ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

ĐẶNG LÊ BẢO CHƯƠNG
NGUYỄN BẢO DUY

KHÓA LUẬN TỐT NGHIỆP

XÂY DỰNG ONTOLOGY PHỤC VỤ CHO VIỆC PHÂN LOẠI HÀNG HÓA TỰ ĐỘNG

KỸ SƯ NGÀNH PHÁT TRIỂN ỨNG DỤNG WEB VÀ DI ĐỘNG

Tp. Hồ Chí Minh, 2015

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

ĐẶNG LÊ BẢO CHƯƠNG - 10520010
NGUYỄN BẢO DUY - 10520011

KHÓA LUẬN TỐT NGHIỆP

XÂY DỰNG ONTOLOGY PHỤC VỤ CHO VIỆC PHÂN LOẠI HÀNG HÓA TỰ ĐỘNG

KỸ SƯ NGÀNH PHÁT TRIỂN ỨNG DỤNG WEB VÀ DI ĐỘNG

GIẢNG VIÊN HƯỚNG DẪN
TS. NGUYỄN ANH TUẤN

Tp. Hồ Chí Minh, 2015

DANH SÁCH HỘI ĐỒNG BẢO VỆ KHÓA LUẬN

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số ... ngày .... của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

1. ..... - Chủ tịch.
2. ..... - Thư ký.
3. ..... - Ủy viên.
4. ..... - Ủy viên.

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh Phúc

Tp.HCM, Ngày 19 tháng 1 năm 2015

NHẬN XÉT KHÓA LUẬN TỐT NGHIỆP

(CỦA CÁN BỘ HƯỚNG DẪN)

Nhóm sinh viên thực hiện : XÂY DỰNG ONTOLOGY PHỤC VỤ CHO VIỆC PHÂN LOẠI HÀNG HÓA TỰ ĐỘNG

Nhóm sinh viên thực hiện :		**Cán bộ hướng dẫn :**
Đặng Lê Bảo Chương	MSSV:10520010	TS. Nguyễn Anh Tuấn
Nguyễn Bảo Duy	MSSV:10520011	

Đánh giá khóa luận :

1. Về cuốn báo cáo:
   Số trang ..... Số chương .....
   Số bảng số liệu ..... Số hình vẽ .....
   Số tài liệu tham khảo ..... Sản phẩm .....
   Một số nhận xét về hình thức cuốn báo cáo: .....
   .....
   .....
   .....

2. Về nội dung nghiên cứu: .....
   .....
   .....
   .....

3. Về chương trình ứng dụng: .....
   .....
   .....
   .....

4. Về thái độ làm việc của sinh viên: .....
   .....
   .....
   .....

Đánh giá chung: .....
.....
.....
.....

Điểm từng sinh viên:

- Đặng Lê Bảo Chương: ...../10
- Nguyễn Bảo Duy: ...../10

Tp.HCM, Ngày 19 tháng 1 năm 2015
Người nhận xét
(Ký tên và ghi rõ họ tên)

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh Phúc

Tp.HCM, Ngày 19 tháng 1 năm 2015

NHẬN XÉT KHÓA LUẬN TỐT NGHIỆP

(CỦA CÁN BỘ PHẢN BIỆN)

Nhóm sinh viên thực hiện : XÂY DỰNG ONTOLOGY PHỤC VỤ CHO VIỆC PHÂN LOẠI HÀNG HÓA TỰ ĐỘNG

Nhóm sinh viên thực hiện :		**Cán bộ phản biện :**
Đặng Lê Bảo Chương	MSSV:10520010	.....
Nguyễn Bảo Duy	MSSV:10520011	

Đánh giá khóa luận :

1. Về cuốn báo cáo:

Số trang	.....	Số chương	.....
Số bảng số liệu	.....	Số hình vẽ	.....
Số tài liệu tham khảo	.....	Sản phẩm	.....

   Một số nhận xét về hình thức cuốn báo cáo: .....
   .....
   .....
   .....

2. Về nội dung nghiên cứu: .....
   .....
   .....
   .....

3. Về chương trình ứng dụng: .....
   .....
   .....
   .....

4. Về thái độ làm việc của sinh viên: .....
   .....
   .....
   .....

Đánh giá chung: .....
.....
.....
.....

Điểm từng sinh viên:

- Đặng Lê Bảo Chương: ...../10
- Nguyễn Bảo Duy: ...../10

Tp.HCM, Ngày 19 tháng 1 năm 2015
Người nhận xét
(Ký tên và ghi rõ họ tên)

Lời cảm ơn

Đầu tiên, chúng em xin chân thành cám ơn Khoa Mạng máy tính và Truyền thông, trường Đại Học Công Nghệ Thông Tin, Đại Học Quốc Gia TP.HCM đã tạo điều kiện cho chúng em hoàn thành tốt khoá luận này.

Chúng em xin chân thành cám ơn Thầy Nguyễn Anh Tuấn, đã tận tình hướng dẫn, dạy dỗ, chỉ bảo chúng em từ những ngày đầu định hình khoá luận cho đến khi hoàn thành. Nhờ sự tận tình của thầy, chúng em đã hoàn thành tốt khoá luận này, bên cạnh đó cũng học hỏi được nhiều kiến thức quý báu từ thầy.

Chúng em xin chân thành cảm ơn quý Thầy Cô trong Khoa Mạng máy tính và truyền thông, trong những năm qua đã không quản ngại mệt mỏi, tận tình giảng dạy, trang bị cho chúng em những kiến thức cần thiết để hoàn thành khoá luận.

Chúng em xin ghi nhớ công ơn sinh thành dưỡng dục của cha mẹ, sự giúp đỡ của các anh, chị, bạn bè trong những năm học, cũng như sự an ủi, động viên trong những lúc khó khăn, vất vả. Dù chúng em đã dùng tất cả nỗ lực của bản thân để hoàn thành tốt khoá luận này, tuy nhiên không thể tránh khỏi những sơ suất, thiếu sót. Kính mong quý Thầy Cô tận tình chỉ bảo. Một lần nữa, chúng em xin chân thành cảm ơn và mong nhận được nhiều tình cảm chân thành của tất cả mọi người.

Tp.HCM, Ngày 19 tháng 1 năm 2015

Sinh viên thực hiện khóa luận

Đặng Lê Bảo Chương & Nguyễn Bảo Duy

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh Phúc

Tp.HCM, Ngày 19 tháng 1 năm 2015

ĐỀ CƯƠNG CHI TIẾT

1. **Tên đề tài:** Xây dựng Ontology phục vụ cho việc phân loai hàng hóa tự động
2. **Cán bộ hướng dẫn:** TS. NGUYỄN ANH TUẤN.
3. **Thời gian thực hiện:** Từ ngày 5/9/2014 đến ngày 19/1/2015
4. **Sinh viên thực hiện:**

Đặng Lê Bảo Chương	MSSV: 10520010
Nguyễn Bảo Duy	MSSV: 10520011

5. **Nội dung đề tài:**

   Xây dựng ứng dụng dùng để thiết kế và chỉnh sửa ontology với các tính năng gần như tương tự với chương trình Protege - tuy nhiên chúng em chọn xây dựng ứng dụng trên nền web, qua đó giúp người dùng chỉnh sửa và phát triển các tài liệu OWL 2 ontology từ khắp mọi nơi. Đồng thời ứng dụng này cũng gồm có cơ chế suy luận các phát biểu từ OWL 2 Ontology (dùng để thực hiện phân loại tự động), tính năng giải thích và hỗ trợ phân loại. Cuối cùng là thiết kế một ontology để trình bày tính năng phân loại tự động.

6. **Kế hoạch thực hiện:**

   - Tìm hiểu và nghiên cứu về Semantic Web, chi tiết ngôn ngữ OWL.
   - Tìm hiểu và nghiên cứu ngôn ngữ SWRL Rule, cách suy luận của OWL 2 và SWRL, từ đó xây dựng nên cách thức phân loại tự động.
   - Tìm hiểu và nghiên cứu tính nhất quán của ontology, các nguyên nhân gây mất tính nhất quán.

- Tìm hiểu về Vaadin Framework, sử dụng làm nền tảng để xây dựng ứng dụng OWL Editor dùng phục vụ cho việc phát triển ontology phân loại và thực hiện phân loại.
- Thiết kế một ontology đề trình bày khả năng phân loại tự động.
- Viết báo cáo về quá trình thực hiện từ tìm hiểu nghiên cứu về ngôn ngữ Ontology Web Language 2, Semantic Web Rule Language, thiết kế và xây dựng hệ thống

Tp.HCM, Ngày 19 tháng 1 năm 2015

Xác nhận của CBHD

(Ký tên và ghi rõ họ tên)

Sinh viên

(Ký tên và ghi rõ họ tên)

Mục lục

Danh sách hình vẽ

Danh sách bảng

Danh sách các từ viết tắt

CWA	**C**losed **W**orld **A**ssumption
DL	**D**escription **L**ogic
IRI	**I**nternationalized **R**esource **I**dentifier
OWL2	**O**ntology **W**eb **L**anguage **2**
OWA	**O**pen **W**orld **A**ssumption
RDF	**R**esource **D**escription **F**ramework
SWRL	**S**emantic **W**eb **R**ule **L**anguage
XML	e**X**tensible **M**arkup **L**anguage

Lời mở đầu

Thế giới của chúng ta đang liên tục vận động theo chiều hướng tích cực. Đây là nguyên nhân chủ yếu cho sự phát triển và thay đổi hằng ngày của mọi lĩnh vực đời sống, đặc biệt là khoa học công nghệ nói chung, và ngành công nghệ thông tin nói riêng. Hiện nay, hầu hết mọi nơi trên thế giới đều đã biết đến sự có mặt của công nghệ thông tin, máy tính, và kể cả internet. Việc internet ra đời là một sự kiện đã làm thay đổi cả thế giới. Thay thế cho việc gọi điện thoại hằng ngày, chúng ta có thể liên lạc qua internet, với việc có thể thấy được hình ảnh của người đối diện chứ không chỉ riêng giọng nói. Thay thế cho những tờ báo bằng giấy, chúng ta đã có những trang web, với những thông tin đầy đủ hơn, hình ảnh sinh động hơn, và cả những đoạn video minh hoạ. Những thông tin trên internet được lan truyền với tốc độ chóng mặt, dẫn đến việc những tin tức nóng nhất được cập nhật liên tục trong từng phút từng giây. Cách tiếp cận thông tin của con người thay đổi, nên sự chuyển động của thông tin ngày càng nhanh hơn, và đến mức nào đó, thông tin sẽ không mang theo đủ những gì mà con người mong muốn truyền tải. Đó là lúc mà con người nghĩ đến việc thay đổi. Và đó cũng chính là lúc Semantic Web được ra đời. Semantic Web mang sứ mệnh lớn lao trong việc thay đổi công nghệ web. Trước đây, máy tính chỉ đóng vai trò là trung tâm "chứa đựng" và "duy trì" các trang web. Tuy nhiên, với Semantic Web, máy tính sẽ phải làm nhiều hơn thế, sẽ phải "suy nghĩ" và "sử dụng" trang web một phần nào đó thay thế cho con người. Để làm được điều này không phải dễ dàng, vì máy tính là vật vô tri vô giác. Do đó, một cách tiếp cận đơn giản để giải quyết vấn đề này, bằng cách thêm vào các trang web thông thường metadata, để các máy tính có thể "đọc" được, như một ngôn ngữ chung của các máy tính.

Nhận thấy được những tiềm năng và những lợi ích to lớn của Semantic Web, chúng em đã lựa chọn đề tài này. Trước tiên, chúng em đã tập trung nghiên cứu và tìm hiểu sâu hơn về Semantic Web, cụ thể hơn là ngôn ngữ OWL2, và SWRL Rule. Ứng dụng của Semantic Web và Ontology rất đa dạng và phong phú. Mục đích chính của nhóm hướng đến việc chứng minh tính khả thi của việc sử dụng kết hợp ngôn ngữ OWL, SWRL Rule và hoạt động của reasoner trong việc phân loại tự động cá thể. Sau khi chứng minh được điều đó, chúng em quyết định xây dựng một công cụ chỉnh sửa ontology trên web, vì nhận thấy

chưa có một công cụ nào đáp ứng được nhu cầu đó. Sau đó chúng em sẽ tiếp tục phát triển hơn khả năng phân loại tự động đó trong từng lĩnh vực cụ thể, chẳng hạn như việc phân loại hàng hoá xuất nhập khẩu tại các cảng hải quan, hay việc phân loại các gói tin trong lưu lượng mạng...

Trong quá trình nghiên cứu và phát triển ứng dụng, tuy gặp phải những khái niệm và công nghệ hoàn toàn mới lạ và ít được công bố, chúng em vẫn cố gắng tìm hiểu bằng mọi cách. Sau hơn 6 tháng làm việc, kết quả đạt được của nhóm là một chương trình chỉnh sửa file Ontology trên nền tảng web, tích hợp khả năng phân loại tự động các cá thể. Hơn nữa, điều này cũng chứng minh tính khả thi trong việc áp dụng công nghệ ontology vào trong việc phân loại tự động các đối tượng thực tế.

Lĩnh vực Semantic Web là rất rộng lớn, với một khoảng thời gian có hạn, chúng em chỉ có thể tìm hiểu được những vấn đề được coi là cơ bản và tất yếu nhất của Semantic Web và Ontology. Dù vậy, chúng em rất hài lòng và tự tin với những gì tìm hiểu, nghiên cứu và phát triển được sẽ mang lại nhiều lợi ích, đóng góp vào công cuộc nghiên cứu khoa học chung.

Chương 1

Giới thiệu

1.1 Tên đề tài

Xây dựng Ontology phục vụ cho việc phân loai hàng hóa tự động

1.2 Nội dung và giới hạn đề tài

1.2.1 Nội dung đề tài

Từ *Ontology* được sử dụng trong ngành khoa học thông tin và ngành khoa học máy tính dùng để chỉ một mô hình hay hệ thống có khả năng biểu diễn hay thể hiện các đối tượng trong một lĩnh vực nào đó thông qua các định nghĩa về loại (phân lớp), thuộc tính và mối quan hệ giữa các đối tượng đó với nhau. Các mô hình ontologies thường được phát triển và sử dụng trong các lĩnh vực như trí tuệ nhân tạo, Web Ngữ nghĩa, phát triển phần mềm, công nghệ thông tin - sinh học, ... nhằm giảm sự phức tạp của hệ thống, tổ chức một khối lượng thông tin lớn và có nhiều ngữ nghĩa. Ngoài ra ontology còn có khả năng ứng dụng vào việc giải quyết vấn đề (problem solving).

Không giống với những ứng dụng phân loại đã có, trong nội dung khóa luận này chúng

em đã chọn ngôn ngữ Ontology Web Languge và Semantic Web Rule Languge là những công nghệ nền tảng xây dựng tính năng phân loại không chỉ hàng hóa mà bất cứ đối tượng nào có thể định nghĩa bằng các ngôn ngữ trên. Chi tiết hơn về các công nghệ trên sẽ nằm trong nội dung báo cáo này.

Trong quá trình nghiên cứu và tìm hiểu về các ngôn ngữ Ontology Web Language 2 (OWL 2), Semantic Web Rule Language (SWRL) vốn là nền tảng lý thuyết để xây dựng ứng dụng phân loại tự động, chúng em nhận thấy rằng để phát triển một mô hình Ontology dựa trên các loại ngôn ngữ nêu trên khá phức tạp, đòi hỏi phải xây dựng một trình biên tập các dữ liệu OWL 2, SWRL (chúng em sẽ trình bày rõ hơn trong nội dung của báo cáo). Trên thực tế, cũng đã có sẵn một số trình biên tập OWL 2 trên môi trường desktop như Protege, Swoop, hoặc trên môi trường web có WebProtege, tuy nhiên những trình biên tập này đã xuất hiện khá lâu, và một vài chương trình trong số đó đã ngừng phát triển hoặc không còn được cập nhật những thư viện mới nhất. Với những nguyên nhân kể trên, chúng em quyết định không sử dụng lại những trình biên tập đã cũ, mà sẽ tự xây dựng một ứng dụng biên tập và phát triển các OWL 2 Ontology, sau đó sử dụng các ontology này để hiện thực quá trình phân loại tự động.

Đề tài khóa luận đã làm những việc sau:

- Tìm hiểu về Semantic Web.
- Tìm hiểu về ngôn ngữ Web Ontology Language (OWL) và Semantic Web Rule Language(SWRL) - hai ngôn ngữ này là nền tảng lý thuyết cho .
- Tìm hiểu về tính nhất quán trong Ontology.
- Tìm hiểu về OWLAPI và SWRL API.
- Tìm hiểu về Vaadin Framework.
- Xây dựng thành công một trình chỉnh sửa OWL2 Ontology online với các tính năng biên tập ontology, suy luận reasoning, và hỗ trợ phân loại.

1.2.2 Giới hạn của đề tài

Lĩnh vực Semantic Web và Ontology là rất rộng lớn. Trong phạm vi của đề tài, chúng em chỉ tập trung nghiên cứu những cơ sở lý thuyết cơ bản nhất, và áp dụng để xây dựng một ứng dụng biên tập ontology trong môi trường web và trình bày tính năng phân loại của ngôn ngữ OWL2, SWRL Rule. Tính năng phân loại chỉ mang tính chất là một bản mẫu thực nghiệm, chứng minh các lý thuyết mà các công nghệ mới như OWL2 Ontology mang lại, chúng không có mục đích thay thế các hệ thống phân loại đang hoạt động và sử dụng rộng rãi.

1.3 Cấu trúc của khóa luận

Khóa luận được chia thành các chương với nội dung như sau :

- Chương 1 giới thiệu về đề tài.
- Chương 2 giới thiệu những cơ sở lý thuyết và các công nghệ được sử dụng trong đề tài.
- Phần thiết kế chương trình chỉnh sửa UIT-OWL Editor hỗ trợ phân loai tự động được mô tả chi tiết trong chương 3.
- Chương 4 trình bày cách mà chúng em hiện thực và xây dựng chương trình này.
- Chương 5 nói về kết luận và hướng phát triển của đề tài.

Chương 2

Cơ sở lý thuyết

Giới thiệu Chương này sẽ tập trung giới thiệu từ khái quát đến chi tiết về những đặc tính lý thuyết của Web Ngữ Nghĩa - Semantic Web, ngôn ngữ Ontology Web Language, ngôn ngữ Semantic Web Rule Language (SWRL) và tính nhất quán của ontology - Ontology Consistency. Đây là những nền tảng lý thuyết cơ bản nhất giúp chúng em xây dựng nên hệ thống phân loại tự động. Cuối chương, chúng em xin giới thiệu khái quát về nền tảng và các thư viện lập trình được chúng em sử dụng để xây dựng hệ thống.

2.1 Semantic Web

Như chúng ta đã biết hệ thống Web hiện tại mà chúng ta được sử dụng được gọi là Web 2.0 thì Semantic Web hay còn gọi là Web ngữ nghĩa được các nhà nghiên cứu từ tổ chức World Wide Web Consortium (W3C) kỳ vọng sẽ trở thành Web 3.0 trong tương lai gần. Semantic Web khuyến khích tích hợp đặc tính ngữ nghĩa vào các tài nguyên Web hiện có bằng việc đặt ra mục tiêu chuyển đổi hệ thống Web hiện tai - vốn được hình thành từ những dữ liệu không được tổ chức hoặc chỉ được tổ chức một phần - thành một tập hợp dữ liệu, tài nguyên thống nhất (A Web of Data). Để làm được việc đó, tổ chức W3C đó đề ra những quy ước chung cho các thành phần của Semantic Web. Mục tiêu cuối cùng mà Semantic Web như trong phát biểu của ngài Tim Berners-Lee - người phát minh ra

WWW: "Semantic Web cung cấp một bộ công cụ thống nhất (framework) để dữ liệu có thể được chia sẻ, tái sử dụng giữa các ứng dụng với nhau, giữa các doanh nghiệp, tổ chức hay trong cộng đồng người dùng web với nhau" [3]. Sau đây, chúng em xin giới nguyên lý và các thành phần của Semantic Web.

2.1.1 Semantic Web dựa trên giả định thế giới mở (Open World Assumption)

Với đặc điểm của thông tin trên Web là những thông tin luôn luôn cần được thêm vào hay chỉnh sửa (thông tin trên web cũng giống như kiến thức vì thế nó sẽ không bao giờ bị giới hạn) nên thay vì chọn tuân theo giả định thế giới đóng (Closed World Assumption - CWA), vốn đã tồn tại rất lâu trong các cơ sở dữ liệu quan hệ (SQL), các nhà nghiên cứu đã chọn giả định thế giới mở làm nguyên lý cho mọi lý thuyết và định nghĩa của Semantic Web. Một so sánh ngắn gọn giữa giả định Thế Giới Mở (Open World Assumption - OWA)[4] được Semantic Web chấp nhận và giả định Thế Giới Đóng (CWA).

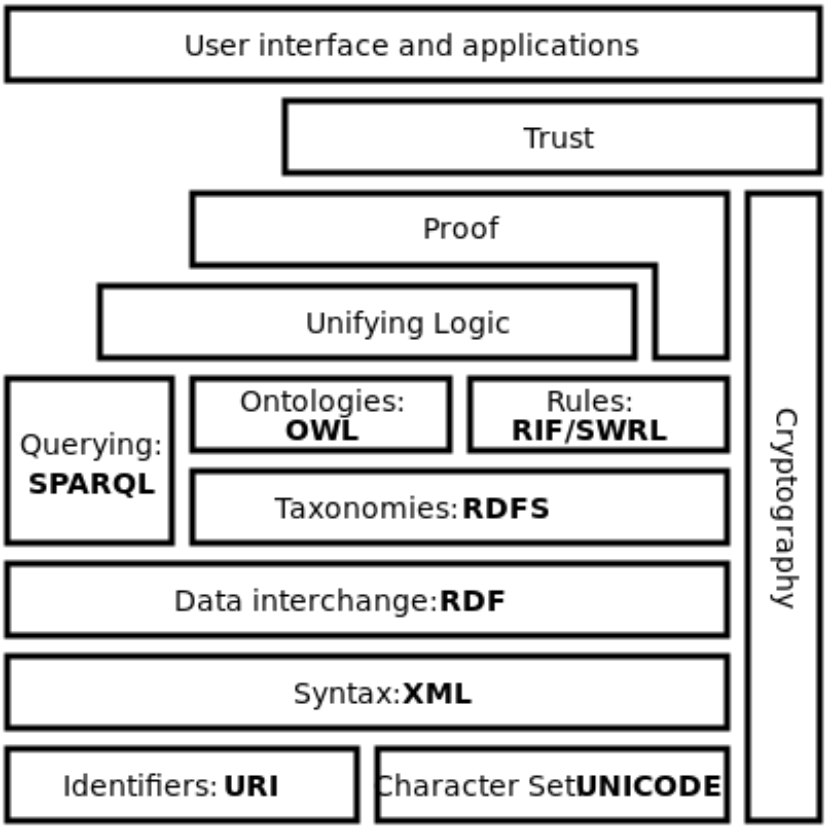
Closed World Assumption Giả định Thế Giới Đóng (CWA) là giả định mà những điều không chắc hoặc không có cơ sở để chứng minh là **đúng** sẽ được chấp nhận là **sai**.

Open World Assumption Giả định Thế Giới Mở (OWA) thì ngược lại, với những điều không chắc hoặc không có cơ sở để chứng minh là **đúng** sẽ được chấp nhận là **chưa biết**.

Ví dụ Xem xét một câu nói sau đây: "A là một công dân của nước Hoa Kỳ". Nếu có ai đó hỏi "A có phải là một công dân của Việt Nam hay không ?". Xét theo CWA, câu trả lời là *không*, ngược lại với OWA thì câu trả lời là *chưa biết*.

2.1.2 Các thành phần của Semantic Web

Khái niệm "Semantic Web" thường được sử dụng cụ thể hơn nhằm chỉ đến những định dạng và công nghệ để hiện thực hóa nó. Việc tổ chức, tập hợp và phục hồi dữ liệu liên kết thực hiện được nhờ vào các công nghệ đặc tả chính thức về các khái niệm, định nghĩa và mối quan hệ trong một lĩnh vực tri thức được biết đến. Tất cả các công nghệ này đều được quy định thành một tiêu chuẩn của tổ chức World Wide Web Consortium (W3C) . Các tiêu chuẩn được liệt kê trong [5] và dựa vào các thành phần đó các nhà nghiên cứu đã đưa ra các thành phần của Semantic Web như sau



Hình 2.1: The Semantic Web Stack

Hình Semantic Web Stack [3] miêu tả kiến trúc của Semantic Web:

1. **XML** là định dạng tài liệu chính để lưu trữ các mô hình dữ liệu, thông tin mà Semantic Web diễn đạt. Ngoài XML cũng tồn tại các định dạng thay thế khác như Turle *.

2. **XML Schema** quy định tiêu chuẩn cho các tài liệu Semantic Web, có thể ví nó như phần mở rộng trong tên tập tin, giúp phân biệt các tiêu chuẩn, các loại tài liệu Semantic khác nhau như OWL/XML, RDF/XML.

3. **Resource Description Framework** (RDF) [6] là ngôn ngữ đầu tiên được quy định bởi W3C để diễn tả các mô hình dữ liệu và mối quan hệ của chúng. RDF có thể được lưu trữ dưới nhiều dạng định dạng khác nhau ví dụ như: RDF/XML, N3, Turtle và RDFa. Có thể nói RDF chính là thành phần cơ bản và quan trọng nhất của Semantic Web.

4. **RDF Schema** (RDFS) [7] mở rộng RDF bằng những vốn từ vựng mới dùng miêu tả các thuộc tính và phân lớp trong các tài nguyên web hay mô hình dữ liệu được biểu diễn bằng RDF.

5. **Ontology Web Language** (OWL) là ngôn ngữ với cú pháp hoàn toàn mới dùng với mục tiêu tương tự RDFS, tuy nhiên ngoài việc biểu diễn được lớp, thuộc tính như RDFS, OWL còn có khả năng các mối quan hệ giữa các đối tượng trong mô hình dữ liệu chi tiết hơn nhiều. Ví dụ: quy định các lớp không được phép có chung cá thể nào (disjointness), các phát biểu ràng buộc số lượng (cardinality), cung cấp nhiều loại dữ liệu cho các thuộc tính, và mô tả được thuộc tính bằng các tính chất (đối xứng/ bất đối xứng, và các lớp liệt kê, ...).

6. SPARQL là một giao thức và ngôn ngữ truy vấn dữ liệu dành cho tài nguyên của Semantic Web (RDF, RDFS, OWL).

7. RIF (W3C Rule Interchange Format) là một ngôn ngữ XML để biểu diễn điều luật web mà máy tính có thể thực thi.

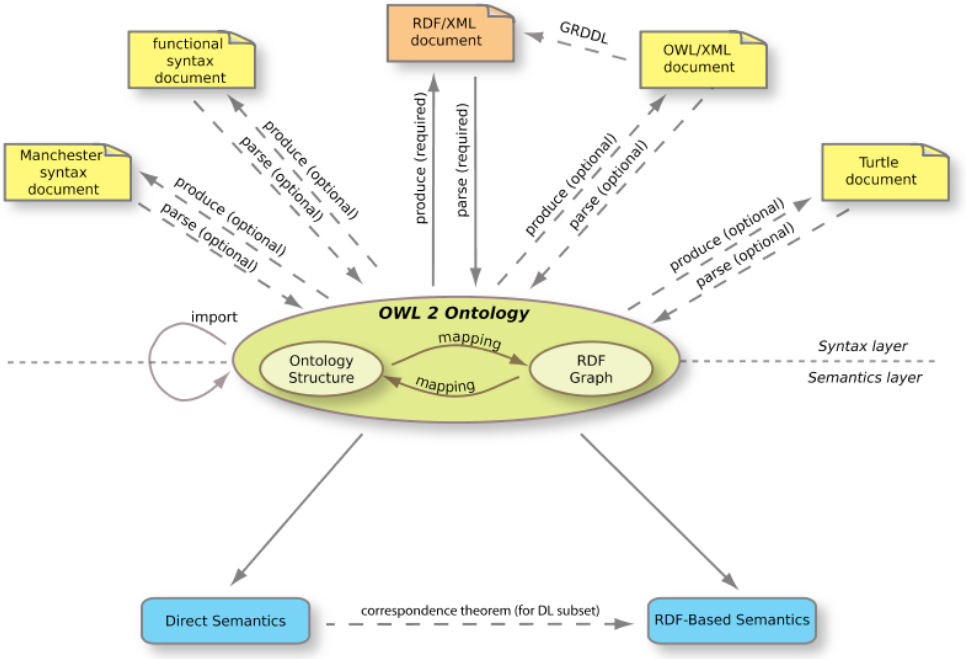
---

* *Turtle: http://en.wikipedia.org/wiki/Turtle_ (syntax)*

2.2 Ontology Web Language 2

Là một thành phần chính của Semantic Web, nhiệm vụ của Ontology Web Language (OWL) chính là một Semantic Web. OWL được tổ chức W3C khuyến khích sử dụng vì những thành phần từ vựng mới của nó giúp đặc tả các thực thể trong một lĩnh vực nào đó hiệu quả hơn so với RDFS hay RDF. Phiên bản OWL hiện thời là phiên bản 2 [8]. Về mặt lý thuyết, OWL là một ngôn ngữ ontology tuân theo Description Logic (DL) $SROIQ_{(D)}$ [9], với ưu điểm là ngoài khả năng đặc tả vừa nêu, nó còn biểu diễn được những suy luận được suy ra từ những đặc tả được khai báo. Khả năng suy luận của OWL chính là điểm quyết định cho tính khả thi của hệ thống phân loại tự động.

2.2.1 Đặc điểm tổng quan



Hình 2.2: Cấu trúc của OWL 2

Hình trên cho chúng ta cái nhìn tổng quan về các định dạng tài liệu, các loại cú pháp và các khả năng chuyển đổi thành RDF Graph của Ontology. Trong hình, hình ê-líp ở giữa thể hiện các khái niệm mà một ontology muốn thể hiện, các thông tin này có khả năng

chuyển đổi qua lại thành định dạng đồ thị RDF [10] (RDF Graph - là định dạng chính trong các cơ sở dữ liệu đồ thị ngữ nghĩa). Ontology có thể được biểu diễn dưới nhiều dạng cú pháp và lưu trữ dưới nhiều dạng tài liệu khác nhau (Syntax Layer trong hình), các định dạng và cú pháp này hoàn toàn có thể chuyển đổi qua lại với nhau. Lớp ngữ nghĩa trong hình (Semantic Layer) cho thấy ngữ nghĩa được quy định theo 2 tiêu chuẩn kỹ thuật khác nhau là Direct Semantics và RDF-Based Semantics.

2.2.1.1 Cú pháp và định dạng

Cú pháp cần thiết cho việc biểu diễn hay xây dựng ontology, định dạng cần cho việc lưu trữ và trao đổi giữa các ứng dụng với nhau.Trong OWL, mỗi định dạng tài liệu sẽ gắn liền với một loại tài liệu. Định dạng/cú pháp chính của OWL là RDF/XML [RDF Syntax] [11]. Ngoài ra, còn có các định dạng/cú pháp khác cũng nằm trong quy định của tổ chức W3C.

2.2.2 Một số ví dụ của các định dạng/cú pháp khác nhau

Functional Syntax

```
Declaration(Class (Grass))) # Khai báo lớp
Declaration(ObjectProperty (canEat))  # Khái báo thuộc tính
SubClassOf(Cow Animal)  # Khai báo lớp con
```

RDF/XML Syntax

```
T(Animal) rdf:type owl:Class
T(canEat) rdf:type owl:ObjectProperty
T(Cow) rdfs:subClassOf T(Animal)
```

OWL/XML Syntax

```
<Declaration>
<Class IRI="#Animal"/> // Khai báo lớp
</Declaration>
<Declaration>
<ObjectProperty IRI="#canEat"/> // Khai báo thuộc tính
</Declaration>
<SubClassOf>
<Class IRI="#Cow"/>
<Class IRI="#Animal"/>  // Khai báo lớp con
</SubClassOf>
```

Manchester Syntax

```
Class: Cow  # Khái báo lớp chung với lớp con
SubClassOf: Animal
SubClassOf: canEat some Grass
```

2.2.3 Các thành phần chi tiết của một OWL 2 ontology

2.2.3.1 Ontology IRI và Version IRI

Mỗi ontology đều có thể gồm *một ontology IRI* [12] * (Internationalized Resource Identifier), dùng để định danh cho ontology. Nếu một ontology có một ontology IRI, thì ontology này có thể có thêm một version IRI, dùng để xác định phiên bản cho ontology này. Version IRI có thể trùng hoặc không nhất thiết phải trùng với ontology IRI. Một ontology không có ontology IRI thì không có version IRI.

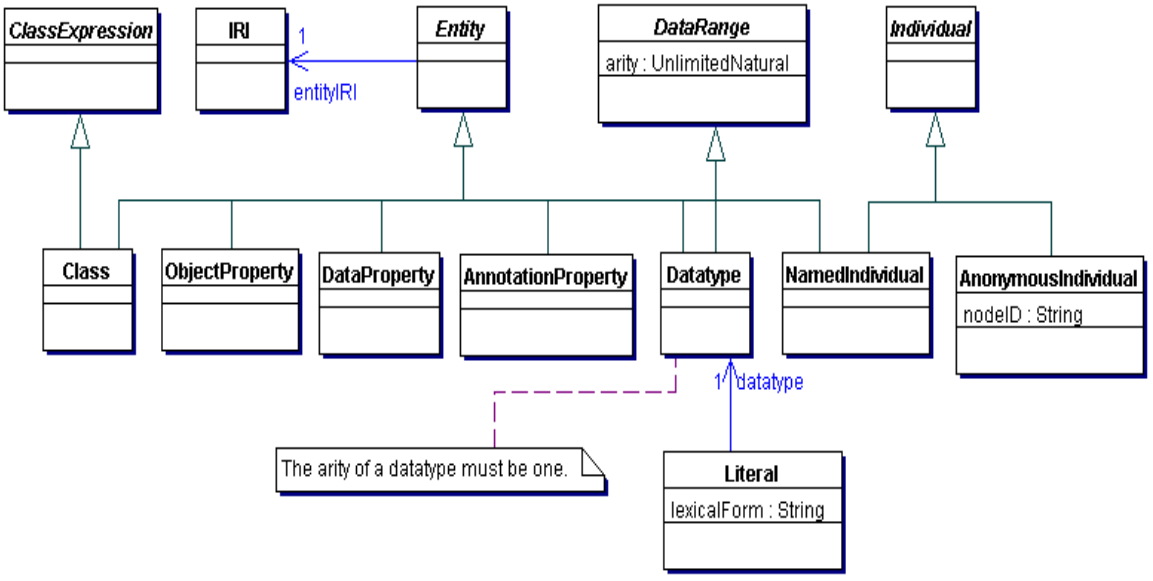
Ontology IRI và các version IRI kết hợp với nhau giúp định danh một phiên bản cụ thể của của ontology từ một bộ chứa tất cả các phiên bản của một ontology cụ thể nào đó được định danh chung bằng ontology IRI. Trong mỗi bộ ontology như vậy, sẽ có chính

* *Internationalized Resource Identifier: giao thức bổ sung cho Uniform Resource Universal Character Set (Unicode/ISO 10646).*

xác một ontology được xác định như một ontology hiện hành - khi dùng ontology IRI để truy vấn ontology mà không đề cập đến version IRI, mặc định ontology có verison IRI hiện hành sẽ được trả về.

2.2.3.2 Thực thể, trực nghĩa và cá thể ẩn danh - Entities, Literals and Anonymous Individuals

Các thực thể (entities) là thành phần cơ bản nhất của OWL2 Ontology, chúng định nghĩa các từ vựng - cụ thể là những đặt tên ra các khái niệm (named term) - của một ontology. Bên cạnh các thực thể, OWL 2 ontologies thường có thêm các trực nghĩa (literals), như strings hay integers.



Hình 2.3: Entities, Literals, Anonymous Individuals trong OWL2

Cấu trúc của các thực thể và trực nghĩa trong OWL 2 được thể hiện trong hình bên. Các lớp (classes), kiểu dữ liệu (datatypes), thuộc tính đối tượng (object properties), thuộc tính dữ liệu (data properties), thuộc tính chú thích và các cá thể có tên đều được gọi chung là các thực thể (entities), tất cả chúng được được định danh bằng một IRI duy nhất.

- Lớp (class) đại diện cho một tập gồm nhiều cá thể(individuals).

- Kiểu dữ liệu (datatype) là một tập của các trực nghĩa như strings hoặc integers.
- Thuộc tính đối tượng và dữ liệu (object & data property) được sử dụng để biểu diễn các mối quan hệ giữa các cá thể với cá thể khác và giữa cá thể với trực nghĩa (literal) trong một lĩnh vực (domain) nào đó.
- Thuộc tính chú thích (annotation) được dùng để đưa thêm những thông tin không có tính ngữ nghĩa (non-logical) như các chú thích, giải nghĩa, ngôn ngữ gắn với ontologies, các phát biểu/tiên đề (axioms) và các thực thể.
- Các cá thể có tên có thể được dùng để biểu diễn một đối tượng cụ thể từ một lớp nào đó.

Bên cạnh các cá thể có tên, OWL2 còn cung cấp một khái niệm gọi là các cá thể ẩn danh (anonymous individuals) - là cá thể tương tự với các node trống (blank nodes) trong RDF Concept [13] và được truy xuất ngay bên trong ontology mà chúng được sử dụng. Cuối cùng, OWL 2 cung cấp thêm cho cac trực nghĩa (literals), một dạng dữ liệu string gọi là định dạng nghĩa bằng từ ngữ (lexical form) và một dạng dữ liệu để chỉ dẫn cách ontology có thể hiểu chuỗi này.

Lớp (Classes) Lớp được hiểu như tập hợp các cá thể. Hai lớp với IRIs *owl:Nothing* và *owl:Thing* là các lớp được định nghĩa sẵn trong OWL2 với ý nghĩa như sau:

- **owl:Thing** là tập hợp gồm tất cả các cá thể.
- **owl:Nothing** là tập hợp rỗng.

Không nên sử dụng 2 định nghĩa trên để gán cho bất kì lớp nào trong OWL 2 DL Ontology. Ví dụ:

```
SubClassOf( a:Child a:Person)
```

Giải thích: Mỗi đứa trẻ đều là một người.

Kiểu dữ liệu (datatypes) Kiểu dữ liệu là thực thể được xem như tập hợp của các giá trị dữ liệu. Như vậy, kiểu dữ liệu cũng tương tự lớp, khác biệt chính là thay vì chứa các cá thể (individuals) như lớp thì lại chứa các giá trị dữ liệu như strings, numbers,... Kiểu dữ liệu có thể được dùng tạo ra các dữ liệu giới hạn (datarange). Ví dụ kiểu dữ liệu *xsd:positiveInteger* đại diện cho tập hợp gồm tất cả các số nguyên dương. Nó được sử dụng để quy định kiểu dữ liệu mà thuộc tính *hasAge* có thể chấp nhận:

```
DataPropertyRange( a:hasAge xsd:positiveInteger)
```

Giải thích: thuộc tính dữ liệu a:hasAge chỉ được phép là các số nguyên dương.

