LỜI CẢM ƠN

Trong suốt quá trình thực hiện luận văn, ngoài nỗ lực chính bản thân, tôi còn nhận được rất nhiều nguồn động viên cũng như nhiều sự giúp đỡ từ các thầy, các cô, bạn bè và gia đình. Đây là một trong những nguồn động lực vô cùng to lớn đã đồng hành cùng tôi và giúp tôi hoàn thành chương trình học cũng như luận văn này. Vì vậy, tôi xin bày tỏ lòng biết ơn chân thành đến:

- Thầy TS. Nguyễn Tuấn Đăng - người hướng dẫn khoa học nghiêm túc và nhiệt tâm, đã hướng dẫn, định hướng và truyền đạt cho tôi những kiến thức chuyên môn cũng như những kinh nghiệm quý báu trong nghiên cứu khoa học và đồng thời tạo mọi điều kiện cho tôi trong suốt thời gian thực hiện đề tài.
- Các thầy, cô trong khoa Khoa Học Máy Tính thuộc trường Đại Học Công Nghệ Thông Tin, TP. HCM đã luôn hỗ trợ kinh nghiệm, chia sẽ những khó khăn trong công việc và động viên tôi.
- Bạn Huỳnh Nguyên Phát và các anh chị cao học khoá 3 đã giúp đỡ tôi trong lúc thực hiện đề tài.
- Ba, Mẹ và em đã dành cho con một tình thương vô bờ bến, một nguồn động lực và là chỗ dựa tinh thần vững chắc nhất giúp con hoàn thành luận văn này.

TP. Hồ Chí Minh, tháng 11 năm 2009

Lương Quý Tịnh Hà

DANH MỤC HÌNH

DANH MỤC BẢNG

MỤC LỤC

CHƯƠNG I - TỔNG QUAN

1.1 Đặt vấn đề

Hiện nay công nghệ phục vụ cho máy tính ngày càng phát triển nhanh chóng, đặc biệt là internet, và đã đem lại nhiều lợi ích to lớn cho con người. Song song với sự phát triển đó, thì lượng thông tin cũng ngày càng lớn dần lên và nhu cầu tìm kiếm thông tin trên internet cũng ngày càng gia tăng. Để đáp ứng nhu cầu tìm kiếm đó, hàng loạt các công cụ tìm kiếm ra đời (google, yahoo, Teoma…). Tuy nhiên, hiện nay hầu hết các công cụ tìm kiếm chủ yếu tìm kiếm dựa trên từ khóa hay một cụm từ nhập vào, và cũng có một số công cụ tìm kiếm có tìm kiếm phân loại theo chủ đề [7].

Vì hầu hết các công cụ tìm kiếm chủ yếu dựa trên từ khóa hay cụm từ nên khi thực hiện việc tìm kiếm sẽ cho ra rất nhiều trang web chứa từ khóa hay cụm từ đó. Vì vậy, để có được thông tin chính xác mà người tìm kiếm cần thì họ cũng phải tốn rất nhiều thời gian để duyệt rất nhiều trang web có chứa các từ khóa hay cụm từ đó, đôi khi không thể tìm được thông tin cần thiết (do thông tin bị ẩn) bởi vì do bản quyền của bài báo hay trang web.

Mặc dù đã có rất nhiều công cụ tìm kiếm (dựa trên từ khóa là chủ yếu) nhưng công cụ tìm kiếm hỗ trợ cho tiếng Việt hầu như rất ít, cho kết quả tìm kiếm còn hạn chế. Vì vậy, việc tìm kiếm thông tin chính xác, đặc biệt là tìm kiếm cho tiếng Việt, là nhu cầu và mục đích cuối cùng của người tìm kiếm. Để đáp ứng mục đích này, chúng tôi thực hiện đề tài:

"Xây dựng công cụ tìm kiếm tài liệu học tập bằng các truy vấn ngôn ngữ tự nhiên trên kho học liệu mở tiếng Việt"

Đề tài này không thực hiện việc tìm kiếm dựa trên từ khoá hay cụm từ và không trả về các trang web chứa từ khoá hay cụm từ, mà trả về câu trả lời tương ứng với những câu hỏi do người dùng nhập vào. Nói cách khác, khi người dùng nhập câu hỏi bằng tiếng Việt, thì công cụ tìm kiếm trả về câu trả lời chính xác cho thông tin người dùng.

1.2 Phạm vi nghiên cứu

1.2.1 Giới hạn vấn đề nghiên cứu

Mục tiêu của luận văn cho phép người dùng nhập câu hỏi bằng tiếng Việt và hệ thống trả về câu trả lời. Câu hỏi người dùng chỉ gồm các vấn đề liên quan đến khoá học: tên (name), mã (id), ngôn ngữ (language), tóm tắt (summary), từ khóa (keywords), loại tài liệu (document type), bản quyền (license), tác giả (authors), chịu trách nhiệm xuất bản (copyright holders), chỉnh lý (maintainers), phiên bản (version), ngày tạo (created), ngày chỉnh sửa (revised).

Câu trả lời trả về thông tin thuộc kho học liệu mở Việt Nam (www.vocw.edu.vn).

1.2.2 Ý nghĩa khoa học

Đáp ứng được nhu cầu tìm kiếm thông tin chính xác.

Việc tìm kiếm thông tin chính xác không dựa trên từ khoá hay cụm từ mà dựa trực tiếp vào câu hỏi tiếng Việt của người dùng để trả lời chính xác cho câu hỏi đó.

1.3 Hiện trạng nghiên cứu

1.3.1 Tình hình nghiên cứu trên thế giới

Trên thế giới hiện nay, đã có rất nhiều công cụ tìm kiếm hầu như dựa trên từ khóa ra đời như google, yahoo, msn, alibaba… Và kết quả tìm kiếm cũng phần nào làm thỏa mãn nhu cầu tìm kiếm đối với người dùng mặc dù phần lớn tốn nhiều thời gian để duyệt qua.

Tuy nhiên, công cụ tìm kiếm liên quan đến việc hỏi-đáp (ngữ nghĩa) để trả lời chính xác câu hỏi của người dùng thì không nhiều.

Một số công cụ tìm kiếm nước ngoài [7]:

Công cụ tìm kiếm	Cơ sở dữ liệu	Từ khóa hay hỏi-đáp (ngữ nghĩa)	Khả năng tìm kiếm
Google www.google.com	Toàn bộ văn bản các trang web, pdf, MS.Office, PostScript…	Từ khóa	- Hỗ trợ tìm kiếm nâng cao. - Dùng * để rút gọn (thay thế từ trong cụm từ). - Dùng dấu “” để

			tìm cụm từ. - Tìm theo vùng. - Tìm các trang liên quan. - Tìm hình ảnh… - Tìm kiếm theo nhiều ngôn ngữ.
AllTheWeb www. allTheWeb.com	Toàn bộ văn bản của các trang web, PDF, MS Office, PostScript…	. Từ khóa	- Không rút gọn, dùng " " tìm cụm từ - Tìm theo vùng. - Tìm hình ảnh và video…
AltaVista www.altavista.com	Toàn bộ văn bản của các trang web.	Từ khóa	- Dùng " " tìm cụm từ . - Dùng * để rút gọn. - Phân biệt chữ hoa, chữ thường. - Tự động xác định cụm từ và kiểm tra chính tả. - Tìm hình ảnh, âm thanh, video và tin tức. - Tìm kiếm theo nhiều ngôn ngữ khác nhau.
Teoma teoma.com	Toàn bộ văn bản của các trang web.	Từ khóa	- Không rút gọn - Dùng " " tìm cụm từ . - Tìm kiếm theo các

			giới hạn như ngày, ngôn ngữ, vị trí, độ sâu liên kết của trang WEB. - Có thể gom nhóm các kết quả.
Ixquick www.ixquick.com	Searches AltaVista, Ask Jeeves /Teoma, MSN, Yahoo & more.	Từ khóa	- Hỗ trợ tìm kiếm tin tức, file mp3, file ảnh. - Tập hợp và phân hạng các kết quả, hạn chế sự trùng lắp thông tin.
AskJeeves www.ask.com	Đáp ứng hàng triệu các yêu cầu từ các site ước lượng.	Từ khóa	- Dùng dấu nháy kép để tìm kiếm cụm từ trong Teoma. - Hoạt động hiệu quả đối với các yêu cầu đơn giản.

1.3.2 Tình hình nghiên cứu tại Việt Nam

Vấn đề tìm kiếm là một trong những vấn đề rất quan trọng khi tìm tài liệu mong muốn. Mặc dù hiện nay trên thế giới đã xuất hiện rất nhiều công cụ tìm kiếm hỗ trợ nhiều ngôn ngữ nhưng ở Việt Nam, công cụ tìm kiếm hỗ trợ cho tiếng Việt không nhiều, đặc biệt là công cụ tìm kiếm theo ngữ nghĩa.

Tuy vậy, cũng đã có một số công cụ tìm kiếm đã ra đời như NetNam, VinaSeek, ZingSearch… cũng góp một phần nào đó trong việc tìm kiếm dựa trên từ khóa tiếng Việt.

Một số công cụ tìm kiếm trong nước [7]:

Công cụ tìm kiếm	**Cơ sở dữ liệu**	**Từ khóa hay hỏi-đáp (ngữ nghĩa)**	**Khả năng tìm kiếm**
NetNam www.pan.vietnam.com	Toàn bộ văn bản của các trang web.	Từ khóa	- Dùng “ ” tìm cụm từ . - Phân biệt chữ hoa và thường. - Sử dụng từ khóa để lọc các.
VinaSeek www.vinaseek.com Tìm kiếm.	Toàn bộ văn bản của các trang web.	Từ khóa	- Dùng “ ” tìm cụm từ . - Phân biệt chữ hoa và thường. - Sử dụng từ khóa để lọc các tìm kiếm.

CHƯƠNG II - CƠ SỞ LÝ THUYẾT

2.1 Đối tượng nghiên cứu

Thông tin từ trang học liệu mở (www.vocw.edu.vn).

