ "có cha mẹ (hasParents)" với cá thể nào đó không? Và liệu cá thể đó (cá thể mà A có quan hệ hasParent) liệu có quan hệ anh em trai "hasBrother" với cá thể B hay không? Tập luật hỗ trợ cho các khái niệm cho việc tạo ra quan hệ hasUncle thông qua hai thành phần thông tin này.

Việc xây dựng các tập luật suy diễn (inference rules) là rất quan trọng, đó là yếu tố công nghệ cũng như là các chuẩn cho Web ngữ nghĩa. Bởi vì các tập luật làm mở rộng cơ sở dữ liệu và Ontology với dạng tri thức có cấu trúc (Structured Knowledge), qua đó nó thể hiện được sự mạnh mẽ, mềm dẻo, và năng động cho thế hệ Web ngữ nghĩa.

Những quy tắc ngữ nghĩa (Semantic rules) là công cụ giúp suy diễn hiệu quả trên mô hình bản thể học. Quy tắc ngữ nghĩa là một dạng biểu diễn tri thức sử dụng thường xuyên và được phân loại theo cấu trúc phân cấp theo cấu trúc như sau:

- Reaction rule: Chứa một sự kiện (Event) dùng để "bẫy" (Trigger) sự thực thi của tập luật hay bẫy các điều kiện cần thiết để thực thi một hành động mà tập luật định nghĩa, hành động tự thân (Action itself), cũng như điều kiện trước sau (pre – and post Condition). Một ví dụ điển hình cho Reaction Rule là Trigger trong ngôn ngữ SQL. Reaction có 2 thành phần đó là:

  + *Integrity Constraints* (ràng buộc toàn vẹn): Chứa các mệnh đề logic. Loại tập luật này định nghĩa các phát biểu là đúng trong tất cả các phát biểu (States) và chỉ ra sự chuyển tiếp của một hệ thống động rời rạc (Discrete Dynamic System) thành

những gì mà chúng được định nghĩa. Ngôn ngữ điển hình cho dạng luật này là SQL.

+ *Derivation Rules*: Chứa đựng một hay nhiều điều kiện và một kết luận, chúng đóng vai trò quan trọng trong một công thức logic (Logical Formula). Ví dụ cụ thể: "A car us available for rental if is not assigned to any client and is not scheduled for service"

Trong đó quy tắc suy diễn (inference rules) là một dạng đặc biệt của Derivation Rule dành cho Web ngữ nghĩa. Tập luật này có dạng là các phát biểu điều kiện (Condition Statement) đặc trưng là dạng mệnh đề if – then mà Antecedent (phần theo trước then) và Consequent (phần theo sau then).

Thông thường, antecedent sẽ sử dụng một hay nhiều các biến tự do (Free Variable) và consequent sẽ sử dụng các biến đã khai báo đó

Antecedent sẽ là sự liên kết của không hay nhiều mệnh đề (Clause). Nếu tất cả các mệnh đề trong antecedent đúng thì mệnh đề trong consequent được suy luận đúng.

Antecedent rỗng (Emty Antecedent) được xem là đúng. Do đó, Consequent phải được suy luận là đúng

Consequent rỗng (Empty consequent) được xem là sai. Do đó, antecedent không được suy luận là đúng.

2.2. Ngôn ngữ biểu diễn quy tắc suy diễn

Mục đích của ngôn ngữ đánh dấu quy tắc là cho phép sử dụng lại, trao đổi và xuất bản các tập luật. Các ngôn ngữ đánh dấu quy tắc khác nhau là thành phần cho việc sử dụng các tập luật trên Web và trong những hệ thống phân tán khác. Chúng cho phép thực thi, xuất bản, và giao tiếp các tập luật trên một mạng. Nói cách khác, ngôn ngữ đánh dấu luật cho phép chúng ta xác định những luật nghiệp vụ (Bussiness Rule) như các modun đơn vị độc lập (Stand alone unit) trong các báo cáo, và cho phép chúng ta xuất bản và trao đổi chúng giữa các hệ thống hay các công cụ khác. Một số ngôn ngữ đánh dấu luật là: RuleML, SWRL.

2.2.1. RuleML

RuleML (Rule Markup Language) [9] là một sự khởi đầu cho việc tạo ra các ngôn ngữ đánh dấu luật hỗ trợ nhiều dạng luật khác nhau và ngữ nghĩa khác nhau.

Tuy nhiên phiên bản 0.91 của RuleML vẫn còn chứa đựng một vài giới hạn cho một số dạng luật. Cụ thể, RuleML vẫn chưa có cú pháp tổng quát cho tập luật dạng Integrity và Reaction như đã giới thiệu ở trên.

RuleML được xây dựng trên mô hình lập trình logic (Logic Programming Paradigm) của logic vị từ (Predicate Logic)

Một ví dụ cụ thể diễn tả việc khai báo thuộc tính hasParent và thuộc tính hasBrother để cài đặt thuộc tính hasUncle

```
<Implies>
    <head>
    <Atom>
        <Rel>hasParent</Rel>
        <Var>x1</Var>
        <Var>x2</Var>
    </Atom>
    <Atom>
        <Rel>hasBrothe</Rel>
        <Var>x2</Var>
        <Var>x3</Var>
    </Atom>
</head>
<body>
    <Atom>
        <Rel>hasUncle</Rel>
        <Var>x1</Var>
        <Var>x3</Var>
    </Atom>
</body>
</Implies>
```

Một số thẻ cơ bản của RuleML:

+ <Implies>: là thẻ dùng để cài đặt một luật cụ thể. Chứa hai phần con là <Head> và <Body>
+ <Head>: tương đương với phần then trong phát biểu if – then
+ <Body>: tương đương với phần then trong phát biểu if – then
+ <Atom>: là một mệnh đề chứa quan hệ (Relation) được áp lên các biến logic (Logic Argument) của nó
+ <Rel>: khai báo một quan hệ (thuộc tính)
+ <Var>: khai báo các biến logic, như trong lập trình logic
+ <Ind>: khai báo hằng cá thể (Individual Constant) giống như tài nguyên RDF

2.2.2. SWRL

Dựa trên nền tảng của RuleML kết hợp với hai ngôn ngữ thành phần của OWL là OWL – Lite và OWL – DL, SWRL [9] là sự kết hợp giữa Antecedent (Body) và consequent (head) chứa tập hợp (có thể rỗng) các atom. Mỗi luật SWRL là một sự kéo theo (Implication), có nghĩa là nếu tất cả các atom của antecedent là đúng thì consequent la đúng.

Các atom thường xuất hiện ở các dạng:

atom::= description ‘(’ i-object ‘)’

| dataRange ‘(’d-object ‘)’

| individualvaluedPropertyID ‘(’i-object i-object ‘)’

| datavaluedPropertyID ‘(’i-object d-object ‘)’

| sameAs ‘(’i-object i-object ‘)’

| differentFrom ‘(’i-object i-object ‘)’

| builtIn ‘(’builtinID {d-object} ‘)’

builtinID ::= URIreference

Trong đó:

i-object: tương ứng với biến cá thể hoặc là cá thể trong OWL

d-object: tương ứng với biến dữ liệu hoặc là giá trị dữ liệu trong OWL

individualvaluedPropertyID ‘(’i-object d-object ‘)’ : tương ứng với ObjectProperty trong OWL

datavaluedPropertyID ‘(’i-object d-object ‘)’ : tương ứng với DataTypeProperty trong OWL

sameAs ‘(’i-object i-object ‘)’ : nếu cả hai i-object thể hiện cho cùng một cá thể, hay cùng một biến cá thể.

differentFrom ‘(’i-object i-object ‘)’: nếu cả hai i-object thể hiện cho hai cá thể, hai biến cá thể khác nhau

builtIn(r, x, …): là các phép toán về toán học (Mathematics), chuỗi (Strings), thời gian (Date, Time), … được xây dựng sẵn.

(1) Namespace:

Cú pháp SWRL XML Concrete sử dụng để cài đặt tập luật cho Ontology thông qua các phần tử được chỉ ra trong các namespace như sau:

swrlx: http://www.w3.org/2003/11/swrlx

ruleml: http://www.w3.org/2003/11/ruleml

owlx: http://www.w3.org/2003/05/owl-xml

xsd: http://www.w3.org/2001/XMLSchema

(2) Thuộc tính gốc của tài liệu OWL trong cú pháp thể hiện XML (XML Presentation)

<swrlx:Ontology>

<swrlx:Ontology swrlx:name = xsd:anyURI>

Content:

</swrlx:Ontology>

Thuộc tính swrlx:name tham chiếu đến tên của Ontology dựa trên định danh xsd:anyURI

(3) <ruleml:var>

<ruleml:var>xsd:String</ruleml:var>

Đây là phần tử dùng để khai báo biến trong tập luật. Ví dụ: ta khai báo biến x1 như sau: *<ruleml:var>x1</ruleml:var>*

(4) <ruleml:imp>

<ruleml:imp>

Content: (_rlab, owlx:Annotation, _body, _head)*

</ruleml:imp>

Phần tử này dùng để khai báo một mệnh đề kéo theo logic giữa antecedent (_body) và consequent (_head) của một tập luật. Phần tử chứa các phần tử con như _rlab, owlx:Annotation, _body, _head

(5) <ruleml:_rlab>

<ruleml:_rlab ruleml:href = xsd:anyURI>

Content:

</ruleml:_rlab>

Phần tử này dùng để khai báo tên và định danh của một luật

Ví dụ: *<ruleml:_rlab ruleml:href = "#ruleExample1">*

(6) <ruleml:_body>

<ruleml:_body>

Content: (swrlx:atom)*

</ruleml:_body>

Đây là phần tử dùng để định nghĩa ra phần tử body (consequent) của luật. Phần tử này chứa các phần tử con là các swrlx:atom

(7) <swrl:_head>

<ruleml:_head>

Content: (swrl:atom)*

</ruleml:_head>

Đây là phần tử dùng để định nghĩa phần tử head (antecedent) của luật. Phần tử này chứa các phần tử con là các swrlx:atom

(8) <swrlx:classAtom>

<swrlx:classAtom>

Content: (owlx:description, swrlx:iObject)

</swrlx:classAtom>

classAtom chứa các mô tả hay các tên của các cá thể, tên của các biến. Các mô tả có thể là tên lớp hay có thể là một mô tả phức hợp (Description Complex) sử dụng các ràng buộc.

Ví dụ:

<swrlx:classAtom>

<owlx:Class owlx:name = "Person" />

<ruleml:var>x1</ruleml:var>

</swrlx:classAtom>

Khai báo biến x1 có kiểu là thực thể thuộc lớp Person

<swrlx:classAtom>

<owlx:IntersectionOf>

<owlx:Class owlx:name= "Person" />

<owlx:ObjectRestriction owlx:property = "hasParent">

<owlx:someValuesFrom owlx:class = "Physician" />

</owlx:ObjectRestriction>

</owlx:IntersectionOf>

</swrlx:classAtom>

Khai báo biến x2 có kiểu là thực thể của lớp giao (Intersection) của hai lớp. Một lớp Person, hai là những lớp có thuộc tính là hasParent và có ít nhất một giá trị thuộc tính thuộc vào lớp Physician

(9) <swrl:datarangeAtom>

<swrl:datarangeAtom>

Content: (owlx:datarange, swrlx:dObject)

</swrl:datarangeAtom>

Phần tử này dùng để khai báo các khoảng giá trị, có khi là tập hợp các nguyên tử (Literal)

Ví dụ:

<swrlx:datarangeAtom>

<owlx:Datatype owlx:name = "&xsd;int" />

<ruleml:var>x1</ruleml:var>

</swrlx:datarangeAtom>

Khai báo biến x1 có kiểu int

<swrlx:datarangeAtom>

<owlx:OneOf>

<owlx:DataValue owlx:datatype = "&xsd;int">5</owlx:DataValue>

<owlx:DataValue owlx:datatype = "&xsd;int">5</owlx:DataValue>

</owlx:OneOf>

</swrlx:datarangeAtom>

Khai báo biến x2 có kiểu int và nhận một trong hai giá trị 5 hoặc 10

(10) *<swrlx:individualPropertyAtom>*

<swrlx:individualPropertyAtom swrlx:property = xsd:anyURI>

Content: (swrlx:iObject, swrlx:iObject)

</swrlx:individualPropertyAtom>

Phần tử này dùng để khai báo thuộc tính cá thể (Individual Property). Lưu ý, OWL không cho phép khai báo các thuộc tính phức hợp, một thuộc tính chỉ có một tên thuộc tính.

Ví dụ:

<swrl:individualPropertyAtom swrlx:property = "hasParent">
<ruleml:var>x1</ruleml:var>
<owlx:individual owlx:name = "John" />
</swrl:individualPropertyAtom>

Khai báo quan hệ hasParent giữa hai cá thể x1 đại diện và cá thể John

(11) <swrlx:dataValuedPropertyAtom>

<swrlx:dataValuedPropertyAtom swrlx:property = xsd:anyURI>
Content: (swrlx:iObject, swrlx:dObject)
</swrlx:dataValuedPropertyAtom>

Phần tử này dùng để khai báo thuộc tính kiểu dữ liệu (DataValued Property). Lưu ý rằng OWL không cho phép khai báo các thuộc tính phức hợp, một thuộc tính chỉ có một tên thuộc tính.