Thuộc tính đối tượng (object properties) Thuộc tính đối tượng kết nối các cặp cá thể - tạo ra mối liên hệ (relationship) giữa các cá thể. Tương tự như lớp cũng có 2 thuộc tính đối tượng được định nghĩa sẵn trong OWL 2 với ý nghĩa như sau:

- **owl:topObjectProperty** kết nối tất cả các cặp cá thể có thể kết nối.
- **owl:bottomObjectProperty** không kết nối bất kì cặp cá thể nào.

Không nên sử dụng 2 định nghĩa trên để gán cho bất kỳ thuộc tính đối tượng nào trong OWL 2 DL Ontology. Ví dụ:

```
ObjectPropertyAssertion( a:parentOf a:Peter a:Chris)
```

Giải thích: Peter là ba mẹ của Chris. Thuộc tính đối tượng *a:parentOf* được dùng để nói lên mối quan hệ giữa các cá thể trong ví dụ trên.

Thuộc tính dữ liệu (data properties) Thuộc tính dữ liệu liên kết các cá thể với các trực nghĩa. Trong một vài hệ thống biểu diễn tri thức, thuộc tính dữ liệu chức năng được gọi là thuộc tính. Hai định nghĩa sẵn *owl:topDataProperty* và *owl:bottomDataProperty* có ý nghĩa như sau :

- **owl:topDataProperty** liên kết tất cả cá thể với tất cả các trực nghĩa.
- **owl:bottomDataProperty** không liên kết bất kì cá thể với trực nghĩa nào.

Tương tự lớp và thuộc tính đối tượng, 2 phát biểu *top* và *bottom* trên cũng không nên được sử dụng để gán cho bất kì thuộc tính dữ liệu nào. Mỗi thuộc tính dữ liệu *a:hasName* chứa tên đầy đủ của mỗi người. Ví dụ nó có thể được sử dụng như trong phát biểu sau:

```
DataPropertyAssertion( a:hasName a:Steve "Steve Job")
```

Giải thích: Tên của Steve là "Steve Job".

Cá thể (Individuals) Cá thể trong OWL2 là một đối tượng cụ thể thuộc một tập/lớp (domain/class). Có 2 dạng cá thể trong cú pháp OWL2. *Cá thể có tên* được khai báo tên một cách rõ ràng để có thể sử dụng trong bất kì ontology nào bằng cách truy vấn tới IRI có chứa tên của nó. Ngược lại, cá thể ẩn danh (Anonymous Individuals) không có tên gọi toàn cục và chỉ truy vấn được trong nội bộ của ontology chứa chúng.

Cá thể có tên (Named Individuals) được định danh bằng một IRI. Vì vậy, nên cá thể có tên cũng là một thực thể (entity) tương tự lớp, thuộc tính và kiểu dữ liệu. Ví dụ khai báo một các thể thuộc 1 lớp:

```
ClassAssertion( a:Person a:Peter)
```

Giải thích: Peter là một người.

Cá thể ẩn danh (Anonymous Individuals) Nếu cần một cá thể chỉ sử dụng ở nội bộ ontology, chúng ta có thể sử dụng cá thể ẩn danh, được định danh bằng một node ID cục bộ thay vì sử dụng IRI toàn cục. Cá thể ẩn danh tương tự như một node rỗng trong đồ thị RDF [13]. Ví dụ khai báo thuộc tính đối tượng giữa cá thể ẩn danh với cá thể có tên:

```
ObjectPropertyAssertion( a:liveAt a:Peter _:a1)
ObjectPropertyAssertion( a:city _:a1 a:HCM)
ObjectPropertyAssertion( a:district _:a1 a:ThuDuc)
```

Giải thích: Peter sống ở một địa chỉ nào đó (chưa biết). Mà địa chỉ chưa biết này nằm trong thành phố Hồ Chí Minh và nằm trong quận Thủ Đức.

Trực nghĩa (Literals) Trực nghĩa biểu diễn các giá trị dữ liệu như chuỗi và số nguyên. Mỗi trực nghĩa gồm một chuỗi định dạng được định nghĩa bởi người dùng (lexical form), và một kiểu dữ liệu được hỗ trợ bởi OWL2 [14]. Một trực nghĩa gồm một chuỗi định dạng *"abc"* và một kiểu dữ liệu (Datatype) định danh bởi IRI *datatypeIRI* được định nghĩa như sau `"abc"^^datatypeIRI`. Thêm nữa, các trực nghĩa mà kiểu dữ liệu của chúng là *rdf:PlainLiteral* có thể được viết tắt trong functional-syntax của OWL2 ghi lại trong tài liệu thành dạng trực nghĩa rỗng RDF [13]. Cú pháp viết tắt này chỉ đơn giản là một định dạng ngắn gọn hơn, chúng không có ảnh hưởng đến ý nghĩa của khai báo trực nghĩa. Viết tắt chủ yếu là để phục vụ cho việc parsing:

- Trực nghĩa dạng `"abc@"^^rdf:PlainLiteral` được viết tắt thành `"abc"`.
- Trực nghĩa dạng `"abc@langTag"^^rdf:PlainLiteral` trong đó *"langTag"* không rỗng được viết tắt thành `"abc"@langTag`.

Một số ví dụ:

```
"1"^^xsd:integer  // trực nghĩa biểu diễn số nguyên dương 1
"abc"^^xsd:string // trực nghĩa biểu diễn chuỗi "abc"
```

Khai báo thực thể Mỗi IRI I được sử dụng trong OWL 2 ontology O cần được khai báo để sử dụng. Phát biểu khai báo một thực thể I nhằm đảm bảo rằng O phải chứa I. Hai mục tiêu của phát biểu này:

- Khẳng định sự tồn tại trong I trong O
- Khai báo gắn với loại của thực thể I - phân loại xem I có phải là một lớp, một kiểu dữ liệu, một thuộc tính đối tượng, một thuộc tính dữ liệu, một đặt tính chú thích hay một cá thể.

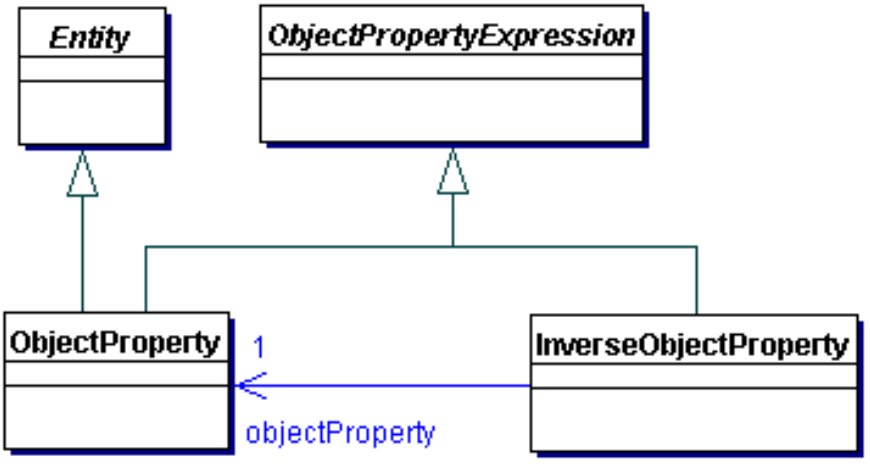
Bối cảnh sử dụng khái báo **Declaration** thường gắn liền với chức năng *Add New Class/Property/Datatype* trong một Ontology Editor nào đó. Ví dụ:

```
Declaration( Class( a:Person ) )
Declaration( NamedIndividual( a:Peter ) )
```

2.2.3.3 Mô tả thuộc tính (Property Expression)

Các thuộc tính được sử dụng trong OWL 2 để tạo ra các mô tả thuộc tính.

Mô tả thuộc tính đối tượng (Object Property Expression) Thuộc tính đối tượng được sử dụng trong OWL 2 để tạo thành các mô tả thuộc tính đối tượng (object property expression), diễn tả các mối quan hệ giữa các cặp cá thể. Chúng được diễn giải trong tài liệu cấu trúc chi tiết của OWL 2 như trong hình sau. Như được thể hiện trong



Hình 2.4: Mô tả thuộc tính đối tượng OWL 2

hình, OWL 2 hỗ trợ 2 loại mô tả thuộc tính đối tượng. Thuộc tính đối tượng là dạng đơn giản của mô tả thuộc tính đối tượng, thuộc tính đối tượng nghịch đảo (inverse object properties) cho phép thể hiện các mối quan hệ 2 chiều giữa biểu hiện các mô tả lớp (class expression) và các phát biểu (axiom).

Mô tả thuộc tính đối tượng nghịch đảo

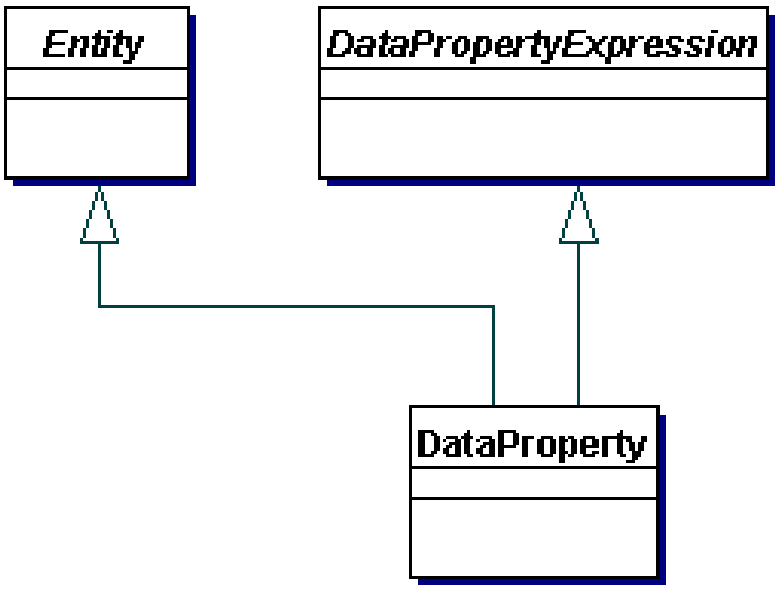
```
ObjectPropertyAssertion( a:fatherOf a:Peter a:Steve)
// Peter là ba của Steve
```

Với phát biểu trên, ontology sẽ hiểu *a:Steve* liên kết với *a:Peter* qua một thuộc tính nghịch đảo của *a:fatherOf* là *ObjectInverseOf(a:fatherOf)*. Chúng ta cũng có thể khai báo tường minh nghịch đảo của *a:fatherOf* là *a:childOf* bằng phát biểu *InverseObjectProperties(a:fatherO a:childOf)*.

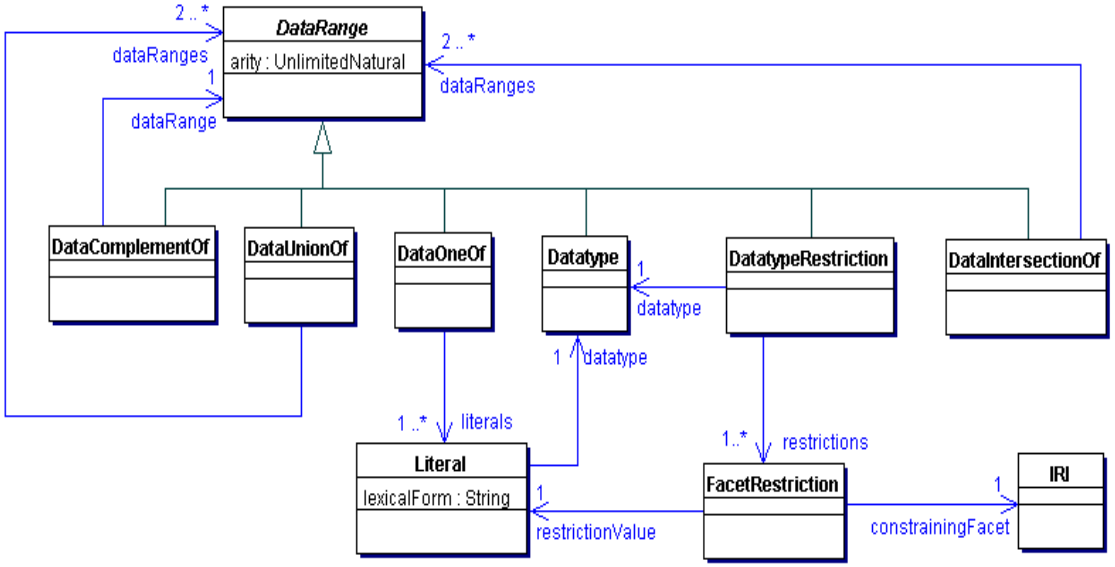
Mô tả thuộc tính dữ liệu (Data Property Expression) Tương đương với biểu hiện thuộc tính đối tượng, trong tài liệu cấu trúc chi tiết của OWL2 cũng giới thiệu định nghĩa mô tả thuộc tính dữ liệu (data property expressions), nhằm biểu diễn mối quan hệ giữa một cá thể và một trực nghĩa. Cấu trúc của mô tả thuộc tính dữ liệu thể hiện trong hình bên. Như chúng ta thấy thuộc tính dữ liệu (data property) cũng chính là 1 mô tả thuộc tính dữ liệu (data property expression), cấu trúc như vậy được xây dựng nhằm tạo thuận lợi cho nhu cầu mở rộng sau này nếu có.

2.2.3.4 Miền dữ liệu (Data Range)

Kiểu dữ liệu như*xsd:string* hoặc *xsd:integer* và trực nghĩa `"1"^^xsd:integer` được dùng để biểu diễn miền dữ liệu - tập hợp danh sách có thứ tự (tuples) của các trực nghĩa, mà mỗi phần tử trong danh sách này chỉ chứa một trực nghĩa duy nhất để định nghĩa chính nó. Miền dữ liệu được dùng để tạo ra ràng buộc (hay những dữ liệu hợp lệ) cho thuộc tính dữ liệu. Cấu trúc của miền dữ liệu trong OWL 2 được mô tả trong hình. Thành phần đơn giản của miền dữ liệu chính là kiểu dữ liệu : một kiểu dữ liệu (Datatype) cũng chính là một miền dữ liệu (Data Range).



Hình 2.5: Cấu trúc của mô tả thuộc tính dữ liệu OWL 2



Hình 2.6: Cấu trúc miền dữ liệu (data range) quy định theo OWL 2

Giao của các miền dữ liệu (Intersection of Data Ranges) Giao của miền dữ liệu *DataIntersectionOf(DR_1...DR_n)* chứa tất cả các danh sách của các trực nghĩa nằng trong DR_i với $1 <= i <= n$. Tất cả miền dữ liệu DR_i phải có cùng số lượng tham số, và cũng phải có cùng số lượng số kết quả trả về. Ví dụ về miền dữ liệu chỉ chứa số 0:

```
DataIntersectionOf( xsd:nonNegativeInteger xsd:nonPositiveInteger )
```

Hội của các miền dữ liệu (Union of Data Ranges) Điều kiện về số lượng tham số và số lượng kết quả trả về của từng DR_i với điều kiện $1 <= i <= n$ phải giống nhau. Cú pháp *DataIntersectionOf(DR_1...DR_n)*. Ví dụ về việc miền dữ liệu chứa tất cả các chuỗi và số nguyên:

```
DataUnionOf( xsd:string xsd:integer )
```

Phủ định của miền dữ liệu (Complement of Data Ranges) Cú pháp *DataComplementOf(DR)* chứa tất cả các miền dữ liệu còn lại trong miền dữ liệu *DR*. Yêu câu số lượng tham số và kết quả phải bằng *DR*

```
DataComplementOf( xsd:positiveInteger )
```

Giải thích: miền dữ liệu trên chứa tất cả số nguyên âm, số 0; và chứa tất cả chuỗi (strings) vì chuỗi không phải số nguyên dương.

Liệt kê trực nghĩa (Enumeration Of Literals) Một danh sách liệt kê các trực nghĩa (literals) với cú pháp *DataOneOf(lt_1...lt_n)*, lt_i với $1 <= i <= n$. Miễn dữ liệu chỉ áp dụng lên một trực nghĩa nằm trong danh sách (*"oneOf"*). Ví dụ khai báo miền dữ liệu cho thuộc tính dữ liệu *canMoveOnOrIn* chỉ có thể là một trong 4 giá trị chuỗi *"OnRoadOrOffRoad"*, *"Rail"*, *"Sky"* và *"Water"*.

```
DataPropertyRange(:canMoveOnOrIn
DataOneOf("OnRoadOrOffRoad" "Rail" "Sky" "Water"))
```

Ràng buộc cho kiểu dữ liệu (Datatype Restrictions) Một ràng buộc cho kiểu dữ liệu (hay một tập các giá trị hợp lệ của kiểu dữ liệu) *DatatypeRestriction(DT F_1 lt_1 ... F_n lt_n)* gồm một kiểu dữ liệu đơn (unary datatype) *DT* và n cặp (F_i, lt_i). Vùng dữ

liệu hợp lệ được tính ra bằng cách hạn chế vùng giá trị của DT và lấy giao tất cả các cặp (F_i, v_i) trong đó v_i là giá trị dữ liệu của trực nghĩa lt_i. Ví dụ miền dữ liệu sau chỉ gồm đúng các số 5,6,7,8,9:

```
DatatypeRestriction( xsd:integer
xsd:minInclusive "5"^^xsd:integer xsd:maxExclusive "10"^^xsd:integer )
```

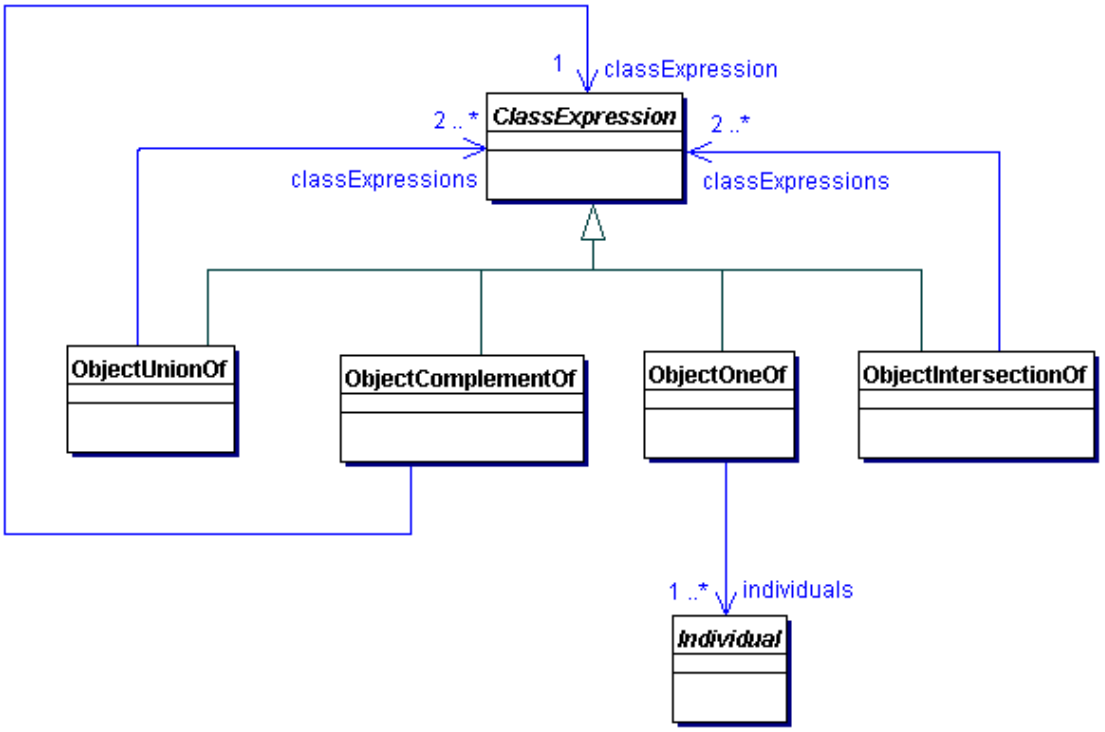
2.2.3.5 Mô tả lớp (Class Expression)

Trong OWL2, các lớp và mô tả thuộc tính (property expressions) được sử dụng để xây dựng nên các mô tả lớp, hay chúng được gọi là các miêu tả hoặc trong thuật ngữ của Description Logic chúng được gọi là *những khái niệm phức tạp*. Các mô tả lớp biểu diễn tập hợp các cá thể bằng cách chính thức đặt ra các điều kiện quy định cho các thuộc tính của các cá thể này; cá thể nào có những thuộc tính phù hợp hay thỏa những quy định đó được xem là một thành viên của mô tả lớp đang xét. Có 5 dạng mô tả lớp chính. Sau đây chúng em xin được trình bày 3 dạng mô tả lớp.

Các mệnh đề logic và liệt kê cá thể OWL2 cho phép liệt kê các cá thể của một lớp, khai báo các mệnh đề logic như trong hình sau. Các khai báo *ObjectIntersectionOf*, *ObjectUnionOf*, và *ObjectComplementOf* cho phép thực hiện các phép toán luân lý (and ,or, not) trên những mô tả lớp. *ObjectOneOf* quy định chính xác những cá thể cho mô tả lớp đang khai báo.

```
ClassAssertion(a:Engineer a:Peter)  // Peter là kĩ sư.
ClassAssertion(a:Teacher a:Peter) // Peter là giáo viên.
```

Giải thích: với khai báo như trên, một hàm ý sẽ được suy ra mà không cần khai báo đó là *Peter vừa là giáo viên vừa là kĩ sư* hay tương đương với phát biểu *ObjectIntersectionOf(a:Engineer a:Teacher)*.



Hình 2.7: Mô tả lớp trong OWL 2

Union of Class Expressions Tập hội của lớp *ObjectUnionOf(CE_1 ... CE_n)* chức tất cả các cá thể là thành viên của của ít nhất một mô tả lớp CE_i với $1 <= i <= n$. Ví dụ:

```
ClassAssertion( a:Man a:Peter )  // Peter là nam
ClassAssertion( a:Woman a:Lois ) // Lois là nữ
```

Giải thích: với khai báo như trên, một hàm ý sẽ được suy ra mà không cần khai báo đó là *cả Peter và Lois đều thuộc lớp nam hoặc nữ* hay tương đương với phát biểu O*bjectUnionOf(a:Man a:Woman)*.

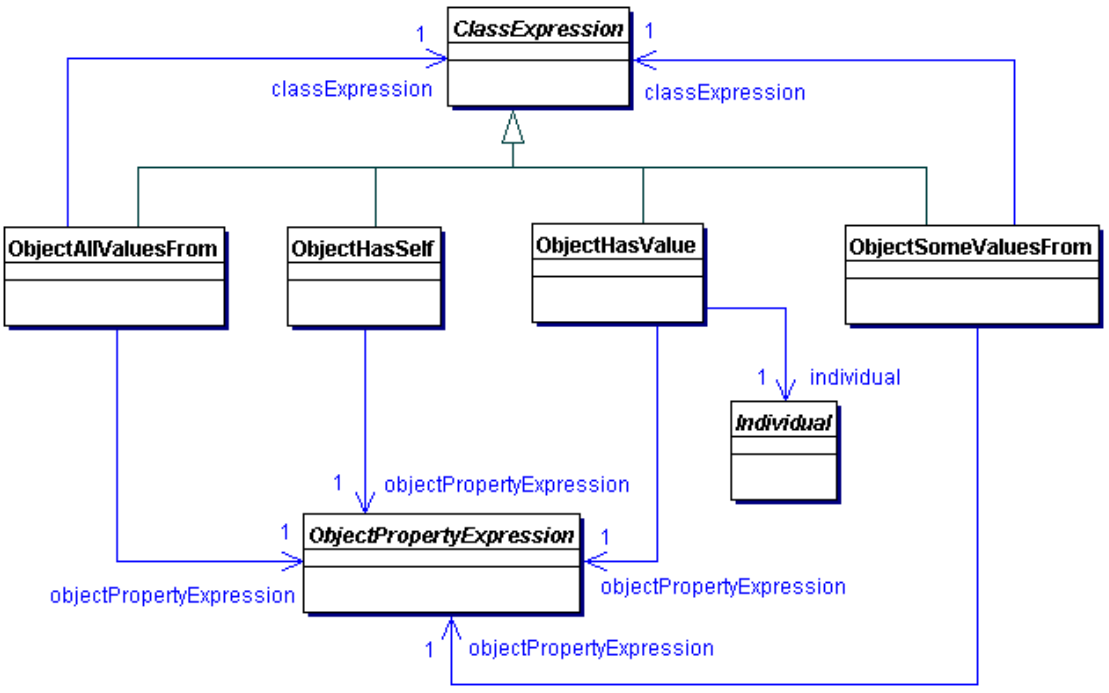
Complement of Class Expressions Phủ định của một mô tả lớp chứa tất cả các phần tử không thuộc mô tả lớp đó. Cú pháp *ObjectComplementOf(CE)*. Ví dụ:

```
DisjointClasses( a:Man a:Woman)
```

```
// Không có cá thể nào vừa làm nam vừa là nữ
ClassAssertion( a:Woman a:Lois)
// Lois là nữ
```

Giải thích: Vì *Lois* là nữa và không có cá thể nào vừa là nam vừa là nữ nên *Lois* thuộc lớp *không phải nam* tương đương với phát biểu *ObjectComplementOf(a:Man)*.

Enumeration of Individuals Liệt kê các cá thể *ObjectOneOf(* a_1 ... a_n *)* chứa duy nhất một cá thể a_i với $1 <= i <= n$.



Hình 2.8: Object Property Restrictions trong OWL 2

Ràng buộc theo thuộc tính đối tượng (Object Property Restrictions) Các mô tả lớp trong OWL 2 có thể được tạo ra bằng cách kết hợp chúng với các thuộc tính đối tượng bằng phát biểu như trong hình.

Existential Quantification Một mô tả lớp *ObjectSomeValueFrom(OPE CE)* thể hiện mối quan hệ giữa những cá thể được kết nối bởi mô tả thuộc tính đối tượng *OPE* đến **ít nhất một** cá thể thuộc mô tả lớp *CE*. Ví dụ:

```
ObjectPropertyAssertion( a:fatherOf a:Peter a:Steve )
// Peter là ba của Steve
ClassAssertion( a:Kid a:Steve )
// Steve là trẻ con.
```

Giải thích: với khai báo như trên, một hàm ý sẽ được suy ra mà không cần khai báo đó là *Peter là ba của một vài (ít nhất một) đứa trẻ* hay tương đương với phát biểu *ObjectSomeValuesFrom(a:fatherOf a:Kid)*.

Universal Quantification Mô tả lớp *ObjectAllValuesFrom(OPE CE)* thể hiện mối quan hệ giữa những cá thể được kết nối bởi mô tả thuộc tính đối tượng OPE đến **tất cả** các cá thể thuộc mô tả lớp CE. Ví dụ:

```
ObjectPropertyAssertion( a:hasPet a:Peter a:Tom)
// Tom là vật nuôi của Peter
ClassAssertion( a:Cat a:Tom)
// Tom là một con mèo
ClassAssertion( ObjectMaxCardinality( 1 a:hasPet ) a:Peter )
// Peter chỉ được phép có một vật nuôi
```

Giải thích: với khai báo như trên, một hàm ý sẽ được suy ra mà không cần khai báo đó là *Peter thuộc mô tả lớp "có tất cả vật nuôi là mèo"* hay tương đương với phát biểu *ObjectAllValuesFrom(a:hasPet a:Cat)*.

Individual Value Restriction Mô tả *ObjectHasValue(OPE a)* thể hiện mối quan hệ giữa tất cả các cá thể được kết nối bởi mô tả thuộc tính đối tượng *OPE* và cá thể a. Ví dụ:

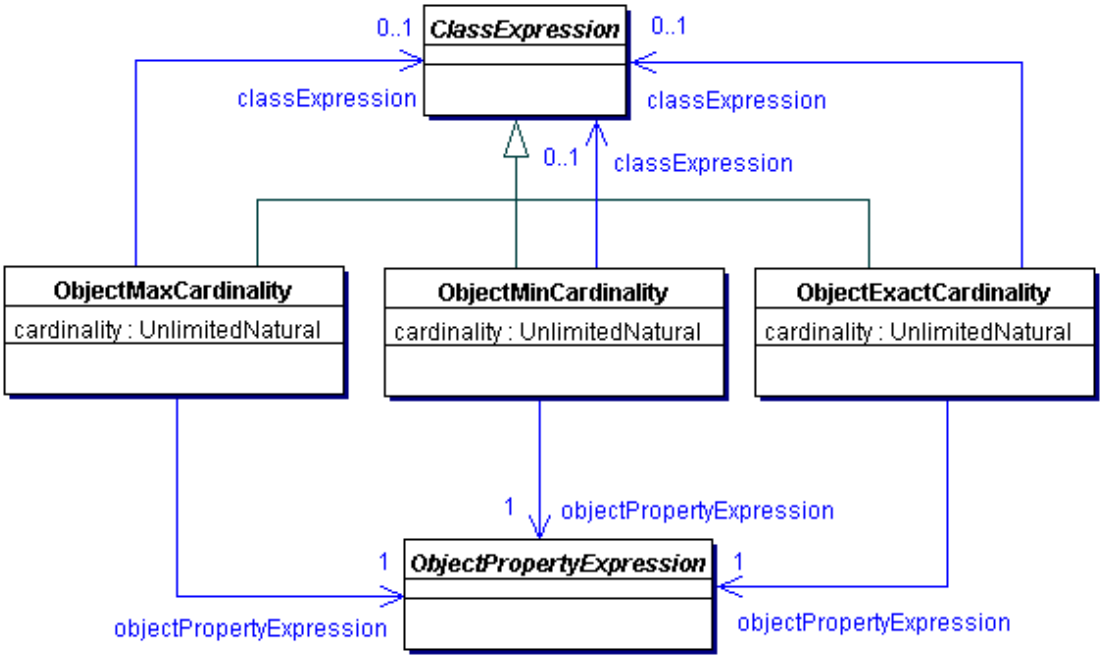
```
ObjectPropertyAssertion( a:fatherOf a:Peter a:Steve)
// Peter là ba của Steve
```

Giải thích: với khai báo như trên, *Peter thuộc mô tả lớp sau* mà không cần khai báo là *ObjectHasValue(a:fatherOf a:Steve)*.

Self-Restriction Mô tả *ObjectHasSelf(OPE)* thể hiện mối quan hệ giữa tất cả các cá thể được kết nối bởi mô tả thuộc tính đối tượng *OPE* với chính chúng. Ví dụ:

```
ObjectPropertyAssertion( a:quayQuanhTruc a:TraiDat a:TraiDat )
// Trái đất quay quan trục của trái đất.
```

Giải thích: với khai báo như trên, *a:TraiDat thuộc mô tả lớp sau* mà không cần khai báo là *ObjectHasSelf(a:quayQuanhTruc)*.



Hình 2.9: Object Property Cardinality Restrictions trong OWL 2

Ràng buộc thuộc tính đối tượng theo số lượng Các mô tả lớp trong OWL 2 có thể được tạo ra bằng cách đặt ra những số lượng hạn chế các cá thể mà các thuộc tính đối tượng có thể liên kết.

Số lượng tối thiểu (Minimum Cardinality) - mô tả *ObjectMinCardinality(n OPE CE)* biểu diễn mọi các thể được kết nối bởi mô tả thuộc tính đối tượng *OPE* đến số lượng tối thiểu n cá thể (khác nhau) thuộc mô tả lớp *CE* với n là số nguyên không âm. Nếu *CE* không được khai báo trong phát biểu trên mặc định sẽ là $owl : Thing$. Ví dụ:

```
ObjectPropertyAssertion( a:fatherOf a:Peter a:Steve )
ObjectPropertyAssertion( a:fatherOf a:Peter a:Bush )
ClassAssertion( a:Kid a:Steve )
ClassAssertion( a:Kid a:Bush )
```

Giải thích: với các điều kiện như trên thì Peter sẽ được ngầm hiểu thành viên của mô tả lớp sau $ObjectMinCardinality($ 2 a:fatherOf a:Kid) - Peter là bố của ít nhất 2 đứa trẻ.

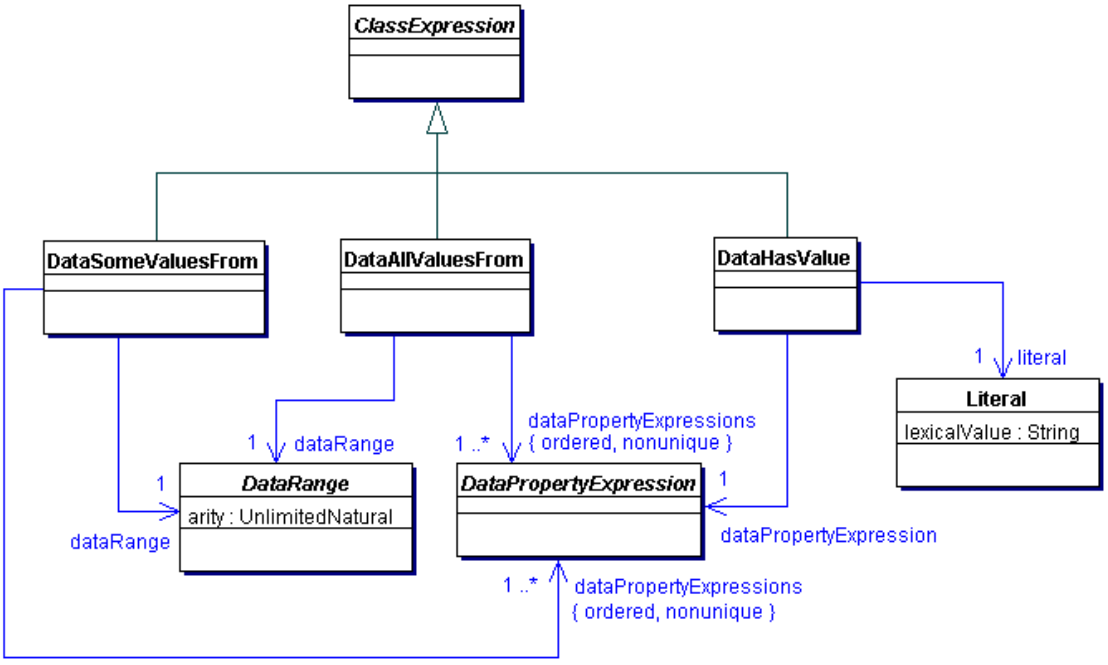
Số lượng tối đa (Maximum Cardinality) Mô tả *ObjectMaxCardinality(n OPE CE)* biểu diễn mọi các thể được kết nối bởi mô tả thuộc tính đối tượng OPE đến số lượng tối đa n cá thể (khác nhau) thuộc mô tả lớp *CE* với n là số nguyên không âm. Nếu CE không được khai báo trong phát biểu trên mặc định sẽ là *owl:Thing*. Ví dụ:

```
ObjectPropertyAssertion( a:hasPet a:Peter a:Tom)
// Peter có vật nuôi là Tom
ClassAssertion( ObjectMaxCardinality( 1 a:hasPet ) a:Peter )
// Peter thuộc lớp "những ai có tối đa 1 vật nuôi"
```

Giải thích: theo 2 điều kiệu trên thì Peter sẽ thuộc mô tả lớp *ObjectMaxCardinality(2 a:hasPet)* chỉ những "ai có tối đa 2 vật nuôi" (số vật nuôi $<= 2$), vì 2 phát biểu đầu tiên chỉ ra rằng Peter chắc chắn chỉ có một vật nuôi.

Số lượng chính xác (Exact Cardinality) - mô tả *ObjectExactCardinality(n OPE CE)* biểu diễn mọi các thể được kết nối bởi mô tả thuộc tính đối tượng *OPE* đến số lượng chính xác n cá thể (khác nhau) thuộc mô tả lớp CE với n là số nguyên không âm. Nếu CE không được khai báo trong phát biểu trên mặc định sẽ là *owl:Thing*. Mô tả lớp này tương được khi lấy giao của 2 mô tả số lượng tối đa và số lượng tối thiểu cùng n.

```
ObjectIntersectionOf( ObjectMinCardinality( n OPE CE )
ObjectMaxCardinality( n OPE CE ) ).
```



Hình 2.10: Data Property Restrictions trong OWL 2

Ràng buộc theo thuộc tính dữ liệu (Data Property Restrictions) Mô tả lớp dạng này được tạo ra bằng cách đặc các ràng buộc về kiểu dữ liệu, giới hạn các giá trị dữ liệu lên mô tả thuộc tính dữ liệu (data property expression) như trong hình. Những mô tả này cũng tương tự như ràng buộc thuộc tính đối tượng vừa nêu ra ở trên.