Trang học liệu mở chứa các khóa học. Vào thời điểm 11/8/2009, trang học liệu chứa 225 khóa học (course). Mỗi khóa học đều chứa thông tin metadata bao gồm:

- Tên (Name)
- Mã (ID)
- Ngôn ngữ (Language)
- Tóm tắt (Summary)
- Từ khóa (Keywords)
- Loại tài liệu (Document Type)
- Bản quyền (License)
- Tác giả (Authors)
- Chịu trách nhiệm xuất bản (Copyright Holders)
- Chỉnh lý (Maintainers)
- Phiên bản (Version)
- Ngày tạo (Created)
- Ngày chỉnh sửa (Revised)

Và mỗi tác giả đều có trang chứa thông tin cá nhân. Thông tin trong trang cá nhân có thể chứa trường dữ liệu:

- Đơn vị công tác (địa chỉ liên lạc) của tác giả
- Câu hỏi tiếng Việt

Xét các câu hỏi bằng tiếng Việt liên quan đến các thành phần thuộc đối tượng thông tin (metadata, tác giả) từ trang học liệu mở.

2.2 Phương pháp nghiên cứu

2.2.1 Phương pháp xây dựng ontology

Hầu hết thông tin của web hiện tại hầu như không có cấu trúc và vẫn còn nhiều hạn chế trong việc tìm kiếm thông tin (dựa trên từ khóa), rút trích thông tin (tốn nhiều thời gian để duyệt), bảo trì thông tin (thông tin lỗi thời và các mâu thuẫn giữa các

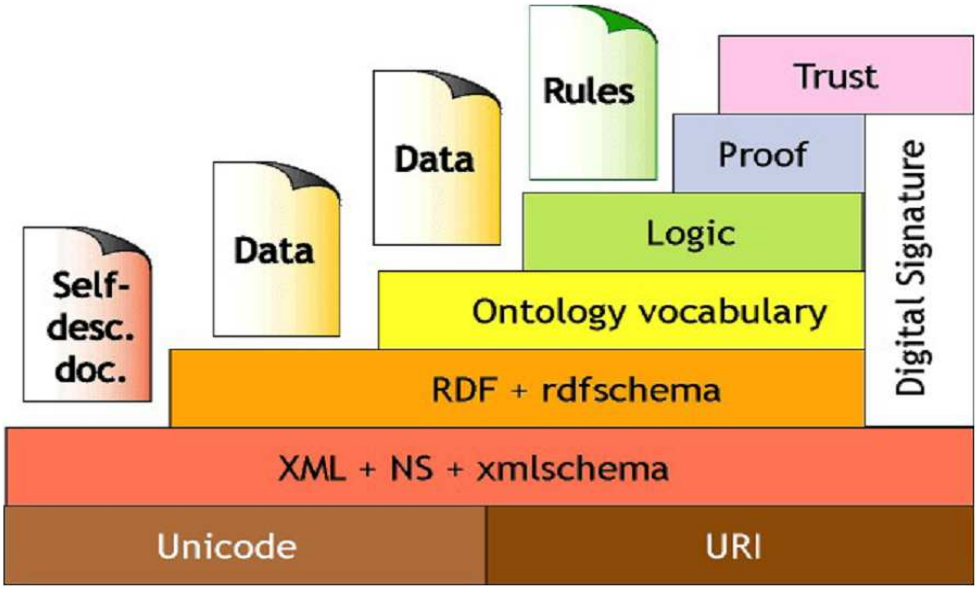
thuật ngữ không thể loại bỏ), khám phá thông tin (thu thập thông tin) và xem thông tin (bỏ qua những thông tin hệ thống web đó chính là web ngữ nghĩa, nhằm tạo ra những hệ thống quản lý tri thức tiên tiến hơn). Vì vậy, cần có một hệ thống web mới ra đời để khắc phục những hạn chế trên:

- Tri thức sẽ được tổ chức theo các khái niệm và ngữ nghĩa của nó.
- Các công cụ tự động sẽ hỗ trợ việc bảo trì bằng cách kiểm tra những mâu thuẫn và rút trích tri thức mới.
- Thay thế việc tìm kiếm dựa trên từ khóa bằng việc trả lời câu hỏi (hỏi-đáp): những tri thức được yêu cầu sẽ được rút trích, lấy ra và biểu diễn theo cách thuận lợi nhất cho con người.
- Hỏi-đáp đối với một vài tài liệu sẽ được hỗ trợ.
- Có thể xác định người mà xem một phần thông tin (thậm trí một phần tài liệu).

Sự phát triển web ngữ nghĩa diễn ra theo từng bước, mỗi bước xây dựng nên một tầng nằm trên tầng khác. Trong quá trình xây dựng một tầng web ngữ nghĩa, cần tuân theo 2 nguyên tắc:

- Tính tương thích từ trên xuống (downward compatibility): các tác tử (Agent) nhận biết hoàn toàn một tầng cũng có thể suy diễn và sử dụng thông tin được viết ở mức thấp hơn. Ví dụ, các Agent hiểu ngữ nghĩa của OWL có thể sử dụng các thông tin được viết trong RDF và RDF Schema.
- Sự thông hiểu từng phần từ dưới lên (upward partial understanding): các Agent nhận biết hoàn toàn một tầng thì ít nhất có thể sử dụng một phần thông tin ở mức cao hơn. Ví dụ, Agent chỉ hiểu RDF và ngữ nghĩa RDF Schema có thể biểu diễn tri thức được viết ở một phần nào đó trong OWL.

Hình 1 là kiến trúc phân tầng của web ngữ nghĩa (do Tim Berners-Lee đề ra) mô tả những tầng chính và việc thiết kế web ngữ nghĩa:



Hình 1.1 Kiến trúc phân tầng của web ngữ nghĩa

(Nguồn từ: A Semantic web Primer (2004), Grigoris Antoniou and Frank van Harmelen, page 18, figure 1.3)

- Tầng XML: môt ngôn ngữ cho phép tạo ra các tài liệu web có cấu trúc với từ vựng do người dùng định nghĩa. XML phù hợp với việc gởi các tài liệu thông qua web.
- Tầng RDF và RDF Schema: RDF là mô hình dữ liệu cơ bản nhằm viết các phát biểu cơ bản về các đối tượng web. Mô hình dữ liệu RDF không dựa trên XML, nhưng RDF có cú pháp dựa trên XML. RDF Schema được xem là ngôn ngữ cơ bản để viết ontology. Tuy nhiên, cũng cần có nhiều ngôn ngữ ontology mở rộng mạnh hơn để biểu diễn các mối quan hệ phức tạp hơn giữa các đối tượng web.
- Tầng Ontology vocabulary (ontology): cung cấp bộ từ vựng chung, khả năng biểu diễn ngữ nghĩa cho tài nguyên web và khả năng hỗ trợ lập luận.
- Tầng Logic: dùng để tăng cường ngôn ngữ ontology hơn và cho phép viết các tri thức khai báo ứng dụng chuyên biệt (application-specific declarative knowledge).
- Tầng Proof: liên quan đến quá trình suy diễn sự thật cũng như sự biểu diễn các Proof trong ngôn ngữ web (từ các lớp thấp hơn) và sự xác nhận tính hợp lệ của Proof.

- Tầng Trust: sử dụng các ký hiệu kỹ thuật số (digital signatures) và những loại tri thức khác. Đôi khi web Trust được sử dụng để biểu diễn Trust đó có sự sắp xếp theo cách hỗn độn và được sắp xếp theo một kiểu nhất định như chính WWW. Trust là mức cao nhất và là khái niệm cốt yếu.

2.2.1.1 Ontology [1][2][3]

A. Thuật ngữ ontology

Từ ontology có nguồn gốc từ tiếng Hy Lạp: ontos nghĩa là “being” (tồn tại, sống, thực sự) và logos nghĩa là “word” (từ) [1]. Trong triết học, ontology thuộc một lĩnh vực nhỏ của triết học, nghiên cứu về sự tồn tại của tự nhiên, chủ yếu xác định những thuật ngữ thông dụng nhất, những thứ tồn tại thực sự, và bằng cách nào để mô tả chúng [2].

B. Định nghĩa ontology

Trong trí tuệ nhân tạo và trong tin học cũng như trong khoa học máy tính, có rất nhiều định nghĩa về ontology, nhưng một trong những định nghĩa súc tích nhất, đó là theo T.R. Gruber (An ontology is an specification of a conceptualization) [1], sau đó được cải tiến bởi R. Studer: “ontology là một sự cụ thể hóa hình thức và tường minh của trừu tượng hóa” (An ontology is an explicit and formal specification of a conceptualization) [2]. Trừu tượng hóa (conceptualization) có nghĩa là một ý niệm trừu tượng (abstract), một quan điểm đơn giản hóa của thế giới. Còn cụ thể hóa (specification) có nghĩa là một sự biểu diễn khai báo và hình thức. Trong cấu trúc dữ liệu, việc biểu diễn ontology, các khái niệm và các ràng buộc cần phải được khai báo, tường minh và có sử dụng ngôn ngữ hình thức [1].

Thường thì khi nói đến ontology là nói đến một ontology. Một ontology là một mô hình dữ liệu biểu diễn một lĩnh vực (miền), và dùng nó để suy luận về các đối tượng trong lĩnh vực đó và mối quan hệ giữa chúng [2].

C. Thành phần của ontology

Ontology cung cấp một bộ từ vựng (terminology) chung bao gồm các khái niệm (thuật ngữ), các thuộc tính (property) quan trọng, các định nghĩa về các khái niệm và các thuộc tính này, và mối quan hệ giữa chúng. Mối quan hệ thường gồm các phân cấp lớp. Một phân cấp chỉ rõ một lớp C là lớp con của lớp C’ khác nếu mọi đối tượng trong C cũng đều thuộc trong C’. Ngoài ra, ontology còn cung cấp các ràng buộc, và các ràng buộc này có thể được xem như các giả định nền tảng của bộ từ vựng và được sử dụng trong một miền nào đó [2].

Bộ từ vựng của ontology được xây dựng dựa trên tầng RDF và RDFS trong kiến trúc phân tầng của web ngữ nghĩa. Nó cung cấp khả năng biểu diễn ngữ nghĩa cho tài nguyên web và có khả năng hỗ trợ lập luận [2].