Ví dụ:

<swrlx:datavaluedPropertyAtom swrlx:property = "grade">
<ruleml:var>x1</ruleml:var>
<owlx:DataValue owlx:datatype = "&xsd;int">4</owlx:DataValue>
</swrlx:datavaluedPropertyAtom>

Khai báo quan hệ grade giữa x1 là biến đại diện và dữ liệu kiểu int có giá trị là 4

(12) <swrlx:sameIndividualAtom>

<swrlx:sameIndividualAtom>
Content: (swrlx:iObject)*
</swrlx:sameIndividualAtom>

Phần tử này dùng để xác định các cá thể là của chung một cá thể. Ví dụ:

<swrlx:sameIndividualAtom>
<ruleml:var>x1</ruleml:var>
<ruleml:var>x2</ruleml:var>
<owlx:individual owlx:name = "Clinton" />
<owlx:individual owlx:name = "Bill_Clinton" />
</swrlx:sameIndividualAtom>

Khai báo biến x1 và x2 đại diện cho hai cá thể có tên là Clinton và Bill_Clinton thực ra chỉ là một.

(13) <swrlx:differentIndividualAtom>

<swrlx:differentIndividualAtom>
Content: (swrlx:iObject)*
</swrlx:differentIndividualAtom>

Phần tử này dùng để xác định các cá thể là khác nhau

(14) <swrlx:builtinAtom>

<swrlx:builtinAtom swrlx:builtin = xsd:anyURI>
Content: (swrlx:dObject)*
</swrlx:builtinAtom>

Phần tử này dùng để khai báo sử dụng một Builtin được định nghĩa sẵn thông qua xsd:anyURI của Builtin đó

Ví dụ sau mô tả trường hợp cài đặt thuộc tính hasUncle bằng cách cài đặt luật thông qua hai thuộc tính khác là hasParent và hasBrother. Luật trên được mô hình trực quan trong môi trường soạn thảo Protégé như sau:

hasParent(?x1, ?x2)^ hasBrother(?x2, ?x3) → hasUncle(?x1, ?x3)

được cài đặt bằng SWRL như sau:

<ruleml:imp>

<ruleml:_rlab ruleml:href = "#example1" />

<ruleml:_body>

<swrlx:individualPropertyAtom swrlx:property = "hasParent">

<ruleml:var>x1</ruleml:var>

<ruleml:var>x2</ruleml:var>

</swrlx:individualPropertyAtom>

<swrlx:individualPropertyAtom swrlx:property = "hasBrother">

<ruleml:var>x2</ruleml:var>

<ruleml:var>x3</ruleml:var>

</swrlx:individualPropertyAtom>

</ruleml:_body>

<ruleml:_head>

<swrl:individualPropertyAtom swrlx:property = "hasUncle">

<ruleml:var>x1</ruleml:var>

<ruleml:var>x3</ruleml:var>

</swrl:indivudualPropertyAtom>

</ruleml:_head>

</ruleml:imp>

2.2.3. SPARQL

SPARQL [14] là một ngôn ngữ truy vấn dữ liệu theo định dạnh RDF: SPARQL được thiết kế để phục vụ cho các use – case và requirement của RDF.

Với dữ liệu được thiết kế dưới dạng TURTLE, ta sẽ thực hiện một số truy vấn đơn giản sau:

- Truy vấn với dòng dữ liệu:
  Dữ liệu:

<http://example.org/book/book1><http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .

Truy vấn:

```
SELECT ?title
WHERE {
    <http://example.org/book/book1><http://purl.org/dc/elements/1.1/title>
"SPARQL Tutorial" .
}
```

Kết quả:

Title
"SPARQL Tutorial"

Giải thích: truy vấn này được dùng để tìm tiêu đề của một quyển sách. Truy vấn gồm hai phần: SELECT chỉ định tên biến sẽ hiển thị trong phần kết quả, mệnh đề WHERE dùng để tìm ra đúng dữ liệu. Trong mệnh đề WHERE lại là một sơ đồ RDF mà object của nó là một biến (điều này không thể có trong dữ liệu RDF thường)

- Truy vấn có nhiều kết quả:
  Dữ liệu:

@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.com> .
_:c foaf:mbox <mailto:carol@example.com> .

Truy vấn:

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

Kết quả:

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

Giải thích: kết quả của một truy vấn SPARQL là một chuỗi các giải pháp (solution) tùy vào sự phù hợp (match) mà mệnh đề WHERE thực hiện, có thể không có hay có từ một kết quả cho mỗi truy vấn. Trong truy vấn này, ta chọn ra tên (name) và địa chỉ email (mbox) của người nào có cả name và mbox (được xác định với cùng biến x trong mệnh đề WHERE)

- Truy vấn với phần nguyên tố RDF (RDF Literal)

Dữ liệu:

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

Chú ý rằng: Trong TURTLE “cat”@en có nghĩa là “cat” và thẻ ngôn ngữ là “en” (English). “42”^^xsd:integer là một giá trị với kiểu dữ liệu là http://www.w3.org/2001/XMLSchema#integer

Với truy vấn mà dữ liệu có chứa thẻ ngôn ngữ: *SELECT ?v WHERE {?v ?p “cat”}* sẽ không có được kết quả nào vì “cat” là khác với “cat” @en. Để có được kết quả như ý muốn ta phải viết truy vấn:

SELECT ?v WHERE {?v ?p “cat”@en}

Khi đó kết quả sẽ là:

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

Với truy vấn dữ liệu dạng số integer.

Số integer được sử dụng trực tiếp trong SPARQL và coi như là một giá trị, do đó với truy vấn: *SELECT ?v WHERE {?v ?p 42}* sẽ vẫn thực hiện được:

Kết quả:

v
<http://example.org/ns#y>

Truy vấn với dạng dữ liệu trừu tượng

Bộ xử lý truy vấn không hiểu về giá trị trong không gian dữ liệu, do đó có thể truy vấn chính xác cần đặt cả IRI của dữ liệu vào trong mệnh đề WHERE. Ví dụ:

SELECT ?v

WHERE {?v ?p “abc”^^<http://example.org/datatype#specialDatatype>}

Kết quả:

v
<http://example.org/ns#z>

- Nút rỗng trong kết quả truy vấn

Kết quả truy vấn có thể chứa những nút rỗng là nút được ký hiệu dưới dạng "_:" kèm với nhãn của nút. Sử dụng nút với cùng một nhãn trong kết quả là cùng một nút:

Dữ liệu:

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .

_:b foaf:name "Bob".

Truy vấn:

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?x ?name

WHERE {?x foaf:name ?name}

Kết quả:

x	Name
_:c _:d	"Alice" "Bob"

x	Name
_:r _:s	"Alice" "Bob"

Ta chú ý rằng kết quả không chứa _:a, :_b giống như trong dữ liệu mà có thể chứa nút với nhãn bất kỳ. Nghĩa là nút rỗng.

- Xây dựng mô hình RDF

SPARQL có một vài hình thức truy vấn: truy vấn SELECT dùng để truyền giá trị trả về vào biến, truy vấn CONSTRUCT trả về một mô hình RDF

Dữ liệu:

@prefix org: <http://example.com/ns#> .

_:a org:employeeName "Alice" .

_:a org:employeeId 12345 .

_:b org:employeeName "Bob" .

_:b org:employeeId 67890 .

Truy vấn CONSTRUCT

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

PREFIX org: <http://example.com/ns#>

CONSTRUCT {?x foaf:name ?name}

WHERE {?x org:employeeName ?name }

Kết quả:

@prefix org: <http://example.com/ns#> .

_:x foaf:name "Alice" .

_:y foaf:name "Bob" .

Có thể sắp xếp theo định dạng RDF/XML như sau:

<rdf:RDF

xmlns:rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#

xmlns:foaf = http://xmlns.com/foaf/0.1/>

<rdf:Description>

<foaf:name>Alice</foaf:name>

</rdf:Description>

<rdf:Description>

<foaf:name>Bob</foaf:name>

</rdf:Description>

</rdf:RDF>

- Lọc lấy kết quả truy vấn

Khi sử dụng SPARQL để truy vấn có thể sẽ trả ra nhiều kết quả phù hợp với mô hình RDF trong mệnh đề WHERE. Vấn đề đặt ra là ta chỉ muốn lấy một số kết quả thỏa mãn một tiêu chuẩn nào đó thôi. SPARQL sử dụng FILTER để làm điều đó, xem trình bày sau:

Dữ liệu:

@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 .

Lọc giá trị chuỗi:

SPARQL FILTER sử dụng regex để kiểm tra các nguyên tố RDF. Hàm regex chỉ có thể kiểm tra các chuỗi text bình thường (plain text không có kèm thẻ language)

Truy vấn:

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

SELECT ?title

WHERE {?x dc:title ?title

FILTER regex(?title, "^SPARQL")}

Kết quả:

Title
"SPARQL Tutorial"

Lọc giá trị số:

SPARQL có thể lọc ra giá trị của các biểu thức số học

Truy vấn:

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 .

}

Kết quả:

title	Price
“The Semantic Web”	23

Giải thích: truy vấn dùng để lọc ra tiêu đề và giá của quyển sách với điều kiện là giá phải nhỏ hơn 30.5

IRIs là sự tổng quát hóa của URI, hoàn toàn tương thích với URIs và URLs. Dưới đây là một số cách khác nhau để viết URIs

<http://example.org/book/book1>

BASE <http://example.org/book/>

<book1>

PREFIX book: <http://example.org/book/>

book:book1

Literal

Cú pháp chung cho các literal là một chuỗi (được để trong dấu nháy đơn hay nháy kép) và có thể chứa thẻ language (bắt đầu với @ hay một IRI của kiểu dữ liệu hay một tiền tố tên (được bắt đầu với ^^))

Điều tiện lợi là số Integer có thể viết một cách trực tiếp. Những con số với dấu "." Và không có chứa mũ (e) được giải thích với xsd:double. Giá trị xsd:Boolean có thể viết là true hay false

Để dễ dàng trong việc viết các giá trị mà có chứa sẵn trích dẫn hay dấu xuống dòng hay giá trị quá dài người ta còn dùng thêm dấu bao với ba dấu nháy đơn hay nháy đôi.

Một số ví dụ về cách viết các literal:

"price"

'price'@fr với thẻ ngôn ngữ fr (france)

1 tương tự như "1"^^xsd:integer

1.3 tương tự như "1.3"^^xsd:decimal

1.300 tương tự như "1.300"^^xsd:decimal

1.0e6 tương tự như "1.0e6"^^xsd:double

True tương tự như "true"^^xsd:Boolean

Biến truy vấn trong SPARQL có phạm vi toàn cục, sử dụng cùng một tên biến ở bất kỳ nơi nào trong truy vấn có nghĩa là cùng một biến. Biến được khai báo bằng dấu "?" hay dấu "$" trước tên biến, các dấu này đảm bảo không nằm trong tên biến. Tên biến được đặt theo quy định của ngữ pháp SPARQL

Nút rỗng trong mô hình đồ thị hoạt động giống như một biến không phân biệt, nó không tham chiếu tới một nút cụ thể nào giống trong dữ liệu được truy vấn.

Nút rỗng được chỉ định bởi hình thức nhãn như: "_abc" hay hình thức "[]". Một nút rỗng mà chỉ dùng trong một nơi trong truy vấn thì có thể được chỉ định bằng [].

Cấu trúc [:p ?v] có thể dùng trong mô hình bộ ba, nó tạo ra một nút rỗng được dùng như Subject cho cặp Predicate – Object chứa bên trong.

Cú pháp mẫu bộ ba

Những mô hình bộ ba với Subject chung có thể được viết bằng cách ghi Subject một lần và sau đó là các bộ Predicate – Object bằng nhau bằng ";"

?x foaf:name ?name ;

foaf:mbox ?mbox .

Nếu một mô hình cùng sử dụng chung Subject và Predicate với các Object khác nhau thì mỗi Object được viết cách nhau bằng dấu ","

?x foaf:nick “Alice” , “Bob” .

Liệt kê Object có thể kết hợp với liệt kê Predicate – Object

?x foaf:name ?name ; foaf:nick “Alice” , “Bob” .

RDF Collections có thể được viết dưới dạng mô hình RDF sử dụng cú pháp “(element1 element2…)”. Hình thức “()” là một thay thế cho IRI http://www.w3.org/1999/02/22-rdf-syntax-ns#nil. Khi sử dụng với nhân tố tập hợp, mô hình bộ ba với nút rỗng được sử dụng. Nút rỗng ở đầu của tập hợp có thể được sử dụng như là Subject hay Object cho các mô hình bộ ba khác.

Ví dụ:

(1 ?x 3 4) :p “w” .

Có thể được viết chính xác như:

```
_:b0   rdf:first   1 ;
       rdf:rest    _:b1 .
_:b1   rdf:first   ?x ;
       rdf:rest    _:b2 .
_:b2   rdf:first   3 ;
       rdf:rest    _:b3 .
_:b3   rdf:first   4 ;
       rdf:rest    rdf:nil .
_:b0   :p          “w” .
```

Từ khóa “a” có thể được viết thay thế cho IRI http://www.w3.org/1999/02/22-rdf-syntax-ns#type. Chú ý ta phân biệt chữ hoa chữ thường

Ví dụ:

?x a :Class1 .

[a :appClass] :p “v” .