Existential Quantification Một mô tả lớp *DataSomeValuesFrom(DPE_1 ... DPE_n DR)* gồm n mô tả thuộc tính dữ liệu DPE_i, $1 <= i <= n$, và một miền dữ liệu (data

range) DR với số lượng tham số phải bằng n. Mô tả lớp này biểu diễn mối quan hệ dữ liệu giữa tất cả các cá thể kết nối với DPE_i tới các trực nghĩa lt_i, $1 <= i <= n$ với (lt_1 ... lt_n) trong DR. Ví dụ:

```
DataPropertyAssertion( a:hasAge a:Steve "17"^^xsd:integer ) // Steve 17 tuổi.
```

Giải thích: Vì Steve 17 tuổi nên chúng ta ngầm hiểu rằng Steve thuộc những ai có số tuổi là số nguyên và không lớn hơn 20 tuổi, tương đương phát biểu sau

```
DataSomeValuesFrom( a:hasAge
DatatypeRestriction( xsd:integer xsd:maxExclusive "20"^^xsd:integer ) )
```

Universal Quantification Một mô tả lớp *DataAllValuesFrom(DPE_1 ... DPE_n DR)* gồm n mô tả thuộc tính dữ liệu DPE_i, $1 <= i <= n$, và một miền dữ liệu (data range) DR với số lượng tham số phải bằng n. Mô tả lớp này biểu diễn mối quan hệ dữ liệu giữa tất cả các cá thể **chỉ** *(only)* kết nối với DPE_i tới các trực nghĩa lt_i, $1 <= i <= n$ với (lt_1 ... lt_n) trong DR. Ví dụ:

```
DataPropertyAssertion( a:hasZIP _:a1 "70000"^^xsd:integer )
// Mã vùng của _:a1 là số nguyên 70000
FunctionalDataProperty( a:hasZIP)
// Mỗi đối tượng chỉ có duy nhất một mã vùng
```

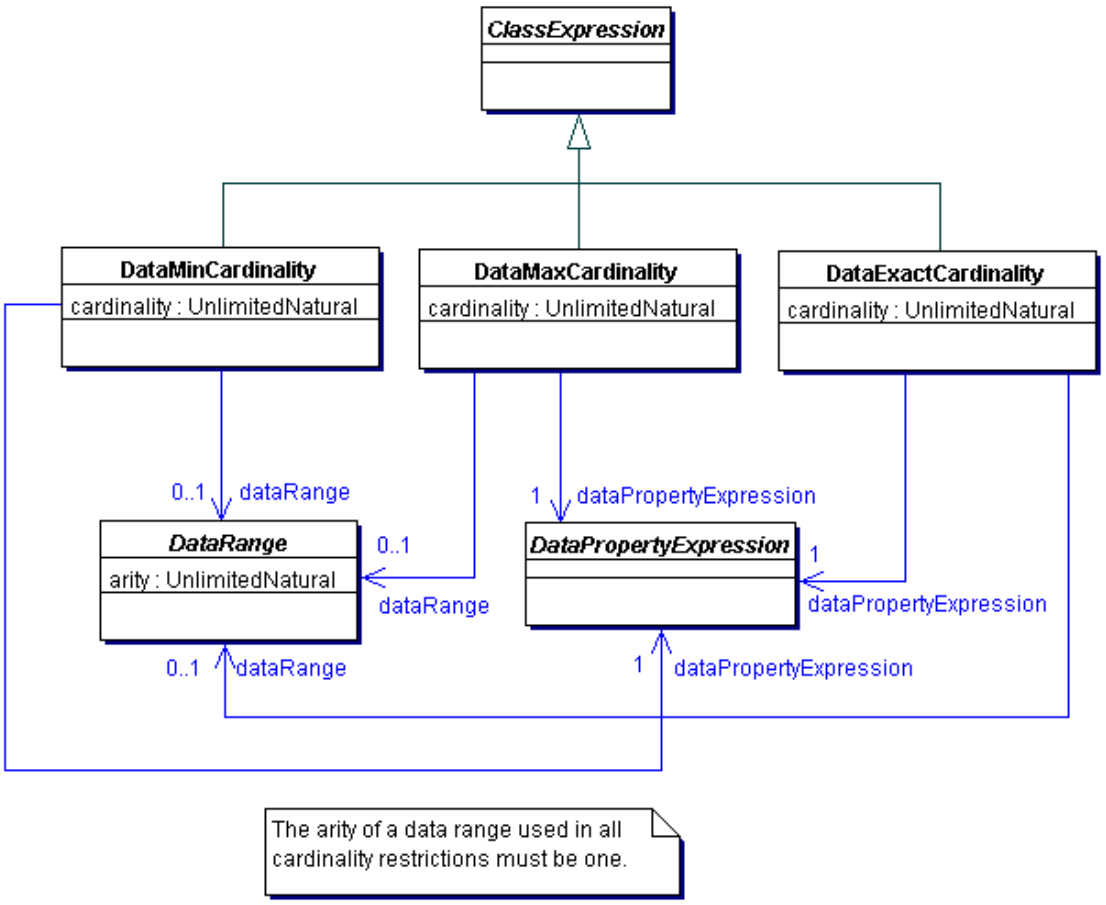
Giải thích: Mã vùng của một số quốc gia như Anh và Canada là dạng chuỗi (chứa ký tự và số), dựa vào 2 phát biểu trên chúng ta có thể hiểu rằng mã vùng của `_a:1` thuộc mô tả những mã vùng **chỉ** gồm số nguyên $DataAllValuesFrom(a : hasZIPxsd : integer)$

Ràng buộc bằng giá trị trực nghĩa - Literal Value Restriction Một mô tả *DataHasValue(DPE lt)* gồm một mô tả thuộc tính dữ liệu DPE và một trực nghĩa lt, nó biểu diễn quan hệ của những cá thể nào qua DPE tới lt. Ví dụ:

```
DataPropertyAssertion( a:hasAge a:Steve "17"^^xsd:integer )
```

Giải thích: Steve là thành viên của mô tả lớp sau mà không cần khai báo "những ai 17 tuổi".

```
DataHasValue( a:hasAge "17"^^xsd:integer )
```



Hình 2.11: Data Property Cardinality Restrictions trong OWL 2

Ràng buộc thuộc tính dữ liệu theo số lượng Tương tự ràng buộc thuộc tính đối tượng theo số lượng, OWL 2 cũng cho phép chúng ta tạo ra các mô tả lớp bằng cách các giá trị dữ liệu (thay vì cá thể) mà thuộc tính dữ liệu có thể liên kết đến. Cấu trúc được mô tả như trong hình.

Số lượng tối thiểu (Minimum Cardinality) Một mô tả lớp *DataMinCardinality(n DPE DR)* biểu diễn mọi cá thể được kết nối bởi thuộc tính dữ liệu *DPE* đến số lượng tối thiểu n các trực nghĩa khác nhau thuộc miền dữ liệu *DR* với n số nguyên không âm. Nếu *DR* không được khai báo trong phát biểu trên mặc định sẽ là *rdfs:Literal*. Ví dụ:

```
DataPropertyAssertion( a:hasPhoneNumber a:Steve "0123456789" )
DataPropertyAssertion( a:hasPhoneNumber a:Steve "0987654321" )
```

Giải thích: Với 2 phát biểu như trên tồn tại, có thể hiểu Steve thuộc lớp những ai có ít nhất 2 số điện thoại *DataMinCardinality(2 a:hasPhoneNumber)*.

Số lượng tối đa (Maximum Cardinality) Một mô tả lớp *DataMaxCardinality(n DPE DR)* biểu diễn mọi cá thể được kết nối bởi thuộc tính dữ liệu DPE đến số lượng tối đa n các trực nghĩa khác nhau thuộc miền dữ liệu DR với n số nguyên không âm. Nếu DR không được khai báo trong phát biểu trên mặc định sẽ là $rdfs : Literal$. Ví dụ:

```
DataPropertyAssertion( a:hasID a:Steve "0001" ) // Steve số CMND là 0001
FunctionalDataProperty( a:hasID ) // Mỗi người chỉ có duy nhất một số CMND
```

Giải thích: Với 2 phát biểu như trên tồn tại, có thể hiểu Steve thuộc lớp "những ai có nhiều nhất 2 số CMND" (số CMND <=2), *DataMaxCardinality(2 a:hasID)*.

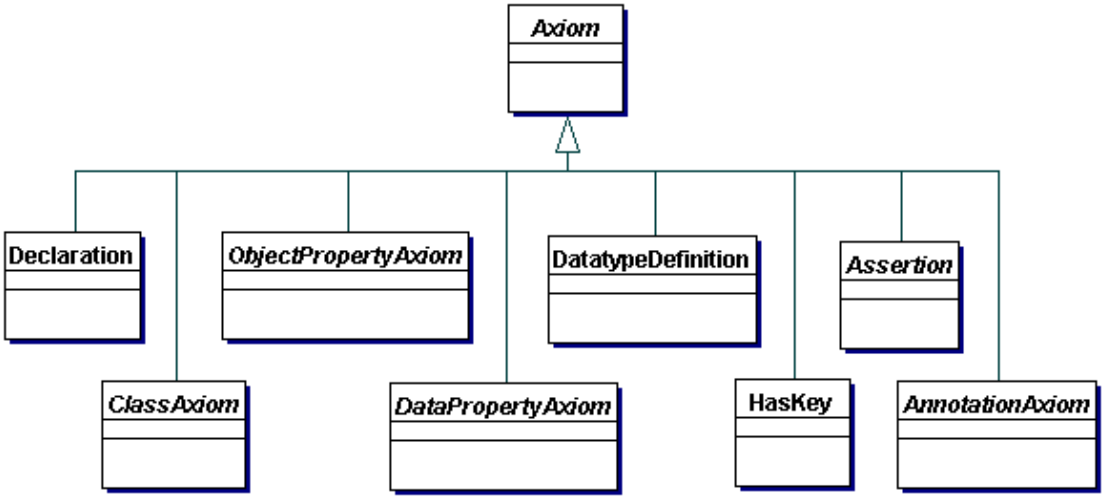
Số lượng chính xác (Exact Cardinality) Một mô tả lớp *DataExactCardinality(n DPE DR)* biểu diễn mọi cá thể được kết nối bởi thuộc tính dữ liệu *DPE* đến số lượng chính xác n các trực nghĩa khác nhau thuộc miền dữ liệu *DR* với n số nguyên không âm. Nếu *DR* không được khai báo trong phát biểu trên mặc định sẽ là *rdfs:Literal*. Ví dụ:

```
DataPropertyAssertion( a:hasID a:Steve "0001" ) // Steve số CMND là 0001
FunctionalDataProperty( a:hasID ) // Mỗi người chỉ có duy nhất một số CMND
```

Giải thích: Với 2 phát biểu như trên tồn tại, có thể hiểu Steve thuộc lớp "có duy nhất 1 CMND" *DataExactCardinality(1 a:hasName)*.

2.2.3.6 Các tiên đề (Axioms)

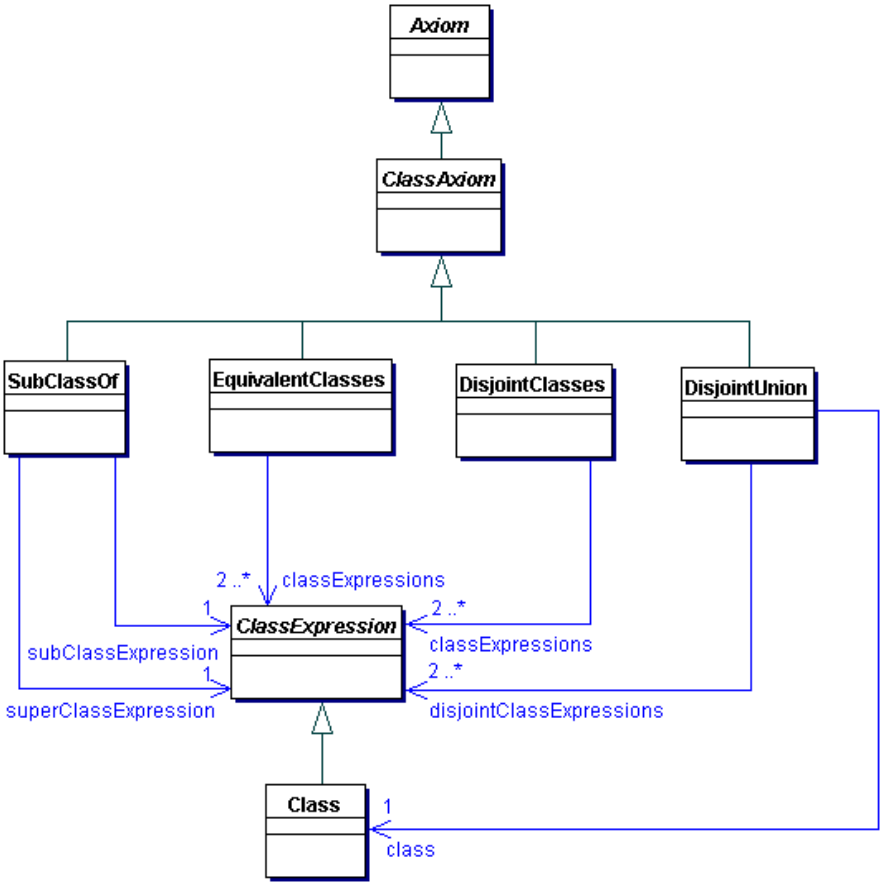
Thành phần chính của một OWL 2 Ontology chính là một tập hợp gồm các tiên đề * - những phát biểu mà chúng ta cho là đúng trong một lĩnh vực. OWL 2 cung cấp một loạt các loại phát biểu được mở rộng (thừa kế) từ lớp **Axiom** như trong hình. Các phát biểu có thể là các phát biểu khai báo (declarations), phát biểu về lớp, phát biểu về thuộc tính đối tượng/dữ liệu, phát biểu định nghĩa kiểu dữ liệu, khóa (HasKey), phát biểu khẳng định (assertions), và các phát biểu chú thích (annotations). Có thể nói những phát biểu (tiên đề) chính là những viên gạch để chúng ta xây dựng nên các tầng ngữ nghĩa (semantic) cho ontology.



Hình 2.12: Các dạng phát biểu của OWL 2

Những phát biểu về mô tả lớp - Class Expression Axioms Những phát biểu này cho phép biểu diễn các loại quan hệ giữa những mô tả lớp (Class Expressions) với nhau. Có 4 phát biểu được mở rộng ra từ phát biểu này như trong hình.

* Lưu ý: *Trong nội dung báo cáo này chúng em sẽ gọi vắng tắt các tiên đề - những phát biểu hiển nhiên đúng trong một lĩnh vực là những phát biểu nhằm diễn giải các lý thuyết, nguyên lý trở dễ hiểu hơn.*



HÌNH 2.13: Các dạng phát biểu về lớp của OWL 2

Phát biểu lớp con (SubClass Axioms) - một phát biểu $SubClassOf(CE_1\ CE_2\ )$ nói rằng mô tả lớp CE_1 là lớp con của mô tả lớp CE_2. Phát biểu này là phát biểu cơ bản trong OWL 2 được sử dụng để xây dựng một phân cấp lớp trong Ontology. Ví dụ:

```
SubClassOf( a:Driver a:Adult ) // Mỗi tài xế là một người trưởng thành.
SubClassOf( a:Adult a:Person ) // Mỗi người trưởng thành là một người.
ClassAssertion (a:Driver a:Steve ) // Steve là một tài xế.
```

Giải thích: nhờ 2 phát biểu đầu tiên, chúng ta ngầm hiểu tài xế cũng là một người trưởng thành, cũng là một người, vì vậy tất cả những cá thể là tài xế, cũng là người trưởng thành và cũng là người -> Steve là cũng là thành viên của 2 lớp *Adult* và *Person*.

Phát biểu lớp tương đương (Equivalent Classes) - một phát biểu *EquivalentClasses(CE_1 CE_n)* nói rằng mỗi mô tả lớp CE_i, $1 <= i <= n$ đều đồng nghĩa với các tất cả các mô tả lớp còn lại trong phát biểu này. Phát biểu *EquivalentClasses(CE_2 CE_1)* tương đương *EquivalentClasses(CE_1 CE_2)*. Ví dụ:

```
EquivalentClasses( a:Bike  a:Bicycle )
```

Disjoint Classes - một phát biểu *DisjointClasses(CE_1 ... CE_n)*nói rằng không cá thể nào của CE_i thuộc CE_j và ngược lại với $i \neq j$, $1 <= i, j <= n$. Ví dụ:

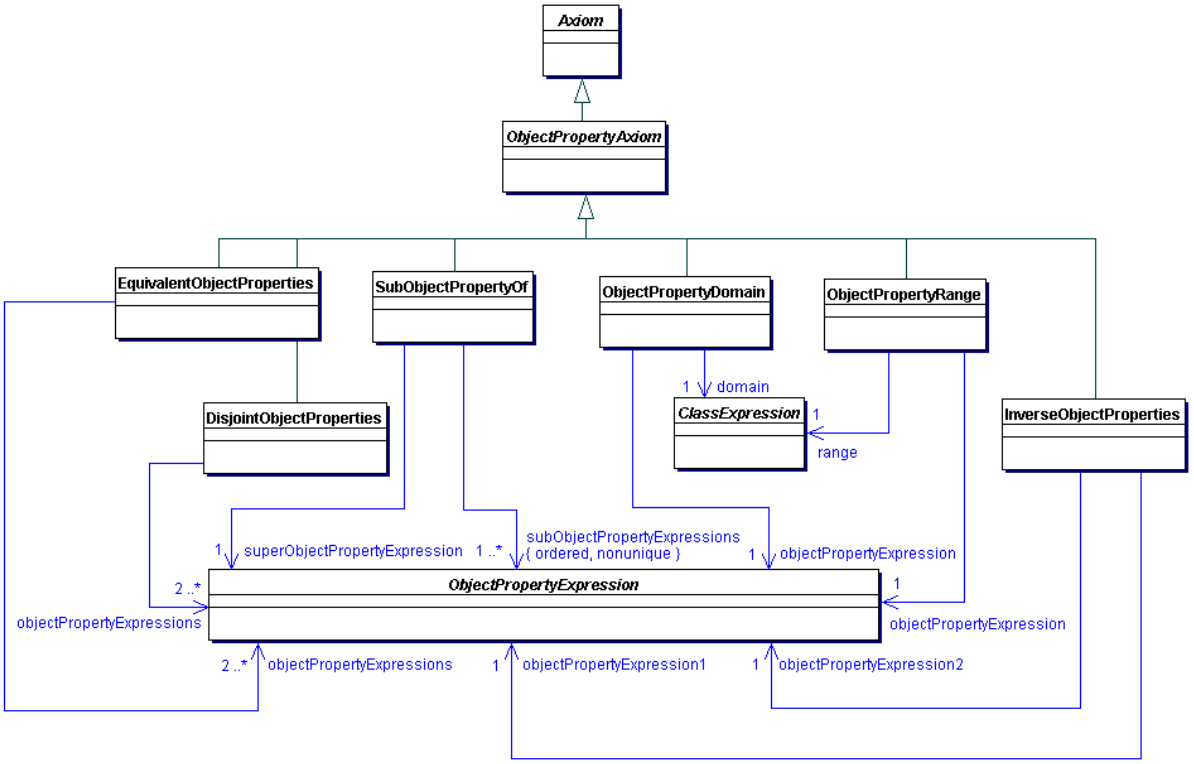
```
DisjointClasses( a:Man a:Woman )
ClassAssertion( a:Man a:Steve )
```

Giải thích: Steve không là nữ - *ObjectComplementOf(a:Girl)*, nếu chúng ta khai báo thêm $ClassAssertion(\ a : Girl\ a : Steve)$ sẽ làm cho ontology bị thiếu tính nhất quán (sẽ được nếu rõ hơn ở các chương sau).

Phát biểu về thuộc tính đối tượng (Object Property Axioms) Những phát biểu này cho phép biểu diễn các loại quan hệ giữa những mô tả thuộc tính đối tượng (Object Property Expressions) với nhau.

Thuộc tính đối tượng con (SubObjectPropertyOf) - một phát biểu *SubObjectPropertyOf(OPE_1 OPE_2)* nói rằng mô tả thuộc tính đối tượng OPE_1 là thuộc tính con của mô tả thuộc tính đối tượng OPE_2, có nghĩa nếu một cá thể x được liên kết bởi OPE_1 với y thì x cũng được liên kết bởi OPE_2 đến y. Ví dụ:

```
SubObjectPropertyOf( a:hasCat a:hasPet )
ObjectPropertyAssertion( a:hasCat a:Peter a:Tom) // Peter có con mèo là Tom
```



Hình 2.14: Các dạng phát biểu về thuộc tính đối tượng của OWL 2 (phần 1)

Giải thích: Ai có mèo thì cũng có thể hiểu là người đó có vật nuôi theo phát biểu đầu tiên, dựa theo phát biểu thứ hai chúng ta có ngầm hiểu là *Peter có vật nuôi là Tom* - *ObjectPropertyAssertion(a:hasPet a:Peter a:Brian)*.

Thuộc tính đối tượng tương đương (Equivalent Object Properties) - một phát biểu *EquivalentObjectProperties(* OPE_1 ... OPE_n *)* nói rằng mỗi mô tả thuộc tính OPE_i, $1 <= i <= n$ đều đồng nghĩa với các tất cả các mô tả thuộc tính còn lại trong phát biểu này phát biểu *EquivalentObjectProperties(* OPE_2 OPE_1 *)* tương đương *EquivalentObject-Properties(* OPE_1 OPE_2 *)*. Ví dụ:

```
EquivalentObjectProperties( a:hasBrother a:hasMaleSibling )
ObjectPropertyAssertion( a:hasBrother a:Chris a:Steve )
// Steve là anh của Chris
ObjectPropertyAssertion( a:hasMaleSibling a:Steve a:Chris )
// Chris là anh chị em ruột (mà là nam) của Steve
```

Giải thích: Do phát biểu đầu tiên có nghĩa "có anh" đồng nghĩa với "có anh chị em ruột (là nam)", từ 2 phát biểu sau chúng ta có thể suy ra các phát biểu sau mà cần khai báo chúng *ObjectPropertyAssertion(a:hasMaleSibling a:Chris a:Steve)* và *ObjectPropertyAssertion(a:hasBrother a:Stewie a:Chris)*.

Disjoint Object Properties - một phát biểu *DisjointObjectProperties*(OPE_1 ... OPE_n) nói rằng không cá thể được liên kết bởi cả của OPE_i và OPE_j với $i \neq j$, $1 <= i, j <= n$. Ví dụ:

```
DisjointObjectProperties( a:hasFather a:hasMother ) Có ba khác có mẹ.
ObjectPropertyAssertion( a:hasFather a:Steve a:Peter ) Peter là ba của Steve.
ObjectPropertyAssertion( a:hasMother a:Steve a:Lois ) Lois là mẹ của Steve.
```

Nếu thêm phát biểu *ObjectPropertyAssertion(a:hasMother a:Steve a:Peter)* sẽ dẫn đến tính thiếu nhất quán.

Thuộc tính đối tượng nghịch đảo - *phát biểu InverseObjectProperties(* OPE_1 OPE_2 *)* nói rằng thuộc tính đối tượng OPE_2 là nghịch đảo của OPE_1. Nếu cá thể x được kết nối bởi OPE_1 tới cá thể y, thì y sẽ được kết nối bởi OPE_2 tới x và ngược lại. Ví dụ:

```
InverseObjectProperties( a:parentOf a:childOf )
ObjectPropertyAssertion( a:childOf a:Peter a:Steve )
// Steve là con của Peter
```

Giải thích: Phát biểu đầu có thể hiểu "nếu x là con y" thì "y là ba mẹ của x", do vậy từ các phát biểu trên ta suy ra được Peter là ba mẹ của Steve tương đương *ObjectPropertyAssertion(a:parentOf a:Steve a:Peter)*.

Domain của thuộc tính đối tượng (Object Property Domain) - phát biểu *ObjectPropertyDomain(OPE CE)* nêu rõ domain của thuộc tính đối tượng là mô tả lớp

CE - nếu một cá thể x được liên kết bởi *OPE* tới cá thể nào đó, thì x phải thuộc mô tả lớp *CE*. Mặc định nếu không có khai báo này thì domain thuộc tính đối tượng sẽ là *owl:Thing*

```
ObjectProertyDomain( a:hasCat a:Person )
// Con người mới nuôi mèo
ObjectPropertyAssertion( a:hasCat a:Peter a:Tom )
// Tom là con mèo của Peter
```

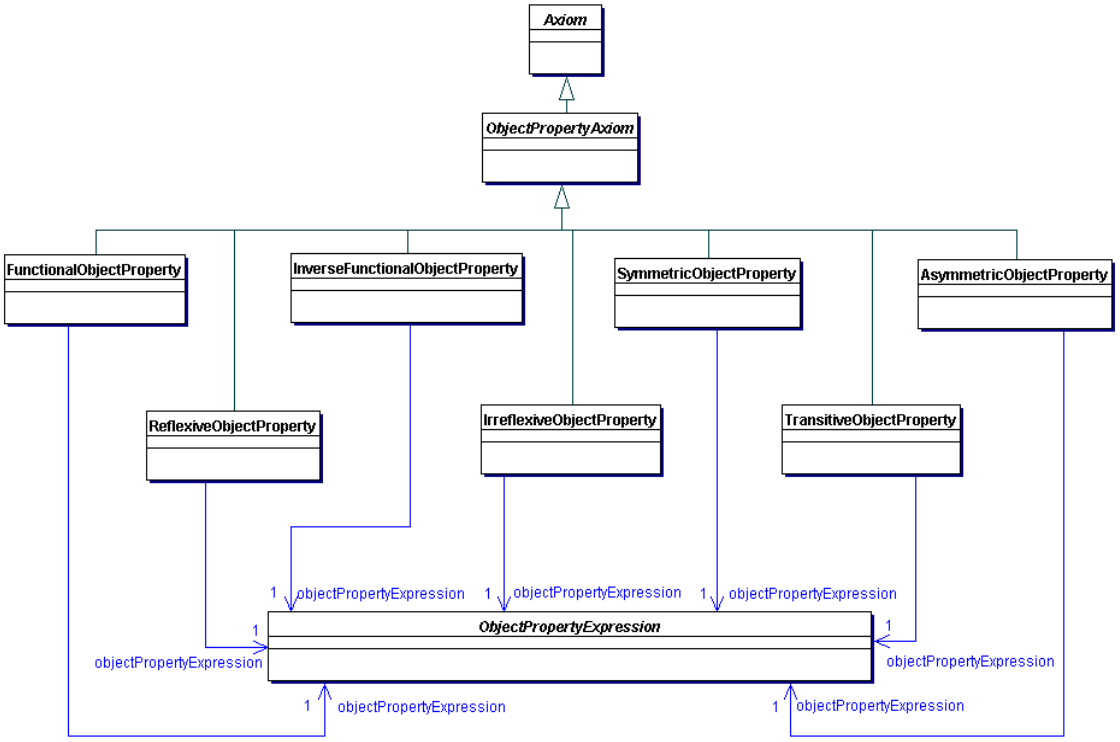
Giải thích: Mặc dù không khai báo Peter là người nhưng xét theo phát biểu đầu tiên chúng ta ngầm hiểu Peter là người, tương đương phát biểu *ClassAssertion(a:Person a:Peter)*.

Miền giới hạn của thuộc tính đối tượng (Object Property Range) - phát biểu *ObjectPropertyRange(OPE CE)* nêu rõ miền giới hạn (range) của thuộc tính đối tượng là mô tả lớp *CE* - nếu cá thể nào đó được liên kết bởi OPE tới cá thể x , thì x phải thuộc mô tả lớp CE. Mặc định nếu không có khai báo này thì range của thuộc tính đối tượng sẽ là $owl : Thing$. Ví dụ:

```
ObjectProertyRange( a:hasPet a:Cat )
// ai có vật nuôi thì bắt buộc đó phải là con mèo
ObjectPropertyAssertion( a:hasPet a:Peter a:Tom )
// Tom là vật nuôi của Peter
```

Giải thích: Mặc dù không khai báo Tom là con mèo nhưng do phát biểu đầu tiên, chúng ta ngầm hiểu Tom là con mèo, tương đương phát biểu $ClassAssertion(a : Cat\ a : Tom)$.

Ngoài các dạng phát biểu được nêu, mô tả thuộc tính đối tượng còn có thêm các phát biểu trong hình sau, nhằm quy định tính chất của mô tả thuộc tính đối tượng hay giữa các cá thể được kết nối bằng các thuộc tính đối tượng này, chẳng hạn thuộc tính *SymmetricObjectProperty* quy định mô tả thuộc tính đối tượng có tính đối xứng - nếu x là bạn y thì y cũng là bạn x.



Hình 2.15: Các dạng phát biểu về thuộc tính đối tượng của OWL 2 (phần 2)

Functional Object Properties phát biểu *FunctionalObjectProperty(OPE)* quy định với từng cá thể x, chỉ tồn tại nhiều nhất một cá thể y mà x được kết nối với y qua OPE. Ví dụ : Mỗi đối tượng chỉ có tối đa một người cha - *FunctionalObjectProperty(a:hasFather)*.

Inverse-Functional Object Properties phát biểu *InverseFunctionalObjectProperty(OPE)* quy định với từng cá thể x, chỉ tồn tại nhiều nhất một cá thể y mà y được kết nối qua OPE tới x . Ví dụ tương tự *FunctionalObjectProperty(OPE)*.

Reflexive Object Properties phát biểu *ReflexiveObjectProperty(OPE)* nói rằng mô tả thuộc tính có tính phản xạ, nghĩa là với từng kết nối bởi OPE tới chính nó. Ví dụ: *ReflexiveObjectProperty(a:knows)*, ai cũng biết bản thân họ, với khai báo *ClassAssertion(a:Person a:Peter)* thì có thể suy ra được phát biểu sau *ObjectPropertyAssertion(a:knows a:Peter a:Peter)*.

Irreflexive Object Properties ngược lại với phát biểu trên *IrreflexiveObjectProperty(OPE)*, không có cá thể nào tự kết nối với chính nó qua *OPE*. Ví dụ: *IrreflexiveObject-Property(a:marriedTo)*, không ai cưới chính mình.

Symmetric Object Properties phát biểu *SymmetricObjectProperty(OPE)* nói rằng *OPE* có tính đối xứng, nếu x nối với y qua *OPE* thì y cũng nối với x bởi *OPE*. Ví dụ: Với *SymmetricObjectProperty(a:friendOf)* và *ObjectPropertyAssertion(a:friend a:Peter a:Bush)* chúng ta suy ra được *ObjectPropertyAssertion(a:friend a:Bush a:Peter)*.

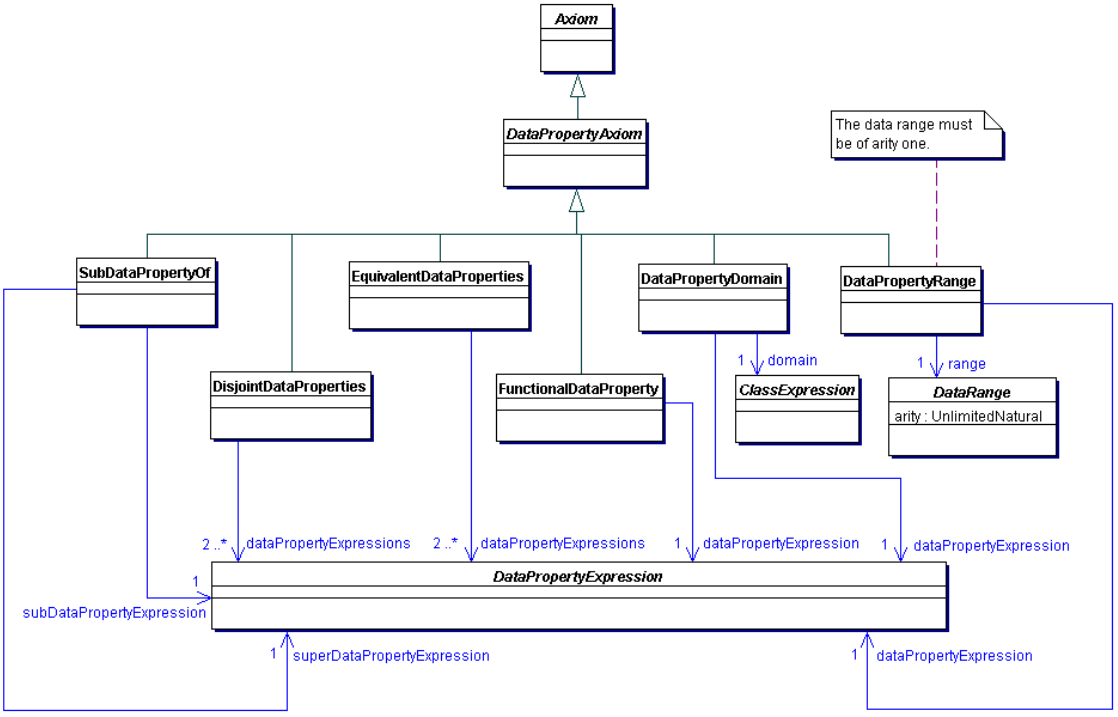
Asymmetric Object Properties phát biểu *AsymmetricObjectProperty(OPE)* nói rằng *OPE* có tính bất đối xứng, nếu x nối với y qua *OPE*, thì y không thể kết nối với x bởi *OPE*. Ví dụ: Với *AsymmetricObjectProperty(a:parentOf)* và *ObjectPropertyAsser-tion(a:parentOf a:Peter a:Bush)* thì nếu ta thêm thêm phát biểu *ObjectPropertyAsser-tion(a:parentOf a:Bush a:Peter)* sẽ làm cho ontology mất tính nhất quán (inconsistent).

Transitive Object Properties phát biểu *TransitiveObjectProperty(OPE)* nói rằng *OPE* có tính bắc cầu, nếu x nối với y bởi *OPE* và y nối với z cũng bằng *OPE* thì x nối với z qua *OPE*. Ví dụ:

```
TransitiveObjectProperty( a:partOf )
ObjectPropertyAssertion( a:partOf a:KhoaMMT a:UIT )
ObjectPropertyAssertion( a:partOf a:UIT a:VNU )
```

Giải thích nếu khoa mạng thuộc UIT, UIT thuộc đại học Quốc Gia (VNU), nhờ phát biểu đầu tiên chúng ta $partOf$ có tính chất bắc cầu nên suy ra được khoa mạng thuộc đại học Quốc Gia - *ObjectPropertyAssertion(a:partOf a:KhoaMMT a:VNU)*

Các phát biểu về thuộc tính dữ liệu Cũng giống như mô tả thuộc tính đối tượng, mô tả thuộc tính dữ liệu cũng được cung cấp các phát biểu như *SubDataPropertyOf*, *EquivalentDataProperties*, *DataPropertyDomain*, *DataProperyRange*.



Hình 2.16: Các dạng phát biểu về thuộc tính dữ liệu của OWL 2 (phần 2)

Phát biểu thuộc tính dữ liệu con (Data subproperties) Phát biểu SubDataPropertyOf(DPE_1 DPE_2) nói rằng mô tả thuộc tính dữ liệu DPE_1 là thuộc tính con của mô tả dữ liệu DPE_2 - có nghĩa nếu một cá thể x được kết nối bởi DPE_1 tới một trực nghĩa y, thì x được kết nối bởi DPE_2. Ví dụ:

```
SubDataPropertyOf( a:hasLastName a:hasName)
// Họ của một người cũng là tên họ của người đó
DataPropertyAssertion( a:hasLastName a:Peter "Smith" )
// Họ của Peter là "Smith"
```

Giải thích: Vì "có họ" **(hasLastName)** là thuộc tính con của "có tên" *(hasName)* nên chúng ta ngầm hiểu phát biểu sau mà không cần khai báo tên của Peter là "Smith", tương đương *DataPropertyAssertion(a:hasName a:Peter "Griffin")*

Phát biểu thuộc tính dữ liệu tương đương (Equivalent Data Properties) Một phát biểu *EquivalentDataProperties(* DPE_1 ... DPE_n*)* nói rằng mỗi mô tả thuộc tính

DPE_i, $1 <= i <= n$ đều đồng nghĩa với các tất cả các mô tả thuộc tính còn lại trong phát biểu này phát biểu *EquivalentDataProperties(DPE_2 DPE_1)* cũng tương đương EquivalentDataProperties(DPE_1 DPE_2). Ví dụ:

```
EquivalentDataProperties( a:hasName a:HoTen )
// a:hasName tương đượng với a:HoTen trong Tiếng Việt
DataPropertyAssertion( a:hasName a:Peter "Peter Nguyen" )
DataPropertyAssertion( a:HoTen a:Peter "Peter Nguyễn" )
```

Giải thích: Do phát biểu đầu tiên, từ phát biểu thứ 2 ta ngầm hiểu *DataPropertyAssertion(a:hasName a:Peter "Peter Nguyễn")* tên tiếng Anh "Peter Nguyen" tương đương tên tiếng Viết "Peter Nguyễn" và ngược lại *DataPropertyAssertion(a:HoTen a : Peter "Peter Nguyen")* .

Domain của thuộc tính dữ liệu (Data Property Domain) Phát biểu *DataPropertyDomain(DPE CE)* nêu rõ domain của thuộc tính dữ liệu là mô tả lớp CE - nếu một cá thể x được liên kết bởi DPE tới vài trực nghĩa nào đó, thì x phải là cá thể thuộc mô tả lớp CE.

```
DataPropertyDomain( a:hasBankAccount a:Person )
// Con người mới có tài khoản ngân hàng
ObjectPropertyAssertion( a:hasBankAccount a:Peter "0002" )
// Peter có tài khoản ngân hàng "0002"
```

Giải thích: Không cần thiết phải khai báo Peter là người vì dựa theo chúng ta cũng kết luận được Peter là người từ 2 phát biểu trên, *ClassAssertion(a:Person a:Peter)*.

Miền giá trị của thuộc tính dữ liệu (Data Property Range) - phát biểu *DataPropertyRange(DPE DR)* nói rằng miền giá trị hợp lệ của mô tả thuộc tính dữ liệu DPE là miền dữ liệu DR - nếu các cá thể được kết nối bởi DPE tới 1 trực nghĩa x, thì x trong miền dữ liệu DR. Ví dụ:

```
DataPropertyRange( a:hasName xsd:string )
DataPropertyAssertion( a:hasName a:Peter "Peter Nguyen" )
```

Nếu khai báo thêm phát biểu sau sẽ làm cho ontology bị thiếu nhất quán (inconsistent):

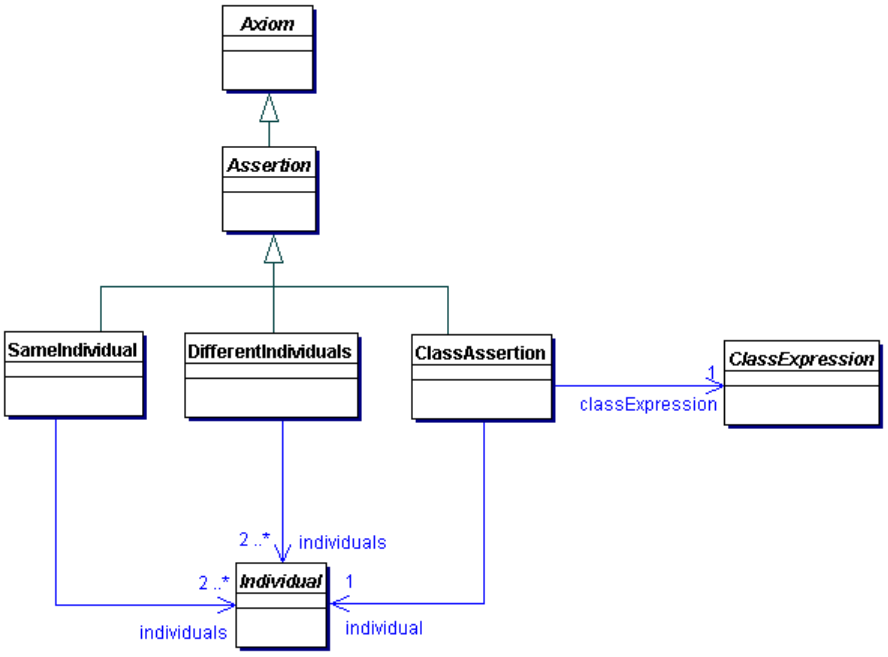
```
DataPropertyAssertion( a:hasName a:Peter "42"^^xsd:integer)
```

Functional Data Properties - phát biểu *FunctionalDataProperty(DPE)* nói rằng với mỗi x, chỉ có nhiều nhất một trực nghĩa y mà x được gán cho bởi *DPE*. Ví dụ:

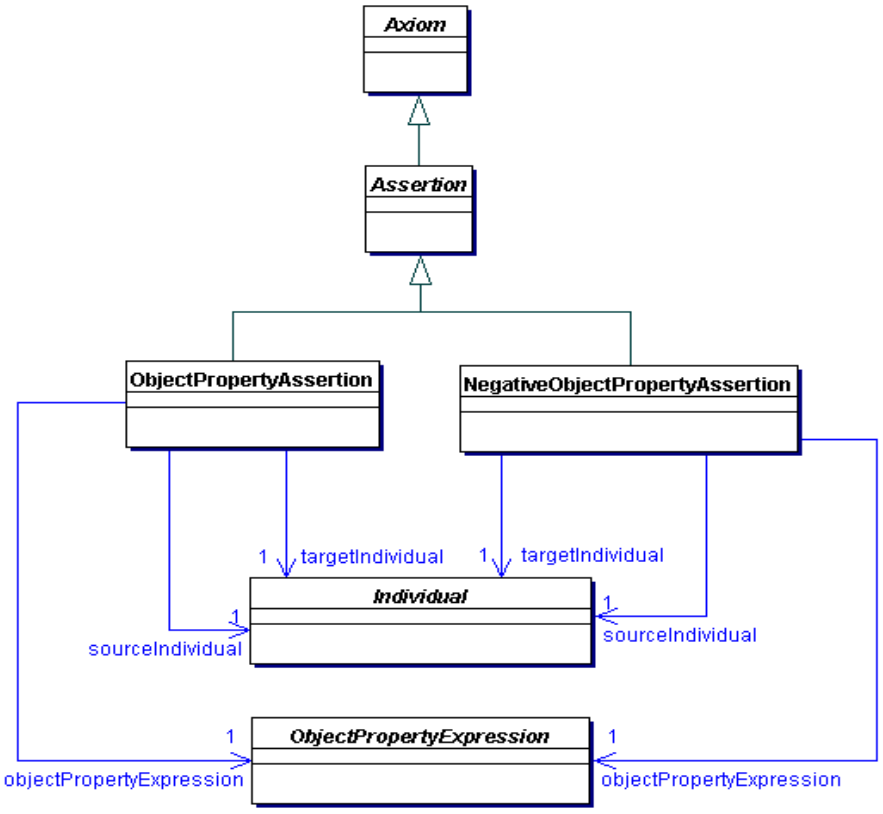
```
FunctionalDataProperty( a:hasAge )
// Mọi đối tượng chỉ có tối đa một số tuổi
DataPropertyAssertion( a:hasAge a:Steve "17"^^xsd:integer )
```

Các phát biểu trên sẽ bị thiếu nhất quán nếu chúng ta thêm vào phát biểu sau :

```
DataPropertyAssertion( a:hasAge a:Steve "15"^^xsd:integer)
```



Hình 2.17: Các phát biểu gán lớp cho cá thể và cá thể giống nhau/khác nhau trong OWL 2



Hình 2.18: Các phát biểu về thuộc tính đối trượng của cá thể trong OWL 2

Các phát biểu về cá thể (Assertions) Các phát biểu về cá thể thường được gọi là *facts*. Để rõ hơn, các loại assertions khác nhau được thể hiện trong các hình sau.