Cá thể (individual): cá thể có thể là các đối tượng cụ thể như con người, động vật, cái bàn… cũng như các cá thể trừu tượng như các thành viên hay các từ. Một ontology có thể không chứa cá thể nào.

Lớp (class): là một nhóm, tập hợp các đối tượng, có thể chứa các cá thể, các lớp khác. Một lớp có thể chứa các lớp con, có thể là một lớp tổng quát (chứa mọi thứ), hoặc chỉ chứa các cá thể riêng lẻ. Một lớp có thể xếp gộp hoặc được xếp gộp vào các lớp khác.

Thuộc tính (property): dùng để mô tả các đối tượng trong ontology. Mỗi một thuộc tính đều có tên và giá trị của thuộc tính đó. Thuộc tính dùng để lưu trữ các thông tin của đối tượng. Giá trị của một thuộc tính có thể là các kiểu dữ liệu phức tạp.

Mối liên hệ (relationship): là một thuộc tính của một đối tượng nào đó trong ontology. Lớp này có thể gộp vào lớp kia tạo nên một kiểu quan hệ xếp gộp, cho biết đối tượng nào là thành viên của lớp nào.

D. Tại sao cần xây dựng Ontology? [1][2][3]

Vì ontology cung cấp bộ từ vựng chung chứa những thông tin (dữ liệu) có cấu trúc về một miền nào đó, có khả năng biểu diễn ngữ nghĩa cho tài nguyên web, có khả năng hỗ trợ lập luận, và thông tin (dữ liệu) của ontology máy có thể hiểu và xử lý được, nghĩa là ontology cung cấp một kiến trúc quan trọng cho việc chuyển web của thông tin và dữ liệu sang web của tri thức (web ngữ nghĩa), nên cần phải xây dựng ontology để có thể:

- Chia sẻ hiểu biết cấu trúc của thông tin giữa người dùng với nhau hoặc giữa các ứng dụng
- Tái sử dụng miền tri thức nào đó
- Tạo ra miền dữ liệu rõ ràng, đơn giản, dễ nhớ và dễ sử dụng
- Phân tích, rút trích thông tin trong miền tri thức nào đó

E. Xây dựng ontology như thế nào? [2][3]

Phát triển một ontology thường là một quy trình lặp lại. Nền tảng để xây dựng bộ từ vựng ontology là tầng RDF và RDF Schema (RDFS).

Các bước xây dựng ontology:

- **Xác định miền (domain) và phạm vi (scope) của ontology**
  - Phát triển một ontology giống như định nghĩa một bộ dữ liệu và cấu trúc của chúng cho chương trình khác để sử dụng. Và ontology là một mô hình của một miền cụ thể, được xây dựng với mục đích cụ thể.
  - Những câu hỏi cơ bản phải trả lời trong bước này:
    - Miền mà ontology sẽ chứa là gì?
    - Chúng ta sử dụng ontology với mục đích gì?
    - Ontology cung cấp các câu trả lời cho những loại câu hỏi nào?
    - Ai sẽ sử dụng và bảo trì ontology?

- **Xem xét sự tái sử dụng các ontology đã tồn tại**
  - Việc xem xét các ontology đã có cũng rất quan trọng, giúp chúng ta có thể lọc và mở rộng các nguồn đã có cho mục đích tạo ontology của chính chúng ta. Ngoài ra, việc tái sử dụng ontology đã có đôi khi rất cần thiết khi hệ thống chúng ta cần tương tác với các ứng dụng khác.
  - Chúng ta có thể tái sử dụng các ontology sẵn có trong cùng một miền miễn sao phù hợp với mục đích tạo ontology của chúng ta. Hơn nữa, việc chuyển một ontology từ dạng này sang dạng khác thì thường không dễ. Do đó, việc xem xét tái sử dụng các ontology có sẵn giúp tiết kiệm thời gian xây dựng ontology của chúng ta.

- Tuy nhiên, nếu ontology không có sẵn, chúng ta phải tạo ra ontology mới.

➢ **Liệt kê các thuật ngữ quan trọng**

- Xác định các thuật ngữ có liên quan.
- Viết ra một danh sách các thuật ngữ liên quan chưa có cấu trúc. Thường thì các danh từ cho tên lớp (class), động từ (hoặc cụm động từ) cho tên thuộc tính (property), chẳng hạn như “is part of”, “has component”.
- Xác định các mối quan hệ giữa các thuật ngữ, phân cấp các lớp và định nghĩa các thuộc tính của các khái niệm có quan hệ với nhau.

➢ **Xác định lớp và phân loại lớp (taxonomy)**

- Sắp xếp, phân loại các thuật ngữ có liên quan theo cấp bậc, theo nhóm, có thể theo mô hình từ trên xuống (top-down) hoặc từ dưới lên (bottom-up) hoặc phối hợp cả hai (combination).
- Mô hình top-down: sẽ bắt đầu từ việc định nghĩa các khái niệm chung nhất, kế đến là cụ thể của các khái niệm. Ví dụ như, lớp “rượu vang” (wine) là một khái niệm chung nhất, còn các rượu vang trắng (white wine), rượu vang hồng (rose wine) và rượu vang đỏ (red wine) là lớp con của các khái niệm cụ thể. Như vậy, rượu vang (wine) sẽ chứa (gồm) các rượu vang trắng, vang hồng và vang đỏ.
- Mô hình bottom-up: sẽ bắt đầu với việc định nghĩa các lớp cụ thể nhất, sau đó nhóm các lớp này lại thành những khái niệm chung hơn. Ví dụ như, đầu tiên chúng ta định nghĩa 2 lớp rượu: lớp rượu vang Pauillac và Margaux. Sau đó, tạo ra một siêu lớp (superclass) chung cho 2 lớp này là Medoc.
- Mô hình kết hợp (combination): đây là mô hình kết hợp giữa mô hình top-down và bottom-up. Đầu tiên, chúng ta định nghĩa các

khái niệm nổi bật hơn. Sau đó, khái quát hóa và cụ thể hóa chúng tương ứng một cách phù hợp. Chúng ta có thể bắt đầu với một vài khái niệm mức đỉnh (top) (ví dụ như wine) và một vài khái niệm cụ thể (ví dụ như Margaux). Sau đó, chúng ta nối chúng với một khái niệm mức trung (middle) (ví dụ như Medoc). Như vậy, chúng ta tạo ra các lớp rượu vang và cũng tạo ra nhiều khái niệm mức trung.

- ❖ Tùy vào mục đích cụ thể mà có thể sử dụng thích hợp các mô hình trên, có thể sử dụng một mô hình hoặc kết hợp nhiều mô hình đó lại với nhau.
- ❖ Chúng ta có thể phân cấp các thuật ngữ thành các lớp. Sau đó, các lớp này tiếp tục được phân cấp (phân cấp lớp). Tuy nhiên, sự phân cấp này cần tuân theo: nếu một lớp A là một lớp cha (superclass) của lớp B, thì mọi thực thể (instance) của B cũng là thực thể của A.

➢ **Xác định các thuộc tính (properties)**

- ❖ Khi chúng ta định nghĩa một vài lớp, chúng ta phải mô tả cấu trúc bên trong của các khái niệm.
- ❖ Đối với mỗi thuộc tính, chúng ta phải xác định nó mô tả lớp nào. Các thuộc tính này có thể là những thuộc tính được gắn vào các lớp. Thông thường, có một vài loại thuộc tính đối tượng như thuộc tính bên trong (intrinsic), thuộc tính bên ngoài (extrinsic).
- ❖ Thuộc tính (slot/property) biểu diễn các mối quan hệ giữa hai cá thể (individual). Thuộc tính liên kết cá thể của miền (domain) và cá thể của phạm vi (range).
- ❖ Có 3 loại thuộc tính:
  - Thuộc tính đối tượng (object property): liên kết cá thể (individual) này với cá thể khác. Các loại thuộc tính đối tượng:

- ✓ Thuộc tính đảo (inverse): ví dụ như has_parent là đảo của has_child.
- ✓ Thuộc tính hàm (functional): ví dụ như has_birth_mother.
- ✓ Thuộc tính bắc cầu (transitive): ví dụ như has_anchestor.
- ✓ Thuộc tính đối xứng (symmetric): ví dụ như has_sibling.

- Thuộc tính kiểu dữ liệu (data type property): liên kết một cá thể (individual) với giá trị kiểu dữ liệu của XML Schema hoặc kiểu nguyên thể (literal) của rdf. Ví dụ như has_birth_of_date.
- Thuộc tính chú thích (annotation property): được dùng để thêm thông tin (metadata) vào các lớp, các cá thể và thuộc tính kiểu dự liệu hoặc thuộc tính đối tượng.

❖ Tất cả các lớp con của một lớp kế thừa thuộc tính của lớp đó. Một thuộc tính có thể được gắn vào lớp tổng quát nhất (có thể mang thuộc tính đó).

❖ Việc xác định các thuộc tính còn giúp phát hiện những mâu thuẫn, những khái niệm sai.

➢ **Xác định các facet (giới hạn của thuộc tính hay giới hạn luật)**

❖ Facet được dùng để biểu diễn thông tin về các thuộc tính (slots), đôi khi còn được gọi là các giới hạn luật.

❖ Các thuộc tính có thể có các facet khác nhau mô tả các kiểu giá trị, các giá trị được phép, các số của giá trị (cardinality) và các đặc tính (feature) khác mà thuộc tính có thể nhận.

❖ Một số facet:

- Số của giá trị (cardinality)

- ✓ Cardinality biểu diễn số chính xác của giá trị được xác nhận cho thuộc tính của lớp đó, và nó xác định có bao nhiêu giá trị mà thuộc tính có.
- ✓ Nhiều hệ thống chỉ phân biệt giữa cardinality đơn (nhiều nhất một giá trị) và cardinality đa (bất kỳ số giá trị nào) (minimum cardinality và maximum cardinality). Minimum cardinality N có nghĩa là một thuộc tính phải có ít nhất N giá trị. Maximun cardinality M có nghĩa là một thuộc tính có tối đa M giá trị.