Là cách viết của:

```
?x rdf:type   :Class1 .
_:b0   rdf:type    :apClass .
_:b0   :p     “v” .
```

Mẫu đồ thị

SPARQL dựa trên sự phù hợp của mô hình đồ thị, nhiều mô hình đồ thị phức tạp có thể được thực hiện bằng cách kết nối các mô hình với nhau theo nhiều cách nào đó. Ở đây chúng ta sẽ tìm hiểu hai mô hình là mô hình đồ thị đơn giản (basic graph pattern) và mô hình đồ thị nhóm (group graph pattern)

Basic Graph Pattern (BGP)

BGP là một tập hợp của các mô hình bộ ba. Mô hình đồ thị trong SPARQL được định nghĩa là sự kết nối các kết quả từ các BGP

Khi sử dụng nút rỗng với hình thức _:abc, nhãn của nút nằm trong phạm vi của BGP. Một nhãn chỉ có thể sử dụng trong một BGP trong bất cứ truy vấn nào

Group Graph Pattern (GGP)

Trong một chuỗi truy vấn của SPARQL một GGP được phân định bằng dấu {}. Ví dụ, truy vấn là một GGP của một BGP:

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

Hay:

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

Là một GGP có chứa hai BGP

- Empty group pattern

Mô hình nhóm {} phù hợp với bất cứ đồ thị nào kể cả đồ thị rỗng

- Phạm vi của Filter:

Một ràng buộc được chỉ định bởi từ khóa Filter là một giới hạn cho tất cả group nơi mà Filter xuất hiện

- Ví dụ về GGP

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

Là một nhóm của một BGP và BGP đó có chứa hai mô hình bộ ba. Trong khi:

{
?x foaf:name ?name . Filter regex(?name, "Smith")
?x foaf:mbox ?mbox .
}

Là một nhóm của một BGP và một Filter. Filter không tách BGP thành hai BGP

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

Là một nhóm của ba nhân tố: một BGP với một mô hình bộ ba, một nhóm rỗng, và một BGP khác với một mô hình bộ ba.

Giá trị tùy chọn

BGP cho phép ứng dụng thực hiện truy vấn với kết quả phải đầy đủ, tức là các biến phải được xác định giá trị nếu không kết quả sẽ không chấp nhận. Để có thể thêm thông tin vào nếu thông tin có sẵn nhưng kết quả không bị loại bỏ (do đó phần không phù hợp) ta sẽ sử dụng OPTIONAL

Ví dụ:

Dữ liệu:

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_: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" .

Truy vấn:

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

Kết quả:

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

Không có chứa giá trị cho mbox đối với tên "Bob". Truy vấn này tìm kiếm tên người trong dữ liệu, nếu có một bộ ba với Predicate mbox và Subject giống thì kết quả sẽ chứa Object của bộ ba trong OPTIONAL.

Ta cũng có thể ràng buộc giá trị trong OPTIONAL. Ví dụ:

Dữ liệu:

@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 .

Truy vấn:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE{
    ?x dc:title ?title .
    OPTIONAL {?x ns:price ?price . FILTER {?price <30}}
}
```

Kết quả:

Title	Price
"SPARQL Tutorial"	
"The Semantic Web"	23

Không có price cho title "SPARQL Tutorial" bởi vì mô hình OPTIONAL có giới hạn để đưa ra price là < 30

Ta có thể sử dụng nhiều mô hình OPTIONAL trong một truy vấn, khi đó kết quả có thể chứa nhiều ô rỗng với từng mô hình OPTIONAL

Phép hợp kết quả

SPARQL cung cấp một phương tiện kết nối các mô hình đồ thị mà một hay vài đồ thị thay thế có thể phù hợp, tất cả những kết quả có thể sẽ được tìm thấy.

Dữ liệu:

```
@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)" .

Truy vấn:

```
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}
      ONION {?book dc11:title ?title}
}
```

Kết quả:

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (Update)"
"SPARQL Query Language Tutorial"

API để sử dụng SPARQL trong Jena là gói com.hp.hpl.jena.query. Các lớp chính trong gói này là:

- Query: Một lớp biểu diễn cho chính câu truy vấn. Đối tượng của Query thông thường được tạo ra bằng cách gọi một trong các phương thức của QueryFactory mà chúng cung cấp sự truy nhập đến các bộ phân tích câu truy vấn.

- QueryExecution: biểu diễn cho một sự thực hiện câu truy vấn

- QueryExecutionFactory: Một nơi để lấy về thể hiện của QueryExecution

- DatasetFactory: Một nơi để tạo ra Dataset, bao gồm tạo ra DataSource (một Dataset có thể cập nhật)

- Cho câu truy vấn SELECT:

+ QuerySolution: Một kết quả đơn của truy vấn

+ ResultSet: Tất cả QuerySolution là một bộ lặp

+ ResultSetFormatter: Chuyển một ResultSet về nhiều dạng, text, RDF Graph hay XML

Ví dụ xây dựng và thực hiện câu lệnh truy vấn trên một model:

```
import com.hp.hpl.jena.query.* ;
Model model = ... ;
String queryString = " .... " ;
Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
try {
 ResultSet results = qexec.execSelect() ;
 for ( ; results.hasNext() ; )
 {
  QuerySolution soln = results.nextSolution() ;
  RDFNode x = soln.get("varName") ;  // Get a result variable by name.
  Resource r = soln.getResource("VarR"); // Get a result variable - must be a resource
  Literal l = soln.getLiteral("VarL") ; // Get a result variable - must be a literal
 }
} finally { qexec.close() ; }
```

Ví dụ xây dựng và thực hiện câu truy vấn CONSTRUCT:

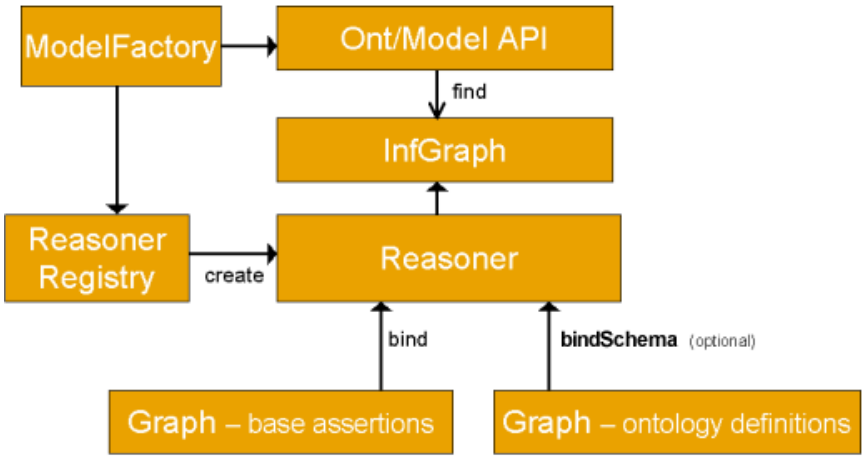
```
Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
Model resultModel = qexec.execConstruct() ;
qexec.close() ;
```

2.3. Jena framework

Jena [13][20] là một Framework Java để xây dựng các ứng dụng Web có ngữ nghĩa. Nó cung cấp một môi trường lập trình cho các RDF, RDFS, OWL và SPARQL và bao gồm một động cơ suy diễn (inference engine) dựa trên quy tắc. Các tính năng chính của Jena 2 là:

- RDF API – một API cho các thao tác với mô hình RDF là một tập hợp của RDF triples, phân tích cú pháp tích hợp và lý thuyết cho RDF/XML, N3 và N – TripleS, và cũng hỗ trợ cho các kiểu literals.
- ARP – một RDF/XML Parser. Jena phiên bản 2 là phù hợp với các khuyến nghị Core RDF . ARP thường được gọi bằng các hoạt động đọc Jena, nhưng cũng có thể sử dụng độc lập.
- Persistence – một mở rộng của lowpss Model Jena cung cấp sự bền vững cho mô hình thông qua việc sử dụng một cơ sở dữ liệu back – end
- Reasoning subsystem – mộ suy diễn dựa trên động cơ quy tắc cấu hình cho RDFS và cho các tập hợp con của OWL Lite của OWL Full.
- Ontology subsystem – một API để làm việc với OWL, DAML + OIL và RDFS
- ARQ – một bộ công cụ truy vấn, mà thực hiện cả hai ngôn ngữ truy vấn SPARQL và RDQL. SPARQL là một ngôn ngữ truy vấn RDF và giao thức được phát triển của W3C (mà ta đã giới thiệu ở phần trên).

Các cấu trúc của bộ máy suy diễn được minh họa dưới đây:



Hình 2-1: Cấu trúc bộ máy suy diễn của Jena

Các ứng dụng thường truy cập các máy suy diễn bằng cách sử dụng các *ModelFactory* liên kết một tập dữ liệu với một số suy luận để tạo ra một mô hình mới. Truy vấn đến các mô hình (model) sẽ trả lại những báo cáo (statements) mà đã có trong các dữ liệu ban đầu, nhưng cũng có báo cáo thêm bổ sung có thể bắt nguồn từ các dữ liệu ban đầu bằng cách sử dụng quy tắc hoặc các cơ chế suy diễn khác được thực hiện bởi suy luận.

Như minh họa các máy suy diễn (inference machinery) được thực sự thực hiện ở cấp độ Graph SPI, vì vậy mà bất kỳ giao diện mô hình (Model interfaces) khác nhau có thể được xây dựng xung quanh một một inference Graph. Đặc biệt các Ontology API cung cấp các phương tiện để liên kết những suy luận thích hợp vào *OntModel S* mà nó được xây dựng. Là một phần mở rộng chung của RDF API chúng tôi cũng cung cấp *InfModel*. Đây là mọt phần mở rộng của giao diện Model bình thường mà cung cấp điều khiển bổ sung và truy cập vào mọt inference graph cơ bản.

Các API suy luận hỗ trợ các khái niệm đặc biệt một suy luận bằng cách gắn nó vào một tập hợp các sơ đồ hoặc dữ liệu ontology sử dụng gọi *bindSchema*. Các suy luận đặc biệt sau đó có thể được gắn vào bộ dữ liệu khác nhau, ví dụ sử dụng các gọi *bind*. Trong trường hợp các thông tin cùng một giản đồ được sử dụng nhiều lần với các bộ dữ liệu khác nhau, ví dụ sau đó kỹ thuật này cho phép đối với một số tái sử dụng các kết luận trên sử dụng khác nhau của lược đồ. Trong RDF không có sự phân mạnh mẽ giữa giản đồ (aka Ontology AKA tbox) dữ liệu và ví dụ (AKA abox) dữ liệu và do đó bất kỳ dữ liệu, cho dù các class hoặc các instance liên quan, có thể được bao gồm trong một hoặc hai *bind* hoặc *bindSchema* khi gọi – những cái tên gợi ý chứ không phải là hạn chế.

Để giữ cho các thiết kết như là kết thúc mở càng tốt Jena cũng bao gồm một ReasonerRegistry. Đây là một lớp tĩnh mặc dù đó tập các reasoners hiện được kiểm tra. Có thể đăng ký các loại suy luận mới và tự động tìm kiếm suy luận của một kiểu nhất định. Việc *ReasonerRegistry* này cũng cung cấp truy cập thuận tiện để các trường hợp dựng sẵn của reasoners đã cung cấp chính.

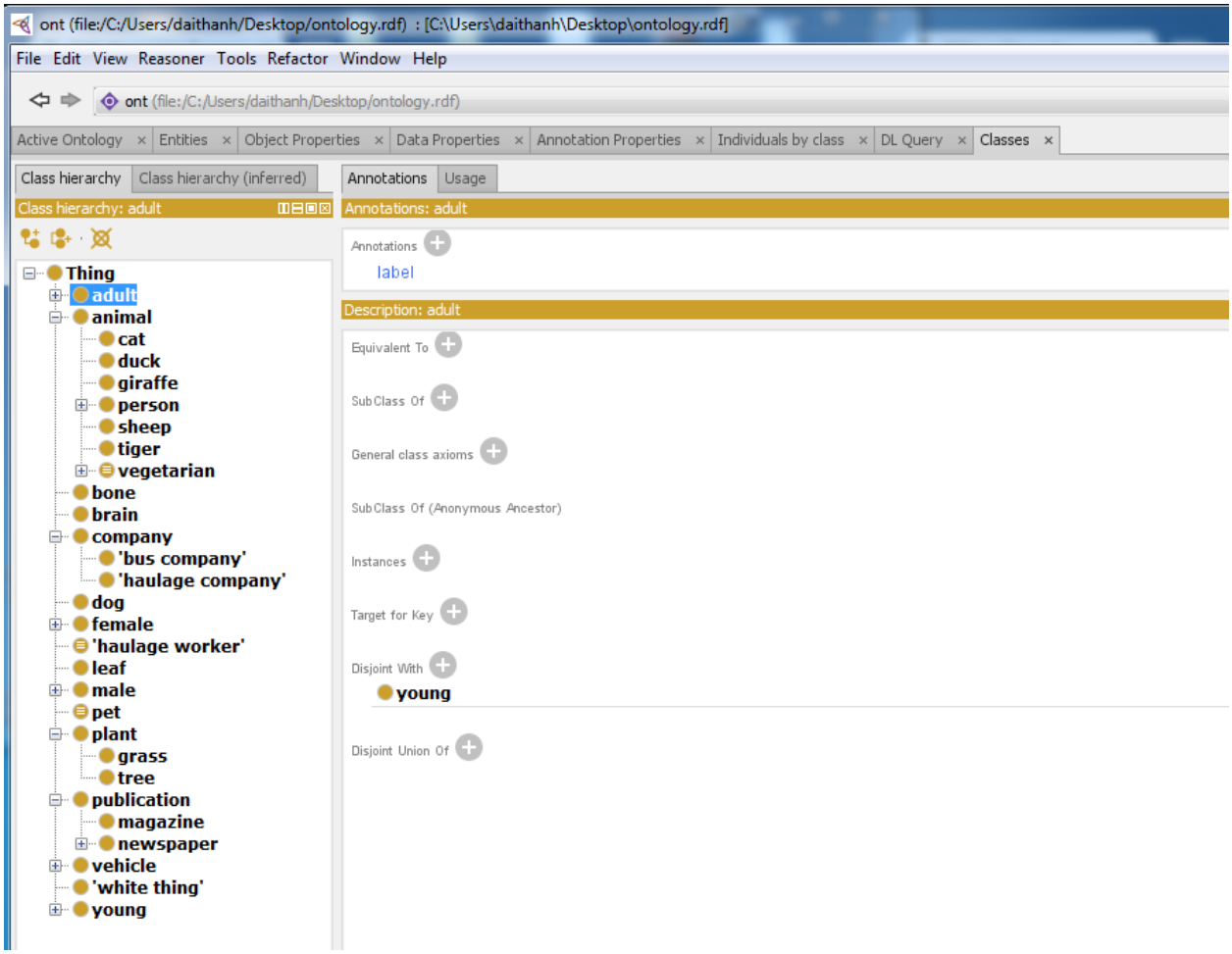
2.4. Một số ví dụ suy diễn trên mô hình bản thể học

Phần này sẽ trình bày một số ví dụ đơn giản các Ontology OWL và thảo luận một số suy diễn mà có thể được thực hiện về các lớp (class) và những cá thể (individuals) trong những bản thể học.