SameIndividual - khẳng định sự tương đồng của các cá thể trong phát biểu.

DifferentIndividuals - khẳng định sự khác nhau của các cá thể trong phát biểu.

ClassAssertion - khẳng định một cá thể thuộc một mô tả lớp/ lớp cụ thể trong phát biểu.

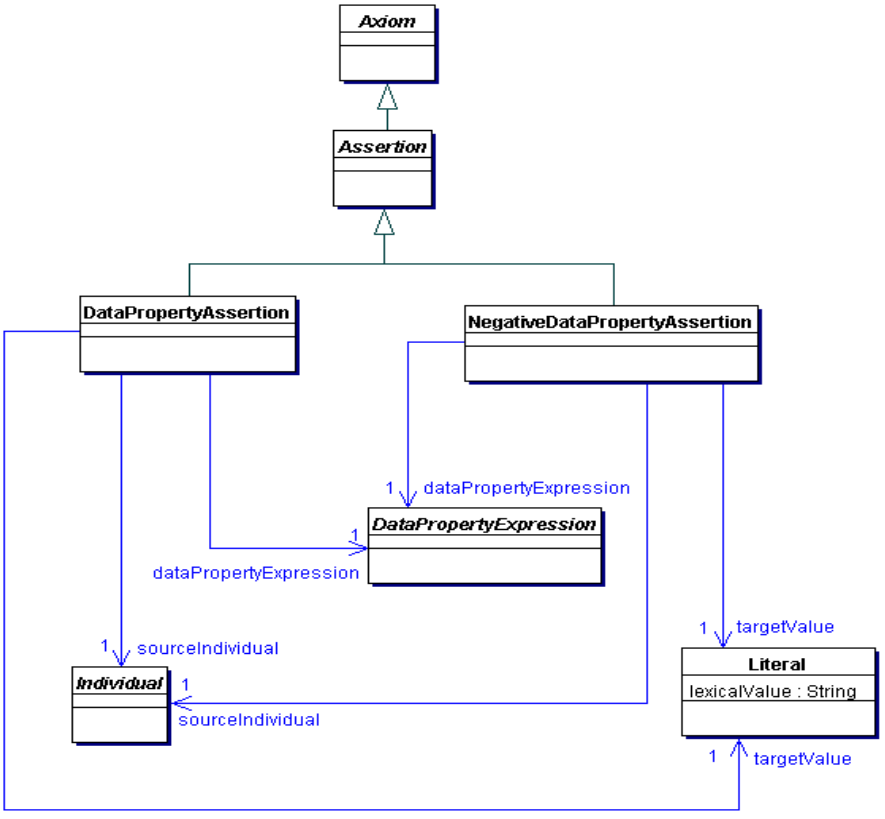
ObjectPropertyAssertion - khẳng định mối quan hệ từ một cá thể tới một cá thể nào đó bởi một mô tả thuộc tính đối tượng.

NegativeObjectPropertyAssertion - phủ định mối quan hệ từ một cá thể tới một cá thể khác bởi một mô tả thuộc tinh đối tượng.

DataPropertyAssertion - khẳng định mối quan hệ từ một cá thể tới một trực nghĩa bởi một mô tả thuộc tính dữ liệu.

NegativeDataPropertyAssertion - phủ định mối quan hệ từ một cá thể tới một trực nghĩa bởi một mô tả thuộc tính dữ liệu.

Kết luận Chúng em vừa trình bày qua những thành phần chính của OWL2 cũng như cách sử dụng của chúng trong quá trình suy luận (reasoning) nhằm tìm ra thông tin ẩn bên trong những phát biểu được khai báo, với mục tiếu cuối cùng là trình bày tính năng phân loại tự động. Vì độ dài khóa luận hạn chế, nên có một số thành phần của OWL 2 chưa được nêu ra ở đây như **Datatype Definitions**, **Keys**, **Annotations**, **Datatype Maps**. Thầy cô và các bạn có thể tham khảo thêm ở [14]. Phần tiếp theo chúng em xin được giới thiệu về Semantic Web Rule Language, một ngôn ngữ điều luật có nhiệm vụ chính là tăng cường tính năng suy luận cho ngôn ngữ OWL 2.



Hình 2.19: Các phát biểu về thuộc tính dữ liệu của cá thể trong OWL 2

2.3 Semantic Web Rule Language

Nếu chỉ sử dụng các thành phần của OWL 2 được trình bày ở chương trước thì vẫn chưa đủ diễn tả hết tất cả các mối quan hệ trong ontology. Semantic Web Rule Language (SWRL) là một ngôn ngữ điều luật tuân theo các quy ước Description Logic mà OWL 2 tuân thao [1]. SWRL cho phép người sử dụng khai báo các điều luật dựa trên các khái niệm của OWL như lớp, thuộc tính đối tượng, thuộc tính dữ liệu nhằm cung cấp một khả năng suy luận mạnh mẽ hơn so với chỉ dùng OWL 2. SWRL có ưu điểm là giúp tối ưu khả năng lập luận, tuy nhiên nó cũng có nhược điểm là dễ làm cho các phát biểu bị thiếu nhất quán (inconsistent).

2.3.1 Cấu trúc của một SWRL Rule [1]

Một luật SWRL chứa một phần điều kiện, hay còn gọi là rule body, và một phần kết quả, hay còn gọi là rule head. Cả phần body và head đều là giao của các positive atoms.

$$atom \text{ ^ } atom \text{ ... -> } atom \text{ ^ } atom$$

Có thể hiểu một SWRL Rule theo cách như sau : Khi tất cả các điều kiện nhỏ nhất (atom) trong phần điều kiện (body) đúng thì chắc chắn rằng những ý được nêu ra trong phần kết quả (head) cũng đúng. Một rule *atom* có dạng:

$$p(\ arg_1, arg_2, \ldots arg_n\ )$$

Trong đó p là kí hiệu cho nội dung điều kiện (Predicate) và arg_i, $1 <= i <= n$, là những khái niệm hay tham số của một *Rule Atom*. Trong SWRL, nội dung điều kiện có thể là các lớp, các thuộc tính hoặc kiêu dữ liệu trong OWL 2. Tham số truyền vào có thể là cá thể, giá trị dữ liệu, hoặc biến để gán cho các khái niệm vừa nêu. Tên biến chỉ có hiệu lực trong cùng một rule, vì vậy có thể sử dụng lại tên biến cho 2 rule khác nhau.

2.3.2 Các loại Rule Atom

Sau đây, chúng em xin trình bày các loại *Atom* trong SWRL Rule, đồng thời kèm theo những ví dụ cơ bản thể hiện tính năng quyết định dựa trên các điều kiện của SWRL.

2.3.2.1 Class Atom

Một **Class Atom** gồm một tên lớp hay mô tả lớp trong OWL2 Ontology và một tham số duy nhất đại diện cho cá thể của lớp đó. Ví dụ:

```
Person(?p)
Vehicle(?v)
Man(Peter)
```

Tham số có thể là biến đại diện cho cá thể `?p`, `?v` hoặc tên của cá thể `Peter`. Một rule đơn giản để khẳng định rằng "ai làm nam đều là người":

```
Man(?x) -> Person(?x)
```

2.3.2.2 Individual Property Atom

Một **Individual Property Atom** gồm một thuộc tính đối tượng (object property) và 2 tham số đại diện cho 2 cá thể trong OWL2 ontology, tham số có thể là biến hoặc tên của cá thể. Ví dụ:

```
hasBrother(?x, ?y)   // có anh em
hasSibling(Steve, ?y) // có anh/chị em
```

Trong ví dụ, *hasBrother* và *hasSibling* là các thuộc tính đối tượng, `?x` và `?y` là biến đại diện cho các cá thể, và `Steve` là tên của một cá thể (named individual). Xem ví dụ sau:

```
Person(?p) ^ hasSibling(?p,?s) ^ Man(?s) -> hasBrother(?p,?s)
```

Giải thích: Nếu một người `?p` nào đó có một anh chị em `?s` nào đó, và người `?s` này là nam thì người `?p` có anh/em trai là người `?s`. Rule này cũng có thể được biểu diễn trong bằng các phát biểu về lớp, domain và range OWL2 như sau:

```
SubClassOf(a:Man a:Person)
SubClassOf(a:Woman a:Person)
DisjointClasses(a:Woman a:Man)
SubObjectProperty(a:hasBrother a:hasSibling)
ObjectPropertyRange(a:hasBrother a:Man)
ObjectPropertyRange(a:hasSibling a:Person)
ObjectPropertyDomain(a:hasSibling a:Person)
ObjectPropertyDomain(a:hasBrother a:Person)
```

Trong trường hợp chúng ta có các khẳng định sau về 2 cá thể `Peter` và `Nguyen`

```
ClassAssertion(a:Person a:Peter)
ClassAssertion(a:Man a:Nguyen)
ObjectPropertyAssertion(a:hasSibling a:Peter a:Nguyen)
```

Dựa vào rule đã khai báo ở trên *hoặc* các phát biểu về lớp, domain, range như trên, thì các khẳng định vừa nêu sẽ suy ra cùng một kết quả đó là "Peter có anh/em trai là Nguyen" tương đương với phát biểu *ObjectProperyAssertion(a:hasBrother a:Peter a:Nguyen)*. **Nhận xét:** Qua các ví dụ này, có thể thấy rõ ràng SWRL có lợi thế hơn trong việc suy ra các ẩn ý so với việc chỉ sử dụng các phát biểu của OWL2, tuy nhiên có một nhược điểm đó là tính nhất quán (consistency) của ontology dễ bị vi phạm hơn do rule có thể xung đột với các phát biểu của OWL2. Lấy lại các phát biểu và rule trong ví dụ trên:

```
Person(?p) ^ hasSibling(?p,?s) ^ Woman(?s) -> hasBrother(?p,?s)
Person(?p) ^ hasSibling(?p,?s) ^ Man(?s) -> hasBrother(?p,?s)
ObjectPropertyRange(a:hasBrother a:Man) // Có anh/em trai
```

Giải thích: 2 Rule mâu thuẫn với nhau do phát biểu "có anh/em trai". Bản thân SWRL sẽ không kiểm tra các mâu thuẫn này cho đến khi chúng xảy ra và làm cho ontology bị thiếu nhất quán. Vì vậy, người phát triển ontology cần cẩn thận trong việc kết hợp cả SWRL và OWL2.

2.3.2.3 Data Valued Property Atom

Một Data Valued Property Atom gồm một thuộc tính dữ liệu trong OWL2 và 2 tham số. Tham số đầu tiên được gán cho một các thể trong OWL 2, có thể là biến số hay tên của cá thể. Tham số thứ hai gán cho một giá trị dữ liệu, có thể là biến số hay giá trị. Ví dụ:

```
hasAge(?x, ?age)
hasName(?x, "Nguyen")
hasNumberOfChilds(Peter, ?x)
```

Một rule đơn giản với ý nghĩa "những ai có xe, có bằng lái thì là tài xế":

```
Person(?p) ^ hasCar(?p, true) ^ hasLicense(?p, true) -> Driver(?p)
```

Chúng ta cũng có thể dùng tên của cá thể cụ thể:

```
Person(Steve) ^ hasCar(Steve, true) ^ hasLicense(Steve, true) -> Driver(Steve)
```

Rule này chỉ có tác dụng duy nhất lên cá thể `Steve`.

2.3.2.4 Different Individuals Atom

Một *Atom* dạng này gồm cú pháp *differentFrom* với 2 tham số đại diện cho 2 cá thể. Ví dụ:

```
differentFrom(?x, ?y)
differentFrom(Steve, Peter)
```

2.3.2.5 Same Individuals Atom

Một *Atom* dạng này gồm cú pháp *sameAs* với 2 tham số đại diện cho 2 cá thể. Ví dụ:

```
sameAs(?x, ?y)
sameAs(Steve, Peter)
```

2.3.2.6 Data Range Atom

Một Data Range Atom gồm một dạng dữ liệu hoặc một tập hợp các trực nghĩa (literals) và duy nhất một tham số đại diện cho giá trị dữ liệu. Ví dụ:

```
xsd:int(?x)
xsd:string(?x)
[3,2,1](?x)
xsd:int[>=5,<=9](?x)
```

2.3.2.7 Những Built-in Atom

Một trong những tính năng mạnh nhất của SWRL chính là khả năng hỗ trợ người phát triển tự xây dựng các built-in nhằm mở rộng khả năng ra các điều kiện cho SWRL Rule. Trong phạm vi khóa luận, chúng em sẽ không đề cập đến cách xây dựng các SWRL Built-in [15], mà chỉ sử dụng những core built-in [16] có sẵn để xây dựng nên tính năng phân loại. Dưới đây là một ví dụ về core built-in. Để có nhiều thông tin hơn mời thầy cô và các bạn đọc thêm ở [16].

Built-In dùng cho các phép so sánh Ví dụ:

```
Person(?p) ^ hasAge(?p, ?age) ^ swrlb:greaterThan(?age, 17) -> Adult(?p)
Vehicle(?v) ^ canCarryNumberOfPassenger(?v, ?x) ^
swrlb:greaterThan(?x, 30) -> Bus(?v)
```

Cú pháp	Ví dụ	Ý nghĩa
swrlb:equal	swrlb:equal(?x,9)	$x = 9$?
swrlb:notEqual	swrlb:notEqual(?x,9)	$x \neq 9$?
swrlb:lessThan	swrlb:lessThan(?x, 9)	$x < 9$?
swrlb:lessThanOrEqual	swrlb:lessThanOrEqual(?x, 9)	$x <= 9$?
swrlb:greaterThan	swrlb:greaterThan(?x, 9)	$x > 9$?
swrlb:greaterThanOrEqual	swrlb:greaterThanOrEqual(?x, 9)	$x >= 9$?

Bảng 2.1: Built-In dùng để so sánh

2.4 Tính nhất quán của một ontology

Khái niệm tính nhất quán (consistency) đã được nhắc đến nhiều lần trong những phần trước. Vậy tính nhất quán là gì và nó có ảnh hưởng như thế nào đến một mô hình ontology . **Trả lời:** Tính nhất quán chỉ sự thống nhất về ý nghĩa của những phát biểu trong ontology hay ý nghĩa của những phát biểu này không mâu thuẫn với nhau. Tính nhất quán cực kì quan trọng đối với bất kì một ontology nào, nếu ngữ nghĩa của của những phát biểu trong ontology đã tồn tại sự mâu thuẫn thì những thông tin được suy luận ra từ những phát biểu này cũng trở nên vô lý. Chính vì thế, quá trình suy luận (reasoning) sẽ không thực hiện được nếu ontology không có tính nhất quán. Trong phần này, chúng em xin được dành ra các nguyên nhân thường phổ biến làm cho các phát biểu khai báo mâu thuẫn với nhau, qua đó sẽ giúp chúng ta tránh phải những lỗi này khai phát triển ontology.

2.4.1 Các định nghĩa cần lưu ý [2]

Lớp không hợp lý - Unsatisfiable Class Dùng để chỉ một lớp mà các phát biểu mô tả, khai báo về nó có ý nghĩa mâu thuẫn với nhau.

Ví dụ

```
Cow SubClassOf:  Vegetarian
Vegetarian SubClassOf: Animal and eats only Plant
DisjointClasses: Plant, Animal
```

```
MadCow  SubClassOf: Cow and eats some Sheep
Sheep  SubClassOf: Animal
Individual: Dora type: MadCow
```

Giải thích Trong ví dụ trên thì *MadCow* là một lớp không hợp lý do ý nghĩa các phát biểu về nó mâu thuẫn với nhau. Cow là lớp con của *Vegeterian*, mà *Vegeterian* chỉ ăn Plant (từ *only* đồng nghĩa với *AllValuesFrom* được giải thích trong phần OWL 2). Trong khi đó khai báo của lớp *MadCow* là lớp con của *Cow* và mô tả ăn (eats some - từ **some** tương đương *SomeValuesFrom*) *Sheep* (Sheep là một lớp con của Animal). Từ đây, suy luận từ các phát biểu trên sẽ có khả năng sinh ra mâu thuẫn: Sheep cũng có khả năng là một phần của Plant (do phát biểu *eats only plant*). Một điểm lưu ý với phát biểu DisjointClasses thì Plant và Animal luôn được khẳng định là khác nhau, nói cách khác không tồn tại một cá thể nào vừa thuộc lớp Plant và vừa thuộc lớp Animal, điều này mâu thuẫn với phát biểu ở câu trước.

Ontology có ý nghĩa không mạch lạc - Incoherent Ontology Dùng để chỉ một *ontology/model* có ý nghĩa không mạch lạc rõ ràng do nó có chứa ít nhất một lớp không hợp lý *Unsatisfiable Class* và với điều kiện là trong những *Unsatisfiable Class* này không được chứa bất kì một cá thể nào. Giả sử ta có ontology A chứa các phát biểu trong ví dụ trên ngoại trừ phát biểu cuối cùng `Individual: Dora type: MadCow` thì ta có thể nói ontology A không mạch lạc rõ ràng do nó chứa unsatisfiable class là MadCow. Chúng ta vẫn có thể sử dụng được ontology A vì nó vẫn còn tính nhất quán (*Consistency*) miễn là không có phần tử nào thuộc lớp MadCow.

Ontology không có tính nhất quán - Inconsistent Ontology Một ontology không nhất quán khi đạt đủ 2 điều kiện: có ít nhất một lớp không hợp lý và có ít nhất một cá thể là thành viên của một trong những lớp không hợp lý này. Như đã thể hiện trong ví dụ đầu tiên thì cá thể *Dora* thuộc lớp *MadCow* (Một lớp không hợp lý thì không nên tồn tại bất kì cá thể nào nếu như chúng ta muốn đảm bảo tính nhất quán cho ontology). Lưu

ý: một ontology không nhất quán thì không được dùng để suy luận vì những suy luận từ đây có khả năng đều bất hợp lý.

2.4.2 Các nguyên nhân phổ biến dẫn đến tính thiếu nhất quán

Các nguyên nhân dẫn đến tính thiếu nhất quán trong ontology gây bởi các lỗi được phân loại thành lỗi gây ra bởi phát biểu ở mức độ lớp (Class level - TBox), các lỗi gây ra bởi phát biểu ở mức độ cá thể (Instance/Individual level - ABox)[17] và lỗi gây ra bởi sự kết hợp của cả 2 nguyên nhân vừa nêu trên.

Khởi tạo cá thể cho một lớp không hợp lý - (TBox + ABox)* Khởi tạo cá thể cho một Unsatisfiable Class được xem là nguyên nhân phổ biến nhất gây ra tính thiếu nhất quán trong ontology. Ví dụ:

```
Individual: Dora type: MadCow
```

Như đã giải thích ở phần trên, một lớp không hợp lý thì không nên tồn tại bất kì cá thể nào nếu như chúng ta muốn đảm bảo tính nhất quán cho ontology

Khởi tạo cá thể thuộc 2 class được disjoint với nhau (TBox + ABox) Đây là một trường hợp dễ bắt gặp vì nó sai ngay trong phát biểu về logic.

```
DisjointClasses: Animal, Plant
Individual: Cat Types: Animal, Plant
```

Bản thân phát biểu Disjont đã khẳng định 2 lớp *Animal*, *Plant* sẽ không bao giờ có cá thể chung nên phát biểu thứ 2 đã mâu thuẫn với nó.

* TBox *Chỉ những phát biểu liên quan đến lớp*
ABox: *Chỉ những phát biểu liên quan đến cá thể*

Các phát biểu về cá thể (ABox) mâu thuẫn với nhau Trường hợp này thì tương tự như nguyên nhân ở trên nhưng khác ở chỗ là lần này sự mâu thuẫn nằm trong các biểu ở cấp độ cá thể (ABox). Ví dụ:

```
Individual: Cat Types: Animal, not Animal
```

Phát biểu "oneOf" về lớp (TBox) Phát biểu bao gồm hoặc một trong (phát biểu liệt kê cá thể ObjectOneOf) cho phép khai báo các cá thể (ABox) thuộc một lớp. Sự kết hợp này đôi khi dẫn đến sự thiếu nhất quán. Lấy ví dụ sau:

```
Class: MyFavouriteCat EquivalentTo: {Tom}
Class: AllMyCats EquivalentTo: {Tom, Jerry}
DisjointClasses: MyFavouriteCat, AllMyCats
```

Chúng ta có diễn giải sau đây *MyFavouriteCat* tương đương *Tom*, *AllMyCats* tương đương *Tom, Jerry*, *MyFavouriteCat* và *AllMyCats* không được có chung cá thể nào. Nhưng rõ ràng thì *Tom* là cá thể chung của 2 lớp vì vậy các phát biểu này mâu thuẫn.

Kết luận Trên đây chúng em đã liệt kê những nguyên nhân phổ biến dẫn đến thiếu nhất quán qua những ví dụ đã được đơn giản hoá để dễ dàng nắm bắt được đâu là căn nguyên gây ra sự mâu thuẫn về logic. Trên thực tế với những ontology có số lượng phát biểu lớn và phức tạp rất khó để người dùng có thể nhận diện được đâu là các phát biểu mâu thuẫn trước khi nó gây ra sự thiếu nhất quán. Chúng ta thường sử dụng các công cụ phổ biến để tìm các lớp không hợp lý là debugger (được hỗ trợ trong các thư viện OWL-API, Pellet, Jena) và trên hết là sự cẩn thận của người phát triển ontology để hạn chế việc gây ra tính thiếu nhất quán trong ontology. Trong quá trình tìm hiểu chúng em cũng có đọc được các bài báo hay trong đó đề ra các giải pháp để sửa chữa các lớp bất hợp lý và tìm giải thích cho chúng, tuy nhiên do trong phạm vi khóa luận chúng em cũng không thể giới thiệu được, thầy cô và các bạn có thể tham khảo thêm ở [18], [19] và [20]. Trên đây là toàn bộ những kiến thức đã tìm hiểu được mà chúng em tin rằng chúng nếu

không nắm được chúng thì việc phát triển ontology và thao tác với thư viện lập trình liên quan đến ngôn ngữ OWL sẽ gặp rất nhiều khó khắn.

2.5 Nền tảng và các thư viện lập trình sử dụng

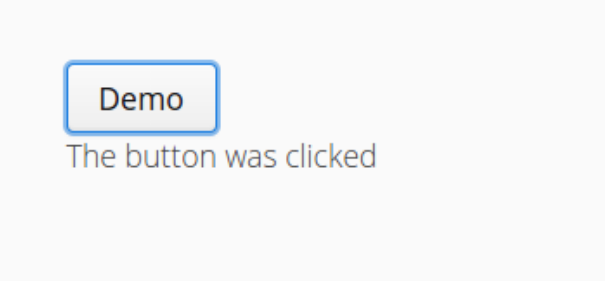
Giới thiệu Trong phần này, chúng em xin giới thiệu khái quát qua nền tảng và các thư viện lập trình (API) được sử dụng để xây dựng hệ thống. Các thư viện sử dụng gồm có OWLAPI [21], SWRL-API [22], Pellet [23] và nền tảng web sử dụng - Vaadin Framework.

2.5.1 Vaadin Framework

Giới thiệu Vaadin Framework là nền tảng xây dựng một ứng dụng web trên Java được thiết kế giúp tạo ra một ứng dụng web chất lượng cao một cách dễ dàng nhất - tập trung chủ yếu vào one-page web application. Không giống như những framework web hiện nay đòi hỏi lập trình viên phải có kiến thức về HTML5, Javascript và ít nhất một ngôn ngữ back-end. Vaadin giúp chúng ta tạm quên việc đi việc phải viết các client Javascript hay từng dòng HTML để xây dựng giao diện người dùng - nói cách khác nó làm cho công việc phát triển front-end trở nên cực kì đơn giản, dễ hình dung nhất là việc phát triển một ứng dụng web trên Vaadin cũng tương tự khi chúng ta phát triển một ứng dụng desktop thông thường với các công cụ Java như AWT, Swing, hay SWT hoặc window form với C#(C-Sharp). Với Vaadin, để phát triển được toàn bộ một ứng dụng, ngôn ngữ duy nhất chúng ta cần nắm đó là Java. Một ví dụ minh họa về tính đơn giản trong khi sử dụng Vaadin:

```
Button button = new Button("Demo");
button.addClickListener( new ClickListener() {
layout.addComponent(new Label("The button was clicked"));
}
```

Đây là kết quả:



Hình 2.20: Vaadin Demo Click

Thử nhìn lại nếu chúng ta xây dựng cùng chức năng trên dù là thủ công hay sử dụng các framework Web thông thường thì đều cần phải có một client script đảm nhận việc bắt sự kiện click của người dùng ở browser (front-end) và truyền nó lên server, ngay khi server nhận được (phải có đoạn code ở server "back-end" để xử lý thông tin click truyền lên từ browser), server sẽ trả về thông báo "The button was clicked", rồi front-end javascript phải thêm hay chèn một đoạn text "The button was clicked" vào HTML. Với Vaadin tác vụ vừa mô tả được thực hiện bằng đoạn code trên một cách đơn giản và dễ hiểu hơn rất nhiều.

2.5.1.1 Kiến trúc

Vaadin hỗ trợ 2 mô hình lập trình: client và server. Mô hình lập trình phía server mạnh mẽ hơn. Mô hình lập trình phía server đảm nhận phần giao diện trên trình duyệt và giao tiếp AJAX giữa trình duyệt và server - hay nói cách khác các giao tiếp giữa server-client nhằm hỗ trợ những thao tác của người dùng đã được xử lý bởi framework và được cài đặt vào bên trong các *Component* của Vaadin. Trong phạm vi ứng dụng mà chúng em xây dựng chỉ sử dụng những thành phần server - chỉ sử dụng các *Component* mà Vaadin cung cấp, cộng với một vài plugin (được viết sẵn cho Vaadin) từ Vaadin Directory [24] để giúp việc phát triển nhanh chóng hơn.

Web Server gồm các thành phần:

Components Các *Built-in Components*, *Add-on Components* đều là những thành phần được xây dựng sẵn bởi Vaadin hoặc được cung cấp dưới dạng các add-on từ Vaadin Directory [24] nhằm giúp việc phát triển UI nhanh chóng hơn. Chúng đảm nhiệm back-end code để giao tiếp với các *Built-in Widgets*, *Add-on Widgets* ở phía browser (client side).

UI Logic, Service và Custom Components là những phần mà lập trình viên phải tự viết code để cài đặt các tác vụ tương tác mà họ mong muốn, tuy nhiên Vaadin cũng cung cấp các abstract class, interface để hỗ trợ việc này.

Back-end Trong một ứng dụng thông thường thì đây chính là nơi để xử lý giao tiếp với các đối tượng trong cơ sở dữ liệu - nơi thực hiện các thao tác Create Read Update Delete (CRUD) [25].

Client-Side Engine gồm các thành phần:

Built-in Widgets Đây chính là thành phần Client-side của *Built-in Components* đảm nhiệm bắt các sự kiện của người dùng với browser và giao tiếp với các *Built-in Components* ở server.

Add-on Widgets là client-side của *Add-on Components*.

Custom-Widgets là client-side của *Custom Components*.

Tóm lại, Vaadin cung cấp sẵn gần như đầy đủ mọi thành phần UI chúng ta cần để phát triển một ứng dụng web tương tác tốt với người dùng một cách nhanh chóng và tiện lợi. Chúng ta sẽ giảm bớt được công việc khi phải viết Javascript, HTML cho cliend-side khi sử dụng các Vaadin UI Components.
Tuy là tích hợp mọi thứ vào UI components của mình, Vaadin tách biệt UI Logic với các thiết kế giao diện. Điều này đồng nghĩa bên cạnh một giao diện mặc định rất tốt của Vaadin chúng ta có thể thiết kế giao diện một cách dễ dàng thông qua các file CSS hoặc cũng có thể tự định nghĩa HTML Template cho riêng mình [*].

[*] *Vaadin Theme: https://vaadin.com/book/-/page/themes.html*

2.5.2 Spring Boot

Spring Boot [26] là một dự án nằm trong bộ *Spring Application Framework* với mục đích giúp người mới sử dụng *Spring* có thể dễ dàng phát triển một ứng dụng một cách nhanh nhất có thể mà không cần phải mất thời gian cấu hình trong các tập tin xml truyền thống.

2.5.3 Spring 4 Vaadin

Dây là một dự án mã nguồn mở [27] vừa được khởi động sử dụng với nền tảng là *Spring Boot* và *Vaadin*, mục tiêu của dự án này là đem các thư viện mạnh mẽ của Spring Framework như Spring Data, Spring Security và đặc biệt là khả năng dependency injection vào Vaadin. Chúng em chủ yếu sử dụng các dự án này vào việc khởi tạo ứng dụng, khởi tạo các Bean Component một lần vào đưa nó (inject) vào lớp **UI** của hệ thống.

2.5.4 Thư viện lập trình OWLAPI

Thư viện lập trình Ontology Web Language là một thư viện mã nguồn mở (phát hành dưới 2 giấy phép **LGPL** và **Apache**) [21] được viết bằng Java với mục đích hỗ trợ các lập trình viên phát triển các ứng dụng có liên quan đến OWL 2 Ontology. Tính đến thời điểm hiện tại thư viện đã được phát triển đến phiên bản 4.0 - cũng là phiên bản được sử dụng trong ứng dụng của chúng em. Thư viện có các thành phần chính như sau:

- API để tương tác với các thành phần của OWL 2 được đề cập trong chương 2.
- Renderer và Parser (dùng đọc và ghi OWL 2 Ontology) nhiều dạng cú pháp khác nhau đã đề cập ở chương 2 như *RDF/XML*, *OWL/XML*, *OWL Functional Syntax*, *Manchester OWL Syntax* và *Turtle*.
- Reasoner Interfaces hỗ trợ các loại reasoners khác nhau nhằm phục vụ cho việc suy luận.

Danh sách các Ontology Reasoner được hỗ trợ trong phiên bản 4.0: FaCT++, Hermit, Pellet [23], JFact. Các đối thành phần (thực thể, mô tả lớp/thuộc tính,...) được giải thích trong lúc giới thiệu OWL 2 đều đươc OWL-API mô hình hóa thành các đối trượng trong Java, với tiếp đầu ngữ *OWL* + tên thành phần. Ví dụ:

```
Class               -> OWLClass  #Lớp
ObjectProperty      -> OWLObjectProperty # thuộc tính đối tượng
OWLClassExpression -> OWLClassExpression # Mô tả lớp
```

Tương tự cho các thành phần OWL 2 khác.

2.5.5 Pellet Reasoner

Như đã được nhắc đến nhiều lần trong báo cáo, suy luận được xem là một điểm đáng giá nhất của ngôn ngữ OWL2. Tuy nhiên, việc suy luận ra các phát biểu hàm ý rõ ràng không phải là một việc dễ dàng nếu chúng ta thực hiện thủ công bằng cách đọc và hiểu các phát biểu như đã làm trong chương 2. Sử dụng Reasoner sẽ làm cho công việc suy ra các mảnh thông tin ẩn chứa bên trong Ontology trở nên dễ dàng hơn rất nhiều, có rất nhiều reasoner được phát triển để thực hiện tác vụ này. Trong số đó Pellet [23] là một thư viện tối ưu nhất và một ưu điểm đặc biệt là khả năng suy luận từ những SWRL Rule. Pellet có thể được sử dụng với OWL-API thông qua reasoner Interface của OWL-API.

```
OWLReasonerFactory rf; // OWLReasoner là inteface reasoner của OWL-API
OWLReasonerFactory rf = PelletReasonerFactory.getInstance();
OWLReasoner rs = rf.createReasoner(ontology, new SimpleConfiguration());
```

2.5.6 SWRL API

SWRL API được xây dựng bởi nhóm phát triển dự án Protege [28] với mục tiêu tổ chức các SWRL Rule theo tên nhằm dễ dàng quản lý, đồng thời họ cũng giới thiệu SQWRL

[22] một ngôn ngữ truy vấn dành cho Ontology. Trong nội dung báo cáo, chúng em chỉ sử dụng tính năng SWRL Rule của SWRLAPI các tính năng còn lại có thể được tham khảo tại [22]. Các tính năng mà chúng em sử dụng gồm render, parse SWRL Rule và thêm/xóa SWRL Rule theo tên của chúng. Một ví dụ nhanh về cách sử dụng API này với OWL-API:

```
OWLOntology ont = // load ontology ../
// Convert OWLOntology -> SWRLAPIOWLOntology
SWRLAPIOWLOntology SWRLOnt =  SWRLAPIFactory.createOntology(ont);
```

Chương 3

Thiết kế hệ thống phân loại tự động

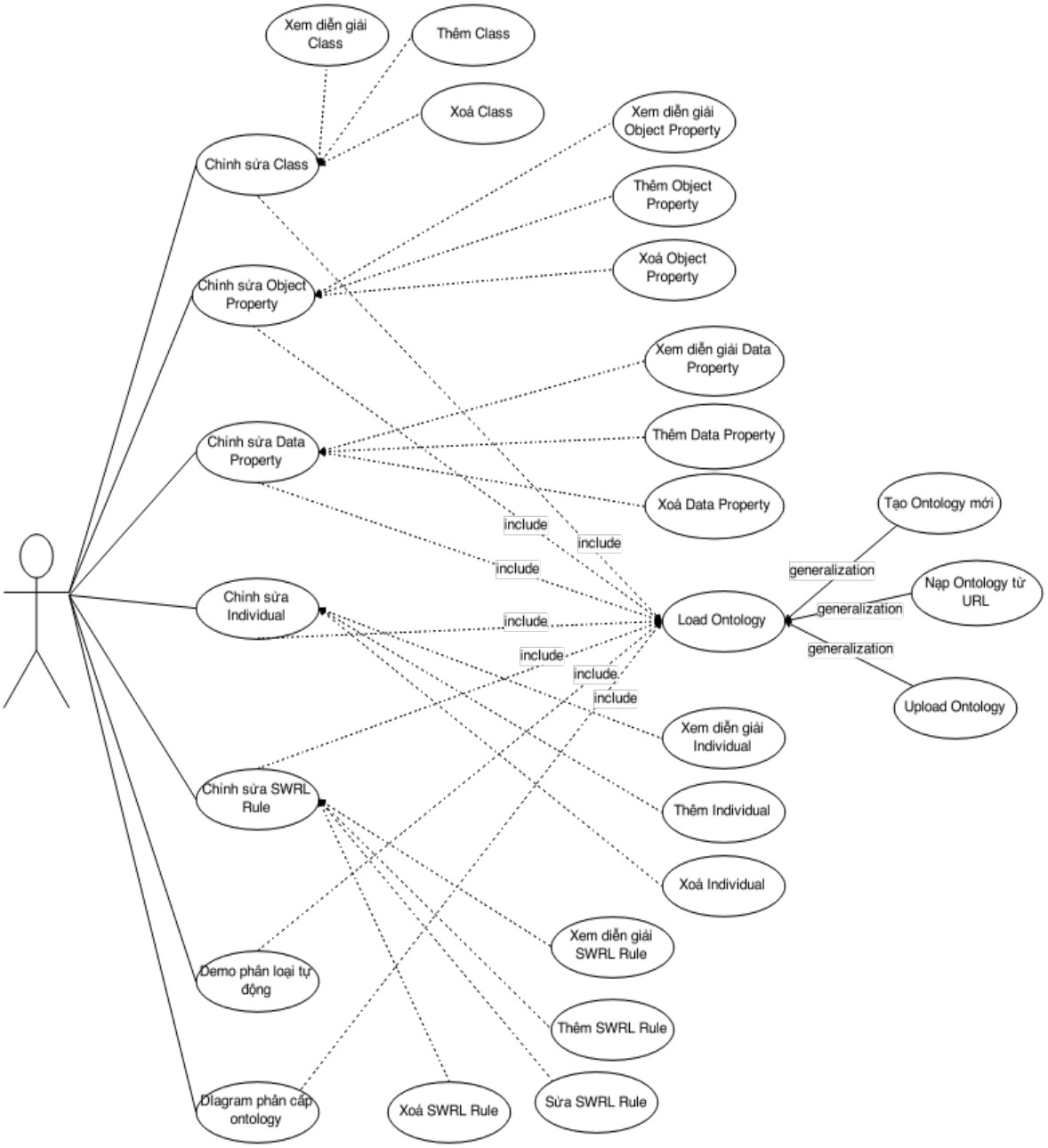
Giới thiệu Về nội dung chương này, chúng em sẽ miêu tả lại quá trình mà chúng em đã thiết kế nên hệ thống gồm các bước như cách áp dụng một số design pattern vào việc xây dựng hệ thống, các phác thảo sơ khai của giao diện, cách kết hợp các thư viện lập trình đã giới thiệu với Vaadin Framework và cuối cùng là thiết kế một ontology dùng để trình bày tính năng phân loại sau khi hệ thống được xây dựng thành công.

3.1 Giải thích về việc lựa chọn nền tảng sử dụng

Như đã được trình bày ở mục trên, Vaadin Framework là một nền tảng xây dựng Web dựa trên ngôn ngữ Java, nhưng không giống với phần lớn các framework Web khác vốn hoạt động theo mô hình Model-View-Controller (MVC). Vaadin chỉ cung cấp một bộ gồm rất nhiều UI Component có sẵn (đã được giải thích ở mục trước) làm cho việc xây dựng giao diện được đơn giản hóa một cách tối đa. Vì thế, lập trình viên sẽ không cần phải quan tâm nhiều đến HTML, JavaScript và ít phải quan tâm đến CSS, điều này giúp tập trung vào việc xây dựng logic của hệ thống tốt hơn. Việc thiết kế cho hệ thống này trên nền web sẽ tương tự như khi thiết kế một ứng dụng GUI trên nên Desktop nhờ có Vaadin.