- Kiểu giá trị (value type)
  - ✓ Facet của kiểu giá trị mô tả kiểu giá trị nào có thể cho vào thuộc tính. Một số các kiểu giá trị:
    - ➔ Kiểu chuỗi (String)
    - ➔ Kiểu số (Number)
    - ➔ Kiểu Boolean
    - ➔ Kiểu liệt kê (Enumerated)
    - ➔ Kiểu instance

➢ **Định nghĩa các thực thể (instances)**

❖ Thực thể là đối tượng hoặc cá thể của lớp. Để định nghĩa một thực thể riêng biệt của một lớp, ta thực hiện như sau:

- Chọn một lớp.
- Tạo một thực thể riêng biệt của lớp đó.
- Thay thế bằng các giá trị thuộc tính.

➢ **Kiểm tra những bất thường (anomalies)**

❖ Nhằm phát hiện tính không nhất quán (mâu thuẫn) trong chính ontology hoặc trong bộ thực thể (set of instances). Những mâu

thuẫn đó có thể là: sự không tương thích với định nghĩa miền và phạm vi (range) cho tính bắc cầu (transitive), tính đối xứng (symmetric) hay các thuộc tính đảo, các thuộc tính số giá trị, các giá trị thuộc tính có thể xung đột với các giới hạn của domain và range.

- ❖ Kiểm tra những bất thường có thể bằng bộ lập luận (reasoner): Pellet, Racer…

Tuy nhiên, trên thực tế, việc phát triển ontology gồm các bước sau:

- ➢ Xác định các lớp trong ontology.
- ➢ Sắp xếp các lớp theo cấp bậc, theo nhóm (lớp cha-lớp con).
- ➢ Xác định các thuộc tính và mô tả các giá trị được phép cho các thuộc tính này.
- ➢ Sắp xếp hoàn chỉnh các giá trị của các thuộc tính cho các thực thể (instances).

F. Kết luận [2]

Như vậy, các ontology rất có ích cho sự tổ chức và điều hướng các website. Ontology cũng cải thiện độ chính xác trong việc tìm kiếm trên web. Các công cụ tìm kiếm có thể tìm những trang web chỉ liên liên đến khái niệm trong một ontology thay vì thu thập tất cả các trang web thường chứa các từ khóa rất mơ hồ, nhập nhằng và không chính xác lắm. Theo cách này, những khác nhau về thuật ngữ giữa các trang web và những truy vấn có thể được khắc phục.

Hơn nữa, tìm kiếm web có thể tìm được những thông tin đặc thù và khái quát. Nếu không thành công trong việc tìm kiếm các tài liệu quan tâm khi đặt câu hỏi, thì các công cụ tìm kiếm sẽ đề nghị người dùng đặt câu hỏi tổng quát hơn. Thậm chí, máy có thể hiểu được và thực hiện các truy vấn đó trước để giảm bớt thời gian đáp ứng trong trường hợp người dùng chấp nhận sự đề nghị đó. Hoặc nếu có quá nhiều câu trả lời được rút trích, thì các công cụ tìm kiếm có thể đưa ra các đề nghị có tính chuyên môn hơn cho người dùng.

Tuy nhiên, không có một ontology nào đúng cho mọi miền. Chất lượng của ontology được xác định thông qua các ứng dụng sử dụng nó.

2.2.1.2 Các ngôn ngữ ontology cho web [2]

- XML: cung cấp cú pháp cho các tài liệu có cấu trúc nhưng không yêu cầu ngữ nghĩa đối với các tài liệu này web.
- XML Schema: là ngôn ngữ để giới hạn cấu trúc của tài liệu XML.
- RDF: là mô hình dữ liệu cho các đối tượng (tài nguyên_resource) và các mối quan hệ giữa chúng. Đây là một ngữ nghĩa đơn giản cho mô hình dữ liệu này, và mô hình dữ liệu này được biểu diễn trong cú pháp của XML.
- RDF Schema: là ngôn ngữ mô tả từ vựng để mô tả các thuộc tính (property) và các lớp (class) của tài nguyên RDF, cùng với ngữ nghĩa cho các phân cấp khái quát hóa của các lớp và các quan hệ đó.
- OWL: là ngôn ngữ mô tả từ vựng phong phú để mô tả các thuộc tính và các lớp, các mối quan hệ giữa các lớp (như disjointness), số của giá trị (cardinality), tính tương đương (equality), định kiểu thuộc tính, đặc tính của thuộc tính (đối xứng) và các lớp đếm.

A. RDF và RDF Schema

a. RDF (Khung mô tả tài nguyên)

a.1 Giới thiệu

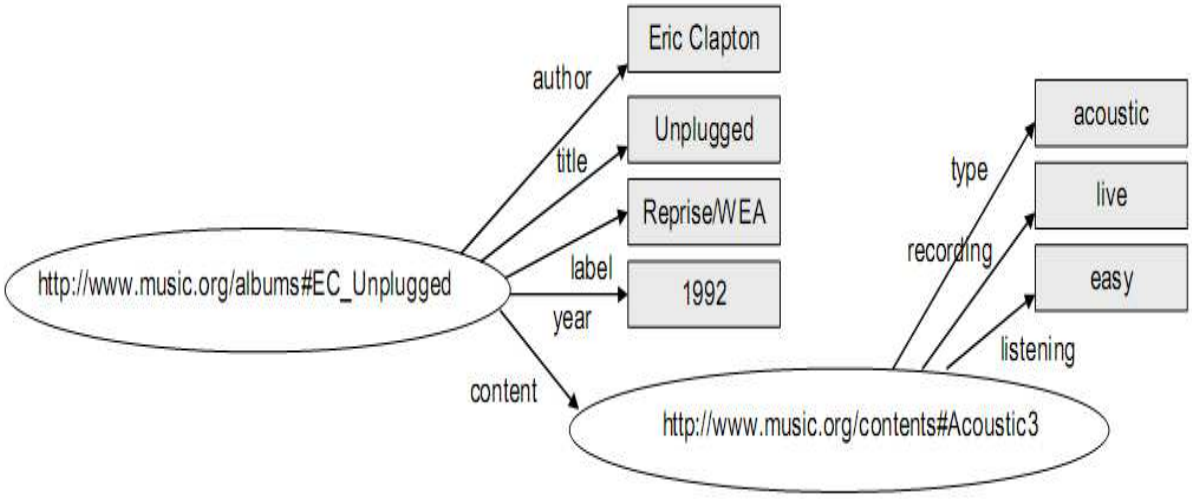
XML là một siêu ngôn ngữ (metalanguage) phổ biến để định nghĩa định dạng (markup). Nó cung cấp cú pháp dễ dùng và một bộ công cụ như bộ phân tích cú pháp (parser) để trao đổi dữ liệu và siêu dữ liệu (metadata) giữa các ứng dụng nhờ sử dụng XML Schema. Tuy nhiên, XML không cung cấp bất kỳ ngữ nghĩa nào của dữ liệu. Vì vậy, cần có một mô hình chuẩn nào đó để biểu diễn các sự kiện của tài nguyên web. Mô hình chuẩn đó là RDF và RDF Schema.

Về cơ bản, RDF là một mô hình dữ liệu. Đó là mô hình bộ ba đối tượng-thuộc tính-giá trị (object-attribute-value triple), được gọi là một phát biểu (statement). Mỗi

phát biểu được biểu diễn dưới dạng một tài nguyên (resource hoặc object), một thuộc tính (property/attribute) của nó và một giá trị (value) của thuộc tính. Giá trị có thể là nguyên thể (literal) hoặc tài nguyên khác.

Một số bộ ba resource-property-value của RDF và lược đồ tương ứng với các bộ ba đó:

RESOURCE	PROPERTY	VALUE
http://www.music.org/albums#EC_Unplugged	author	"Eric Clapton"
http://www.music.org/albums#EC_Unplugged	title	"Unplugged"
http://www.music.org/albums#EC_Unplugged	label	"Reprise/WEA"
http://www.music.org/albums#EC_Unplugged	year	"1992"
http://www.music.org/albums#EC_Unplugged	content	http://www.music.org/contents#Acoustic3
http://www.music.org/contents#Acoustic3	type	"acoustic"
http://www.music.org/contents#Acoustic3	recording	"live"
http://www.music.org/contents#Acoustic3	listening	"easy"



Hình 2.1 Lược đồ bộ ba (resource, property, value) của RDF

(Nguồn từ: Dragan Gas˘evic´, Dragan Djuric´, Vladan Devedz˘ic’ (2006), Model Driven Architecture and Ontology Development, fig.3.4, trang 84)

Những bộ ba trên cũng có thể được biểu diễn như sau (có sử dụng cú pháp của XML):

```
<Album rdf:ID="EC_Unplugged"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.music.org/albums#"
        xml:base="http://www.music.org/albums">
   <author>Eric Clapton</author>
   <title>Unplugged</title>
   <label>Reprise/WEA</label>
   <year>1992</year>
   <content>
     <Content rdf:ID="Acoustic3"
              xmlns="http://www.music.org/contents#">
       <type>acoustic</type>
       <listening>easy</listening>
       <recording>live</recording>
     </Content>
   </content>
</Album>
```

Hình 2.2 Biểu diễn bộ ba (resource, property, value) dưới dạng XML

(Nguồn từ: Dragan Gasˇevic´, Dragan Djuric´, Vladan Devedzˇic' (2006), Model Driven Architecture and Ontology Development, fig.3.4, trang 84)

Mô hình RDF chỉ cung cấp một cơ cấu miền độc lập để mô tả các tài nguyên riêng lẻ. Nó không ưu tiên ngữ nghĩa của bất kỳ miền ứng dụng nào, và cũng không tạo ra giả định nào về miền cụ thể nào đó. RDF thường dùng để mô tả các thực thể của các ontology, trong khi RDF Schema mã hóa các ontology.

a.2 Các khái niệm cơ bản

Tài nguyên (resource)

Có thể xem tài nguyên như một đối tượng, một " thứ" mà chúng ta muốn nói. Tài nguyên có thể là tác giả, sách, nhà xuất bản, địa điểm, con người…

Mỗi resource có một URI (Universal Resource Identifier_bộ định danh tài nguyên chung). Một URI có thể là một URL (Unified Resource Locator_bộ định vị tài nguyên hợp nhất hoặc địa chỉ web) hoặc một vài loại trình định danh (identifier) khác. Thông thường, chúng ta cho rằng một URI là trình định danh của tài nguyên web.