OWL Web Ontology language mô tả một ngôn ngữ cho các Ontology. Ngôn ngữ này được trang bị với một ngữ nghĩa được định dạng trong OWL Web Ontology Semantics và cú pháp trừu tượng (đã trình bày ở chương 1). Sử dụng những ngữ nghĩa, suy luận về bản thể và individuals là có thể được thực hiện. Nó không phải là luôn luôn rõ ràng lý do tại sao những suy diễn đã xảy ra, tuy nhiên. Giải thích về quá trình suy luận là một chủ đề quan tâm nghiên cứu – tuy nhiên điều này vẫn còn là để đạt được một trạng thái nơi nó có hiệu quả.

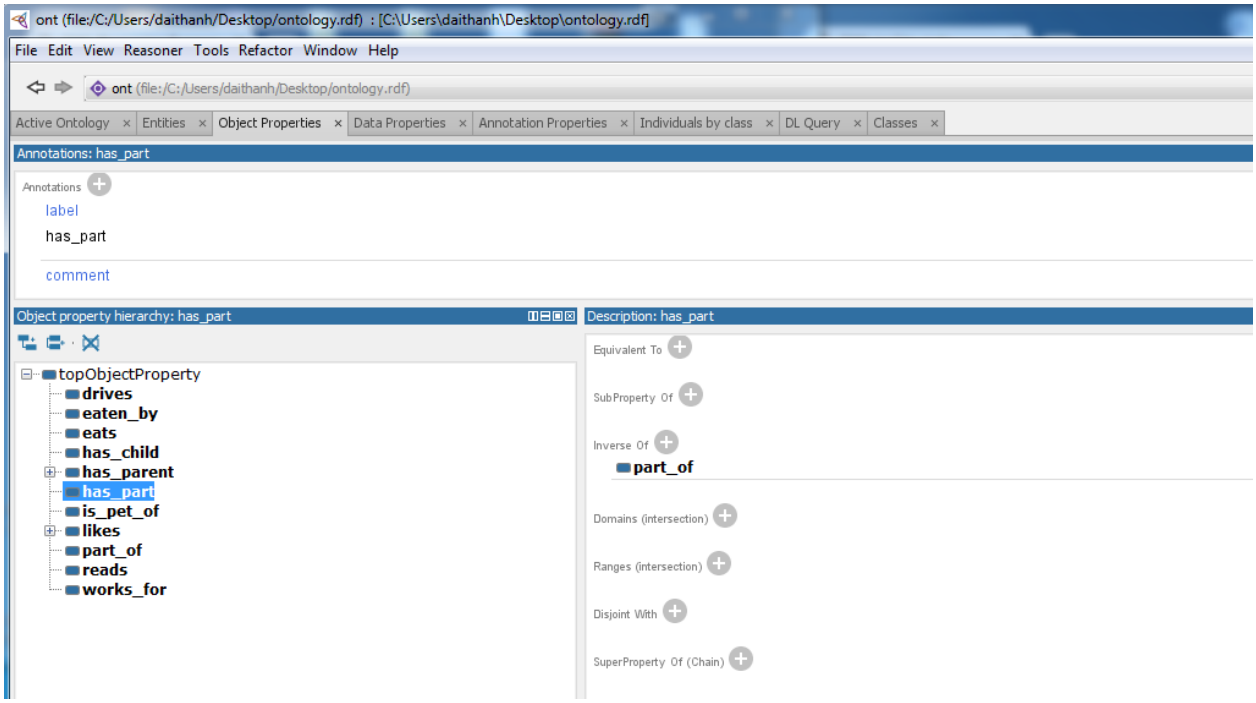
Phần này ta sẽ làm việc với Ontology đơn giản mục tiêu để minh họa quá trình suy diễn làm thế nào có thể được rút ra trong mô hình bản thể học của chúng.

Mộ bản thể học về people được được cung cấp sẵn dưới dạng RDF/XML [1]. Nó có thể được thể hiện trong cú pháp trừu tượng dạng:



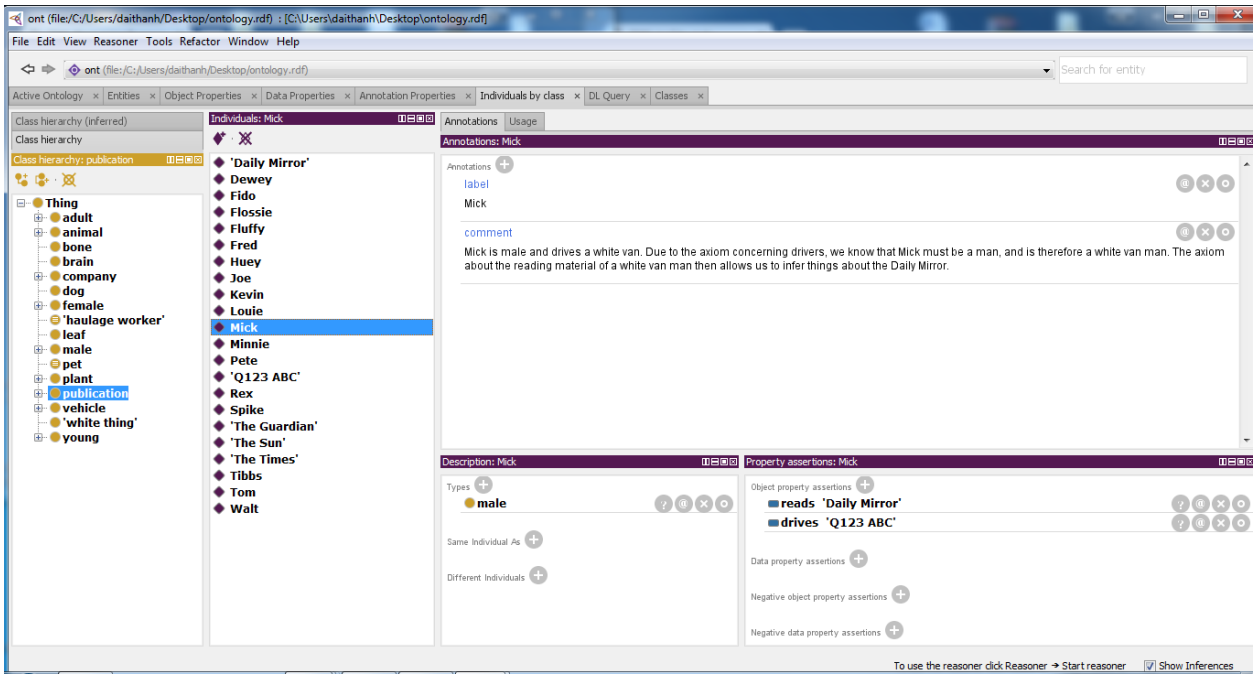
Hình 2-2: Hình ảnh minh họa các lớp của bản thể học

Hình ảnh những thành phần khác của bản thể học này như thể hiện ở các hình dưới:



Hình 2-3: Những thuộc tính đối tượng (Object property)

Mô tả những cá thể:



Hình 2-4: Mô tả những cá thể

Hai mục nhỏ dưới đây trình bày một vài ví dụ về suy diễn trên mô hình bản thể học được mô tả ở trên.

2.4.1. Suy diễn trên lớp (Classes Inferences)

Một trong những tính năng chính của bản thể học được mô tả bằng OWL – DL là chúng có thể được xử lý bởi một suy diễn. Một trong những dịch vụ chính được cung cấp bởi một suy diễn là suy ra được có hay không một lớp (Class) là một lớp con (subClass) của một lớp khác [10 – trang 48,86] bằng cách thực hiện các kiểm tra để hệ thống tính toán xem có thể có sự phân cấp trên lớp không?. Xét ví dụ đơn giản như sau:

Ví dụ 1: Suy diễn ra kết quả: Chủ của con mèo là người yêu thích mèo

Ta có ba luật sau:

1) *Class(a:cat_owner complete intersectionOf(a:person restriction(a:has_pet someValuesFrom (a:cat))))*

2) *SubPropertyOf(a:has_pet a:likes)*

3) *Class(a:cat_liker complete intersectionOf(a:person restriction(a:likes someValuesFrom (a:cat))))*

Trong ví dụ này, luật thứ nhất phát biểu rằng: Chủ của con mèo có thú cưng là mèo, luật thứ ba nói rằng: Người yêu thích mèo chắc chắn sẽ yêu thích một vài đặc điểm nào đó của mèo, luật thứ hai chỉ ra rằng thú cưng (has_pet) là thuộc tính con của thuộc tính likes. Do vậy nhờ luật thứ hai này mà ta có thể suy ra được chủ của con mèo sẽ là người yêu thích mèo. Nói cách khác lớp những người nuôi mèo (cat_owner) là một lớp con của lớp những người yêu mèo (cat_liker)

2.4.2. Suy diễn trên thể hiện (Intence Inferences)

Suy diễn trên các thể hiện là việc sử dụng các thể hiện cụ thể nào đó và các mối quan hệ giữa các thể hiện mà ta có thể suy ra được các thể hiện tương ứng này thuộc về lớp nào. Nói cách khác từ những chi tiết ta có thể suy ra sự tổng quát. Xét ví dụ sau:

Ví dụ 2: Chỉ ra rằng: Pete là người, Spike là động vật thông qua bốn luật sau:

1) *Individual(a:Spike type(owl:Thing) value(a:is_pet_of a:Pete))*

2) *Individual(a:Pete type(owl:Thing))*

3) *ObjectProperty(a:has_pet domain(a:person) range(a:animal))*

4) *ObjectProperty(a:is_pet_of inverseOf(a:has_pet))*

Luật thứ nhất nói rằng Spike là một đối tượng và là một thú cưng của Pete. Luật thứ hai nói rằng: Pete cũng là một đối tượng, tiếp theo luật thứ ba chỉ ra rằng có_thú_cưng (has_pet) là một mối quan hệ giữa hai đối tượng có chiều từ người (person) tới động vật (animal). Luật thứ bốn chỉ ra mối quan hệ là_thú_cưng_của (is_pet_of) chiều ngược lại với quan hệ trong luật ba. Từ bốn luật trên có thể suy luận ra rằng Pete là một thể hiện của lớp người (people) và Spike là một thể hiện của lớp động vật (animal). Hay nói cách khác Pete là người, còn Spkike là động vật.

2.5. Sự phân phối trên những quy tắc

Quy tắc phân phối để định lượng hiện sinh (existentials) tương tự như trong logic mệnh đề cho kết hợp và phân ly, ví dụ như:

Các hiện sinh (existials)

$$A \sqcap (B \sqcup C) \equiv (A \sqcap B) \sqcup (A \sqcap C)$$

$$\exists p.(A \sqcup B) \equiv (\exists p.A) \sqcup (\exists p.B)$$

Trong terms của ngôn ngữ OWL, phiên dịch ra sẽ như sau:

```
restriction(some p unionOf(A B))
    ≡
unionOf(restriction(some (p A))
      restriction(some (p B)))
```

Ngoài ra còn có một số suy luận là yếu hơn so với sự tương đương, xét các phép toán mô tả sau:

$$\exists p.(A \sqcap B) \sqsubseteq (\exists p.A) \sqcap (\exists p.B)$$
$$(\exists p.A) \sqcap (\exists p.B) \sqsubseteq (\exists p.A) \sqcup (\exists p.B)$$
$$(\exists p.A) \sqcap (\exists p.B) \sqsubseteq \exists p.(A \sqcup B)$$

Những phép toán này sẽ tương đương được mô tả bởi ngôn ngữ OWL tương ứng như sau:

```
restriction(some p intersectionOf(A B)
⊑
intersectionOf(restriction(some (p A))
          restriction(some (p B)))

intersectionOf(restriction(some (p A))
          restriction(some (p B)))
⊑
```

unionOf(restriction(some (p A))
restriction(some (p B)))

intersectionOf(restriction(some (p A)) restriction(some (p B)))
⊑
restriction(some p unionOf(A B)

Ta cũng suy ra rằng phép toán Union là phân phối trong existentials, trong khi intersection thì không.