3.2 Thiết kế Use Case của hệ thống

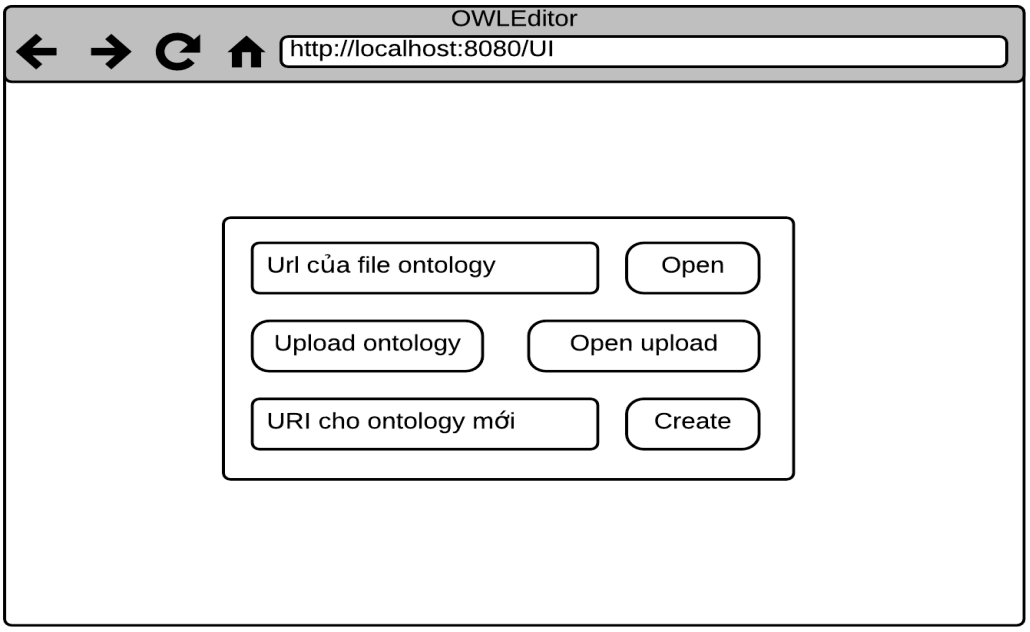
Trước tiên, chúng em xin giới thiệu một cách tổng quan về hoạt động của hệ thống.



Hình 3.1: Sơ đồ Use Case của hệ thống

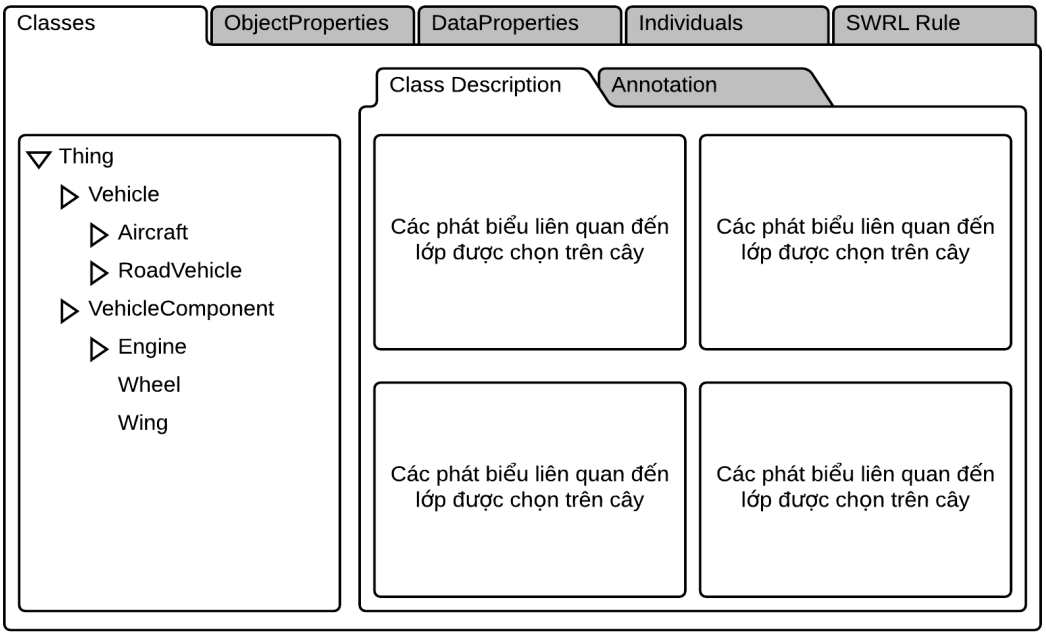
3.3 Thiết kế giao diện phác thảo

Hệ thống hay ứng dụng sẽ gồm 2 view chính: view đầu tiên tạm gọi là EntryView dùng để nạp/tạo mới các tài liệu OWL 2. View thứ hai là view chính của ứng dụng, gọi là MainView cho phép thực hiện việc chỉnh sửa ontology, thực hiện suy luận (phục vụ cho tính năng phân loại tự động). MainView sẽ gồm nhiều tab, mỗi loại thực thể (gồm lớp,

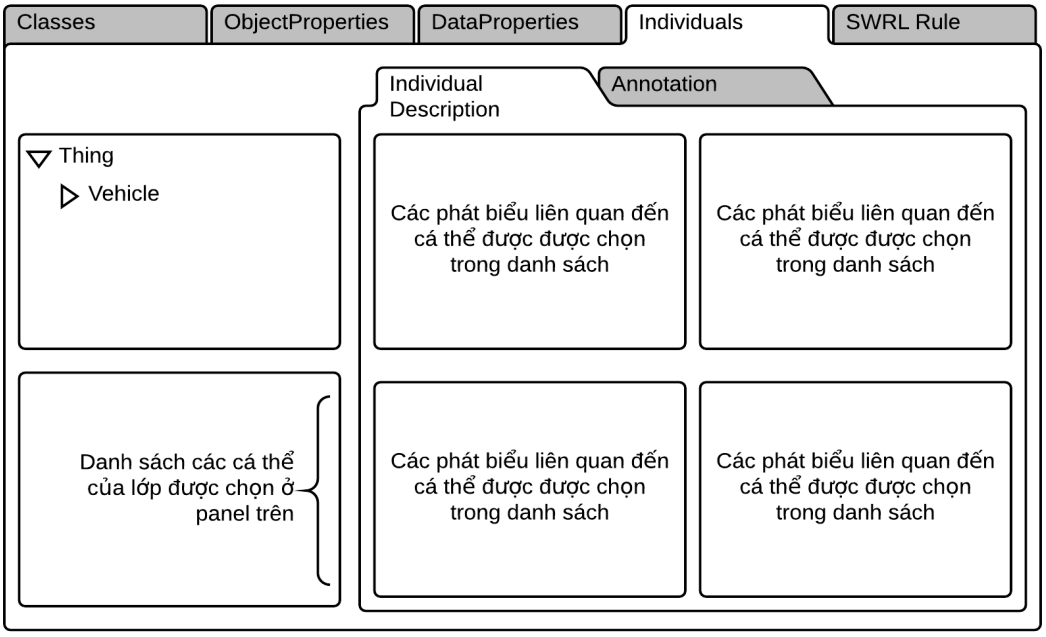


Hình 3.2: Phác thảo Entry View

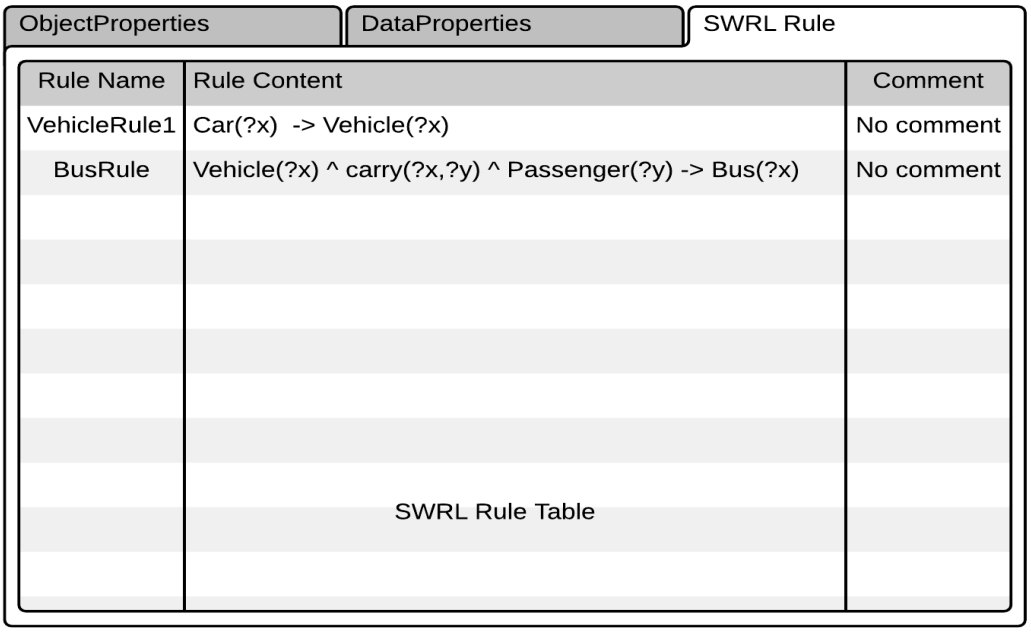
thuộc tính đối tượng, thuộc tính dữ liệu, cá thể) và SWRL Rule sẽ được tổ chức thành từng tab với tên tương ứng. Ngoài ra, còn có thêm 2 tab khác là tab Demo (demo tính năng phân loại tự động) và tab Diagram (sơ đồ phân cấp các thực thể của Ontology). Các tab về lớp (Class), thuộc tính đối tượng (Object Propery) và thuộc tính dữ liệu (Data Propery) sẽ có bố cục giống nhau (Hình 3.3). Bên trái là một cây biểu diễn cấu trúc phân cấp của các loại thực thể này và bên phải là các panel nhỏ. Từng panel sẽ tương ứng với từng loại phát biểu có liên quan đến thực thể được chọn bên trái. Riêng tab về các cá thể sẽ có bố cục hơi khác so với các thực thể còn lại, nó sẽ có thêm một danh sách (sẽ nằm bên dưới cấu trúc cây chứa các lớp - Hình 3.4). Bên trong tab SWRL Rules sẽ



Hình 3.3: Phác thảo Main View - Tab "Classes" (chứa lớp và các mô tả lớp liên quan)



Hình 3.4: Phác thảo Main View - Tab "Individuals" (chứa cá thể và các mô tả liên quan)

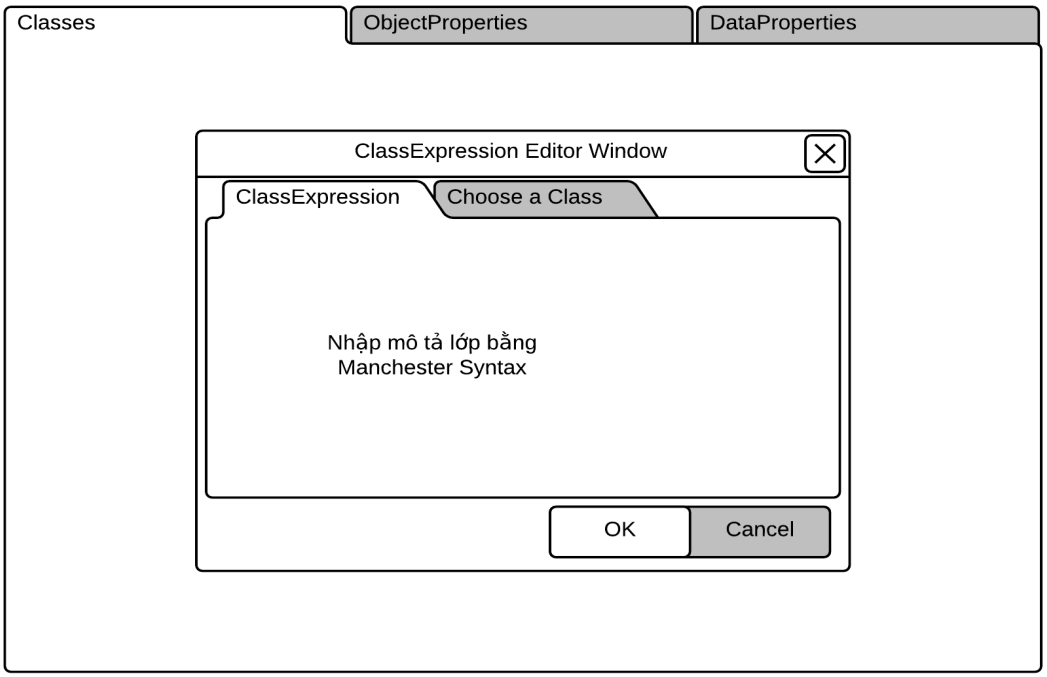


Hình 3.5: Phác thảo Main View - Tab "SWRL Rules"

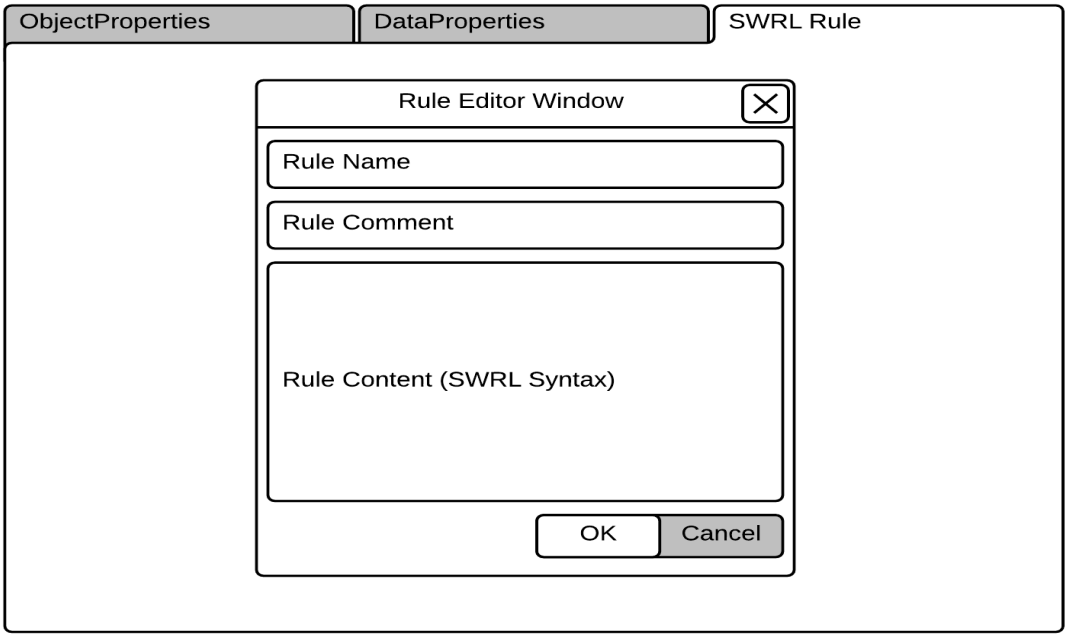
là một bảng gồm chứa các rule, chia thành 3 cột gồm tên, nội dung và lời chú thích cho rule (Hình 3.5). Ngoài các view và tab đã kể ở trên, thì ứng dụng buộc phải có thêm các thành phần giao diện hỗ trợ viêc biên tập mô tả lớp, thuộc tính, cá thể (class/property expression, individual assertion axiom) và việc biên tập SWRL Rule. Các hình 3.6, 3.7 thể hiện phác thảo về các giao diện đảm nhiệm những tính năng vừa nêu.

3.4 Truy xuất đến các thành phần của Ontology trong OWL-API bằng cách áp dụng Visitor Pattern

Trong OWL-API, các đối tượng dữ liệu có cấu trúc tương tự với các thành phần trong OWL 2 Ontology (được trình bày trong mục 2.2.3) - tên của các đối tượng dữ liệu sẽ có thêm tiếp đầu ngữ *"OWL"* cộng với tên của thành phần trong OWL 2 như *Ontology* thì đối tượng trong OWL-API sẽ là *OWLOntology*, *ClassExpression* (mô tả lớp) thì đối tượng tương ứng là *OWLClassExpression*, tương tự cho các thành phần khác. Việc tương tác và



Hình 3.6: Phác thảo cửa sổ biên tập mô tả lớp

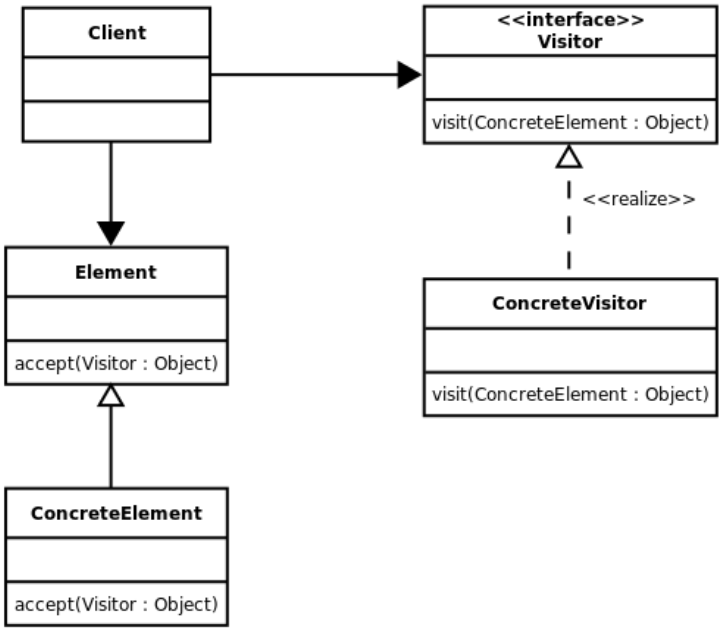


Hình 3.7: Phác thảo cửa sổ biên tập rule

thay đổi sẽ diễn ra trong đối tượng *OWLOntology* - đối tượng Java biểu diễn một OWL 2 ontology.

Như đã được giới thiệu trong mục 2.2.3, với một cấu trúc phức tạp gồm nhiều đối tượng được mở rộng và thừa kế từ các loại đối tượng khác, thì việc truy xuất đến từng thành phần cụ thể và áp dụng các thay đổi khác nhau lên chúng là một tác vụ khó nếu không áp dụng Visitor Pattern.

3.4.1 Visitor Pattern

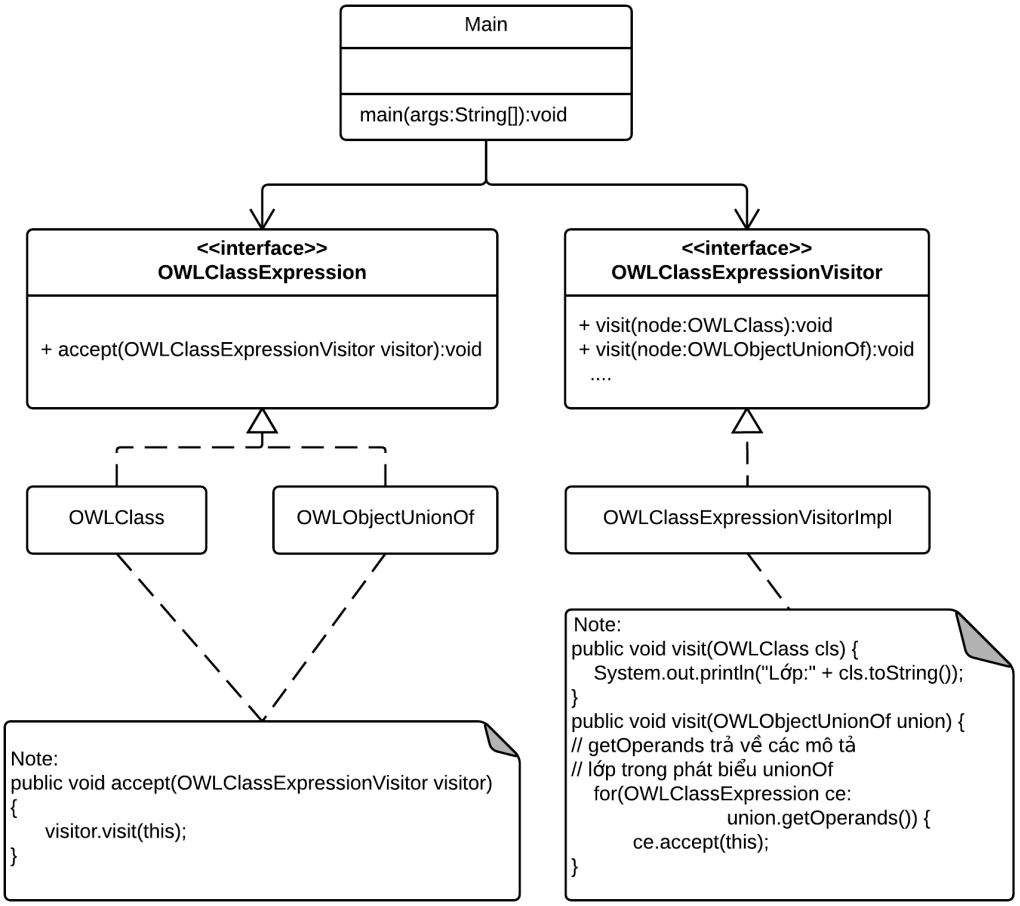


Hình 3.8: Visitor Design Pattern

Visitor Pattern là một design pattern trong lập trình hướng đối tượng với mục tiêu tách biệt các cấu trúc dữ liệu khỏi giải thuật. Nói cách khác chúng ta dễ dàng thay đổi các giải thuật mong muốn lên các thành phần của OWLOntology mà không cần phải thay đổi cấu trúc của chúng. Như biểu diễn trong hình mọi loại đối tượng kế thừa/mở rộng từ IElement đều có khả năng truy xuất bởi các đối tượng áp dụng Interface Visitor, các giải thuật mong muốn sẽ được định nghĩa trong phương thức *visit*.

3.4.2 Visitor trong OWL-API

Trong OWL-API cung cấp sẵn một số các Interface Visitor cho từng loại thành phần cụ thể. Ví dụ như OWLClassExpression sẽ có OWLClassExpressionVisitor, hay OWL-Datatype sẽ có OWLDatatypeVisitor, .v.v.. . Hình sau đây miêu tả cách hoạt động của OWLClassExpressionVistor. Giải sử chúng ta có một mô tả lớp (class expression) bằng



Hình 3.9: Class Diagram của OWLClassExpressionVisitor

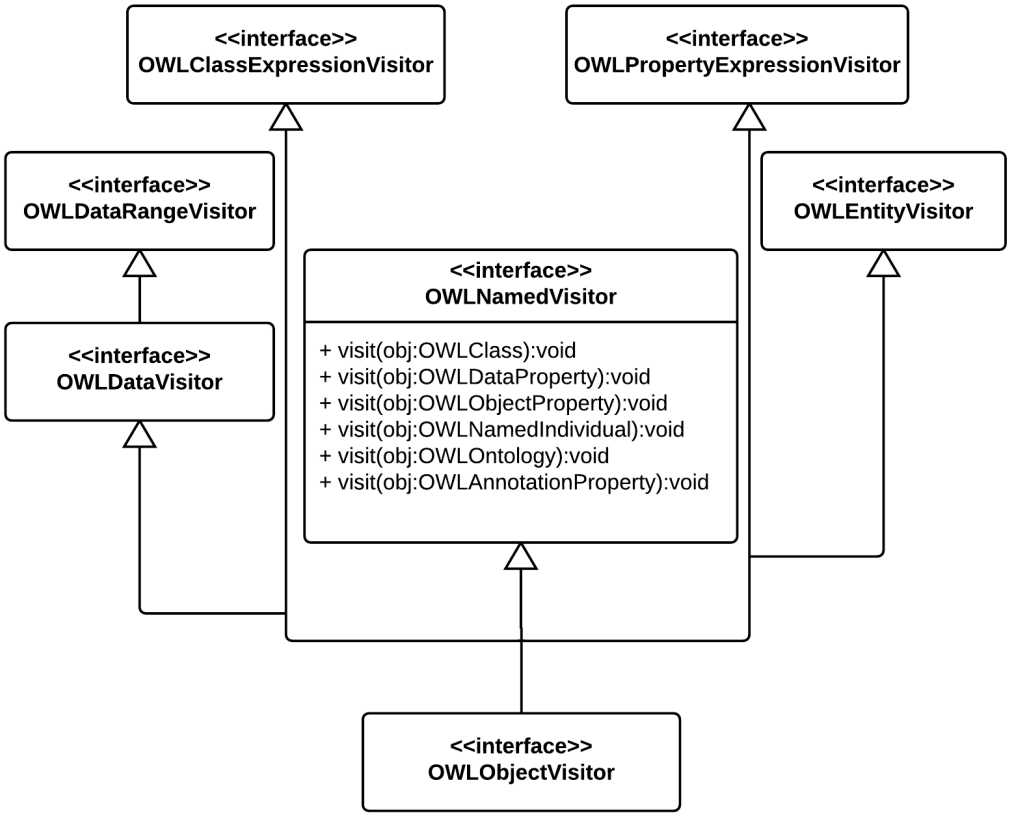
Manchester Syntax *"Car and Bike"* và hàm main trong hình khai báo như sau:

```
// void main
OWLDataFactory factory = OWLManager.getOWLDataFactory();
```

* *OWLDataFactory: nằm trong OWL-API với chức năng dùng như một nhà máy tạo ra mọi loại đối tượng từ lớp, thuộc tính tới các phát biểu.*

```
OWLClass car = factory.getOWLClass("a:Car");
OWLClass bike = factory.getOWLClass("a:Bike");
OWLObjectUnionOf union = factory.getOWLObjectUnionOf(car, bike);
OWLClassExpressionVisitor visitor = new OWLClassExpressionVisitorImpl();
union.accept(visitor);
// Ta sẽ có output là
Lớp: a:Car
Lớp: a:Bike
```

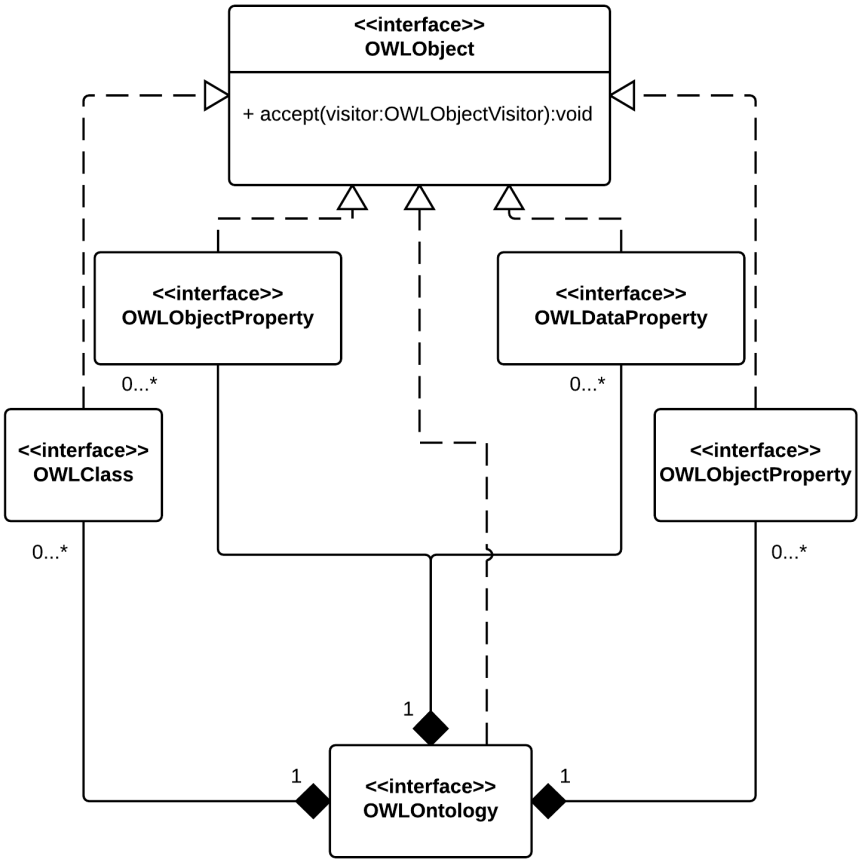
Visitor lớn nhất trong OWL-API là *OWLObjectVisitor* có khả năng truy vấn đến tất cả các loại đối tượng thuộc *OWLOntology*. Visitor này thực ra được tập hợp lại từ nhiều visitor nhỏ hơn (Hình 3.10). Nhờ tận dụng tính đa hình trong lập trình hướng đối tượng



Hình 3.10: Class Diagram của OWLObjectVisitor

nên các loại đối tượng thừa kế/mở rộng từ OWLObject đều có khả năng chấp nhận truy

vấn lên chính nó (visitor.visit(this)) từ *OWLObjectVisitor* (Hình 3.11). Để sử dụng các



Hình 3.11: Class Diagram của OWLObject và OWLObjectVisitor

visitor, chúng em chỉ cần tạo các class áp dụng các interface tương ứng nhằm truy vấn dạng thành phần cụ thế như ví dụ OWLClassExpressionVisitor ở trên. Tuy nhiên, mỗi lần sử dụng là mỗi lần phải viết lại nội dung các phương thức trong Interface Visitor, để hạn chế sự lặp lại này OWL-API cũng cung cấp sẵn hoặc chúng ta cũng có thể tạo các class VisitorAdapter với nội dung mỗi phương thức được cài đặt tính năng mặc định. Ví dụ:

```
ClassExpressionVisitorAdapter implements OWLClassExpressionVistor {
  private void doNothing(OWLClassExpression ce) {};
  public void visit(OWLClass cls) {doNothing(cls);}
  public void visit(OWLObjectUnionOf union) {doNothing(cls);} ... }
```

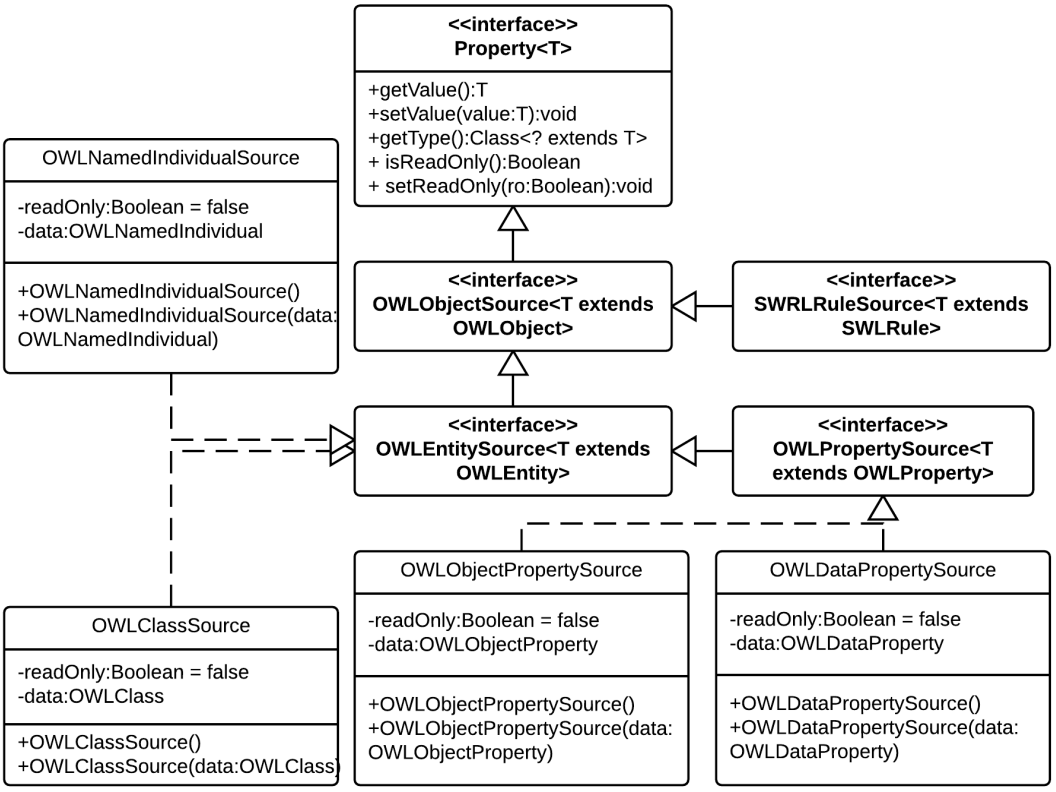
Mỗi khi muốn thay đổi nội dung phương thức nào chỉ cần *Override* lại phương thức đó trong VisitorAdapter là được.

3.5 Các tổ chức dữ liệu (data model) cho hệ thống

Trong Vaadin Framework, dữ liệu trong lúc một ứng dụng đang ở trong tiến trình hoạt động được tổ chức thành các mô hình gọi là Data Model, với 3 cấp khác nhau gồm:Property, Item và Container. Có thể xem các *Data Model* này như một đối tượng bao bọc (encapsulation) lại dữ liệu kiểu dữ liệu mà chúng ta muốn định nghĩa. Nếu câu hỏi là tại sao phải tốn công đưa các dữ liệu vào trong các đối tượng chứa như vậy, thì câu trả lời là vì Vaadin Framework cung cấp một cơ chế mà khi chúng ta tương tác với dữ liệu trong các *Data Model* này thì đồng thời những thành phần giao diện (UI Components) cũng sẽ cập nhật các thay đổi đó. Lý do là vì hầu hết các thành phần giao diện đều được thiết kế độc lập với dữ liệu, dữ liệu được liên kết với các giao diện thông qua các *Data Model* này. Sau đây chúng em xin trình bày cách tổ chức các đối tượng từ OWL-API vào trong các *Data Model* là Property và Container (Item không được sử dụng vì không phù hợp với cấu trúc dữ liệu mà chúng em muốn tổ chức).

3.5.1 Tổ chức các đối tượng OWL-API vào trong Property của Vaadin

Property là *Data Model* đơn giản nhất, được Vaadin cung cấp dưới dạng Interface *Property*<*T*> với T kiểu dữ liệu được chúng ta định nghĩa, các thành phần giao diện như *TextField*, *Label*, *TextArea* sử dụng *Property* để liên kết dữ liệu với các đối tượng dữ liệu được định nghĩa qua T. Công việc phải làm là định nghĩa các *Property* chứa các đối tượng OWL-API này. Do tính chất của OWL 2, ta có thế nói OWLClass (lớp) là một OWLEntity (thực thể), OWLEntity là một OWLObject (thành phần của OWL 2). Theo đó mà chúng em cũng sẽ tạo ra các Property tương ứng OWLObjectSource > OWLEntitySource > OWLClassSource. Việc tận dụng tính đa hình của đối tượng rất

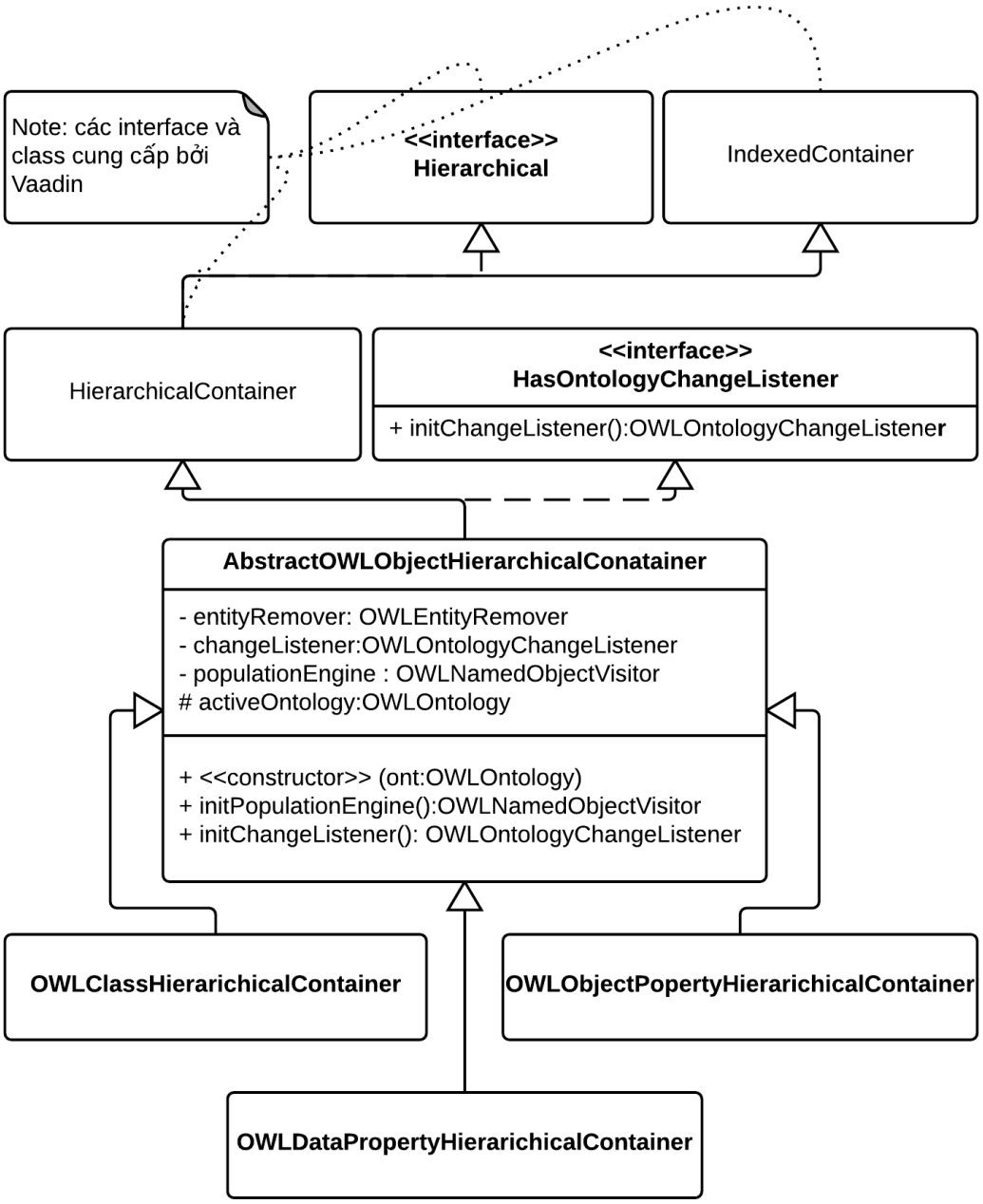


Hình 3.12: Class Diagram của Property<T>

có ích khi xây dựng các UI Component chung cho các loại đối tượng giống nhau như OWLClass, OWLObjectProprerty, OWLDataProperty (cả đều mở rộng từ OWLEntity).