Thuộc tính (property)

Thuộc tính là một kiểu tài nguyên đặc biệt, mô tả các mối quan hệ giữa các tài nguyên, chẳng hạn như “written by”, “age”, “title”. Thuộc tính trong RDF cũng được xác định bởi URI (nhưng thực tế là URL).

Các phát biểu (statement)

Các phát biểu xác định thuộc tính của tài nguyên. Một phát biểu là một bộ ba đối tượng-thuộc tính-giá trị (object-attribute-value triple), gồm một tài nguyên, một thuộc tính và một giá trị. Giá trị có thể là hoặc tài nguyên hoặc nguyên thể (literal). Literal là các giá trị nguyên tử (atomic value) hoặc chuỗi (string).

a.3 Cú pháp của RDF

Cú pháp của RDF dựa trên cú pháp của XML. Tài liệu RDF gồm một phần tử “rdf:RDF”, nội dung (content) là số lượng các mô tả (descriptions).

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XLMSchema#"
    xmlns:uni="http://www.mydomain.org/uni-ns#">
    <rdf:Description  rdf:about="949352">
        <uni:name>Grigoris Antoniou</uni:name>
        <uni:title>Professor</uni:title>
    </rdf:Description>
    <rdf:Description  rdf:about="949318">
        <uni:name>David Billington</uni:name>
        <uni:title>Associate Professor</uni:title>
        <uni:age  rdf:datatype="&xsd;integer">2
    </rdf:Description>
    <rdf:Description  rdf:about="949111">
        <uni:name>Michael Maher</uni:name>
        <uni:title>Professor</uni:title>
    </rdf:Description>
```

</rdf:RDF>

Không gian tên (namespace) trong XML được sử dụng trong những mục đích chung; còn trong RDF, không gian tên là tài liệu RDF định nghĩa tài nguyên, được sử dụng trong việc nhập tài liệu RDF, và cho phép tái sử dụng tài nguyên bởi người khác để chèn thêm các đặc tính vào trong các tài nguyên này.

Thuộc tính "rdf:about" của phần tử "rdf:Description" được dùng để chỉ rằng đối tượng về phát biểu nào được tạo ra đã được định nghĩa ở nơi khác.

Nội dung của phần tử "rdf:Description" được gọi là các phần tử thuộc tính. Ví dụ như, trong mô tả:

<rdf:Description rdf:about="CIT3116">
<uni:courseName>Knowledge Representation</uni:courseName>
<uni:isTaughtBy>Grigoris Antoniou</uni:isTaughtBy>
</rdf:Description>

Các phần tử thuộc tính của mô tả phải được đọc một cách liên tục.

Thuộc tính "rdf:resource"

Được sử dụng trong trường hợp có sự trùng tên ngẫu nhiên. Thuộc tính "rdf:resource" dùng để phân biệt các trường hợp có tên giống nhau nhưng là những người khác nhau.

<rdf:Description rdf:about="CIT1111">
<uni:courseName><Discrete Mathematics></uni:courseName>
<uni:isTaughtBy **rdf:resource="#949318**"/>
</rdf:Description>

<rdf:Description rdf:ID="**#949318**">

<uni:name>David Billington</uni:name>
<uni:title>Associate Professor</uni:title>
</rdf:Description>

Phần tử "rdf:Description"

Các phần tử "rdf:Description" có thể được lồng vào nhau.

```
<rdf:Description  rdf:about="CIT1111">
        <uni:courseName>Discrete Mathematics</uni:courseName>
        <uni:isTaughtBy>
                <rdf:Description  rdf:about="949318">
                        <uni:name>David Billington</uni:name>
                        <uni:title>Associate Professor</uni:title>
                </rdf:Description>
        </uni:isTaughtBy>
</rdf:Description>
```

Phần tử "rdf:type"

Cho phép chúng ta đưa một vài cấu trúc vào tài liệu RDF.

```
<rdf:Description rdf:about="CIT1111">
<rdf:type rdf:resource="&uni;course"/>
        <uni:courseName>Discrete Mathematics</uni:courseName>
        <uni:isTaughtBy rdf:resource="949318"/>
</rdf:Description>
```

Phần tử chứa (container elements)

Các phần tử chứa được dùng để thu thập các tài nguyên hoặc các thuộc tính mà chúng ta muốn tạo các phát biểu như một tổng thể. Có 3 loại phần tử chứa có sẵn trong RDF:

- "rdf:Bag": phần tử chứa không có thứ tự, chứa nhiều sự kiện (occurrence).
- "rdf:Seq": phần tử chứa có thứ tự, chứa nhiều sự kiện.
- "rdf:Alt": một bộ các lựa chọn (alternative).

Nội dung của các phần tử chứa là những phần tử được đặt tên là "rdf:_1", "rdf:_2"…

Ví dụ:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:uni="http://www.mydomain.org/uni-ns#">
    <uni:lecturer  rdf:about="949352"
        uni:name="Grigoris Antoniou"
        uni:title="Professor">
        <uni:coursesTaught>
            <rdf:Bag>
                <rdf:_1  rdf:resource="CIT1112"/>
                <rdf:_2  rdf:resource="CIT3116"/>
            </rdf:Bag>
        </uni:coursesTaught>
    </uni:lecturer>
</rdf:RDF>
```

Thay vì rdf:_1, rdf:_2 ..., có thể viết rdf:li.

```
<uni:course    rdf:about="CIT1111"
        uni:courseName="Discrete Mathematics">
    <uni:lecturer>
        <rdf:Alt>
            <rdf:li  rdf:resource="949352"/>
            <rdf:li  rdf:resource="949318"/>
        </rdf:Alt>
    </uni:lecturer>
</uni:course>
```

a.4 Ngữ nghĩa tiên đề cho RDF

Để máy có thể sử dụng được (machine accessible) ngữ nghĩa tường minh, chúng ta cần tạo ra ngữ nghĩa đó, mô tả ngữ nghĩa của RDF trong ngôn ngữ hình thức giống như logic, và cung cấp một bộ lập luận (reasoner) tự động để xử lý các công thức logic.

Giới thiệu

Tất cả các từ mẫu (primitives) trong RDF được biểu diễn bởi các hằng (constant): Resource, Class, Property, subClassOf… Một vài vị từ có sẵn được sử dụng để biểu diễn các mối quan hệ giữa chúng.

Hầu hết các tiên đề cung cấp thông tin định kiểu. Chẳng hạn như: Type(subClassOf, Property): cho biết 'subClassOf" là một thuộc tính. Các tên biến bắt đầu bằng dấu "?".

Vị từ cơ bản (predicate)

PropVal(P, R, V): một vị từ với 3 đối số, dùng để biểu diễn một phát biểu RDF với tài nguyên R, thuộc tính P và giá trị V.

Type(R, T): viết tắt của PropVal(type, R, T), chỉ rõ tài nguyên R có kiểu T

Type(?r, ?t) $\leftrightarrow$ PropVal(type, ?r, ?t)

Các phát biểu

Một phát biểu RDF (là một bộ ba) (P, R, V) được biểu diễn như *PropV al(P,R,V)*

Lớp (class)

Tất cả các lớp là những thực thể (instance) của Class. Với các hằng (constant): Class, Resource, Property, Literal., các lớp có kiểu *Class*:

Type(Class,Class)
Type(Resource,Class)
Type(Property,Class)
Type(Literal,Class)

Tài nguyên (resource) là lớp phổ biến nhất. Mỗi đối tượng, mỗi lớp và mỗi thuộc tính là những tài nguyên:

Type(?p, Property) → Type(?p,Resource)
Type(?c,Class) → Type(?c,Resource)

Vị từ trong phát biểu RDF phải là một thuộc tính:

PropVal(?p, ?r, ?v) → Type(?p, Property)

Thuộc tính "type"

"type" là một thuộc tính:

Type(type, Property)

Phát biểu này tương đương với PropVal(type, type, Property).

Thuộc tính "FuncProp"

Thuộc tính hàm là một thuộc tính là một hàm. Thuộc tính **"**FuncProp**"** liên kết một tài nguyên với tối đa một giá trị.

FuncProp "hằng" (constant) biểu diễn lớp của tất cả các thuộc tính hàm. P là một thuộc tính hàm nếu và chỉ nếu nó là một thuộc tính, và không có x, y1, và *y2* sao cho *P(x, y1), P(x, y2),* và $y1 \neq y2$

Type(?p, FuncProp) $\leftrightarrow$

Type(?p, Property) Λ $\forall$?r $\forall$?v1 $\forall$?v2
PropVal(?p, ?r, ?v1) Λ PropVal(?p, ?r, ?v2)
$\rightarrow$?v1 = ?v2

Các phát biểu cụ thể hoá (statements)

Phát biểu "constant" biểu diễn lớp của tất cả các phát biểu cụ thể hóa. Tất cả các phát biểu cụ thể hóa là các tài nguyên, và phát biểu là một thực thể (instance) của Class:

Type(?s, Statement) $\rightarrow$ Type(?s, Resource)

Type(Statement, Class)

Một phát biểu cụ thể hóa có thể được phân tách thành 3 phần của bộ ba RDF:

Type(?st, Statement) $\rightarrow$

$\exists$?p $\exists$?r $\exists$?v (PropVal(Predicate, ?st, ?p) Λ
PropVal(Subject, ?st, ?r) Λ PropVal(Object, ?st, ?v))

Chủ thể (subject), vị từ (predicate) và đối tượng (object) là các thuộc tính hàm. Nói cách khác, mỗi phát biểu có chính xác 1 chủ thể, một vị từ và 1 đối tượng:

Type(Subject, FuncProp)

Type(Predicate, FuncProp)

Type(Object, FuncProp)

Thông tin định kiểu của chúng là:

PropVal(Subject, ?st, ?r) →

(Type(?st, Statement) Λ Type(?r,Resource))

PropVal(Predicate, ?st, ?p) →

(Type(?st, Statement) Λ Type(?p, Property))

PropVal(Object, ?st, ?v) →

(Type(?st, Statement) Λ (Type(?v,Resource) V Type(?v,Literal)))

Tiền đề cuối cùng là nếu đối tượng như thuộc tính trong phát biểu RDF, thì nó phải áp dụng cho một phát biểu cụ thể hóa và có như giá trị hoặc là một tài nguyên hoặc là một literal.