Ta hãy xét ví dụ về Ontology đơn giản facts.rdf để chứng tỏ những điều trên:

Namespace(a= <http://oiled.man.example.net/facts#>)

Ontology(

ObjectProperty(a:hasFriend)
ObjectProperty(a:isFriendOf
inverseOf(a:hasFriend))

Class(a:Academic partial
a:Person)
Class(a:Academic partial)
Class(a:Happy partial a:Person)
Class(a:Happy partial)
Class(a:Lecturer partial a:Academic)
Class(a:Person partial)
Class(a:Professor partial a:Academic)
Class(a:Student partial a:Person)

AnnotationProperty(rdfs:comment)
AnnotationProperty(rdfs:label)

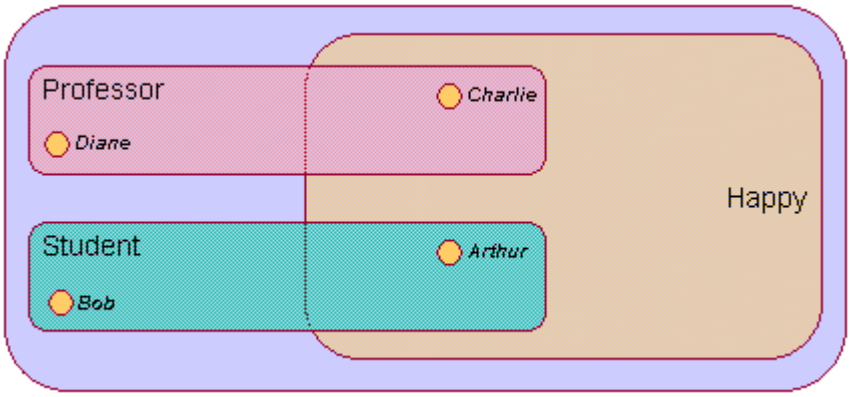
Individual(a:Arthur type(a:Happy) type(a:Student))
Individual(a:Bob type(complementOf(a:Happy)) type(a:Student))
Individual(a:Charlie type(a:Professor) type(a:Happy))
Individual(a:Diane type(a:Professor) type(complementOf(a:Happy)))
Individual(a:Patricia
type(owl:Thing)
value(a:hasFriend a:Arthur))
Individual(a:Quentin
type(owl:Thing)
value(a:hasFriend a:Bob)
value(a:hasFriend a:Charlie))

```
Individual(a:Richard
 type(owl:Thing)
 value(a:hasFriend a:Charlie))
Individual(a:Roberta
 type(owl:Thing)
 value(a:hasFriend a:Bob))
Individual(a:William
 type(restriction(a:hasFriend cardinality(0))))
Individual(a:Xanthe
 type(restriction(a:hasFriend cardinality(1)))
 value(a:hasFriend a:Arthur))
Individual(a:Yolanda
 type(restriction(a:hasFriend cardinality(2)))
 value(a:hasFriend a:Bob)
 value(a:hasFriend a:Charlie))
Individual(a:Zaphod
 type(restriction(a:hasFriend cardinality(1)))
 value(a:hasFriend a:Charlie))
Individual(a:Zeke
 type(restriction(a:hasFriend cardinality(1)))
 value(a:hasFriend a:Bob))

AllDifferent(a:Arthur a:Bob a:Charlie a:Diane a:Patricia
  a:Quentin a:Richard a:Roberta a:William a:Xanthe a:Yolanda
  a:Zaphod a:Zeke)

DisjointClasses(a:Academic a:Student)
)
```

Bản thể chứa một số lớp như là: Person (người), Academic (sự học), Professor (giáo sư) và Student (sinh viên). Ngoài ra còn có một lớp con của Happy Person và một tiên đề (axiom) nói rằng Students (sinh viên) và Academics (học giả) là tách rời nhau. Lưu ý rằng chúng ta có thể suy ra rằng Professors (giáo sư) và Students (sinh viên) thường tách rời do các tiên đề disjointness liên quan đến Academics (học giả) và Student (sinh viên). Bốn Individual (cá thể) Arthur, Bob, Diane và Charlie sau đó chiếm phân vùng khác nhau của domain



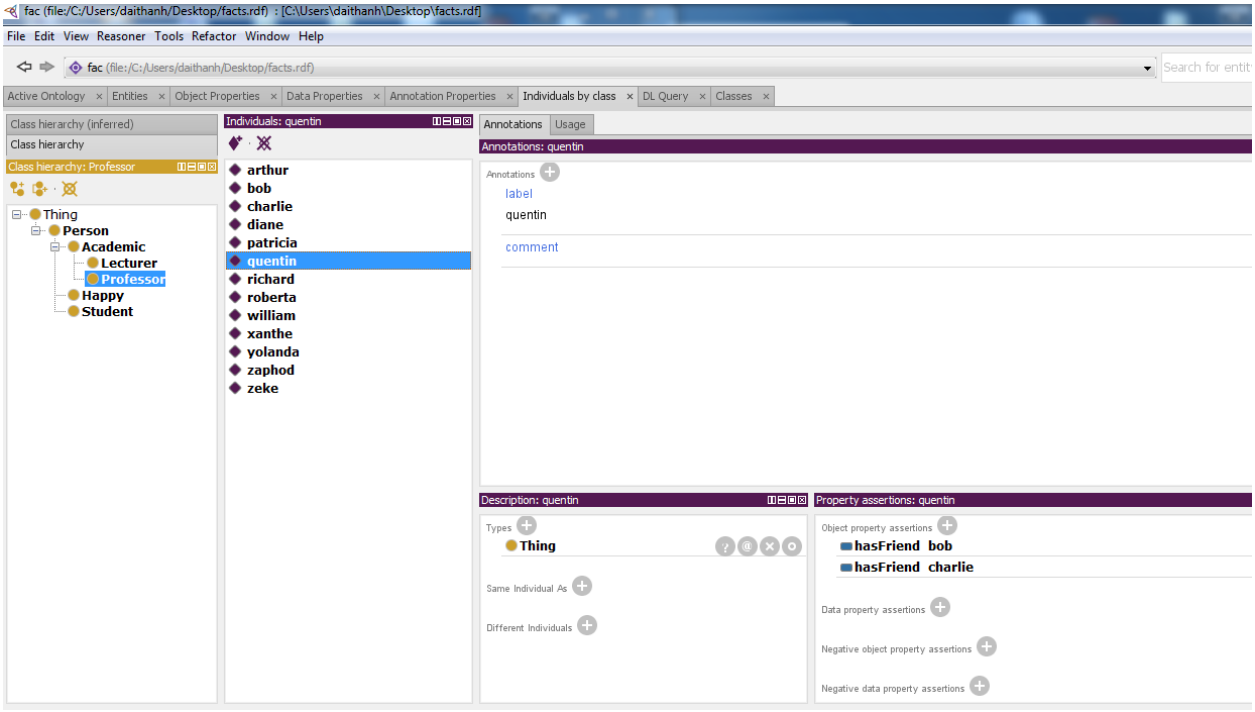
Hình 2-5: mô tả sự phân phối các lớp (class) trong ontology

Các cá thể khác nhau hiện tại cung cấp nhân chứng cho việc không tương đương (non – equivalentce) của các định nghĩa. Ví dụ:

restriction(some p intersectionOf(A B))
/≡
intersectionOf(restriction(some (p A)) restriction(some (p B)))

Quentin có một người bạn thuộc class Happy (tên là Charlie) và một người bạn thuộc lớp Student (tên là Bob). Tuy nhiên Quentin là không biết đến có một người bạn mà thuộc cả hai class Happy và Student. Chúng tôi không thể suy ra rằng Quentin là một thể hiện của lớp thứ hai, nhưng không phải những lớp đầu tiên. Như vậy phần mở rộng của họ là không giống nhau.

Hình dưới thể hiện mô tả phân cấp các lớp và các individuals



Hình 2-6: Phân cấp các lớp và các individuals

Các Universal (phổ quát)

Rules for universal quantification are similar (những quy tắc cho lượng hóa phổ quát tương tự như):

$\forall p.(A \sqcap B) \equiv (\forall p.A) \sqcap (\forall p.B)$

Trong terms của OW, chúng ta có dịch tương ứng như sau:

restriction(all p intersectionOf(A B))
≡
intersectionOf(restriction(all (p A))
restriction(all (p B)))

Có một số suy diễn một lần nữa là yếu hơn so với sự tương đương:

$(\forall p.A) \sqcap (\forall p.B) \sqsubseteq (\forall p.A) \sqcup (\forall p.B)$
$\forall p.(A \sqcap B) \sqsubseteq (\forall p.A) \sqcup (\forall p.B)$
$(\forall p.A) \sqcup (\forall p.B) \sqsubseteq (\forall p.(A \sqcup B))$

Trong OWL như sau:

```
intersectionOf(restriction(all (p A))
               restriction(all (p B)))
⊑
unionOf(restriction(all (p A))
        restriction(all (p B)))

restriction(all p intersectionOf(A B)
⊑
unionOf(restriction(all (p A))
        restriction(all (p B)))

unionOf(restriction(all (p A))
        restriction(all (p B)))
⊑
restriction(all p unionOf(A B)
```

Ở đây ta có thể thấy Intersection là phân phối trong phổ quát (universals), còn union thì không.

CHƯƠNG 3. PHÁT TRIỂN ỨNG DỤNG THỬ NGHIỆM

Nội dung chính của chương này bao gồm:

- Ứng dụng nghiên cứu vấn đề gì?, giải pháp thực hiện như thế nào?

- Phân tích mô hình bản thể học lĩnh vực chuyên ngành cây trồng.

- Phân tích những chức năng và xây dựng những tập luật suy diễn cho hệ thống.

3.1. Đặt vấn đề

Những khó khăn vất vả của người làm nông nghiệp trồng trọt, một trong những yếu tố đó là việc tiếp nhận thông tin hữu ích trong khoa học quy trình của cây trồng như các đặc tính: mùa vụ, loại cây trồng, thời gian dự kiến được thu hoạch…

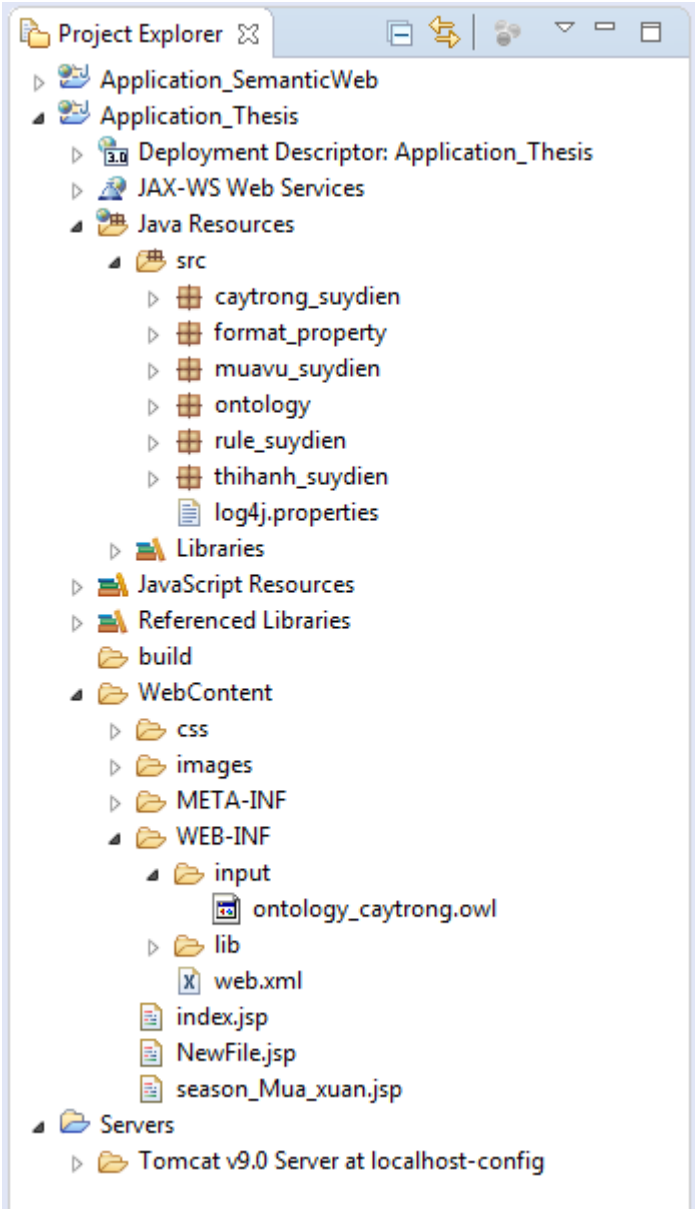
Trên tinh thần muốn hỗ trợ người trồng trọt khai thác thông tin hữu ích về những vấn đề trên, chúng tôi xây dựng một ứng dụng nhỏ nhằm khai thác một số chức năng cần thiết cho người trồng trọt cây trồng trong vườn nhà.

Chương trình ứng dụng có áp dụng sự suy diễn trên mô hình bản thể học, trong đó có bao gồm việc xây dựng một Ontology về lĩnh vực cây trồng, và việc xây dựng các tập luật suy diễn nhằm đưa ra các chức năng thông minh của hệ thống.