3.5.2 Tổ chức các thực thể OWLClass, OWLObjectProperty và OWLDataProperty thành các Container

Container là cấu trúc *Data Model* phức tạp và cao nhất, chúng được chia thành các loại khác nhau tương ứng với tính chất của cấu trúc dữ liệu như *HierarchicalContainer* dùng cho các cấu trúc dữ liệu phân cấp được hỗ trợ trong các thành phần giao diện (UI Component) như *Table*, *Tree*, *ComboBox* và *ListSelect*, đơn giản hơn có *IndexedContainer* dùng cho cấu trúc dữ liệu dạng danh sách được hỗ trợ bởi *ComboBox* và *ListSelect* và *JPAContainer* dùng cho các dữ liệu dạng SQL Table - hỗ trợ bở *Table*, *TreeTable*. Trong ứng dụng, chúng em chỉ sử dụng *HierarchicalContainer* để tổ chức các lớp, thuộc tính đối



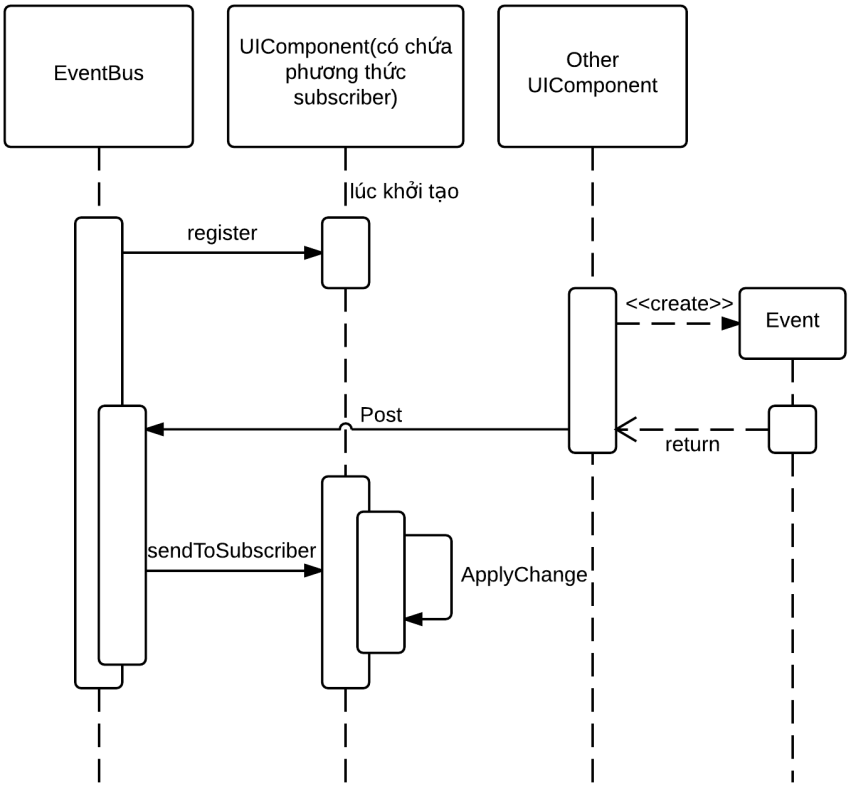
Hình 3.13: Class Diagram của HierarchicalContainer

tượng hay dữ liệu vì chúng cũng có tính chất phân cấp (SubClassOf, SubPropertyOf). Ở đây chúng em cũng xây dựng một AbstractClass rồi mở rộng ra với từng loại OWLEntity tương ứng là OWLClass, OWLObjectProperty và OWLDataProperty (Hình 3.13).

3.6 Thiết kế cách xử lý sự kiện

3.6.1 Sử dụng Guava EventBus

Do hệ thống gồm nhiều tab và các cửa sổ nhỏ (mô tả trong mục phác thảo giao diện), nên chúng em sẽ không áp dụng cách xử lý sự kiện truyền thống trong Java cho toàn hệ thống. Thay vào đó chúng em sử dụng một giải pháp thay thế là thư viện Guava EventBus - một nhánh nhỏ của thư viện Guava phát triển bởi Google [29]. Với Guava EventBus, mọi thành phần giao diện sẽ đăng ký các phương thức Subscriber - mỗi subscriber sẽ xử lý cho một loại sự kiện cụ thể - với EventBus trong lúc khởi tạo hoặc hậu khởi tạo, sau đó EventBus lắng nghe mọi sự kiện được công bố lên eventBus bằng phương thức *post*, cuối cùng EventBus bắt sự kiện và truyền sự kiện này đến đúng cho Subscriber nào đã đăng ký xử lý loại sự kiện này lúc đầu (Hình 3.14).



Hình 3.14: Sequence Diagram của EventBus

UIComponent có thể được khai báo subscriber bằng Annotation @Subscribe cung cấp bởi thư viện Guava như sau:

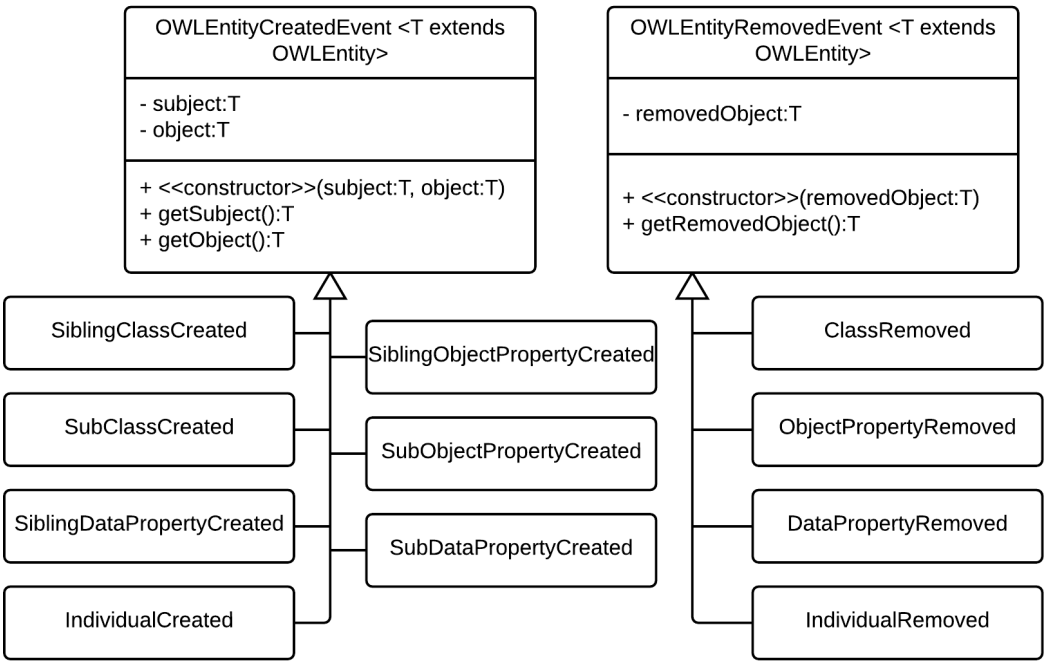
```
public class SubscriberComponent {
  public SubscriberComponent { EventBus.register(this); }
  @Subscribe public void eventHandler(SpecificEventType event) {
   // các thay đổi được áp dụng ở đây  }
}
```

EventBus có thể được sử dụng qua một lớp static với các phương thức register, post, unregister. Phương thức được chọn làm subscriber phải đáp ứng các yêu cầu sau đây:

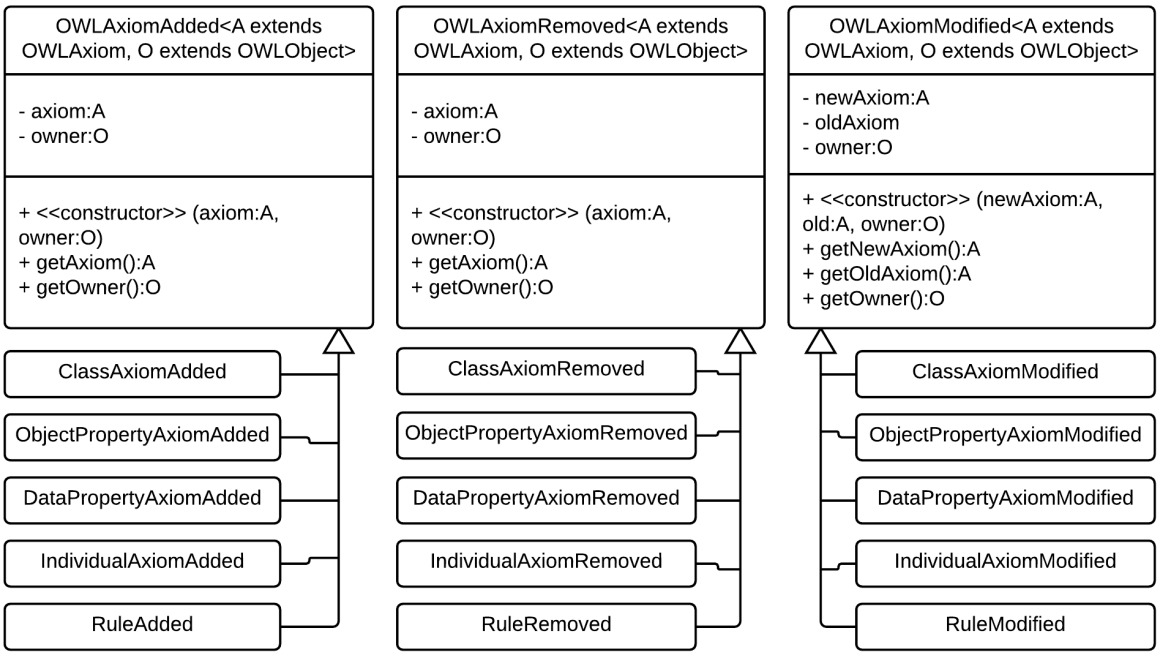
- Khai báo public
- Có duy nhất một tham số là sự kiện (event).
- Không có kiểu trả về (return void).

3.6.2 Các loại sự kiện trong hệ thống

Do EventBus sẽ dựa vào loại Event để truyền đến đúng Subscriber cụ thể. Vì thế các loại sự kiện trong hệ thống sẽ được chúng em phân thành 3 loại dạng là thêm (add/create), xóa(remove), sửa (edit/modify) và phân thành 2 loại lớn là các sự kiện liên quan tới các thực thể và các sự kiện liên quan đến các phát biểu. Tên của các sự kiện cho cá thể gồm phần đầu là tên loại thực thể (Class, ObjectProperty, ...) và phần sau là động từ nói lên đó là sự kiện thêm, xóa hay sửa (Hình 3.15). Đối với sự kiện dành cho các phát biểu, tên của chúng gồm phần đầu nói lên loại phát biểu mà sự kiện đó đang xử lý và phần sau là động từ nói lên hành động thêm, xóa hay sửa (Hình 3.16).



Hình 3.15: Class Diagram của các sự kiện liên quan đến thực thể



Hình 3.16: Class Diagram của các sự kiện liên quan đến các phát biểu

3.7 OWLEditorKit

Interface: `vn.edu.uit.owleditor.OWLEditorKit`

Class implementation: `vn.edu.uit.owleditor.OWLEditorKitImpl`

Đây là thành quan trọng nhất của toàn bộ hệ thống được thiết kế để đảm nhiệm việc khai thác các API từ OWL-API, nạp/tạo Ontology từ IRI, suy luận, giải thích các phát biểu, parse các chuỗi viết theo cú pháp *Manchester* thành các mô tả lớp và dữ liệu. Thực ra, các chứng năng vừa kể trên hầu hết đều được thực thi nhờ các API của OWL-API, *OWLEditorKit* đóng gói tất cả lại thành một đối tượng để dễ dàng sử dụng trong các UI Component hơn, cũng như tránh việc phải khởi tạo nhiều lần các API.

3.7.1 Các chức năng của OWLEditorKit

3.7.1.1 Nạp Ontology

Tương ứng với hàm *loadOntologyFromOntologyDocument*, load các tài liệu OWL2 từ tham số là đối tượng *IRI* của OWL-API, ontology khi load xong sẽ được gán cho đối tượng Active Ontology (sẽ được đề cập ngay sau).

```
OWLEditor kit = new OWLEditorKitImpl();
kit.loadOntologyFromOntologyDocument(IRI.create("some url"));
```

3.7.1.2 Giải thích các phát biểu trong OWL2 Ontology

Việc này được thực hiện thông qua phương thức *explain(OWLAxiom axiom)* của OWLEditorKit. Kết quả trả về là một đối tượng *ExplanationTree*trong đó gồm các OWLAxiom (là các phát biểu giải thích). Cách sử dụng chi tiết hơn trong chương sau.

3.7.1.3 Xóa các thực thể trong OWL2 Ontology

Xóa các phát biểu trong OWL 2 Ontology không phải là một tác vụ dễ dàng vì một phát biểu có thể được sử dụng để xây dựng nên phát biểu khác. Ví dụ:

```
Declaration( Class( A ))
SubClassOf(A B)
EquivalentClasses(A C D)
```

Giả sử chúng ta muốn xóa phát biểu *Declaration(Class(A))* thì **bắt buộc** chúng ta phải xóa luôn những phát biểu có liên quan đến A là *SubClassOf và EquivalentClasses* bởi vì nếu không có phát biểu khẳng định sự tồn tại của A thì những phát biểu còn lại sẽ trở nên vô nghĩa. Tuy nhiên để thực hiện tác vụ này OWL-API cung đối tượng là *OWLEntityRemover* với tính năng duyệt qua mọi phát biểu có liên quan đến thực thể cần xóa (thực ra OWLEntityRemover này cũng là một OWLObjectVisitor), xóa tất cả các phát biểu đó trước khi xóa thực thể. OWLEntityRemover được sử dụng thông phương thức *getEntityRemover()* của OWLEditorKit.

```
OWLClass person = ... ;
person.accept(editorKit.getEntityRemover()); // Xóa lớp person
editorKit.getEntityRemover().reset(); // Cập nhật lại thông tin
```

3.7.1.4 Tạo ra các thành phần OWL 2 bằng OWLDataFactory

Một chức năng thực sự hữu ích cung cấp bởi OWL-API, với số lượng đối tượng (thực thể, mô tả lớp, thuộc tính, phát biểu, ...) rất lớn đã đề cập ở chương 2, rất khó để chúng ta có thể nhớ và sử dụng chúng dễ dàng. *OWLDataFactory* hoạt động như một nhà máy tạo ra hầu hết các loại phát biểu, thực thể, kiểu dữ liệu trong OWL 2 Ontology một cách tiện lợi. Ví dụ như:

```
// Declaration( NamedIndividual(a:Peter) )
```

```
OWLNamedIndividual Peter = factory.getOWLNameIndividual("a:Peter");
// Declaration( DataProperty(a:HasAge) )
OWLDataProperty hasAge = factory.getOWLDataProperty("a:HasAge");
// ClassAssertion( a:Person a:Peter )
OWLLiteral age = factory.getOWLLiteral(22);
```

Trong hệ thống, OWLDataFactory được sử dụng qua phương thức *getOWLDataFactory()* của OWLEditorKit.

3.7.1.5 Tạo ra các Data Model và Event bằng EditorDataFactory

Học tập mô hình *Factory Pattern* * từ *OWLDataFactory*, chúng em cũng tự xây dựng một factory để tạo ra các đối tượng dữ liệu và các event trong ứng dụng. Ví dụ:

```
OWLClass person, man; OWLEditorKit kit = ...;
OWLEditorFactory fc = new OWLEditorFactoryImpl(kit);
ClassAxiomAdded event = fc.getSubClassOfAddEvent(man, person);
OWLClassHierarchicalContainer con =
                fc.getClassHierarchicalContainer(kit.getActiveOntology)
```

3.7.1.6 Parse chuỗi thành các mô tả lớp (OWLClassExpression)

OWLEditorKit cũng được tích hợp một đối tượng *ManchesterOWLSyntaxParser* trong OWL-API, nhiệm vụ là để parse các đoạn text từ người dùng theo cú pháp *Manchester* thành các mô tả lớp (OWLClassExpression) hay các miền dữ liệu (OWLDataRange). Parser này được sử dụng thông qua getter *getParser* hoặc parse trực tiếp qua hàm *parseClassExpression(String stringToParse)* . Ví dụ:

```
OWLEditorKit eKit = new OWLEditorKitImpl(); // nạp ontology ...
OWLClassExpression ce2 = eKit.parseClassExpression("Has some Kid");
```

*: *http://en.wikipedia.org/wiki/Factory_method_pattern*

3.7.1.7 Suy luận các phát biểu

PelletReasoner cũng được chứa trong OWLEditorKit, được sử dụng qua phương thức *getReasoner()*. Cách sử dụng:

```
OWLNamedIndividual peter = ...;
// Tìm xem Peter thuộc những lớp nào
editorKit.getReasoner().getTypes(peter,true).getFlattened();
OWLClass driver = ...;
// Tìm lớp con
editorKit.getResoner().getSubClasses(driver,true).getFlattened();
```

True nghĩa là chỉ tìm lớp gần nhất cho việc giải thích. Giả sử cá thể Peter thuộc các lớp driver mà Driver là lớp con của Person. Nếu để *true* thì kết quả chỉ có Driver và ngược lại thì kết quả gồm Driver và Person. Tương tự cho SubClass, SubProperty, ...

3.7.1.8 Quản lý các Ontologies được nạp và áp dụng các thay đổi

Việc nạp ontology, thực sự diễn ra bên trong lớp OWLEditorKitImpl và được thực hiện bởi *OWLOntologyManager* - api của OWL-API. Ontology nạp vào sẽ được quản lý theo OntologyIRI và VersionIRI (xem ở chương 2).

```
OWLEditorKitImpl implements OWLEditorKit { ...
  OWLOntologyManager manager = new OWLManager.getOWLOntologyManager();
  public void loadOntologyFromOntologyDocument(IRI iri) {
   manager.loadOntologyFromOntologyDocument(iri);
  }
}
```

Ngoài chức năng trên, OWLOntologyManager còn được sử dụng để áp dụng các thay đổi lên Ontology, các thao tác chỉnh sửa sẽ vô nghĩa nếu kết quả của chúng không được áp

dụng lên Ontology. Chúng ta sử dụng OWLOntologyManager cho tác vụ này qua phương thức *getModelManager()* như sau:

```
OWLEditorKit kit = ...; fc = kit.getOWLDataFactory();
OWLClass vehicle = fc.getOWLClass("a:Vehicle");
OWLAxiom clsAxiom = fc.getOWLClassDeclarationAxiom(vehicle);
kit.getModelManager()
   .applyChange(new AddAxiom(clsAxiom, kit.getActiveOntology());
```

3.7.1.9 Quản lý, thao tác với các SWRL Rule

Trong cài đặt của OWLEditorKit, cũng bao gồm SWRLAPIOntology (từ SWRLAPI [22]) dùng quản lý và thêm/xóa/sửa SWRL Rule. Đối tượng này được sử dụng qua phương thức *getSWRLActiveOntology()* của OWLEditorKit. Lưu ý: SWRLAPIRule (từ thư viện SWRL-API) được mở rộng từ SWRLRule (từ thư viện OWL-API).

```
SWRLAPIOntology ruleOntology = editorKit.getSWRLActiveOntology();
Set<SWRLAPIRule> allRules = ruleOntology.getSWRLAPIRules();
```

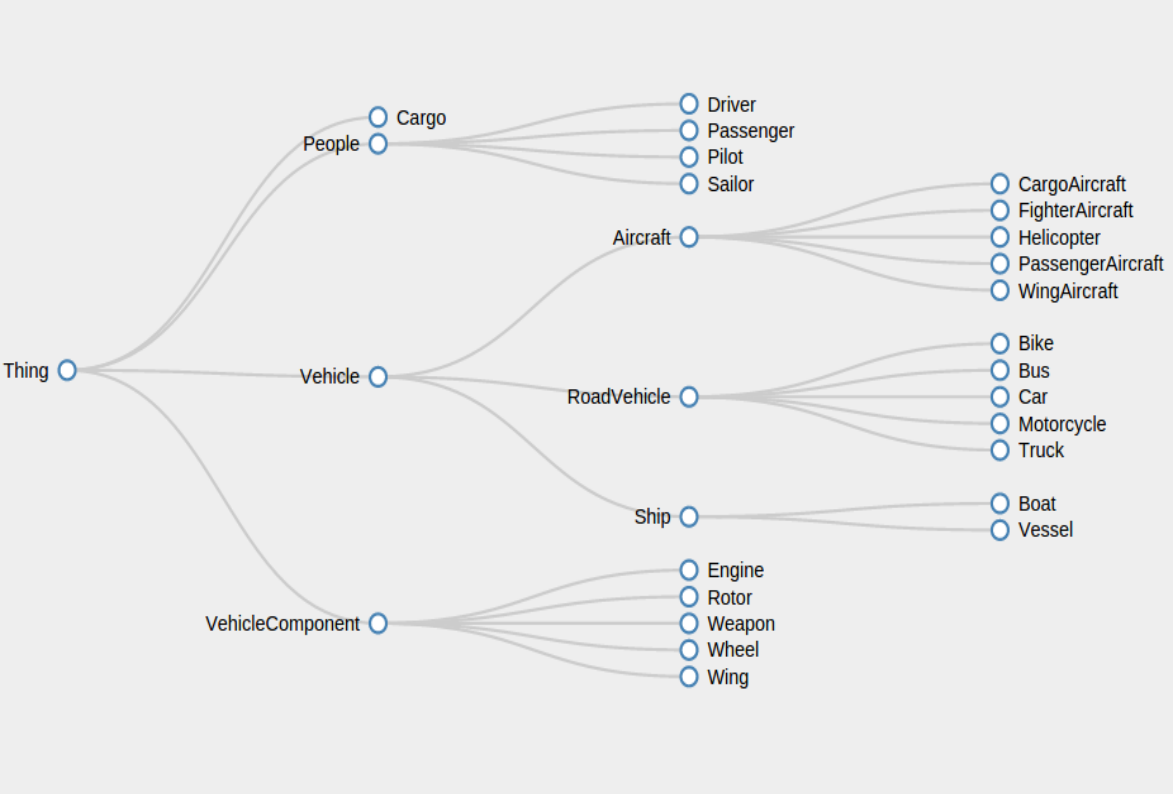
Tóm lại, mọi thao tác lên ontology từ bất kì thành phần giao diện nào đều phải thông qua sử dụng các phương thức mà OWLEditorKit cung cấp.

3.8 Xây dựng một ontology để trình bày quá trình phân loại

Giới thiệu Transport.owl [30] là một ontology do nhóm tự xây dựng nên, đối tượng chủ yếu là các phương tiện giao thông đường bộ, đường thuỷ, đường hàng không, bên cạnh đó là các bộ phận của phương tiện như động cơ, rô-tơ,... các đối tượng liên quan như hàng hoá, hành khách, bộ phận của phương tiện, ...Ontology được thiết kế nhằm biểu diễn

đặc tính phân loại của OWL 2 và SWRL, nên có một vài điểm không bám sát theo đúng với thực tế. Sau đây chúng em xin được miêu tả ontology này. Ontology được chia theo cấu trúc phân lớp, và bao gồm các phân lớp sau :

3.8.1 Lớp



Hình 3.17: Các lớp trong ontology transport.owl

Các lớp mở rộng ra là lớp con của lớp (node trong hình) trước nó.

3.8.2 Thuộc tính đối tượng

Trong bảng 3.1 thì *hasWing*, *hasEngine* và *hasRotor* là thuộc tính con của *hasParts*.

ObjectProperty	Transitive	Domain	Range	Inverse
canCarry		Vehicle	Cargo or People	canBeCarriedBy
isPartOf	x	Vehicle or VehicleComponent	Vehicle or VehicleComponent	hasParts
canBeCarriedBy		Cargo or People	Vehicle	canCarry
control		People	Vehicle	isControlledBy
hasParts	x	Vehicle or VehicleComponent	Vehicle or VehicleComponent	isPartOf
hasWing		WingAircraft	Wing	
hasEngine		Vehicle	VehicleComponent	
hasRotor		Helicopter	Rotor	
isControlledBy		Vehicle	People	control

Bảng 3.1: Bảng các thuộc tính đối tượng trong ontology transport

DataProperty	Functional	Domain	Range
canCarryNumberOfPassenger	x	Vehicle	positiveInteger
canCarryTheAmountOfCargo	x	Vehicle	positiveInteger
hasNumberOfSeats	x	Vehicle	positiveInteger
hasNumberOfWheels	x	RoadVehicle	positiveInteger
canMoveOnOrIn	x	Vehicle	{"Road" , "Sky" , "Water"}

Bảng 3.2: Bảng các thuộc tính dữ liệu trong ontology transport

3.8.3 Thuộc tính dữ liệu

3.8.4 SWRL Rule

Các rule trong ontology này gồm:

```
// Tàu chở hàng hóa trên 100 (đơn vị) là tàu chở hàng Vessel
Ship(?x) ^ swrlb:greaterThan(?y, 100)
         ^ canCarryTheAmountOfCargo(?x, ?y)  -> Vessel(?x)
// Xe cộ có số chỗ ngồi lớn hơn 20 là xe bus
RoadVehicle(?x) ^ hasNumberOfSeats(?x, ?s)
                ^ swrlb:greaterThan(?s, 20) -> Bus(?x)
// Máy bay mang vũ khí là máy bay chiến đấu
Aircraft(?x) ^ canCarry(?x, ?y) ^ Weapon(?y) -> FighterAircraft(?x)
```

```
// Xe cộ có số chỗ ngồi >= 4 và <=7 là xe hơi
RoadVehicle(?x) ^ swrlb:lessThanOrEqual(?s, 7)
^ swrlb:greaterThanOrEqual(?s, 4) ^ hasNumberOfSeats(?x, ?s) -> Car(?x)
```

3.8.5 Cá thể

Lưu ý: các cá thể này đều là các cá thể có tên (NamedIndividual) Ngoài ra còn có các cá

Named Individual	Types	Object propety assertions	Data property assertions
A1	Vehicle	*canCarry* Charlie	*isControlledBy* Anh
A2	Vehicle	*hasWing* XWing	
A3	Vehicle	*isControlledBy* Nguyen *hasParts* Rotor1	
A4	Vehicle	*isControlledBy* Nguyen *canCarry* Missile	
S1	Vehicle		*canMoveOnOrIn* "Water"
S2	Ship		*canCarryTheAmountOfCargo* 102
V1	Vehicle	*canCarry* Peter	*canMoveOnOrIn* "Road"
V1	Vehicle	*canCarry* Container	*canMoveOnOrIn* "Road"
V3	Vehicle		*hasNumberOfSeats* 6 *canMoveOnOrIn* "Road"
V4	RoadVehicle		*hasNumberOfSeats* 21

Bảng 3.3: Bảng các cá thể trong ontology transport

thể thuộc các lớp như sau:

- **Passenger:** Steve, Charlie, Peter, Anh, Nguyen
- **Pilot:** Anh, Nguyen
- **Weapon:** Missile, MachineGun
- **Wing:** XWing, YWing
- **Cargo:** Container, SmallCargo

Trên đây, chúng em đã liệt kê ra các thực thể được chúng em tạo ra trong transport.owl và đặc điểm của chúng. Trong chương sau, chúng em sẽ trình bày tính năng phân loại dựa trên thiết kế ontology này.

Tổng kết Chương này em đã trình bày qua chi tiết từng thiết kế từ giao diện phác thảo, cách tổ chức dữ liệu và xử lý sự kiện cho hệ thống và cuối cùng là thiết kế một ontology nhằm phục vụ cho quá trình phân loại khi hệ thống đã được xây dựng.

Chương 4

Xây dựng hệ thống phân loại tự động

Giới thiệu Qua các chương trước, chúng em đã trình bày các cơ sở lý thuyết gồm OWL 2, SWRL, đề ra các thiết kế cho hệ thống và thiết kế một ontology sử dụng cho việc phân loại. Trong chương này, chúng em sẽ trình bày lại quá trình xây dựng hệ thống phân loại dựa trên các thiết kế ở chương trước.

4.1 UI của ứng dụng - OWLEditorUI

4.1.1 Giới thiệu OWLEditorUI

Lớp này được mở rộng từ lớp **UI** của Vaadin - chức năng cơ bản của **UI** là nơi khởi tạo mọi thành phần giao diện và trình bày nó dưới dạng HTML trên web page, OWLEditorUI mở rộng các chức năng trên như sau:

- Chứa EntryView, MainView (các View này chứa tất cả thành phần giao diện của hệ thống)
- Cập nhật và cài đặt MainView khi ontology được nạp vào EntryView.
- Cập nhật trạng thái khi người dùng refresh trang.

- Nạp (inject) OWLEditorKit và cung cấp nó qua phương thức tĩnh (static).
- Nạp EventBus và cung cấp qua các phương thức tĩnh.
- Nạp HttpSession và cung cấp qua phương thức tĩnh .

4.1.2 Chi tiết các chức năng của OWLEditorUI

Trước khi đi vào chi tiết các chức năng trên, chúng em muốn giới thiệu khái quát về cách thức hoạt động của Spring Boot [26], các Annotation đáng chú ý được sử dụng trong việc xây dựng hệ thống.

Spring Boot : Khi chương trình khởi động - nghĩa là khi chương trình được nạp vào một hệ thống Servlet như Tomcat, Jetty thì điểm xuất phát đầu tiên của chương trình sẽ từ đối tượng sau:

```
@SpringBootApplication public class OWLEditorApplication {
    public static void main(String[] args) {
       SpringApplication.run(OWLEditorApplication.class, args); }
}
```

@SpringBootApplication là một tên thay thế cho 3 annotation gồm **@Configuration**, **@EnableAutoConfiguration** và **@ComponentScan** (sẽ được giới thiệu ở dưới), mục tiêu của đối tượng này là khởi động ứng dụng, duyệt hết trong package hiện hành tới package con để tìm các Component Bean cần thiết để khởi tạo và thực thi các Bean đó. Cụ thể ở đây, nó sẽ quét vào tìm thấy **OWLEditorUI** được đánh dấu bằng **@VaadinUI** và chạy theo cấu hình có săn được định nghĩa trong Spring 4 Vaadin [27]. Từ đây mọi thứ sẽ diễn ra trong **OWLEditorUI**. Ý nghĩa của các Annotation:

- **@VaadinUI** Là annotation của Spring 4 Vaadin [27] khai báo một lớp là một Bean và là một Vaadin **UI** cho hệ thống Spring Boot.

- **@VaadinComponent** Là annotation của Spring 4 Vaadin, tương tự annotation @Component của Spring Framework: khai báo đây là một Bean cơ bản nhất cho hệ thống Spring Boot.
- **@Repository** Là annotation của Spring Framework, khai báo một đối tượng là một Bean với tính chất là nơi để lấy và khai thác dữ liệu, OWLEditorKit được khai báo bằng annotation này.
- **@Autowired** Là annotaion của Spring Framework, dùng nạp tự động các Bean đã được khai báo (bằng các annotation như @VaadinComponent, @Component, @Repository, ...) vào hệ thống, cụ thể chúng em dùng các đối tượng này để nạp OWLEditorKit, HttpSession và OWLEditorEventBus vào OWLEditorUI.
- **@Theme** Là annotation của Vaadin dùng để nạp thành phần tùy chỉnh CSS cho toàn bộ giao diện của hệ thống.
- **@EnableAutoConfiguration** Là annotation của Spring Boot, bật tính năng cấu hình tự động (mọi cấu hình về DataBase, Base Url,... sẽ ở trạng thái mặc định).
- **@ComponentScan** Là annotation của Spring, tự động quét và tìm các Bean từ package hiện hành và khởi tạo chúng.
- **@Nonnull** Là annotation từ JSR-305, đảm bảo tham số nhập vào không được null, nếu null tự động throw NullPointerException.

4.1.2.1 Sử dụng OWLEditorKit trong OWLEditorUI

Như đã nói sơ qua, thì **OWLEditorKit** được định nghĩa là một Bean của hệ thống nên khi khởi động nó sẽ được tạo ra nhờ tính năng duyệt Bean vừa giới thiệu. Trong OWLEditorUI, chúng ta sẽ nạp (inject) nó vào OWLEditorUI và tạo một phương thức tĩnh để có thể sử dụng từ các thành phần giao diện như sau:

```
@Autowired OWLEditorKit eKit;
public static OWLEditorKit getOWLEditorKit() {
```

```
  ((OWLEditorKit) UI.getCurrent()).eKit; }
```

Lưu ý: *UI.getCurrent()* là một hàm tiện ích của Vaadin nhằm trả về UI đang chạy hiện hành, trong hệ thống mà chúng em xây dựng chỉ có duy nhất một UI là OWLEditorUI nên sẽ ép kiểu trực tiếp, trong trường hợp có nhiều UI cần lưu ý vấn đề kiểm tra loại UI. Trong source code [30], chúng em chỉ khai báo Annotation @Repository cho lớp *OWLEditorKitImpl* vậy tại sao @Autowired lại biết mà khởi tạo được ? Câu trả lời là hệ thống sẽ quét và tìm đến các lớp áp dụng interface OWLEditorKit mà có Annotation khai báo để khởi tạo. Để đơn giản ở đây chúng em sẽ dùng hàm khởi tạo mặc định trong OWLEditorKitImpl, ngoài ra còn rất nhiều cách thức @Autowired (hoặc inject) khác có thể tham khảo ở [26].

4.1.2.2 Sử dụng OWLEditorEventBus trong OWLEditorUI

Trong thiết kế chúng em sẽ sử dụng *một* EventBus để xử lý sự kiện cho hệ thống, vì vậy sẽ dễ dàng hơn nếu nó được khởi tạo một lần trong OWLEditorUI và cung cấp dưới dạng phương thức tĩnh. Đầu tiên là cấu trúc của một lớp dùng chứa EventBus thật sự và cách tạo ra các hàm tĩnh.

```
@VaadinComponent public class OWLEditorEventBus {
  private final EventBus realEventBus = new EventBus(this); //từ Guava API
  public static void post(@Nonnull final Object event) {
    OWLEditorUI.getGuavaEventBus().realEventBus.post(event);  }
  public static void register(@Nonnull final Object obj) { ... } ...
}
// Trong OWLEditorUI
@Autowired OWLEditorEventBus eventBus;
public static OWLEditorEventBus getGuavaEventBus() {
  ((OWLEditorKit) UI.getCurrent()).eventBus; }
```

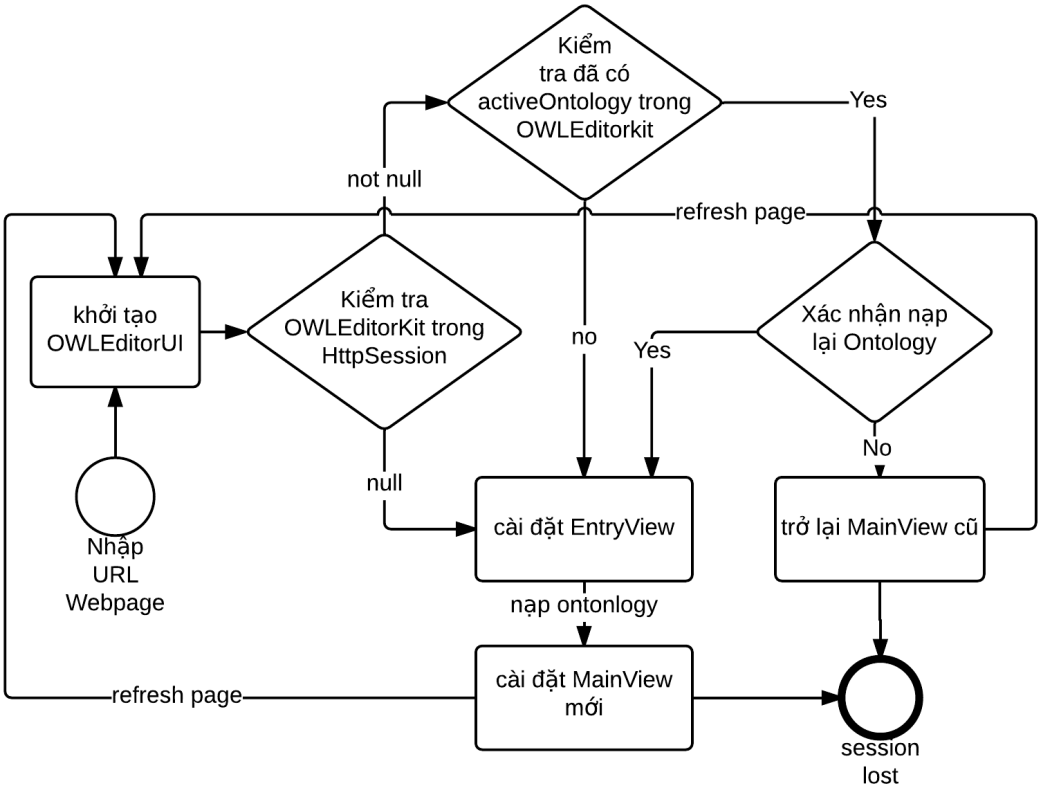
Vậy để sử dụng Guava EventBus trong OWLEditorUI, chỉ cần gọi *OWLEditorUI. getGuavaEventBus* hoặc đơn giản hơn *OWLEditorEventBus. <tên phương thức>*.

4.1.2.3 Sử dụng HttpSession trong OWLEditorUI

Tương tự 2 thành phần trên, HttpSession cũng là một Bean có sẵn của Spring nên chúng ta có thể nạp vào OWLEditorUI và sử dụng như sau:

```
@Autowired HttpSession httpSession;
public static HttpSession getHttpSession() {
 return ((OWLEditorUI) getCurrent()).httpSession; }
```

4.1.2.4 Cập nhật trạng thái giữa EntryView và MainView



Hình 4.1: Vòng đời của các view trong OWLEditorUI

Mỗi lần chúng ta vào trang url của ứng dụng hay refresh thì vòng đời của các View sẽ diễn ra như trong hình 4.1. Việc kiểm tra các điều kiện đều nằm trong phương thức **updateContent** của OWLEditorUI. **Tóm lại** OWLEditorUI là nơi sẽ cung cấp các thành phần lõi của hệ thống như OWLEditorKit, EventBus và HttpSession. Tiếp theo, chúng em sẽ trình bày cách xây dựng các View và thành phần của chúng.

4.2 Xây dựng EntryView

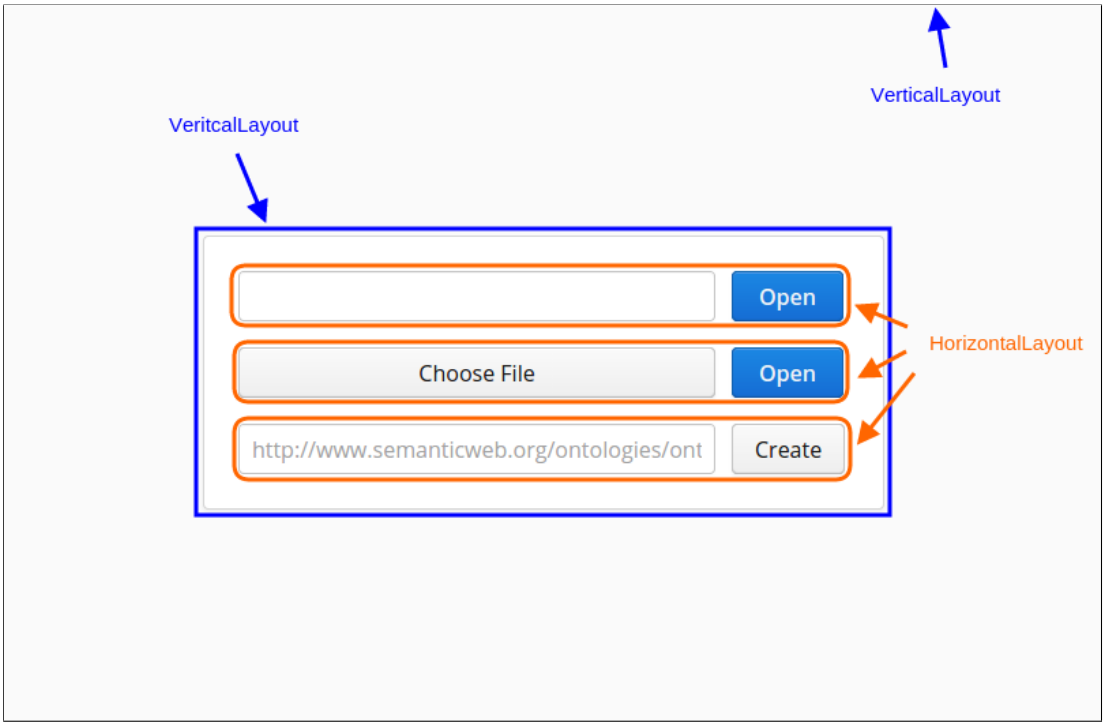
Như được thiết kế thì giao diện của EntryView tương đối đơn giản, chỉ có 1 panel chứa các input để nạp ontology theo 3 tùy chọn:

1. Nạp ontology qua URL. URL này bắt buộc phải là một tài liệu được định dạng theo tiêu chuẩn OWL 2 (RDF/XML, OWL/XML, FunctionalSyntax/XML,...).
2. Upload một tập tin chứa tài liệu Ontology và nạp tài liệu ontology này vào.
3. Tạo mới một tài liệu ontology.