Phần tử container

Tất cả các container là các tài nguyên:

Type(?c,Container) → Type(?c,Resource)

Các container là các danh sách (list):

Type(?c,Container) → list(?c)

Các container là các bag hay các sequence hay các alternative:

Type(?c,Container) ↔ (Type(?c,Bag) V Type(?c, Seq) V Type(?c,Alt))

Các bag và sequence là disjoint (tách biệt):

¬(Type(?x,Bag) Λ Type(?x, Seq))

Đối với mỗi số tự nhiên n > 0, có một bộ chọn (selector), sẽ chọn phần tử thứ n của một container. Đó là một thuộc tính hàm: Type(_n, FuncProp) và chỉ áp dụng cho các container:

PropVal(_n, ?c, ?o) → Type(?c,Container)

b. RDF Schema (Lược đồ RDF)

b.1 Giới thiệu

RDF là một ngôn ngữ phổ biến, cho phép người dùng mô tả các tài nguyên bằng cách sử dụng bộ từ vựng của chính người dùng. RDF không tạo ra các giả định về bất kỳ miền ứng dụng cụ thể nào, cũng không định nghĩa ngữ nghĩa của bất kỳ miền nào. RDF Schema sẽ giúp người dùng làm được điều đó.

XML Schema bắt buộc tài liệu XML phải có cấu trúc, trong khi RDF Schema định nghĩa từ vựng được dùng trong mô hình dữ liệu RDF. Trong RDFS, chúng ta có thể định nghĩa từ vựng, chỉ rõ các thuộc tính nào áp dụng cho các loại đối tượng nào và giá trị nào mà chúng có thể nhận, chỉ rõ các lớp và mô tả các mối quan hệ giữa các đối tượng, giữa các lớp.

b.2 Những khái niệm cơ bản

Lớp và thuộc tính

Một lớp có thể được xem như một bộ các phần tử. Các đối tượng riêng lẻ thuộc về một lớp được tham chiếu đến như các thực thể của lớp đó. Sử dụng “rdf:type” để định nghĩa mối quan hệ giữa các thực thể và các lớp trong RDF.

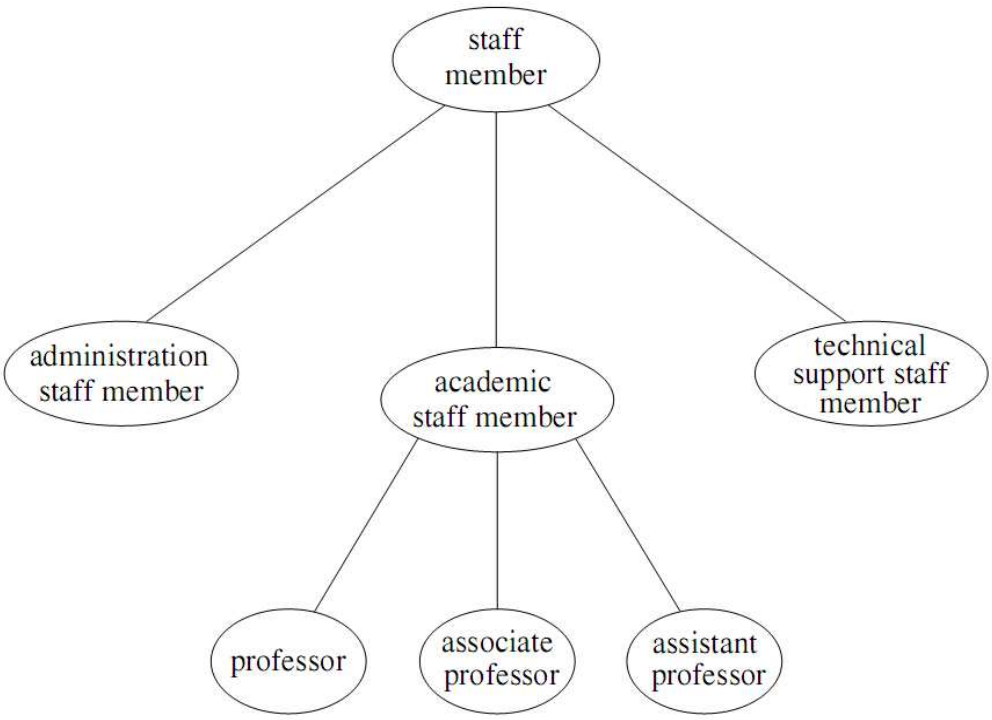
Trong các ngôn ngữ lập trình, việc định kiểu được sử dụng để ngăn ngừa sự vô nghĩa (nonsense) được tạo ra. Để giới hạn các giá trị của thuộc tính, ta giới hạn range của thuộc tính. Để giới hạn các đối tượng của thuộc tính, ta giới hạn domain của thuộc tính.

Các thuộc tính được định nghĩa một cách riêng biệt với các lớp. Mỗi thuộc tính được mô tả bởi rdfs:domain và rdfs:range, và nó giới hạn sự kết hợp các thuộc tính với các lớp. Một thuộc tính có thể được định nghĩa để có nhiều lớp.

Kế thừa và phân cấp lớp

Khi xác định các lớp, chúng ta cần phân loại, phân nhóm các lớp theo cấp bậc để thiết lập các mối quan hệ giữa chúng. Thông thường, A là một lớp con của B nếu mỗi thực thể (instance) của A cũng là một thực thể của B.

Trong RDF Schema, các lớp không nhất thiết phải liên kết với nhau hình thành nên một phân cấp nghiêm ngoặc. Một lớp có thể có nhiều lớp cha (superclass). Nếu lớp A là một lớp con của cả B1 và B2, thì mỗi thực thể của A là thực thể của cả B1 và B2.



Hình 2.3 Sự phân cấp lớp

(Nguồn từ: A Semantic web Primer (2004), Grigoris Antoniou and Frank van Harmelen, page 82, figure 3.5)

Lớp (class), sự kế thừa (inheritance) và thuộc tính (property) tuy có những điểm tương đồng, nhưng cũng có những điểm khác nhau. Trong lập trình hướng đối tượng, một lớp đối tượng xác định các thuộc tính. Để thay đổi một lớp, ta thêm các thuộc tính mới vào trong một lớp. Trong sự phân cấp lớp, các lớp kế thừa các thuộc tính của tổ tiên.

Có thể định nghĩa các thuộc tính mới áp dụng cho một lớp cũ mà không làm thay đổi lớp đó và cũng có thể sử dụng các lớp được định nghĩa bởi các lớp khác, sửa chúng lại sao cho phù hợp với các yêu cầu của người dùng thông qua các thuộc tính mới.

Phân cấp thuộc tính

Các mối quan hệ phân cấp giữa các lớp có thể được định nghĩa. Chẳng hạn như “is taught by”(được dạy bởi) là thuộc tính con của “involves” (có quan hệ): nếu môn học C được dạy bởi giảng viên A, thì C cũng có quan hệ với A. Sự đảo ngược của thuộc tính có thể đúng nhưng đôi khi có thể không đúng.

Thông thường, P là thuộc tính con của Q nếu Q (x, y) chứa P (x, y).

b.3 Ngữ nghĩa tiên đề cho RDF Schema

Để tạo ngữ nghĩa tường minh và máy có thể sử dụng được (machine accessible), chúng ta cần mô tả ngữ nghĩa của RDFS giống như logic. Do đó, cần có sự hỗ trợ của các bộ lập luận (reasoner) tự động để xử lý các công thức logic.

Giới thiệu

Tất cả các từ mẫu (primitives) trong RDF Schema được biểu diễn thông qua các hằng (constant): Resource, Class, Property, subClassOf… Một vài vị từ có sẵn được sử dụng như nền tảng để biểu diễn các mối quan hệ giữa các hằng.

Hầu hết các tiên đề cung cấp thông tin định kiểu. Chẳng hạn như: Type(subClassOf, Property): cho biết 'subClassOf" là một thuộc tính. Các tên biến bắt đầu bằng dấu "?".

Vị từ cơ bản (predicate)

PropVal(P, R, V): một vị từ với 3 đối số, dùng để biểu diễn một phát biểu RDF với tài nguyên R, thuộc tính P và giá trị V.

Type(R, T): viết tắt của PropVal(type, R, T), chỉ rõ tài nguyên R có kiểu T

Type(?r, ?t) $\leftrightarrow$ PropVal(type, ?r, ?t)

Lớp con và thuộc tính con

"subClassOf" là một thuộc tính:

Type(subClassOf, Property)

Nếu C là một lớp con của lớp C', thì tất cả các thực thể (instance) của C cũng là các thực thể của C':

$$\text{PropVal(subClassOf, ?c, ?c')} \leftarrow (\text{Type(?c,Class)} \wedge \text{Type(?c',Class)} \wedge \forall \text{?x(Type(?x, ?c)} \rightarrow \text{Type(?x, ?c')))}$$

Tượng tự đối với "subPropertyOf"; P là một thuộc tính con của P' nếu P' (x, y) chứa P (x, y):

$$\text{Type(subPropertyOf, Property)}$$
$$\text{PropVal(subPropertyOf, ?p, ?p')} \leftrightarrow (\text{Type(?p, Property)} \wedge \text{Type(?p',Property)} \wedge \forall \text{?r} \forall \text{?v(PropVal(?p, ?r, ?v)} \rightarrow \text{PropVal(?p', ?r, ?v)))}$$

Các ràng buộc (constraints)

Mỗi tài nguyên ràng buộc là một tài nguyên:

PropVal(subClassOf,ConstraintResource,Resource)

Các thuộc tính ràng buộc là tất cả các thuộc tính mà cũng là các tài nguyên ràng buộc:

Type(?cp,ConstraintProperty) ↔ (Type(?cp,ConstraintResource) Λ
Type(?cp, Property))

Domain và range là các thuộc tính ràng buộc:

Type(domain,ConstraintProperty)
Type(range,ConstraintProperty)

Domain và range lần lượt xác định domain và range của một thuộc tính. Domain của thuộc tính P là tập hợp của tất cả các đối tượng mà P áp dụng vào. Nếu domain của P là D, thì với mỗi P (x, y), x Є D.