3.2. Giải pháp thực hiện

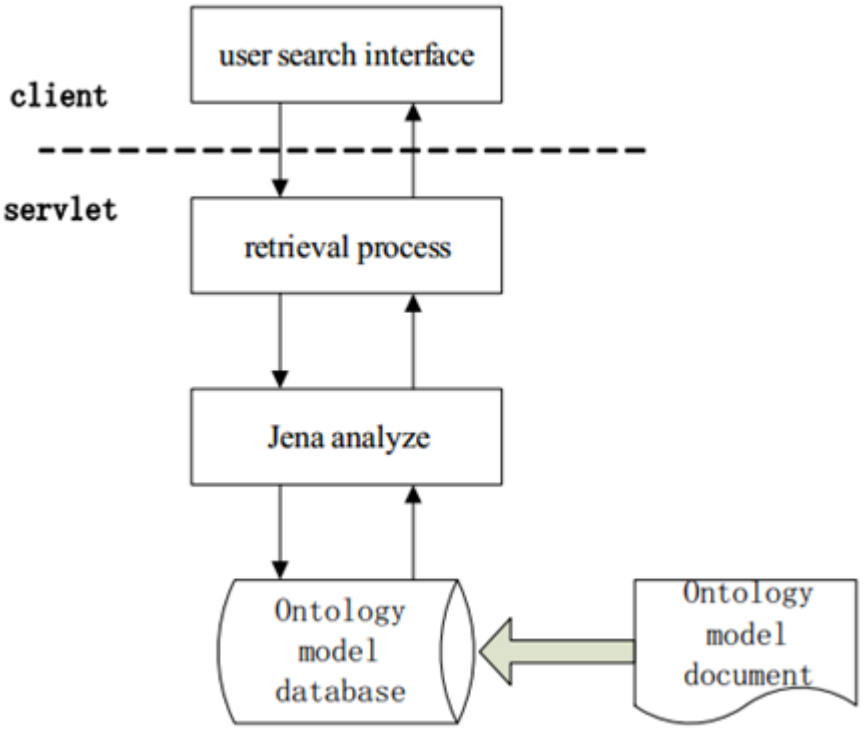
Để thực hiện việc xây dựng chương trình ứng dụng, chúng tôi triển khai hướng một ứng dụng Semantic Web, tức là ứng dụng có khả năng khai thác trên Internet dựa trên việc chỉ ra các đường dẫn URL. Những công việc thực hiện bao gồm: Xây dựng một bản thể học (Ontology) đặt tên là: ontology_caytrong.owl ; tiếp theo nghiên cứu xây dựng các tập luật để hỗ trợ việc suy diễn để tạo các chức năng cho hệ thống.

Trong đó chúng tôi sử dụng các công cụ hỗ trợ như: Protégé 5.0 để xây dựng Ontology, công cụ Eclipse phiên bản neo để soạn thảo chương trình, máy chủ Apache Tomcat để chạy ứng dụng web; sử dụng ngôn ngữ lập trình Java, JSP, Servlet, CSS…



Hình 3-1: Project Application_Theis và Application_SemanticWeb

Sử dụng Jena Framework đóng vai trò máy suy luận, với giao diện hoạt động của người dùng và hệ thống như hình sau:

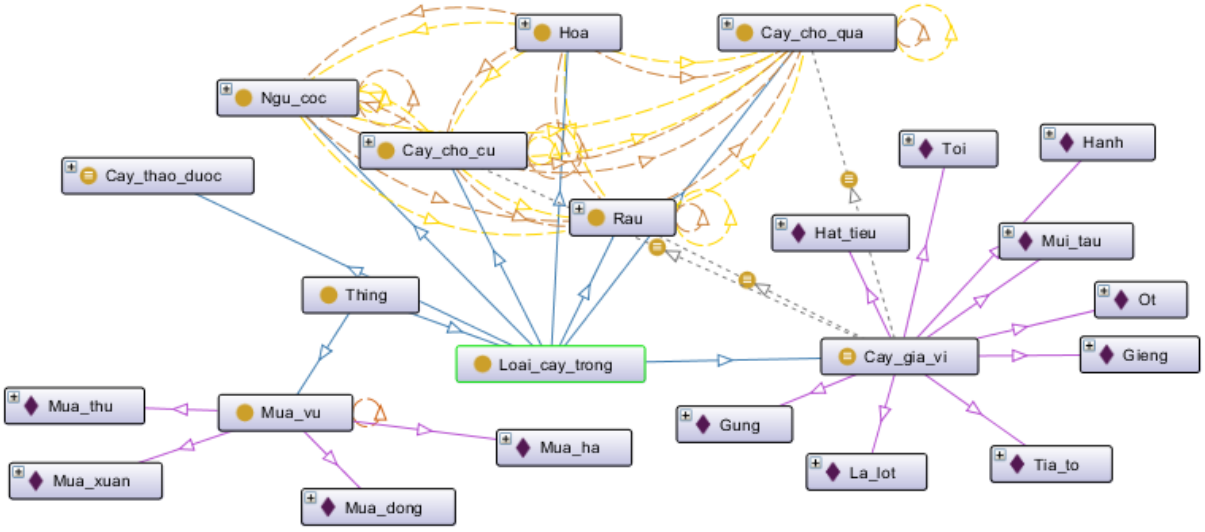


Hình 3-1-1: Cấu trúc mô hình giao tiếp của người dùng với hệ thống

3.3. Xây dựng ứng dụng

3.3.1. Xây dựng bản thể học (Ontology)

Dưới đây là hình ảnh thu gọn của bản thể học:



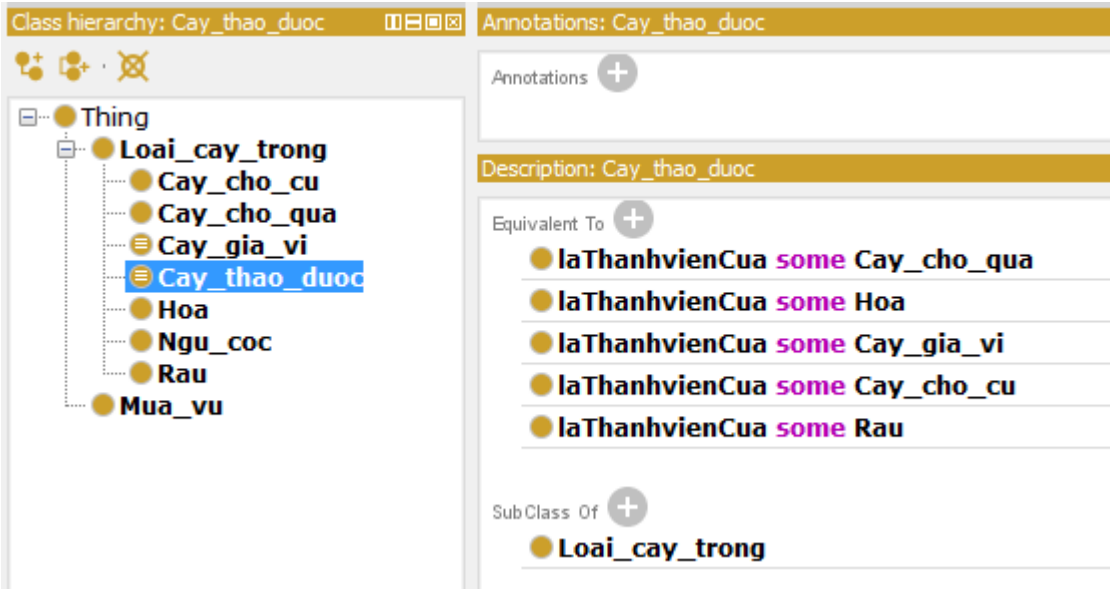
Hình 3-2: Hình ảnh mô hình bản thể học ontology_caytrong.owl

Bảng những loại cây trồng và mùa vụ (những thể hiện – individuals) của ontology bao gồm:

Loại cây trồng và mùa vụ	**Các thể hiện (individuals)**	**Ghi chú**
Cay_cho_cu	Dau_lac, Gieng, Gung, Hanh, Khoai_lang, Khoai_tay, San_day, San_tau, Toi,…	Cây cho củ
Cay_cho_qua	Bau, Bi, Cam, Buoi, Ngo, Tao, Xoai, Dau_xanh, Dau_tuong, Ot,…	Cây cho quả
Cay_gia_vi	Gieng, Gung, Hat_tieu, La_lot, Ot, Hanh, Toi,…	Cây làm gia vị
Cay_thao_duoc	Ac_ti_so, Ngai_cuu, Ngot, Tia_to, Toi,…	Cây làm thảo dược
Hoa	Cuc, Hong, Nhai,…	Cây hoa
Ngu_coc	Dau_den, Dau_tuong, Dau_lac, Dau_xanh, Ngo,…	Ngũ cốc
Rau	Bau, Bi, Cai_bap, Xu_hao, Muop, Ngai_cuu, Muong, Ngot,…	Cây rau
Mua_vu	Mua_xuan, Mua_ha, Mua_thu, Mua_dong	Các mùa trong năm

Bảng 3-1: Những thể hiện (individuals) của ontology

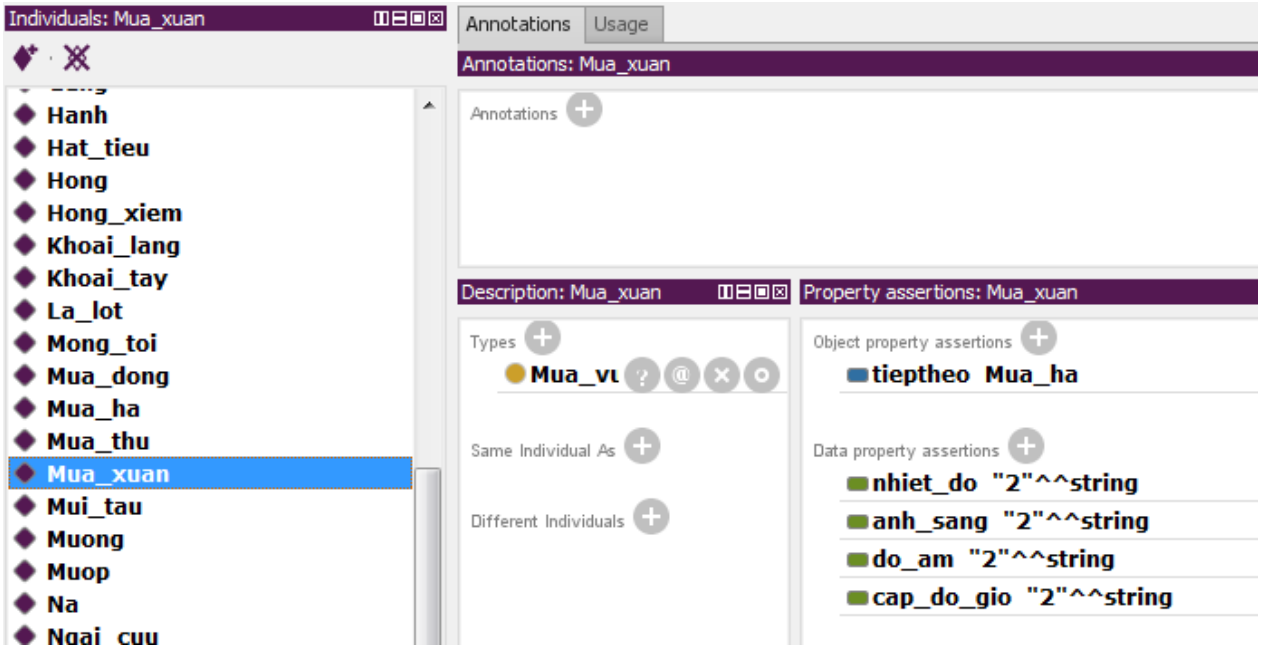
Những lớp (Classes) của Ontology được thể hiện phân cấp như sau:



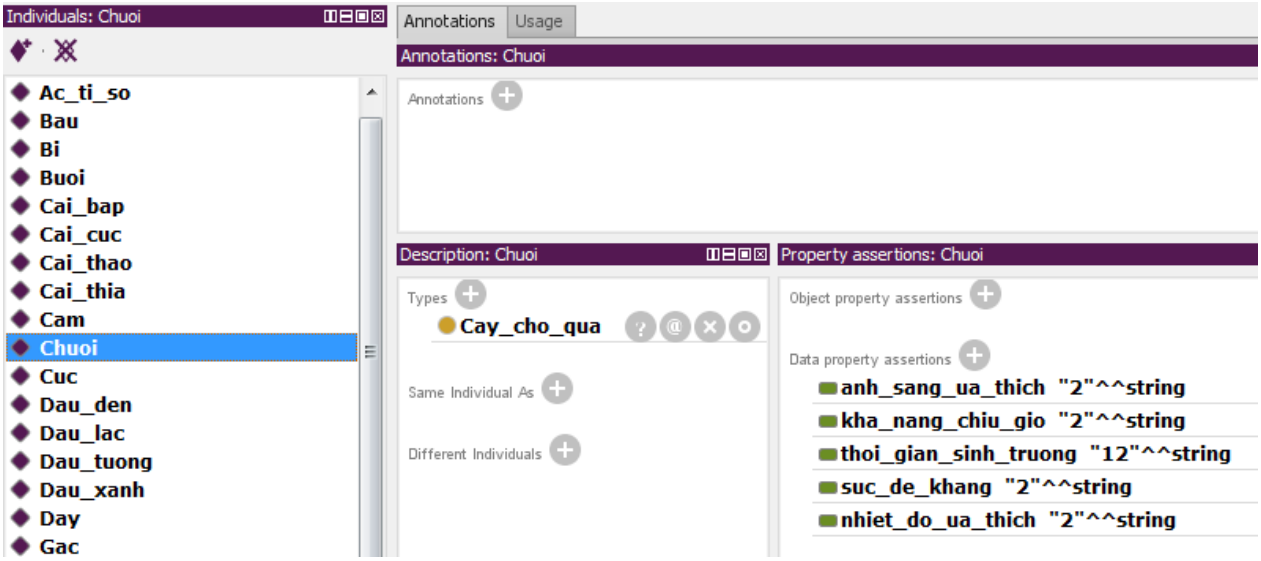
Hình 3-3: Mô tả sự phân cấp lớp trong ontology

Trong đó chú ý mô tả lớp những cây_gia_vi và lớp những cây_thảo_dược

Các Individuals và các thuộc tính đặc biệt của mỗi loại cây trồng và mùa vụ được thể hiện như hình sau:



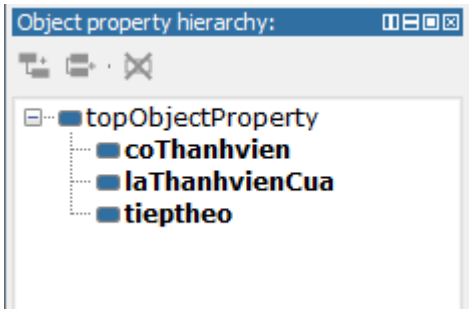
Hình 3-4: Individual Mùa vụ và những thuộc tính mô tả cho nó



Hình 3-5: Individual cây trồng và những thuộc tính mô tả cho nó

Các thuộc tính Object Property và Data Property bao gồm:

- coThanhvien: có thành viên, dùng để mô tả một loại cây trồng có các cây trồng nào.
- laThanhvienCua: là thành viên của, dùng để mô tả một cây trồng cụ thể thuộc một loại cây trồng nào.
- tieptheo: tiếp theo, dùng để mô tả mùa vụ tiếp theo sau một mùa vụ, ví dụ tieptheo của Mua_xuan là Mua_ha,…



Hình 3-6: Các thuộc tính đối tượng

Các thuộc tính Data Property bao gồm:

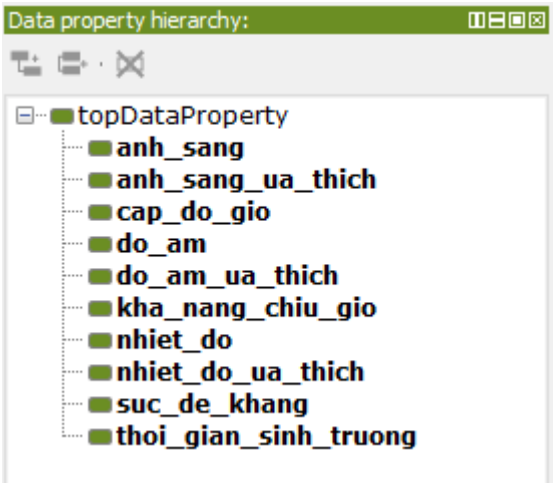
Các thuộc tính của mùa vụ:

- anhsang: mô tả ánh sáng của mùa vụ
- cap_do_gio: mô tả cấp độ gió của mùa vụ
- do_am: mô tả độ ẩm trung bình của mùa vụ
- nhiet_do: mô tả nhiệt độ trung bình của mùa vụ

Các thuộc tính của cây trồng:

- anh_sang_ua_thich: mô tả ánh sáng trung bình mà cây trồng ưa thích
- do_am_ua_thich: mô tả độ ẩm trung bình mà cây trồng ưa thích
- kha_nang_chiu_gio: mô tả khả năng chịu gió của cây trồng
- nhiet_do_ua_thich: mô tả nhiệt độ ưa thích của cây trồng
- suc_de_khang: mô tả sức đề kháng của cây trồng
- thoi_gian_sinh_truong: mô tả thời gian sinh trưởng của cây trồng.

Dưới đây là hình ảnh cho các thuộc tính Data Property:



Hình 3-7: Thuộc tính Data Property

3.3.2. Suy diễn và phát triển hệ thống

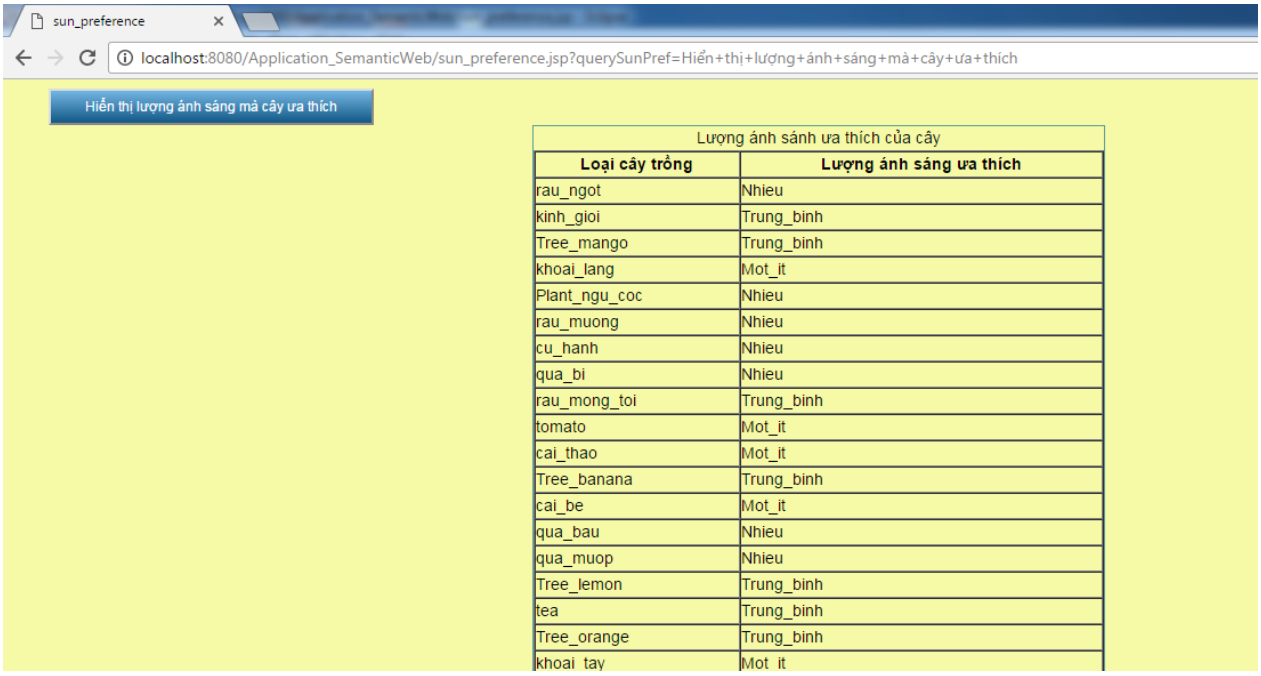
Để suy diễn ra các chức năng của hệ thống ta đi xây dựng các tập luật suy diễn, những tập luật suy diễn được xây dựng nhằm mục đích truy vấn và suy luận trên mô hình bản thể học nhằm đưa ra kết quả phù hợp với yêu cầu. Một luật suy diễn có thể bao gồm nhiều luật con trong nó. Ở đây ta sử dụng ngôn ngữ truy vấn SPARQL (đã được giới thiệu ở chương trước).

Hình ảnh giao diện chính của hệ thống, thực hiện chức năng gợi ý những cây trồng mà người dùng muốn trồng theo các tiêu chí phù hợp

Hình 3-8: Chức năng gợi ý những cây trồng theo tiêu chí của người làm vườn

Những đặc tính quan trọng của cây trồng đó là lượng ánh sáng ưa thích, điều này được suy diễn với quy tắc ngữ nghĩa sau:

```
String queryString = "PREFIX garden: <" + rel + "> "
                                    + "SELECT ?x ?z "
                                + " WHERE {
                                + "?x garden:sun_preference ?z "
                                + "}";
```



Lượng ánh sánh ưa thích của cây

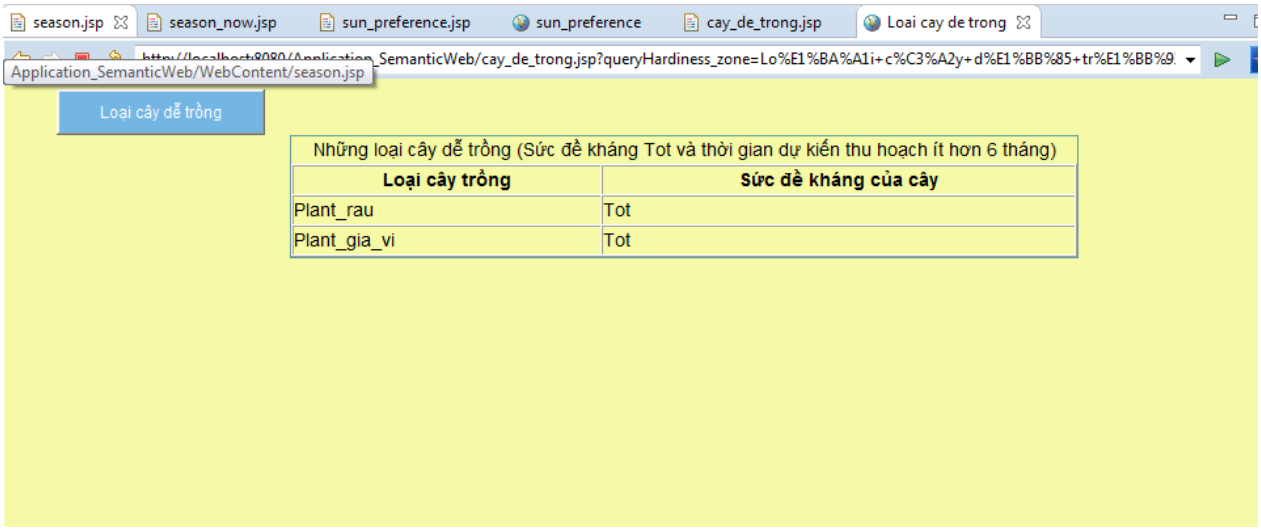
Loại cây trồng	Lượng ánh sáng ưa thích
rau_ngot	Nhieu
kinh_gioi	Trung_binh
Tree_mango	Trung_binh
khoai_lang	Mot_it
Plant_ngu_coc	Nhieu
rau_muong	Nhieu
cu_hanh	Nhieu
qua_bi	Nhieu
rau_mong_toi	Trung_binh
tomato	Mot_it
cai_thao	Mot_it
Tree_banana	Trung_binh
cai_be	Mot_it
qua_bau	Nhieu
qua_muop	Nhieu
Tree_lemon	Trung_binh
tea	Trung_binh
Tree_orange	Trung_binh
khoai_tay	Mot_it

Hình 3-9: Suy diễn ra chức năng yếu tố ánh sáng ưa thích của loại cây trồng

Những cây dễ trồng là những cây mà có Sức đề kháng tốt và thời gian thu hoạch ngắn. Điều này được suy diễn nhờ luật sau:

```
String queryString = "PREFIX garden: <" + rel + "> "
                   + "SELECT ?x ?z "
                   + "WHERE {"
                   + "{?x garden:hardiness_zone ?z . "
                   + " ?x garden:hardiness_zone \"" + "Tot" + "\" ."
                   + " ?x garden:harvest_duration \"" + "it_hon_3" + "\" . "
                   + "}"

                   + "UNION"
                   + "{?x garden:hardiness_zone ?z . "
                   + " ?x garden:hardiness_zone \"" + "Tot" + "\" ."
                   + " ?x garden:harvest_duration \"" + "it_hon_6" + "\" . "
                   + "}"
                   + "}";
```

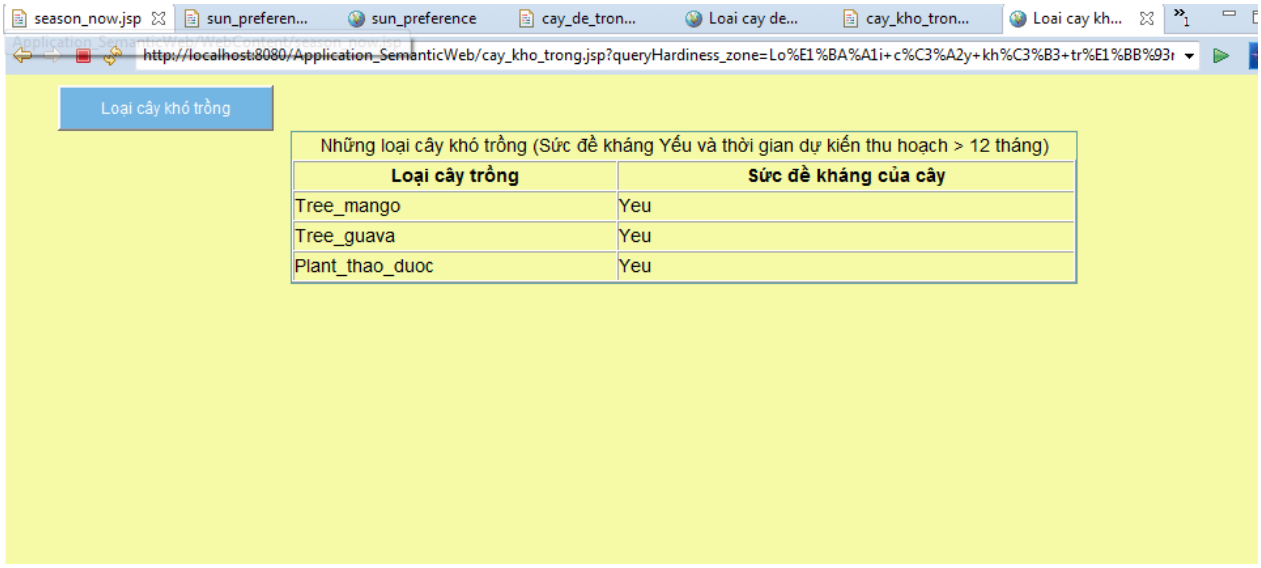


Hình 3-10: Suy diễn ra chức năng đưa ra những cây dễ gieo, trồng

Những cây khó gieo trồng là những cây có Sức đề kháng kém và thời gian thu hoạch dài. Điều này có thể được suy diễn nhờ luật sau:

```
String queryString = "PREFIX garden: <" + rel + "> "
                   + "SELECT ?x ?z "
             + "WHERE {"
                   + "{?x garden:hardiness_zone ?z . "
                   + " ?x garden:hardiness_zone \"" + "Yeu" + "\" ."
                   + " ?x garden:harvest_duration \"" + "it_hon_24" + "\" . "
                   + "}"

                   + "UNION"
                   + "{?x garden:hardiness_zone ?z . "
                   + " ?x garden:hardiness_zone \"" + "Yeu" + "\" ."
                   + " ?x garden:harvest_duration \"" + "it_hon_12" + "\" . "
                   + "}"
                   + "}";
```