Giao diện EntryView được mở rộng từ layout *VerticalLayout*[*], để chứa panel ở giữa - panel này cũng là một *VerticalLayout* tập hợp mỗi dòng là một *HorizontalLayout*, mỗi *HorizontalLayout* này sẽ chứa một *TextField* và một *Button* hoặc một *UploadField* và một *Button* (Hình 4.2) - tất cả các thành phần vừa nêu đều được cung cấp bởi Vaadin. Nhờ vậy, việc xây dựng giao diện khá dễ dàng, chẳng hạn để đặt vị trí trung tâm cho panel trong hình chỉ cần sử dụng dòng code sau :

```
setComponentAlignment(entriesPanel, Alignment.MIDDLE_CENTER);
```

1. *HorizontalLayout* các UI Component được sắp xếp theo chiều ngang, từ trái qua phải.
2. *VerticalLayout* các UI Component được sắp xếp theo chiều dọc, từ trên xuống.
3. *AbsoluteLayout* các UI Component được sắp xếp theo đúng vị trí tọa độ được khai báo.
4. *CssLayout* các UI Component được sắp xếp theo định nghĩa từ các file css tương ứng.



Hình 4.2: EntryView đã xây dựng

Sau khi click "Open", nếu URL chứa tài liệu hợp lệ thì nó sẽ được nạp vào trong OWLEditorKit, nếu đây là lần đầu OWLEditorKit được sử dụng và chưa được lưu trong HttpSession thì lưu nó vào Session và cuối cùng là chuyển giao diện qua *MainView* (Hình 4.1). Quá trình tương tự cũng xảy ra khi nạp tài liệu từ tập tin upload và tạo mới tài liệu, riêng việc tạo tài liệu thì chúng ta có thể tùy chọn đặt 1 IRI cho nó trong TextField ở dòng cuối. Upload file cũng là một plugin được cung cấp qua [24], nên việc sử dụng cũng rất đơn giản.

```
UploadField uf = new UploadField();
// code bên trong "Open" Button ClickListener
File file = (File) uf.getValue();
OWLEditorUI.getEditorKit() // Nạp ontology vào trong OWLEditorKit
           .loadOntologyFromOntologyDocument(IRI.create(file));
OWLEditorUI.getHttpSession() // lưu OWLEditorKit trong Session
           .setAttribute("OWLEditorKit", OWLEditorUI.getEditorKit());
// Đặt giao diện là MainView
```

```
UI.getCurrent().setContent(new MainView());
```

4.3 Xây dựng MainView

Trong thiết kế hệ thống, *MainView* sẽ là nơi chứa tất cả các Tab của ứng dụng, các Tab này được xây dựng dựa trên thành phần *TabSheet* của Vaadin, mỗi Tab đều có thể là bất cứ thành phần giao diện nào của Vaadin (UIComponent và Layout của Vaadin đều có chung Interface *Component*). Do vậy, nội dung của MainView khá đơn giản:

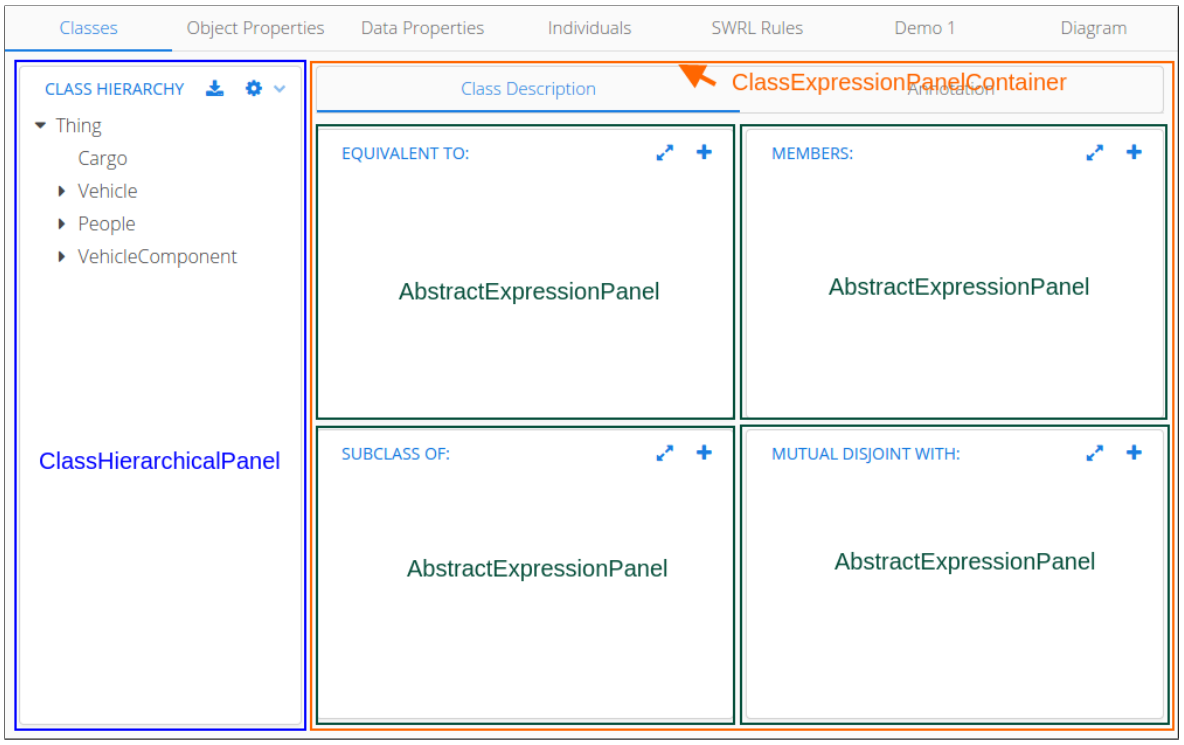
```
public class MainView extends HorizontalLayout {
  final TabSheet root = new TabSheet();
  public MainView() {
    root.addTab(new ClassesSheet(), "Classes");
    addComponent(root); // Thêm TabSheet vào layout của MainView;
    ...}
}
```

4.3.1 Xây dựng Tab Sheet để mô tả lớp, thuộc tính đối tượng/dữ liệu và cá thể

Do bố cục phần lớn các thành phần trong các Tab này cũng tương đối giống nhau, nên chúng em sẽ dùng Tab Classes để trình bày chi tiết, với những Tab còn lại chúng đều có cách xây dựng tương tự.

4.3.1.1 Sheet

Đây là lớp chúng em dùng để tổ chức các thành phần giao diện dành để tương tác với các lớp trong OWL 2 Ontology. ClassSheet cũng là một Layout gồm 2 thành phần con



Hình 4.3: ClassSheet(Tab dành cho lớp) đã xây dựng

là ClassHierarchicalPanel và ClassExpressionPanelsContainer, đây là nơi bắt sự kiện lựa chọn lớp trên cấu trúc cây của ClassHierarchicalPanel để gán lớp này vào làm dữ liệu cho các panel con bên phải (Hình 4.3).

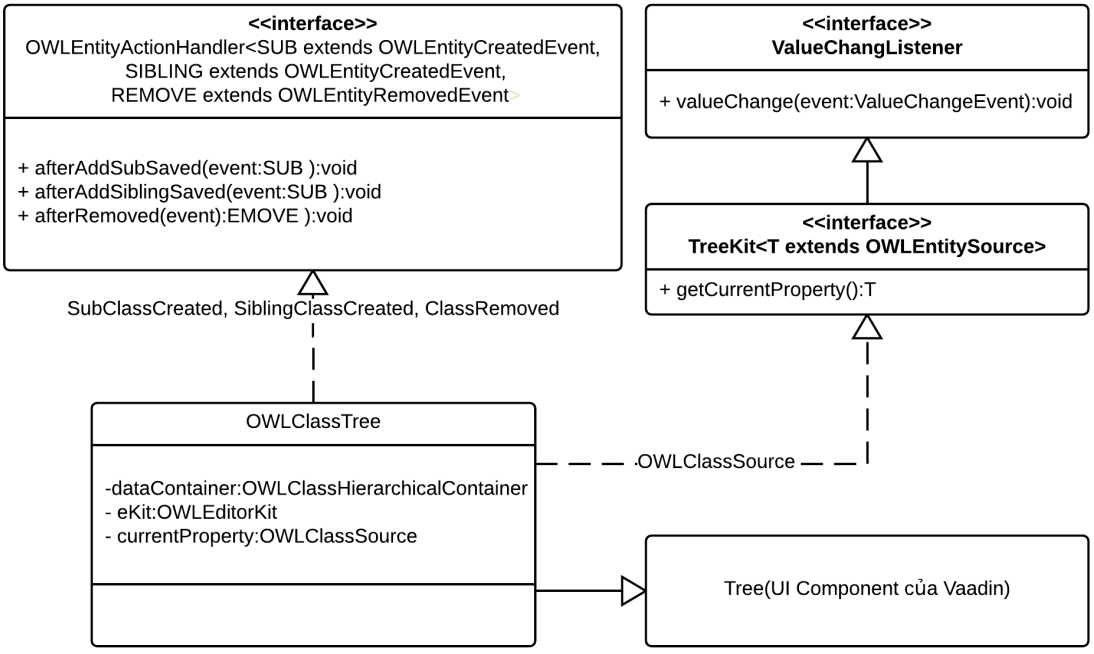
4.3.1.2 AbstractHierarchyPanel

Ở bên trái là một Panel gọi là *ClassHierarchicalPanel* (Hình 4.3) mở rộng từ *AbstractHierarchyPanel* * được chúng em xây dựng với mục tiêu là biểu diễn các lớp trong OWL 2 Ontology, và cho phép thực hiện các thao tác thêm lớp con, lớp cùng họ (sibling) và xóa lớp bằng cách click phải vào bất kì node nào trên cây. Ở góc phải bên trên của panel này, với kí hiệu bánh răng với chức năng thêm xóa tương tự, ngoài ra còn có tính năng bật reasoner cho hệ thống. Kế bên kí hiệu bánh răng, là một kí hiệu tải xuống, khi click sẽ lưu lại trạng thái ontology hiện hành và tải xuống dưới định dạng OWL/XML.

Bên trong *ClassHierarchicalPanel* có thành phần *Tree* đây chính là thành phần sẽ liên

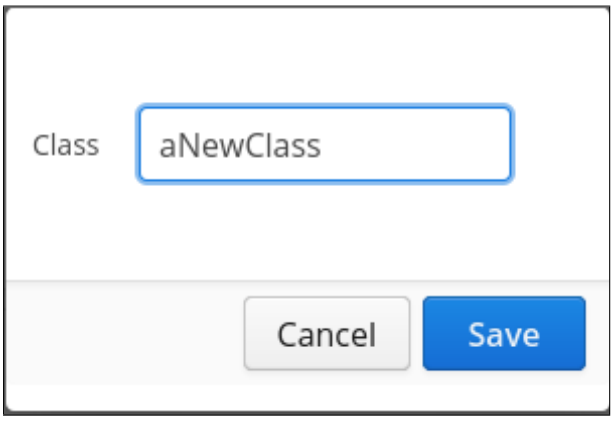
Lưu ý: *ClassHierarchicalPanel*, *ObjectPropertyHierachicalPanel* và *DataPropertyHierarchicalPanel* đều kế thừa từ *AbstractHierarchyPanel*.

kết với dữ liệu từ các *HiearchicalContainer* được giới thiệu trong mục tổ chức dữ liệu ở chương trước, ở đây cụ thể nó sẽ liên kết với các *ClassHierarchicalContainer* - chứa các lớp trong ontology với cấu trúc phân cấp. Nếu đúng như trong thiết kế thì chúng em sẽ sử dụng EventBus để xử lý các sự kiện thêm/xóa lớp ở đây, tuy nhiên sau khi xây dựng xong chúng em gặp phải vấn đề về con-current event - nghĩa là khi thực hiện thao tác thêm/xóa sẽ có 2 event giống nhau được tạo ra và truyền lên EventBus. Do vẫn chưa tìm được nguyên nhân chính xác, nên chúng em đã quay về với thiết kế truyền thống của Java là tạo một interface *OWLEntityActionHandler* vào áp dụng nó vào thành phần *Tree* (Hình 4.4). Thao tác thêm mới cửa sổ dùng để nhập tên lớp mới (Hình 4.5), cửa sổ này là lớp

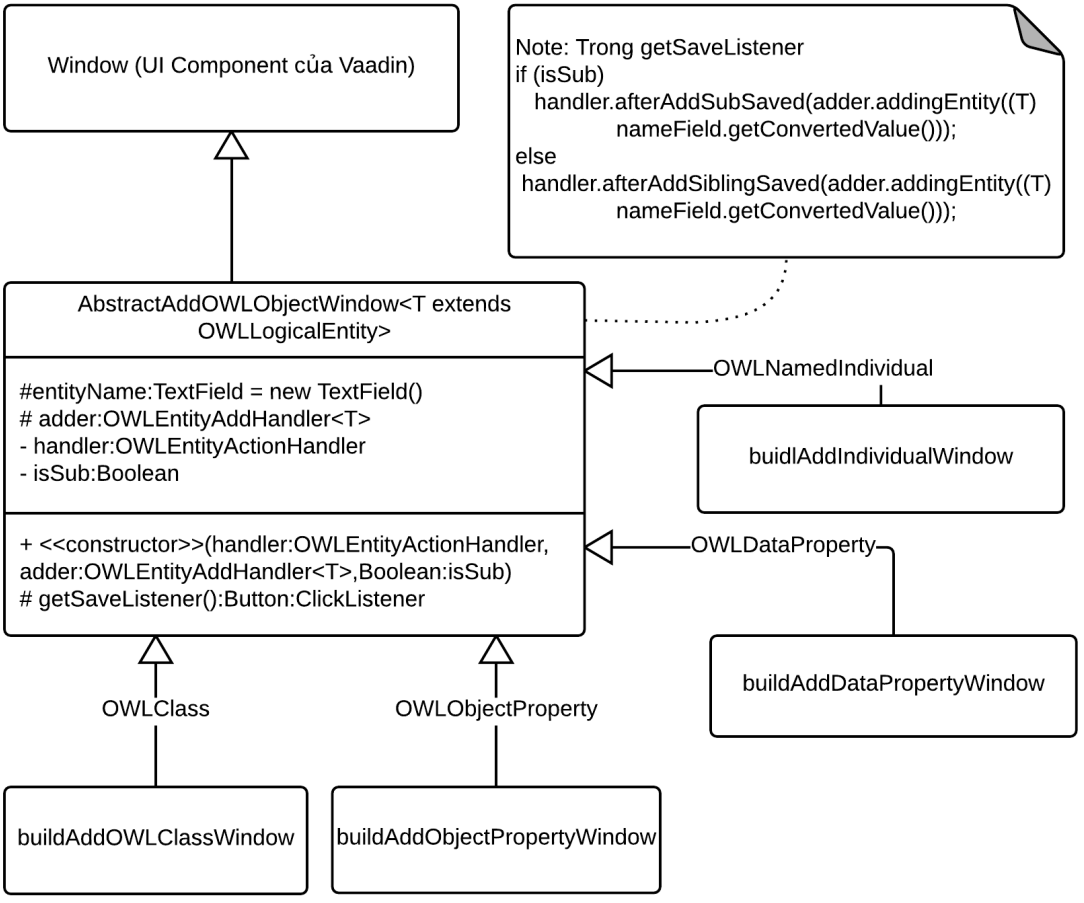


Hình 4.4: Class Diagram của OWLClassTree các interface của nó

buildAddOWLClassWindow mở rộng từ *AbstractAddOWLObjectWindow* (Hình 4.6). Đối tượng này phải biết OWLEntityActionHandler (là OWLClassTree hình 4.4) để khi chọn save nó sẽ sử dụng action handler này để áp dụng các thay đổi lên ontology và lên giao diện cây sẽ tự động thêm node con, hay node cùng họ (Sibling) - tương ứng với tùy chọn Sub Class hay Sibling Class. Chính vì thế nên hàm khởi tạo của lớp này sẽ buộc phải có tham số *OWLEntityActionHandler*, để sử dụng khi click save (phần note Hình 4.6). Ngoài



Hình 4.5: Cửa sổ thêm thực thể *lớp* trong ClassSheet



Hình 4.6: Class Diagram của AbstractAddOWLObjectWindow

interface *OWLEntityActionHandler*, thì trong hình chúng ta còn thấy một đối tượng là *OWLEntityAddHandler* (hàm khởi tạo và phần note hình 4.6). Đây là một Interface hỗ

trợ cho việc tạo ra các sự kiện liên quan thêm các thực thể *OWLEntityCreatedEvent* (giới thiệu trong chương trước). Vì trong lớp Abstract, chúng ta chưa biết phải trả về loại sự kiện tạo lớp hay thuộc tính hay cá thể, nên Interface này sẽ giúp chúng ta nạp vào loại sự kiện phù hợp. Ví dụ chúng ta muốn tạo một cửa sổ con để khai báo mới một thuộc tính dữ liệu con:

```
Window w = new AbstractAddOWLObjectWindo<OWLDataProperty>(
 actionHandler,
 newDataPropery -> new SiblingClassCreated(newDataPropery,
                   owlClassTree.getSelectedProperty),
 true);// True là tạo thực thể con
```

Ngoài ra chúng ta còn tận dụng được một đặc điểm mới của ngôn ngữ Java 8 là Lambda Expression (obj -> function). Rất nhiều Interface dạng này được tạo ra nhằm giảm thiểu việc phải code lại nhiều lần với cùng một mục đích là tạo ra các sự kiện. Đây cũng chính là lý do vì sao các sự kiện lại được thiết kế để khai thác tính đa hình của hướng đối tượng. Như đã trình bày trong chương 3, các thành phần mở rộng từ *Tree* của Vaadin có khả năng liên kết với các dữ liệu dạng phân cấp hay cụ thể chúng em sử dụng OWLClassHierachicalContainer cho OWLClassTree trong trường hợp này. Ngoài ra, OWLClassHierachicalContainer còn được sử dụng để lắng nghe mọi thay đổi bên trong ontology qua:

```
public OWLClassTree() {
  editorKit.getModelManager().addOntologyChangeListener(
  changes -> { for (OWLOntologyChange chg :changes) {
   chg.accept(dataContainer.getOWLOntologyChangeListener());}
});
```

Nhờ vậy, khi dữ liệu bên trong Container thay đổi, nó sẽ thông báo đến các thành phần giao diện mà nó liên kết để cập nhật thay đổi mà không code để thực hiện các thay đổi trên giao diện.

4.3.1.3 AbstractExpressionPanel

Nằm ở bên phải của Tab ClassSheet là một container chứa nhiều panel nhỏ khác (Hình 4.3), mỗi panel nhỏ này đóng vai trò là nơi tương tác với từng loai phát biểu liên quan đến lớp đang được chọn trên cấu trúc cây. Khi một lớp được chọn, trong ClassSheet sẽ gọi đến phương thức sau đây của *ClassExpressionPanelsContainer*.

```
setPropertyDataSource(Property newDataSource) {
  equivPanel.setPropertyDataSource(newDataSource);
  ...}
```

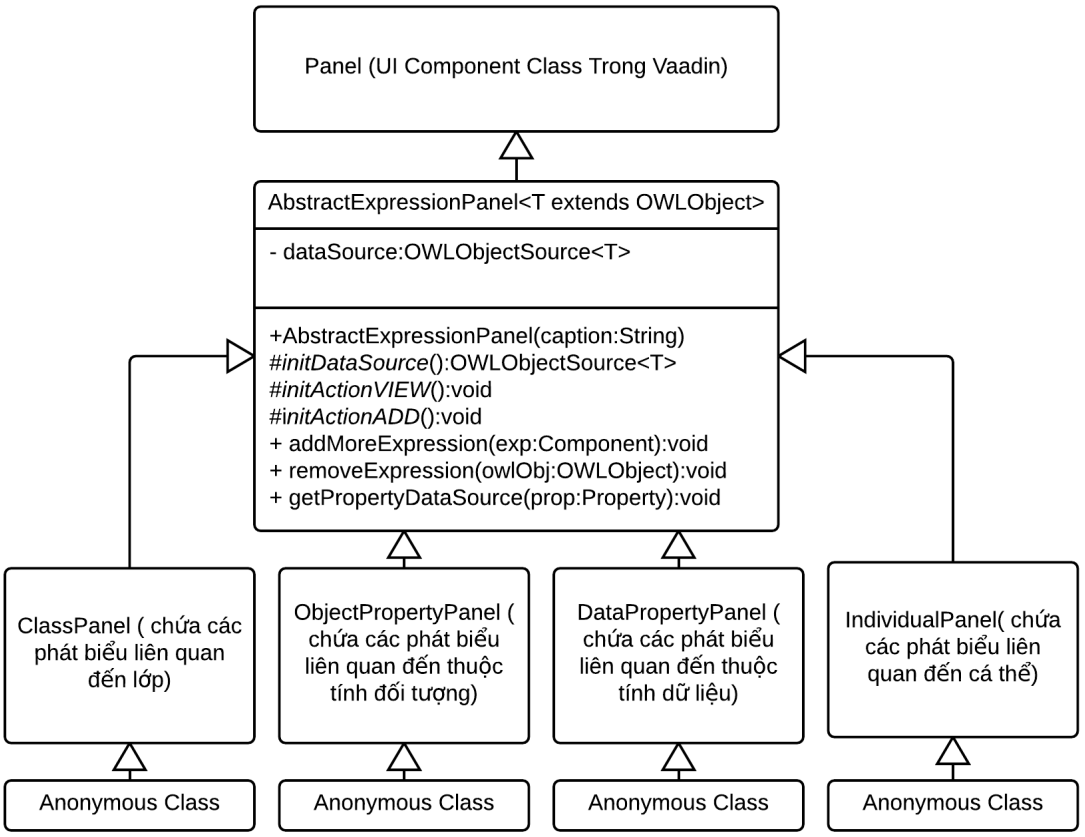
Phương thức này sẽ gọi một phương thức tương tự trong các panel nhỏ bên trong đặt *Property* cho chúng - Property này chỉnh là Data Model dạng Property đã thiết kế trong chương 3. Các panel này tuy nhiều nhưng thật ra chúng em xây dựng chúng đều trên một lớp Abstract gọi là *AbstractExpressionPanel*. Như chúng ta sẽ thấy trong sơ đồ ở hình 4.7 thì tất cả các panel mô tả các phát biểu đều được kế thừa từ đây. Các phương thức in nghiêng trong lớp abstract này chính là các phương thức abstract sẽ được xây dựng cụ thể ở những lớp con của chúng để phù hợp với phát biểu mà chúng muốn diễn tả. Ví dụ chúng em, muốn diễn ta một phát biểu về về lớp con (SubClassOf), thì sẽ khai báo một Anonymous Class từ ClassPanel như sau:

- initActionADD

```
// định nghĩa cụ thể là phải tạo event SubClass
UI.getCurrent().addWindow(new buildAddClassExpressionWindow(
  expression -> eKit.getDataFactory().getSubClassOfAddEvent(...))
);
```

- initActionVIEW

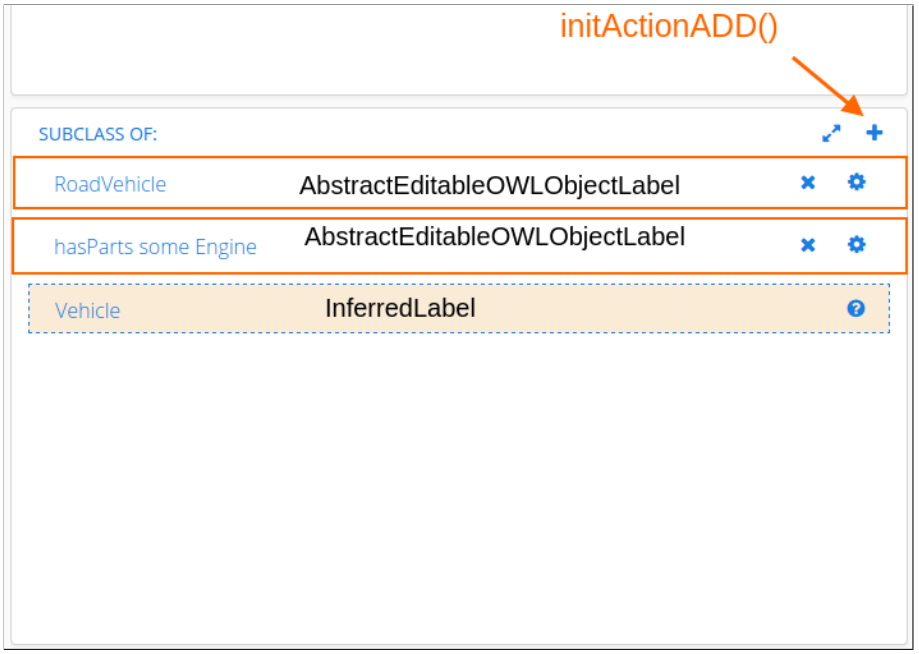
```
//: định nghĩa cụ thể là phải tìm hiển thị các lớp cha của
Collection<OWLClassExpression> ces =
```



Hình 4.7: Class Diagram của AbstractExpressionPanel

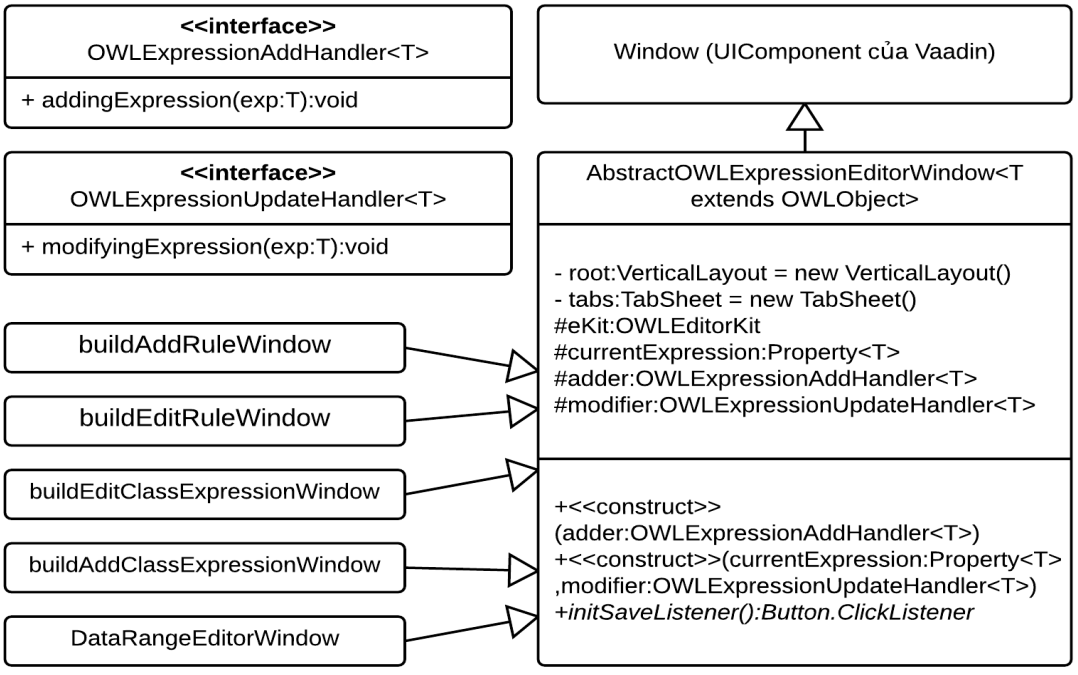
```
            EntitySearcher.getSuperClasses(..);
// tạo các Label trên panel
ces.forEach(ce -> panel.addComponent(new ClassLabel(ce, ...)));
// Nếu trạng thái reasoner bật thì lấy các thông tin về lớp cha
Set<OWLClass> clzz = editorKit.getReasoner().getSuperClasses(...)
// Tạo các label tô đậm trên panel
clzz.forEach(c -> panel.addComponent(new InferredLabel(c)));
```

Khi chọn biểu tượng dấu "+" hoặc bánh răng biểu như trong hình 4.8, sẽ mở ra một cửa sổ mới để chúng ta có thể biên tập các mô tả lớp (ClassExpression) hoặc một cửa sổ để chỉnh sửa các phát biểu này, các loại cửa sổ này đều được xây dựng từ một lớp Abstract là AbstractOWLExpressionEditorWindow.



Hình 4.8: Panel mô tả các phát biểu SubClassOf

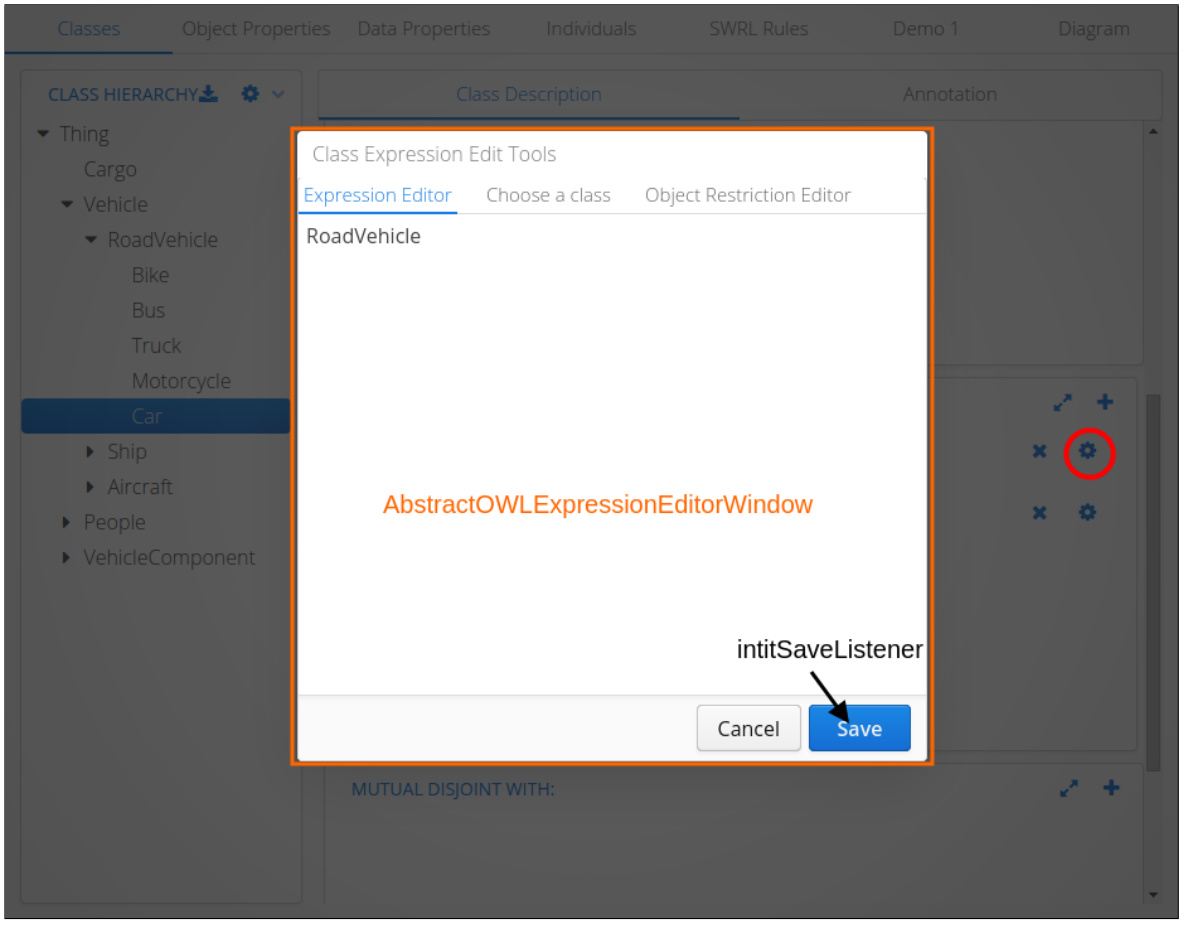
4.3.1.4 AbstractOWLExpressionEditorWindow



Hình 4.9: Class Diagram của AbstractOWLExpressionEditorWindow

Trong hình 4.9 tập hợp hết tất cả các lớp con là các cửa sổ dùng để thêm hoặc sửa mô tả lớp, kiểu dữ liệu trong hệ thống, chúng đều được mở rộng từ abstract này. Đáng chú ý chúng em cũng đang viết các interface dùng trả về các sự kiện phù hợp với từng loại phát biểu (cách hoạt động của chúng như đã trình bày trong mục 4.3.1.2). Một điểm không được thể hiện trong sơ đồ lớp đó chính là việc sử dụng EventBus trong đây. EventBus được sử dụng trong phương thức *initSaveListener*. Ví dụ để cài đặt một event chỉnh sửa một mô tả lớp, chúng ta sẽ đưa đoạn code sau vào initSaveListener.

```
OWLClassExpression ce = editorKit.parseClassExpression(editor.getValue());
OWLEditorEventBus.post(modifyExpression.modifyingExpression(ce));
```



Hình 4.10: Biên tập và chỉnh sửa mô tả lớp

Biến *modifyExpression* là một interface *OWLExpressionUpdateHandler* (Hình 4.9). Interface này sẽ được định nghĩa thông qua một lớp gọi là *AbstractEditableOWLObjectLabel*

(Hình 4.8), Label được thêm vào panel trong *initActionVIEW* ở mục trên, ClassLabel cũng chính là một *AbstractEditableOWLObjectLabel*. Vì thế nếu nhìn chi tiết hơn vào đoạn code ở trên ta có ClassLabel được tạo ra như sau:

```
... = new ClassLabel(new OWLClassExpressionSource(ce),
() -> eKit.getDataFactory().getSubClassOfRemoveEvent(currentCls, ce),
newClsEx -> eKit.getDataFactory()
          .getSubClassOfModEvent(currentCls, newClsEx, currentClsEx))
```

Bởi vì chỉ khi tạo một panel cụ thể cho một phát biểu chúng ta mới biết đó là loại sự kiện gì cho nhóm phát biểu nào (lớp, thuộc hay cá thể). Như đã nói việc xây dựng những interface tiện ích như trên giúp chúng em tiết kiệm rất nhiều công sức khi thực hiện. Trở lại với cửa sổ của biên tập mô tả lớp vừa rồi, khi nhấn "Save" sự kiện sẽ được công bố lên EventBus. Vậy đâu mới là đối tượng tiếp nhận sự kiện vừa rồi. Đó là thành phần sẽ được giới thiệu ngay sau đây.

4.3.1.5 AbstractPanelContainer

Xem lại hình 4.3, đây là thành phần chứa tất cả các *AbstractExpressionPanel* và đặc biệt đây là thành phần chứa 3 loại Subscriber tương ứng với 3 loại sự kiện là thêm, xóa và sửa. Lấy ClassExpressionPanelContainer (lớp kế thừa từ lớp abstract này) làm ví dụ, nó sẽ có 3 subscriber như sau:

```
@Subscribe public void afterClassAxiomAdded(ClassAxiomAdded event)
@Subscribe public void afterClassAxiomRemoved(ClassAxiomRemoved event)
@Subscribe public void afterClassAxiomModified(ClassAxiomModified event)
```

Các thay đổi sẽ được áp dụng trực tiếp vào ontology thông qua:

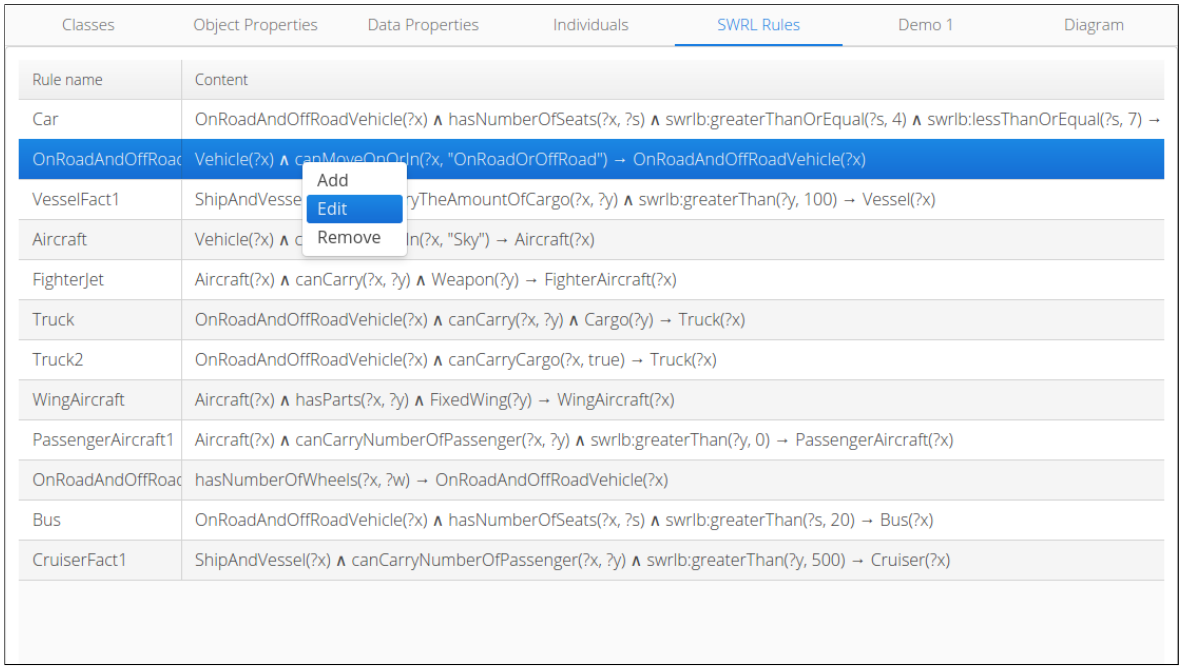
```
@Subscribe public void afterClassAxiomAdded(ClassAxiomAdded e) {
  ChangeApplied ok = eKit.getModelManager()
```

```
.applyChange(new AddAxiom(eKit.getActiveOntology(), e.getAxiom()));
  if(ok == SUCCESSFULLY) {
  e.getAxiom().accept(addHelper(e.getAxiom(), e.getOwner())); }
}
```

Sau khi các thay đổi được áp dụng thành công, chúng sẽ được cập nhật lên ontology bằng addHelper/removeHelper, cả hai đối tượng này đều là *OWLAxiomVisitor* được cài đặt thông qua *OWLAxiomVisitorAdapter* để truy vấn đến những thay đổi có liên quan đến phát biểu có trong sự kiện. Ví dụ: nếu các phát biểu đó liên quan đến lớp thì chúng ta chỉ định nghĩa lại giải thuật cho các phương thức visit(OWLClassAssertionAxiom/ OWLDisjointClassesAxiom/ OWLEquivalentClassesAxiom/ OWLSubClassOfAxiom) trong adapter.