Hình 3-11: Suy luận ra những cây khó gieo trồng

Những cây trồng mà phù hợp với một mùa vụ cụ thể cũng được suy diễn ra nhờ vào luật suy diễn sau:

```
String queryString = "PREFIX caytrong: <" + rel + "> "
                          +"SELECT ?x ?as ?da ?gio ?nd ?sdk ?tgst "
                          +"WHERE {"
                          + "?x caytrong:anh_sang_ua_thich ?as . "
                          + "?x caytrong:do_am_ua_thich ?da . "
                          + "?x caytrong:kha_nang_chiu_gio ?gio . "
                          + "?x caytrong:nhiet_do_ua_thich ?nd . "
                          + "?x caytrong:suc_de_khang ?sdk . "
                          + "?x caytrong:thoi_gian_sinh_truong ?tgst . "
                                + "} " ;
```

Sau đó lọc theo luật mùa vụ phù hợp:

```
String queryString = "PREFIX muavu: <" + rel + "> "
                                + "SELECT ?mua ?as ?cdgio ?da ?nd "
                                + "WHERE {"
                                + "?mua muavu:anh_sang ?as ."
                                + "?mua muavu:cap_do_gio ?cdgio ."
                                + "?mua muavu:do_am ?da ."
                                + "?mua muavu:nhiet_do ?nd ."
                                      + "} ";
```

Ví dụ với mùa xuân ta đi so sánh các thuộc tính của cây trồng và thuộc tính của mua vụ:

```
if (anh_sang_ut_new.equals(as_mx) && do_am_ut_new.equals(da_mx) &&
nhiet_do_ut_new.equals(nd_mx)
                                && kn_chiu_gio_new.equals(cd_gio_mx)) {
```

```
// hiển thị kết quả
System.out.print(i + ", " + mua + " ---> ");
System.out.print(caytrong_new + " ---> ");
System.out.print(as + " |");
System.out.print(da + " |");
System.out.print(gio + " |");
System.out.print(nd + " |");
System.out.print(sdk + " |");
System.out.println(tgst);
}
```

Và kết quả nhận được là:

```
3, Mua_xuan ---> Tia_to ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
6, Mua_xuan ---> Ot ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
10, Mua_xuan ---> Tao ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
16, Mua_xuan ---> San_tau ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
17, Mua_xuan ---> Nhai ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
19, Mua_xuan ---> Na ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
24, Mua_xuan ---> Hong ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
25, Mua_xuan ---> Buoi ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 12 tháng
26, Mua_xuan ---> Hat_tieu ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 12 tháng
27, Mua_xuan ---> Hanh ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
30, Mua_xuan ---> Toi ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
32, Mua_xuan ---> Cam ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 12 tháng
33, Mua_xuan ---> Gung ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 12 tháng
36, Mua_xuan ---> Hong_xiem ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 12 tháng
37, Mua_xuan ---> Quyt ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
41, Mua_xuan ---> Ngai_cuu ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 6 tháng
42, Mua_xuan ---> Gieng ---> Trung bình ánh sáng |Trung bình độ ẩm |Trung bình |Trung bình |Trung bình |Khoảng 12 tháng
```

Hình 3-12: Suy diễn ra những cây trồng phù hợp với mùa xuân

3.4. Đánh giá kết quả ứng dụng

Ứng dụng Semantic Web được xây dựng trong đó gồm hai thành phần cơ bản đó là: Xây dựng một Ontology lĩnh vực cây trồng và các đặc tính của nó, và việc xây dựng nhiều tập luật suy diễn trên mô hình ontology này. Việc xây dựng Ontology đã mô tả khá đầy đủ, chi tiết về những nội dung yếu tố quan trọng nhất của cây trồng – đó là các đặc tính quan trọng của cây trồng, những đặc tính mùa vụ riêng biệt. Những điều này lý giải tại sao một số cây trồng lại trồng ở mùa vụ này, còn một số cây trồng khác lại trồng ở mùa vụ khác.

Việc xây dựng ứng dụng Semantic Web trong đó có sử dụng việc suy diễn trên mô hình bản thể học cây trồng khá hiệu quả. Mặt rất mạnh của ứng dụng loại này là khả năng khai thác các chức năng cần thiết rất hiệu quả và thông minh, mềm dẻo nhờ vào việc xây dựng ra rất nhiều các tập luật suy diễn một cách dễ dàng.

Hệ thống cơ bản trợ giúp cho người dùng khai thác chức năng trên giao diện Web trên Internet dễ dàng bằng cách chỉ ra các đường dẫn URL ở mọi nơi, nên khá dễ dàng và hiệu quả.

KẾT LUẬN

Với nhu cầu thông tin ngày càng lớn của con người, khả năng đáp ứng thông tin càng trở lên bức thiết. Kỹ thuật Web hiện nay vẫn tạo khó khăn trong việc rút trích, bảo trì và phát triển thông tin. Máy tính chỉ được dùng như một thiết bị gửi và trả thông tin. Chúng không thể truy xuất những khả năng thực sự cần thiết, do đó chúng chỉ hỗ trợ ở mức giới hạn nào đó trong việc truy xuất và xử lý thông tin. Kết quả là người sử dụng phải phải gánh trên vai trách nhiệm không những truy cập và xử lý thông tin mà còn rút trích và thông dịch mọi thông tin.

Để khắc phục những yếu điểm của Web hiện tại khái niệm "Semantic Web" đã ra đời. Semantic Web là sự mở rộng của Web hiện tại mà trong đó thông tin được xử lý một cách tự động bằng máy tính, làm cho con người và máy tính có thể hợp tác với nhau.

Suy diễn trên mô hình bản thể học là thao tác giúp ta khai thác hiệu quả trên bản thể học, bởi vì nếu không thực hiện quá trình suy diễn thì bản thể học chỉ có chức năng như kho chứa mà thôi. Suy diễn bằng các quy tắc có thể suy ra kiến thức mới, kiến thức tiềm ẩn cần thiết dựa trên những sự kiện được biết đến trước đó đã mang lại những sự hiệu quả to lớn cho thế hệ Web ngữ nghĩa.

Những điều mà luận văn đã làm được đó là: luận văn nghiên cứu các suy diễn trên mô hình bản thể học bằng cách xây dựng các tập quy tắc suy diễn, qua đó củng cố thêm sự mô tả mô hình bản thể học về một lĩnh vực, cuối cùng là việc xây dựng một ứng dụng trong đó sử dụng những nghiên cứu cho lý thuyết này, tạm đặt tên: "Chương trình hỗ trợ cho người làm vườn".

Trong đó việc nghiên cứu lý thuyết đã làm rõ một số công nghệ như:

- Nghiên cứu tìm hiểu công nghệ Semantic Web
- Nghiên cứu các thành phần cấu thành của mô hình bản thể học, từ đó chủ động xây dựng bản thể học.
- Nghiên cứu ngôn ngữ mô tả dữ liệu OWL (Ontology Web Language)
- Nghiên cứu các phương pháp suy luận trên mô hình bản thể học. Xây dựng các quy tắc suy diễn, truy vấn khai thác thông tin trên bản thể học.
- Nghiên cứu Framework Jena, và việc hỗ trợ bộ máy suy luận (Inference engine)

Việc xây dựng ứng dụng mô tả bao gồm các công việc như sau:

- Xây dựng hoàn chỉnh một bản thể học đặt tên là: ontology_caytrong.owl; trong đó làm rõ được bản chất của tri thức về cây trồng và các thuộc tính mùa vụ của mỗi loại cây trồng.
- Phân tích các thành phần chức năng của cây trồng, và đặc tính mùa vụ của cây trồng.
- Xây dựng một số tập luật suy diễn nhằm mục đích suy diễn ra các chức năng mong muốn của hệ thống như: đưa ra cây trồng phù hợp với mỗi mùa vụ: mùa xuân, mùa hạ, mùa thu, mùa đông; khai thác các đặc tính của mỗi loại cây trồng; khai thác các đặc tính của mỗi mùa vụ; Tìm những cây trồng theo tiêu chí của người trồng cây,….

Luận văn cơ bản đạt được một số kết quả mong muốn, thấy được sự ưu việt của thế hệ Semantic Web – thế hệ Web 3.0, ngôn ngữ truy vấn, và khả năng suy diễn mà sinh ra thông tin mới dựa trên ontology, giúp cải thiện đáng kể khả năng khai thác thông tin trên môi trường web.

Vì thực sự bản thể học(ontology) gắn liền với một lĩnh vực cụ thể của thực tiễn chuyên môn, nó thực sự là một hệ chuyên gia, do vậy việc xây dựng nó đủ tốt để đáp ứng cho ứng dụng là một việc đòi hỏi nhiều công sức nghiên cứu chuyên ngành

Những mặt hạn chế của luận văn:

- Xây dựng bản thể học về lĩnh vực chưa hoàn chỉnh, chưa đủ tốt để phục vụ ứng dụng.
- Xây dựng tập luật để hỗ trợ suy diễn chưa hiệu quả trong việc sinh ra thông tin mới, thông tin hữu ích.

Công việc nghiên cứu tiếp theo: Tìm hiểu về lĩnh vực chuyên môn, sâu hơn của ứng dụng thực tiễn và qua đó xây dựng lên các bản thể học về những lĩnh vực đó. Sau đó xây dựng các suy diễn trên nó, phục vụ mục đích của cuộc sống như: y tế, giáo dục, kinh doanh, môi trường,…

TÀI LIỆU THAM KHẢO

Tiếng Việt:

[27] Hoàn Nguyễn Tuấn Minh, Hoàng Hữu Hạnh (2011). Tạp chí khoa học - Đại học Huế. Các ngôn ngữ truy vấn RDF: Đánh giá tổng quan và So sánh các đặc tính ngôn ngữ.

[29] Vũ Bội Hằng (2005), Phát hiện quan hệ ngữ nghĩa Nguyên nhân – kết quả từ các văn bản, Luận văn cao học, Trường Đại học Công nghệ.

Tiếng Anh:

[1] https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

[2] Vagan Terziyan (2010) AI Department, Kharkov National University of Radioelectronics /MIT Department, University of Jyvaskyla. *Advanced It from multiagents to Semantic Web http://www.cs.jyu.fi/ai/Advanced_IT-2010.pdf*

[3] Christian Bizer, Freie Universität Berlin, Germany Tom Heath, Talis Information Ltd, United Kingdom Tim Berners-Lee, Massachusetts Institute of Technology, USA. Special Issue on Linked Data, International Journal on Semantic Web and Information Systems (IJSWIS). http://linkeddata.org/docs/ijswis-special-issue

[4] D. Lenat and R. Guha (1990) Building Large Knowledge Based Systems: Representation and Inference in the Cyc Project. Addison-Wesley Publishing. http://www.jimdavies.org/summaries/lenat1990-1.html

[5] Jean Vincent Fonou-Dombeu and Magda Huisman (2011) Semantic-Driven e Government: Application of Uschold and King Ontology Building Methodology for Semantic Ontology Models Development

[6] Fox, M.S. and Gruninger, M. (1994). Ontologies for enterprise integration, Proceedings of the Second International Conference on Cooperative Information Systems, pages 82-89.

[7] http://owl.man.ac.uk/2003/why/latest/

[8] Seongwook Youn, Dennis McLeod (2006) University of Southern California, Los Angeles, USA Dennis McLeod University of Southern California, Los Angeles, USA. Ontology Development Tools for Ontology – based Knowledge Management.

[9] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. http://www.w3.org/Submission/SWRL/

[10] Matthew Horridge (2011) A Practical Guide To Building OWL Ontologies Using Protégé4 and CO-ODE Tools Edition 1.3 The University Of Manchester

[20] Jena - A Semantic Web Framework for Java. Project homepage. http://jena.sourceforge.net.

[21] Dave Beckett, R.V. Guha (2004), *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation 10 February 2004, http://www.w3.org/TR/rdf-schema/

[22] Frank Manola, Eric Miller(2004), *RDF Primer*, W3C Recommendation 10 February2004, http://www.w3.org/TR/rdf-primer/

[23] Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale of RuleML(2001): A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium (SWWS'01)*. Stanford University, July/August 2001.

[24] OWL 2: Proflles, http://www.w3.org/TR/owl2-profiles (2009)

[25] G. Meditskos, N. Bassiliades (2009), Rule-based OWL Reasoning Systems: Implementations, Strengths and Weaknesses, Handbook of Research on Emerging Rule Based Languages and Technologies: Open Solutions and Approaches, IGI Global, ISBN Number 978-1-60566-402-6, 2009.

[26] G. Meditskos, N. Bassiliades(2008), Combining a DL Reasoner and a Rule Engine for Improving Entailment-Based OWL Reasoning, in: 7th International Semantic Web Conference (ISWC 2008), Karlsruhe, Germany, 2008.

Website:

[11] http://www.w3.org/People/Berners-Lee/Weaving/Overview.html

[12] http://www.semanticweb.org

[13] https://jena.apache.org/index.html

[14] https://www.w3.org/TR/rdf-sparql-query/

[15] http://dior.ics.muni.cz/~makub/owl/

[16] https://docs.marklogic.com/guide/semantics/inferencing

[17] http://owl.man.ac.uk/2003/why/latest/

[18] http://webprotege.stanford.edu/#List:coll=Home;

[19] http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library