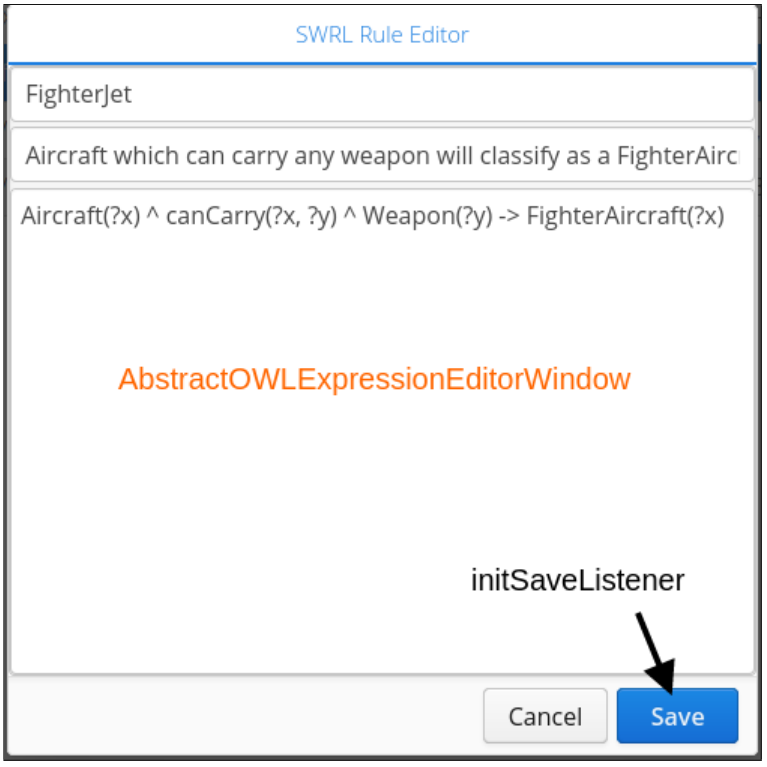
4.3.1.6 SWRL Rule Tab

Tab này có đặt điểm khác sao với các tab còn lại, chỉ gồm một thành phần chính là một Table chứa các SWRL trong tài liệu OWL 2 Ontology. Khi right-click sẽ có các chức năng thêm/xóa/sửa rule được chọn. Khi chọn thêm hay sửa rule, sẽ hiện ra một cửa sổ để thêm



Hình 4.11: SWRL Rule Tab

hoặc chỉnh sửa, về cách thức tổ chức sự kiện thì tương tự với *AbstractOWLExpressionEditorWindow* vì chúng được kế thừa từ lớp Abstract này (Hình 4.9). Duy chỉ có một khác biệt đó là thay vì có một *AbstractPanelContainer* để chứa các Subscriber (phục vụ các sự kiện thêm, xóa, sửa rule) thì các Subscriber nằm trong chính *RuleSheet* - layout chính của tab này.

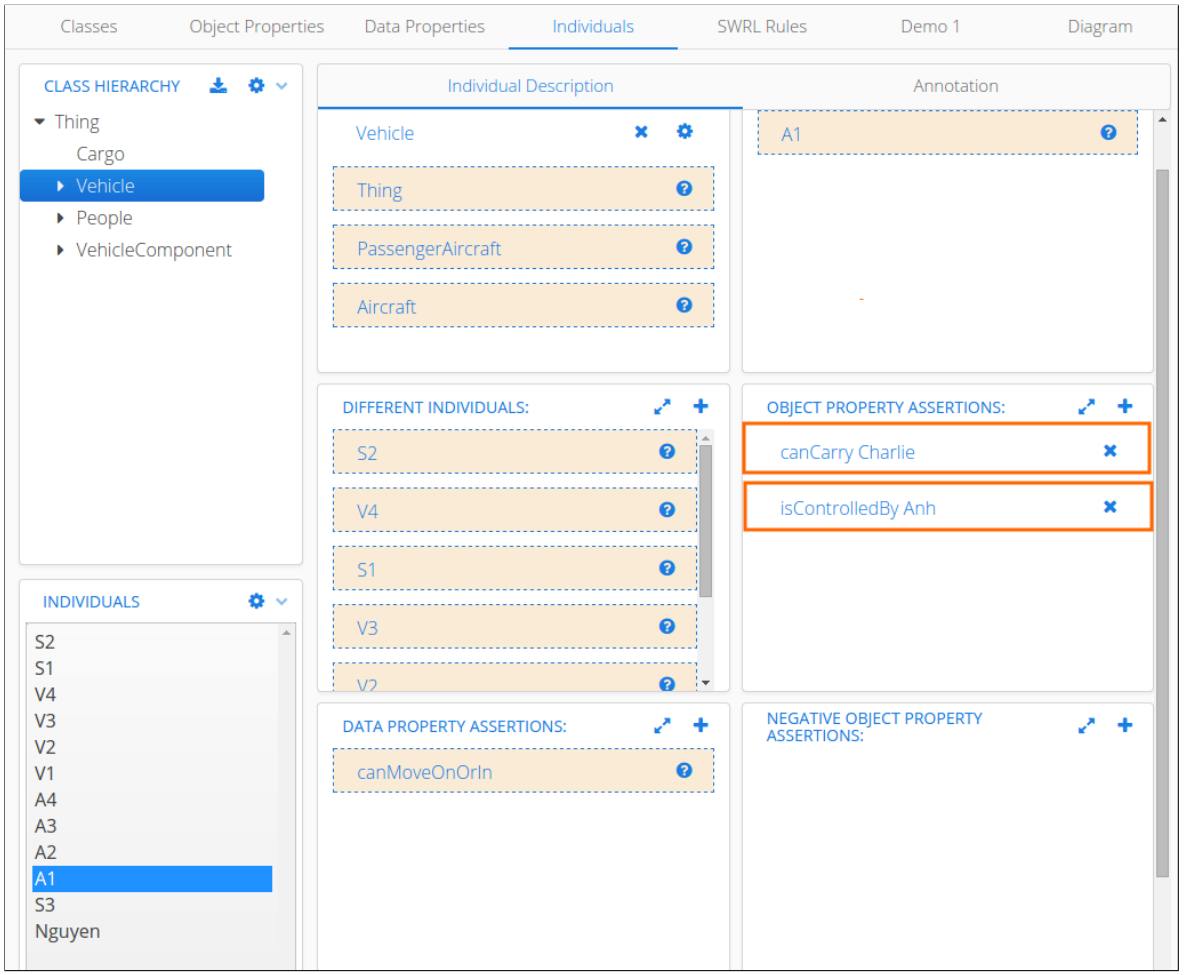


Hình 4.12: Cửa sổ biên tập SWRL Rule

Kết luận Tất cả các thành phần vừa được chúng em vừa giới thiệu, đều được sử dụng để mở rộng thành những thành phần giao diện cụ thể trong ứng dụng. Với số 4 loại thực thể (lớp, thuộc tính đối tượng/dữ liệu và cá thể) và rất nhiều dạng phát biểu (Axiom), chúng em nghĩ việc xây dựng các thành phần giao diện thành các lớp Abstract, sử dụng interface để làm các action handler hoặc xây dựng dữ liệu theo tính đa hình đã phát huy hiệu quả rất lớn là chúng em đã tách biệt thành phần giao diện và dữ liệu (thực thể, phát biểu) với nhau, nhằm tránh việc xây dựng lại từng giao diện cho từng phát biểu, thực thể.

4.4 Hiện thực khả năng phân loại tự động của ontology đã thiết kế

Đây chính là mục tiêu cuối cùng mà cả hệt thông muốn hướng tới, chứng minh tính khả thi trong việc sử dụng ontology để phân loại. Ontology transport [30] được xây dựng giống như thiết kế ở chương 3. Chúng em sẽ thực nghiệm việc suy luận (reasoning) qua đó phân loại các cá thể thuộc lớp *Vehicle* vào trong các lớp con của nó.



Hình 4.13: Phân loại dựa vào các phát biểu tương đương

4.4.1 Phân loại dựa vào các phát biểu tương đương

Cá thể A1 thuộc các lớp Aircraft (Hình 4.13), chọn dấu chấm ? trên phát biểu chúng ta sẽ nhận được giải thích sau:

```
Aircraft EquivalentTo Vehicle and (isControlledBy some Pilot)
Anh Type Pilot
A1 isControlledBy Anh
A1 Type Vehicle
```

Trong mọi loại phát biểu thì phát biểu tương đương Equivalent là loại phát biểu có ràng buộc lớn nhất giữa các lớp với nhau, nghĩa là một cá thể hội đủ điều kiện của *Vehicle* **và** *(isControlledBy some Pilot)* sẽ thuộc loại *Aircraft*. Do A1 đã thuộc *Aircraft* do các phát biểu trên nên nó cũng thuộc *PassengerAircraft* dựa theo các phát biểu sau

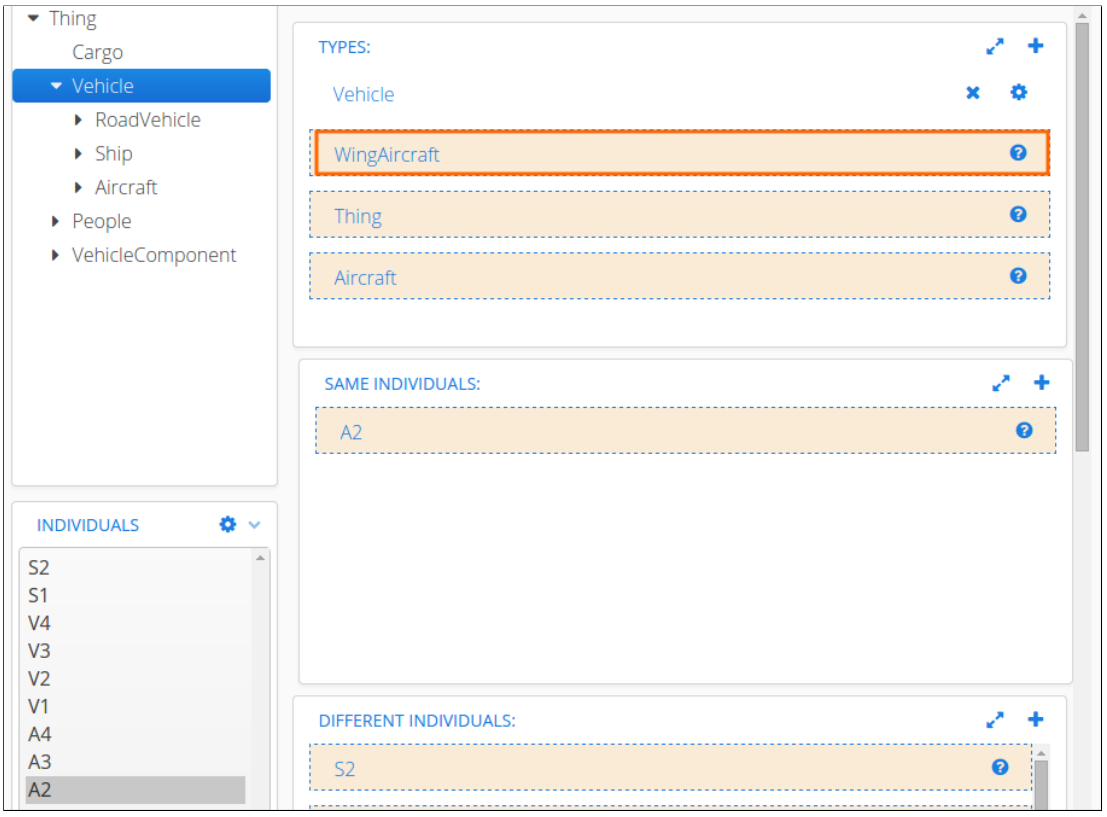
```
PassengerAircraft EquivalentTo Aircraft and (canCarry some Passenger)
Charlie Type Passenger
A1 canCarry Charlie
```

4.4.2 Phân loại dựa trên Domain của thuộc tính đối tượng

Chọn vào dấu ? để xem giải thích (Hình 4.14, chúng ta sẽ nhận được các giải thích sau cho lý dó *A2* là *WingAircraft*

```
hasWing Domain WingAircraft
A2 hasWing XWing
```

Giải thích chỉ có những cá thể thuộc lớp *WingAircraft* mới có thuộc tính đối tượng *hasWing*, mà A2 *hasWing* XWing (không cần biết XWing thuộc lớp nào) nên A2 cũng thuộc *WingAircraft*.



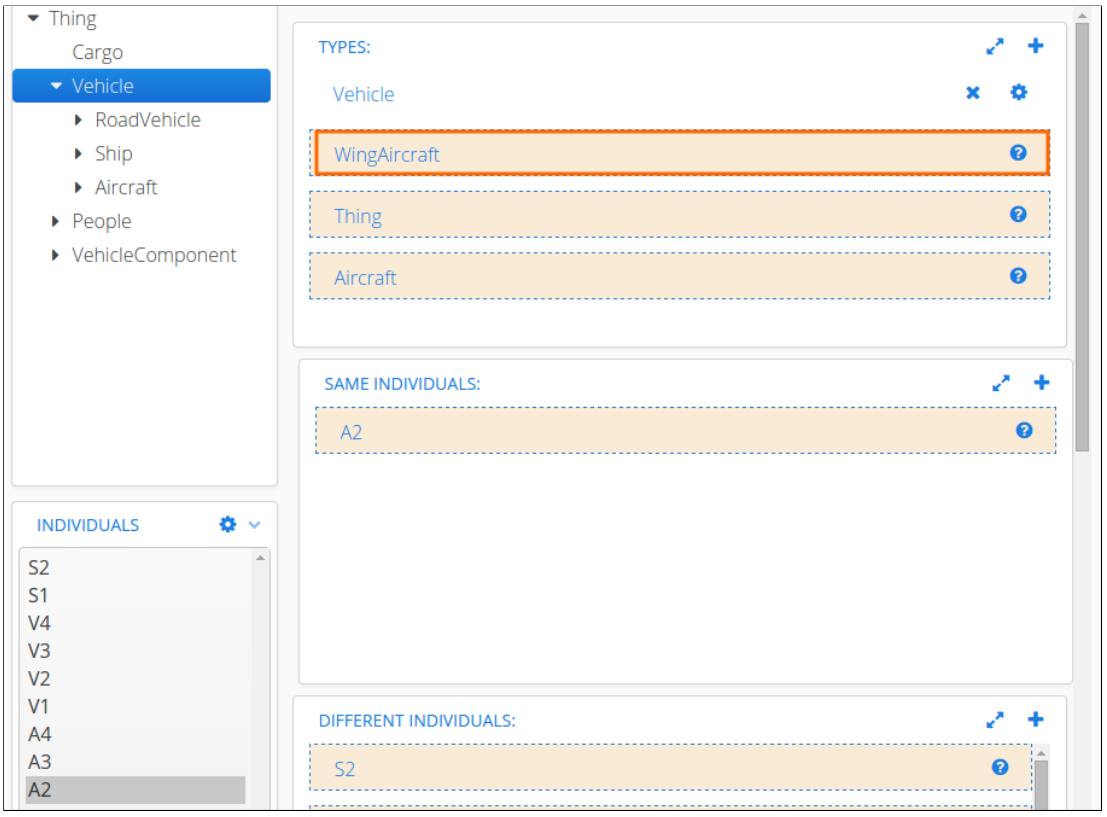
Hình 4.14: Phân loại dựa trên Domain của thuộc tính đối tượnglabeloverflow

4.4.3 Phân loại theo SWRL Rule và thuộc tính dữ liệu

Cá thể V4 đã thuộc lớp *RoadVehicle* trước, khi phân loại chúng ta thấy nó sẽ thuộc vào lớp *Bus*, đây là lý do

```
RoadVehicle(?x) ^ hasNumberOfSeats(?x, ?s)
                ^ swrlb:greaterThan(?s, 20) -> Bus(?x)
V4 hasNumberOfSeats 21
V4 Type RoadVehicle
```

V4 được khai báo là Vehicle có thuộc tính dữ liệu *hasNumberOfSeats* (có số chỗ ngồi) với giá trị 21, mà SWRL Rule nói rằng phương tiện đường bộ nào có số chỗ ngồi lớn hơn hoặc bằng 20 đều là xe Bus (Hình 4.15). Trên đây, là 3 ví dụ khá đơn giản về cách sử dụng các phát biểu của riêng OWL 2 hay sử dụng chúng kết hợp chúng với SWRL Rule

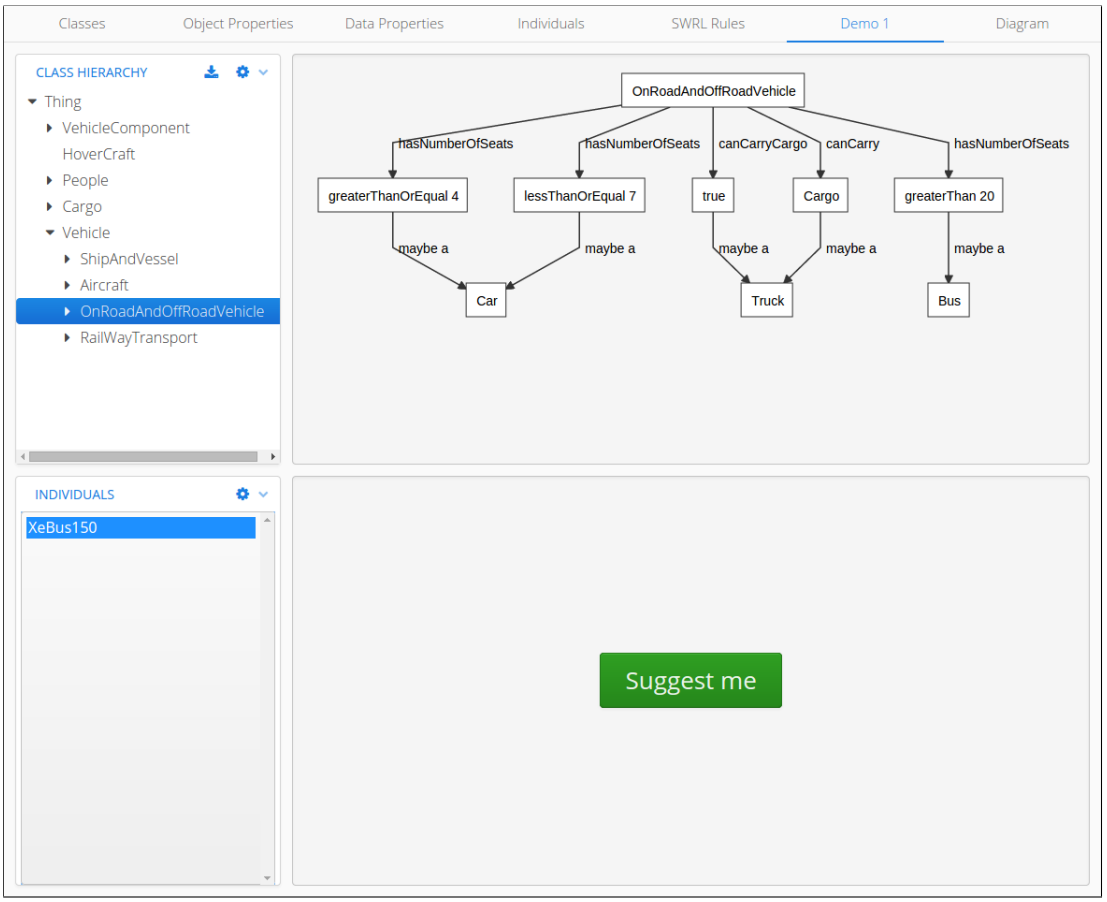


Hình 4.15: Phân loại dựa trên Domain của thuộc tính đối tượnglabeloverflow

để phân loại. Trên thực tế sẽ có những trườgn hợp phức tạp hơn, ví dụ của chúng em chỉ muốn kiểm chứng tính khả thi và khả năng của OWL 2 và SWRL.

4.5 Tính năng hỗ trợ phân loại - Demo Tab

Việc sử dụng SWRL Rule hay đơn giản hơn là đọc và hiểu ý nghĩa có nó thường rất khó đối với người dùng thông thường, chưa kể đến việc chưa có cơ chế tổ chức các SWRL rule một cách khoa học, sẽ có những trường hợp chúng ta vô tình tạo ra những điều kiện trùng nhau dẫn đến 2 kết quả khác nhau hoặc ngược lại. Hiểu được nhược điểm này của SWRL Rule, chúng em đã thiết kế thêm tính năng hỗ trợ phân loại với giao diện là Tab Demo trong ứng dụng, với khả năng vẽ ra một sơ đồ dựa theo chuỗi các điều kiện - kết quả có liên quan đến lớp đang chứa cá thể được chọn để phân loại. Để minh họa cho tính năng này ,chúng em xin trích một đoạn chứa các SWRLRule trong ontology *Transport.owl* [30]



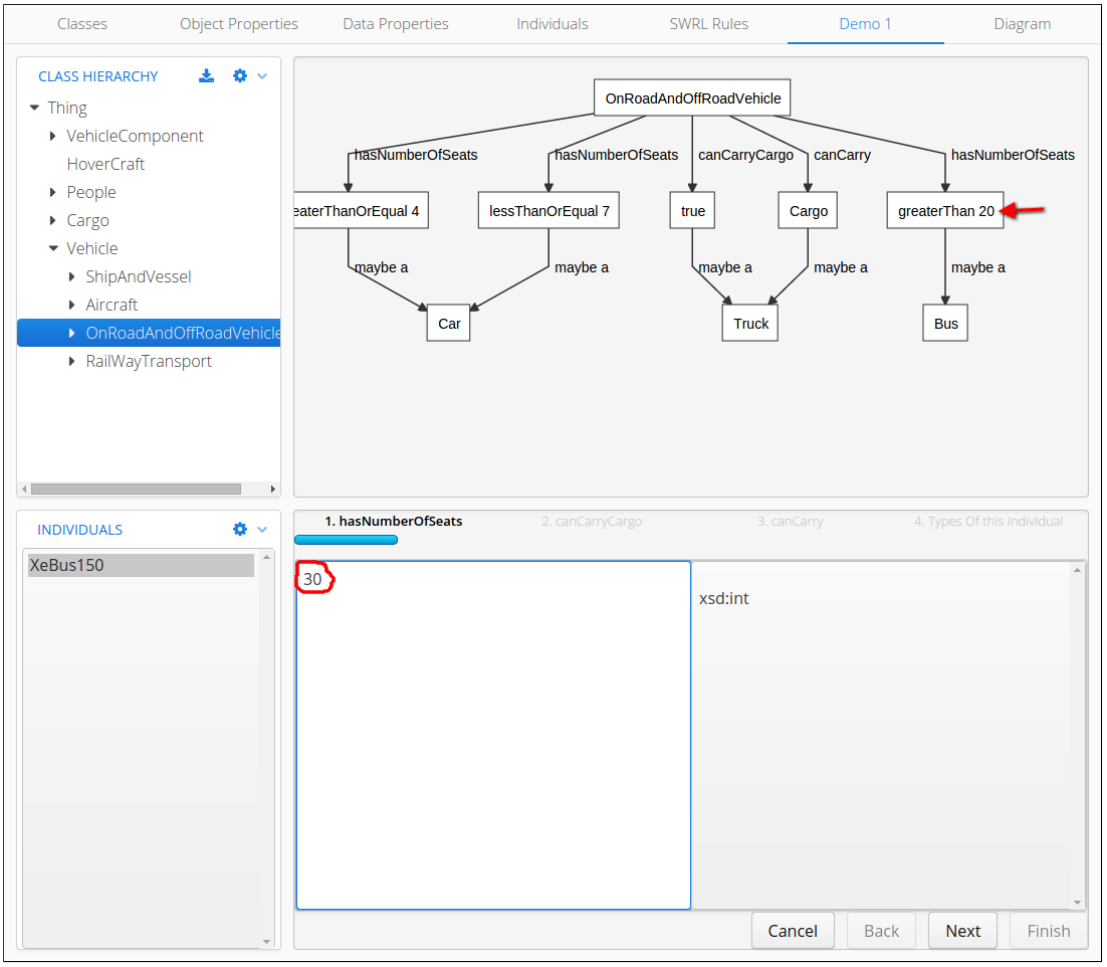
Hình 4.16: Tính năng hỗ trợ phân loại Demo Tab (phần 1)

(được chúng em xây dựng để làm demo cho tính năng phân loại). Hình trên mô tả lại các rule sau:

```
# Phương tiện đường bộ có thể chở hàng hóa (Cargo) là xe tải (Truck)
OnRoadAndOffRoadVehicle(?x) ^ canCarry(?x, ?y) ^ Cargo(?y)  -> Truck(?x)
# Phương tiện đường bộ có khả năng chở hàng là xe tải (Truck)
OnRoadAndOffRoadVehicle(?x) ^ canCarryCargo(?x, true) -> Truck(?x)
# Phương tiện đường bộ có số 4 <= chỗ ngồi <= 7 là xe hơi (Car)
OnRoadAndOffRoadVehicle(?x) ^ hasNumberOfSeats(?x, ?s)
^ swrlb:greaterThanOrEqual(?s, 4)
^ swrlb:lessThanOrEqual(?s, 7)  -> Car(?x)
# Phương tiện đường bộ có số chỗ ngồi >= 20 là xe Bus
OnRoadAndOffRoadVehicle(?x) ^ hasNumberOfSeats(?x, ?s)
```

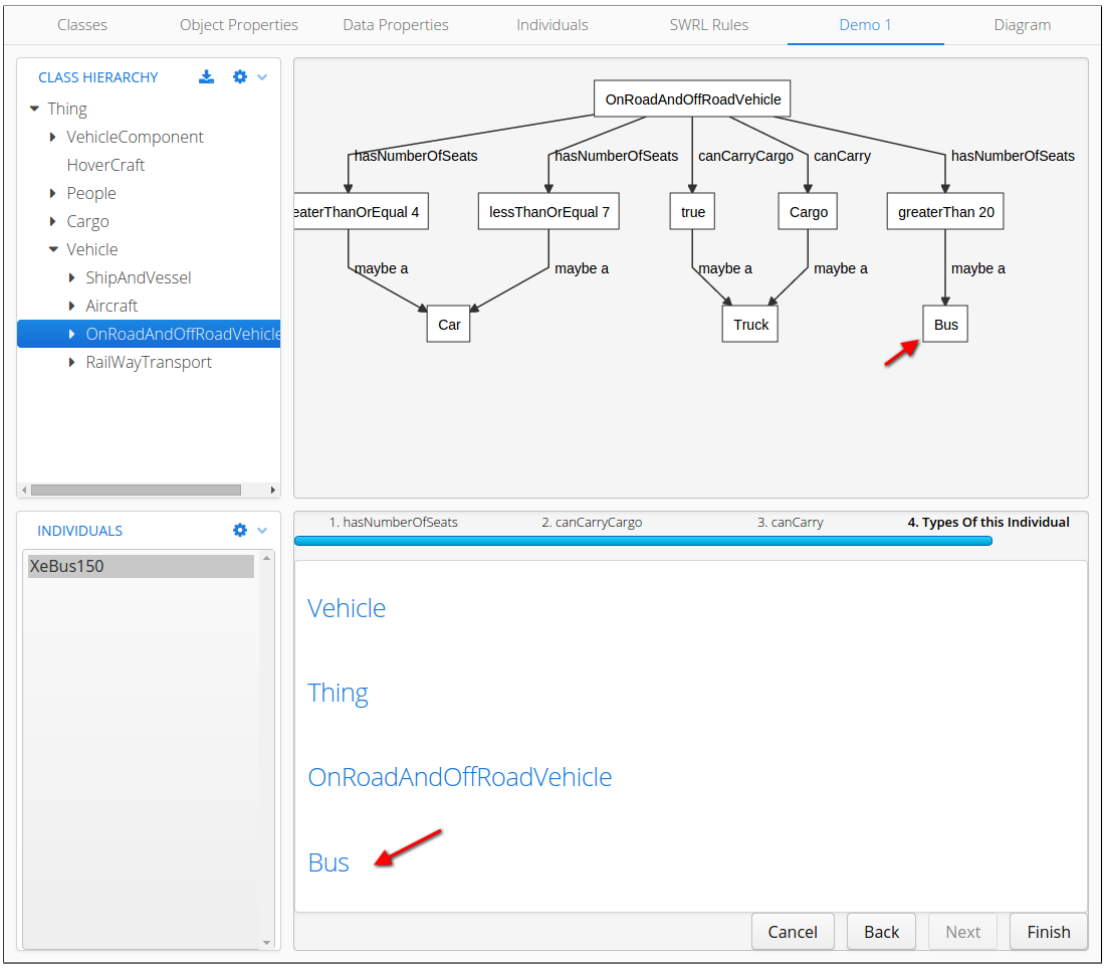
```
^ swrlb:greaterThan(?s,20) -> Bus(?x)
```

Trong hình chỉ miêu tả những rule nào có liên quan đến lớp *OnRoadAndOffRoadVehicle* vốn là lớp chứa cá thể *XeBus150* trong hình, làm vậy để hạn chế những thông tin không cần thiết đối với người dùng. Khi chọn *Suggest Me* sẽ tạo ra một loại các UI Component phục vụ để tạo ra các phát biểu về cá thể *XeBus150*. Trong hình trên mỗi bước tương



Hình 4.17: Tính năng hỗ trợ phân loại Demo Tab (phần 2)

ứng với những phát biểu có liên quan đến *OnRoadAndOffRoadVehicle* như trong sơ đồ. Miền dữ liệu (Range) của thuộc tính dữ liệu và thuộc tính đối tượng cũng sẽ được áp dụng nhằm giảm bớt những thông tin không cần thiết, thuộc tính *hasNumberOfSeats* có DataPropertyRange(xsd:int) nên ở cột bên phải chúng ta chỉ thấy kiểu dữ liệu này thay vì là một loạt kiểu dữ liệu như xsd:string, xsd:byte, v.v... . Khi đến bước cuối cùng,



Hình 4.18: Tính năng hỗ trợ phân loại Demo Tab (phần 3)

reasoner sẽ đánh giá lại các thông tin mà chúng ta vừa nhập vào ở các bước trước để đưa kết quả phân loại phù hợp như đã được biểu diễn trong sơ đồ.

Chương 5

Kết luận

Với sự phát triển ngày càng nhanh chóng và mạnh mẽ của công nghệ Web 3.0 hay Semantic Web, trong tương lai các hệ thống dữ liệu sẽ sử dụng những nguyên lý của Semantic Web sẽ ngày càng tăng lên do những lợi ích mà nó mang lại so với cách tổ chức dữ liệu truyền thống (Open World vs. Closed World Assumption). Điều này có nghĩa là sẽ có nhiều dữ liệu được tổ chức dưới dạng Ontology Web Language hay các phiên bản RDFS của nó nên việc chúng em đề ra một ý tưởng mới là dùng ontology để phân loại các dữ liệu này là một bước đi đón đầu. Bên cạnh đó, với những lý thuyết về việc sử dụng Ontology cho việc phân loại đã được thực nghiệm trong đề tài, chúng em tin rằng khả năng phát triển một hệ thống thực tế với một ontology với lượng phát biểu lớn về dữ liệu và thông tin của chúng là khả thi và có thể thực hiện bằng các công nghệ hiện tại.

5.1 Những công việc đã làm được

- Có được những kiến thức nền tảng chắc chắn về ngôn ngữ Ontology Web Language 2, Semantic Web Rule Language và đặc tính suy luận của chúng.
- Biết được tính nhất quán của ontology và các nguyên nhân phổ biến gây ra tính thiếu nhất quán trong ontology.

- Xây dựng được một ứng dụng dùng để thiết kế ontology trên môi trường Web với giao diện thân thiện, dễ dùng và có đầy đủ các tính năng như suy luận, giải thích hỗ trợ biên tập SWRL Rule.
- Thiết kế môt OWL2 ontology để trình bày tính năng phân loại và tính năng hỗ trợ phân loại từ ứng dụng đã thiết kế.

Tóm lại, đề tài nghiên cứu và "xây dựng ontology phục vụ cho việc phân loại hàng hóa tự động" đã cho thấy tính khả thi trong việc kết hợp ngôn ngữ Ontology Web Language 2 (OWL2), ngôn ngữ SWRL Rule và cách thức hoạt động của reasoner để phân loại các cá thể một cách tự động. Bên cạnh đó, công cụ chỉnh sửa ontology OWL Editor mà nhóm phát triển đem lại nhiều lợi ích trong việc nghiên cứu và phát triển ontology, cụ thể hơn là đơn giản hoá việc thiết kế một ontology theo ý muốn. Sau khi đạt được những thành công kể trên, đề tài còn có thể đạt được những thành công thực tiễn hơn nếu được áp dụng vào các lĩnh vực khác, chẳng hạn như việc phân loại hàng hoá, hoặc phân loại các gói tin trong lưu lượng mạng...

5.2 Những mặt hạn chế

- Trình biên tập còn nhiều điểm phản hồi chưa tốt do chúng em chưa chỉ tập trung vào tính năng , vẫn chưa tính đến hiệu năng cũng như hiệu suất của ứng dụng , như việc phải chịu nhiều tải cùng lúc, khả năng lưu trữ, quản lý người dùng.
- Trình biên tập chưa có cơ chế đăng nhập hoặc quản lý các tài liệu ontology.
- Ý tưởng về việc phân loại còn chỉ ở dạng mô hình, chưa có tính thực nghiệm cao.

5.3 Hướng phát triển

Nhằm khắc phục một số hạn chế kể trên chúng em cũng đã nghĩ đến các hướng phát triển sau :

- Để tăng tốc độ xử lý, chúng em sẽ sử dụng kết hợp Vaadin với Spring Boot (đã thực hiện được một phần), đồng thời tối ưu hóa các đoạn code.
- Trong giai đoạn phát triển cuối cùng, chúng em đã tìm hiểu được là đã có những cơ sở dữ liệu hỗ trợ lưu tài liệu Ontology dưới dạng RDF Graph, tiêu biểu có Stardog - một cơ sở dữ liệu Semantic hỗ trợ OWL 2, SWRL, và đặc biệt tích hợp bản mới nhất của reasoner Pellet 3.0. Sẽ là một ý tưởng tuyệt vời nếu tích hợp sử dụng Stardog là nơi lưu trữ những tài liệu ontology mà người dùng soạn thảo, và giảm tải khả năng suy luận cho cơ sở dữ liệu,
- Thiết kế một ontology với số lượng lớn các phát biểu phục vụ cho việc phân loại trên thực tế, lĩnh vực là rất rộng lớn nhưng chúng em đã nghĩ đến một lĩnh vực thực tiễn và có thể sử dụng các kiến thức chuyên ngành mạng, đó là phân loại thông tin lưu lượng mạng nhằm phục vụ cho các tác vụ như xây dựng Firewall rule, IDS rule.
- Đưa ứng dụng thành một ứng dụng mã nguồn mở.

Đề tài mà chúng em đã thực hiện, với tính năng phân loại tự động đấu giá, có thể được áp dụng vào nhiều lĩnh vực thực tế khác nhau. Dưới đây là một số ứng dụng có tính khả thi cao :

- Phân loại các hàng hoá dựa theo các đặc điểm, tính năng đặc trưng của nó
- Phân loại thông tin lưu lượng mạng nhằm phục vụ cho việc phân tích, ngăn chặn các cuộc tấn công an ninh mạng

Tài liệu tham khảo

[1] "Swrl wiki faq." [Online]. Available: https://github.com/protegeproject/swrlapi/wiki/SWRLLanguageFAQ

[2] R. Stevens, "(i can't get no) satisfiability." [Online]. Available: http://ontogenesis.knowledgeblog.org/1329

[3] "Semantic web introduction." [Online]. Available: http://en.wikipedia.org/wiki/Semantic_Web

[4] J. Squeda, "Introduction to: Open world assumption vs closed world assumption," 2012. [Online]. Available: http://semanticweb.com/introduction-to-open-world-assumption-vs-closed-world-assumption_b33688

[5] "Semantic web standards by w3c." [Online]. Available: http://www.w3.org/2001/sw/wiki/Main_Page

[6] "Resource description framework." [Online]. Available: http://www.w3.org/standards/techs/rdf#w3c_all

[7] "Rdf schema 1.1," 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/

[8] B. Motik, P. F. Patel-Schneider, and B. Parsia, "Ontology web language 2 overview," *W3C Recommendation*, December 2012. [Online]. Available: http://www.w3.org/TR/owl2-overview/

[9] "Description logic." [Online]. Available: http://en.wikipedia.org/wiki/Description_logic

[10] B. Motik, P. F. Patel-Schneider, and B. Parsia, "Mapping to rdf graph," *W3C Recommendation*, December 2012. [Online]. Available: http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/

[11] "Rdf/xml syntax specification." [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[12] "Internationalized resource identifiers." [Online]. Available: http://www.ietf.org/rfc/rfc3987.txt

[13] "Rdf concept." [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[14] B. Motik, P. F. Patel-Schneider, and B. Parsia, "Owl 2 web ontology language structural specification," *W3C Recommendation*, December 2012. [Online]. Available: http://www.w3.org/TR/owl2-syntax/

[15] "Swrl built-in." [Online]. Available: https://github.com/protegeproject/swrlapi/wiki/SWRLBuiltInFAQ

[16] "Swrl core built-ins." [Online]. Available: http://www.daml.org/rules/proposal/builtins.html

[17] S. Bail, "Common reasons for ontology inconsistency"." [Online]. Available: http://ontogenesis.knowledgeblog.org/1343

[18] A. Kalyanpur, B. Parsia, B. Cuenca-Grau, and E. Sirin, "Repairing unsatisfiable concepts in owl ontologies," 2007. [Online]. Available: http://www.cs.ox.ac.uk/people/bernardo.cuencagrau/publications/repair.pdf

[19] A. Kalyanpur, B. Parsia, and E. Sirin, "Axiom pinpointing: Finding (precise) justifications for arbitrary entailments in shoin (owl-dl)," UMIACS 2006, Tech. Rep., 2006.

[20] M. Horridge, "Justification based explanation in ontologies," Ph.D. dissertation, The University of Manchester, 2012. [Online]. Available: http://www.bcs.org/upload/pdf/dd-matthew-horridge.pdf

[21] *Programming with the OWL API*, 2014. [Online]. Available: https://github.com/owlcs/owlapi

[22] *SWRL API*, 2014. [Online]. Available: https://github.com/protegeproject/swrlapi/wiki

[23] *Pellet - OWL 2 Reasoner for Java*, 2013. [Online]. Available: http://github.com/clarkparsia/pellet

[24] "Vaadin directory." [Online]. Available: https://vaadin.com/directory

[25] "Vaadin architecture." [Online]. Available: https://vaadin.com/book/-/page/architecture.html#architecture.overview

[26] *Spring Boot Reference*. [Online]. Available: http://docs.spring.io/spring-boot/docs/1.2.1.RELEASE/reference/htmlsingle/

[27] "Spring for vaadin." [Online]. Available: https://github.com/peholmst/vaadin4spring/

[28] "Protege desktop." [Online]. Available: https://github.com/protegeproject/protege/

[29] *Guava EventBus*. [Online]. Available: https://code.google.com/p/guava-libraries/wiki/EventBusExplained

[30] "Source code của ứng dụng." [Online]. Available: https://bitbucket.org/rexey3s/spring-vaadin-owleditor