PropVal(domain, ?p, ?d) → ∀?x∀?y(PropVal(?p, ?x, ?y) → Type(?x, ?d))

Range của thuộc tính P là tập hợp của tất cả các giá trị P có thể nhận. Nếu range của P là R, thì với mỗi P (x, y), y Є R.

PropVal(range, ?p, ?r) → ∀?x∀?y(PropVal(?p, ?x, ?y) → Type(?y, ?r))

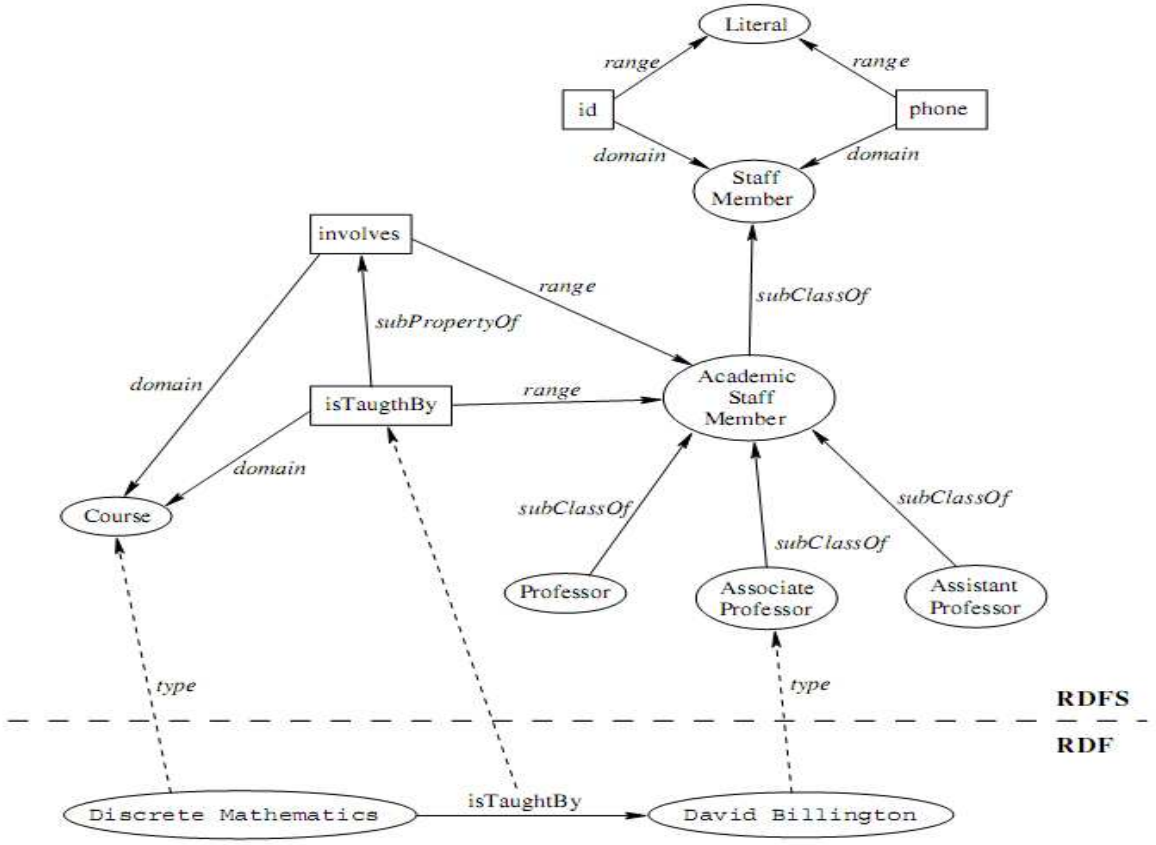
Các công thức có thể được suy ra từ ràng buộc trên:

PropVal(domain, range, Property)
PropVal(range, range,Class)
PropVal(domain, domain, Property)
PropVal(range, domain,Class)

c. Sự khác nhau giữa RDF và RDFS

RDF thường dùng để mô tả các thực thể (instances) của các ontology, trong khi RDF Schema mã hóa các ontology, sử dụng các thành phần Class, subClassOf, Property, subPropertyOf… để chỉ rõ các lớp, các quan hệ giữa các lớp, định nghĩa các thuộc tính và liên kết chúng với các lớp.

Xem hình 4: các hình khối vuông là các thuộc tính, các hình eclipse nằm trên đường gạch nối nằm ngang là các lớp, và các hình elip nằm dưới đường gạch nối nằm ngang là các thực thể (instance).



Hình 2.4 Các lớp RDF và RDFS

(Nguồn từ: A Semantic web Primer (2004), Grigoris Antoniou and Frank van Harmelen, page 84, figure 3.6)

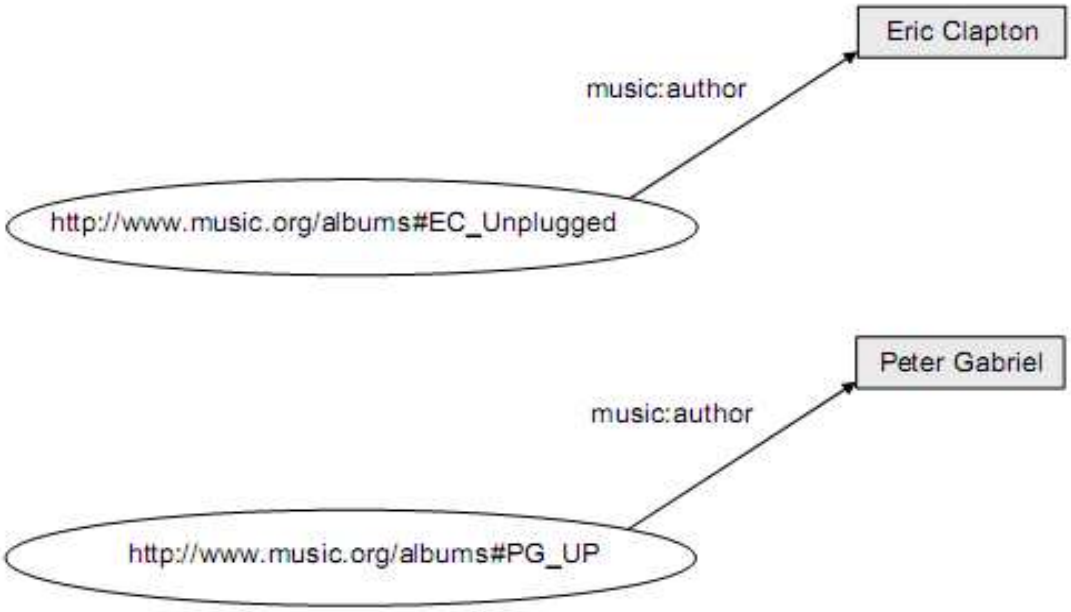
d. Ngôn ngữ truy vấn SPARQL [1]

Không giống OWL và RDF(S), SPARQL không dành cho ontology và sự biểu diễn tài nguyên, nhưng nó dành cho việc truy vấn dữ liệu web. Nó là ngôn ngữ truy vấn cho RDF. SPARQL được sử dụng để:

- Rút trích thông tin từ lược đồ RDF dưới dạng URIs, bNodes và các nguyên thể (literals) được định dạng rõ ràng.
- Rút trích các lược đồ con của RDF.
- Xây dựng các lược đồ RDF mới dựa trên các thông tin trong các biểu đồ được truy vấn.

Các truy vấn của SPARQL phù hợp với (match) các mô hình lược đồ dựa trên lược đồ đích của truy vấn. Có mô hình giống như các lược đồ RDF, nhưng chứa các biến đã được định tên tại một vài nút (tài nguyên) hoặc các liên kết/các vị từ (thuộc tính). Mô hình lược đồ đơn giản nhất giống như bộ ba của RDF (resource-property-value hoặc O-A-V).

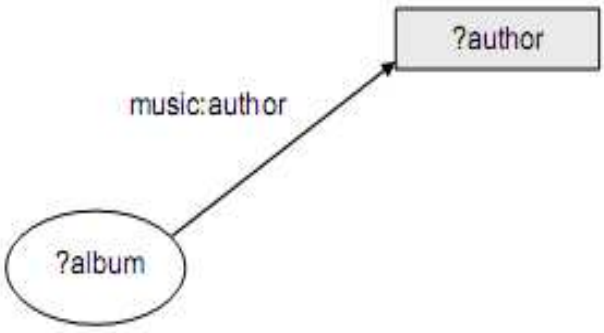
Hãy xem 2 bộ ba RDF sau:



Hình 2.5 Hai bộ ba RDF

(Nguồn từ: Dragan Gas˘evic´, Dragan Djuric´, Vladan Devedz˘ic' (2006), Model Driven Architecture and Ontology Development, fig.3.12, trang 93)

Cả hai bộ ba trên phù hợp với mô hình bộ ba sau:



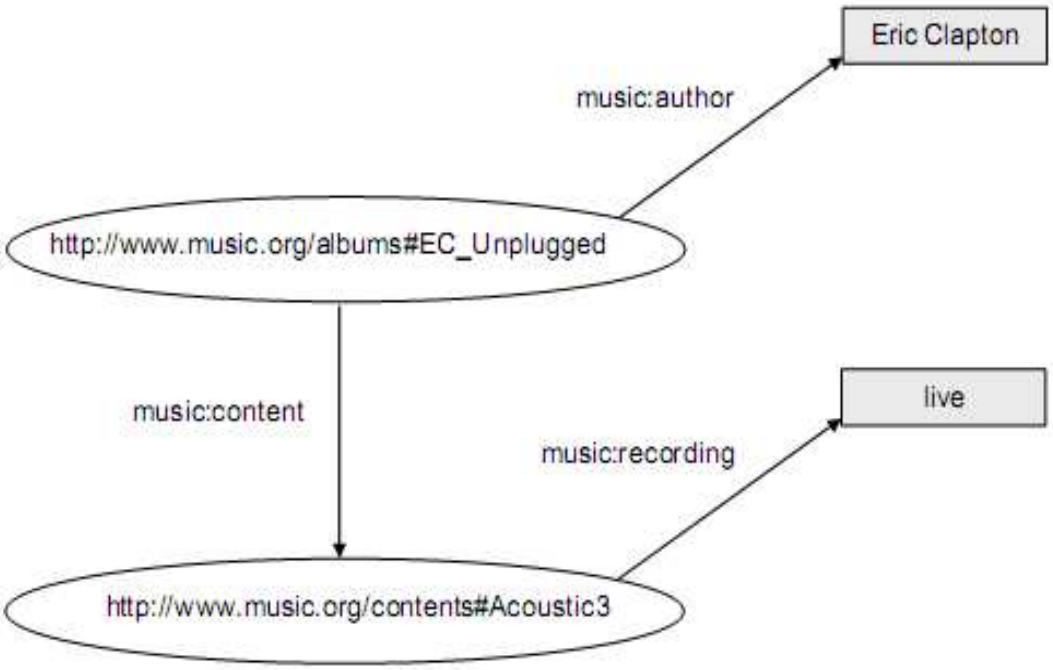
Hình 2.6 Mô hình bộ ba (?album, music:author, ?author) của RDF

(Nguồn từ: Dragan Gas˘evic´, Dragan Djuric´, Vladan Devedz˘ic' (2006), Model Driven Architecture and Ontology Development, fig.3.13, trang 93)

Một sự nối kết là một ánh xạ từ một biến trong một truy vấn đến các thuật ngữ. Mỗi bộ ba của 2 bộ ba trên là một giải pháp mô hình (một tập các nối kết đúng) cho mô hình bộ ba. Các kết quả truy vấn trong SPARQL là tập các giải pháp mô hình. Các kết quả của truy vấn được biểu diễn bởi mô hình bộ ba là những giải pháp mô hình sau:

album	**author**
http://www.music.org/albums#EC_Unplugged	Eric Clapton
http://www.music.org/albums#PG_UP	Peter Gabriel

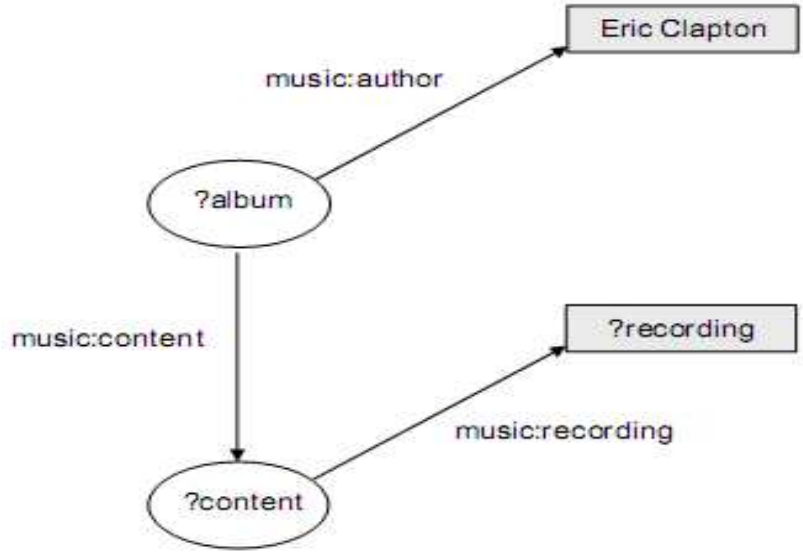
Các mô hình lược đồ đơn giản có thể được kết hợp thông qua việc sử dụng các toán tử khác nhau thành các mô hình lược đồ phức tạp hơn. Chẳng hạn như, lược đồ RDF phức tạp hơn như sau:



Hình 2.7 Lược đồ RDF phức tạp

(Nguồn từ: Dragan Gas˘evic´, Dragan Djuric´, Vladan Devedz˘ic' (2006), Model Driven Architecture and Ontology Development, fig.3.14, trang 94)

tương hợp (match) với mô hình lược đồ SPARQL phức tạp hơn sau:



Hình 2.8 Lược đồ SPARQL phức tạp

(Nguồn từ: Dragan Gas˘evic´, Dragan Djuric´, Vladan Devedz˘ic' (2006), Model Driven Architecture and Ontology Development, fig.3.14, trang 94)

Và giải pháp mô hình sẽ là:

album	http://www.music.org/albums#EC_Unplugged
ccontent	http://www.music.org/contents#Acoustic3
recording	live

Về mặt cú pháp, các truy vấn SPARQL thuộc một trong những dạng được biểu diễn trong hình sau:

```
SELECT    ?author
WHERE     { <http://www.music.org/albums#EC_Unplugged> <http://www.music.org/elements/auhor> ?author }

PREFIX    music: <http://www.music.org/elements/>
SELECT    ?author
WHERE     { <http://www.music.org/albums#EC_Unplugged> music:author ?author }

PREFIX    music: <http://www.music.org/elements/>
PREFIX    : <http://www.music.org/albums>
SELECT    $author
WHERE     { :EC_Unplugged  music:author  $author }
```

Rõ ràng, cú pháp rất giống cú pháp của các ngôn ngữ truy vấn cơ sở dữ liệu như SQL. Mệnh đề SELECT chứa các biến (variables), bắt đầu bằng dấu "?" hoặc dấu "$". Mệnh đề WHERE chứa một mô hình (pattern). Các Prefix được sử dụng như một cơ cấu viết tắt (abbreviation mechanism) cho các URI và áp dụng cho toàn bộ truy vấn.

e. Kết luận

RDF cung cấp một nền tảng cho sự biểu diễn và xử lý siêu dữ liệu.

RDF có mô hình dữ liệu dựa trên biểu đồ, với các khái niệm quan trọng như resouce (tài nguyên), property (thuộc tính) và statement (phát biểu). Một phát biểu (statement) là một bộ ba tài nguyên-thuộc tính-giá trị.

RDF có cấu trúc dựa trên XML để hỗ trợ khả năng tương tác cú pháp. XML và RDF bổ sung cho nhau bởi vì RDF hỗ trợ thao tác cú pháp.

RDF có sự phân quyền và cho phép xây dựng tri thức thặng dư (incremental), tái sử dụng và chia sẽ.

RDF là một miền (domain) độc lập. RDF Schema cung cấp kỹ thuật để mô tả những miền cụ thể.

RDF Schema là ngôn ngữ ontology nguyên thủy. Nó cung cấp các từ mẫu nền tảng và có các khái niệm quan trọng như lớp (class), quan hệ của lớp con (subclass relation), thuộc tính (property), quan hệ của thuộc tính con (subproperty relation), và giới hạn domain và giới hạn range.

RDF và RDFS cũng được hỗ trợ bởi các ngôn ngữ truy vấn..

B. Ngôn ngữ ontology web (OWL)

b.1 Giới thiệu

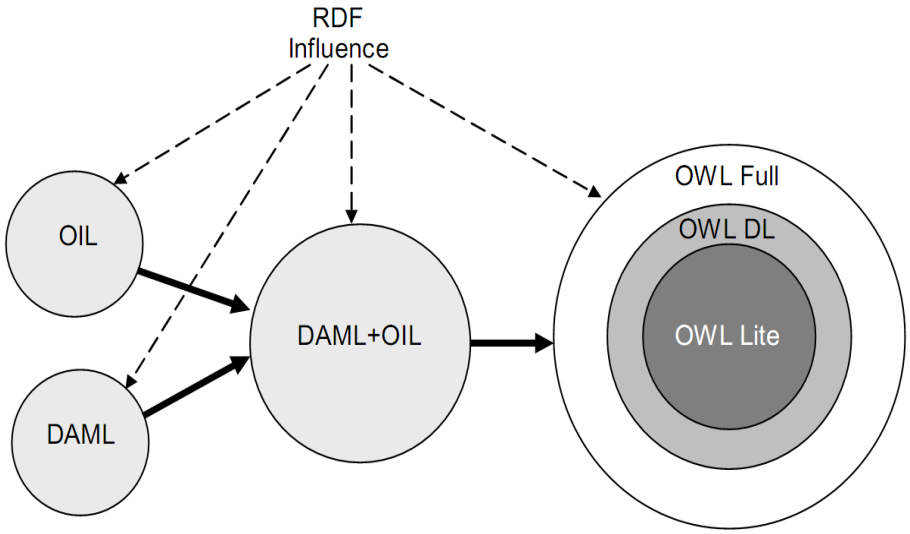
RDF và RDF Schema còn nhiều hạn chế trong việc diễn đạt: RDF bị giới hạn đối với các vị từ nền tảng nhị phân, và RDF Schema bị giới hạn đối với sự phân cấp lớp con (subclass hierarchy) và sự phân cấp thuộc tính (property hierarchy), cùng với các định nghĩa range và domain của những thuộc tính này.

Vì thế, cần có một ngôn ngữ xây dựng mô hình mạnh hơn khắc phục những hạn chế trên. Và ngôn ngữ OWL (Ontology web Language) ra đời.

OWL kế thừa từ DAML+OIL. Tên DAML+OIL là sự kết hợp giữa tên DAML-ONT (http://www.daml.org/2000/10/daml-ont.html) do Mỹ đề xuất và ngôn ngữ OIL (http://www.ontoknowledge.org/oil/) do Châu Âu đề xuất.

Giống như DAML+OIL, bộ từ vựng OWL gồm một tập các thuộc tính và phần tử của XML với ngữ nghĩa được định nghĩa rõ ràng. OWL dùng để mô tả các thuật ngữ của một miền và các mối quan hệ giữa các thuật ngữ trong một ontology.

Sự hình thành OWL:



Hình 2.9 Sự hình thành OWL

(Nguồn từ: Dragan Gas˘evic´, Dragan Djuric´, Vladan Devedz˘ic' (2006), Model Driven Architecture and Ontology Development, fig.3.14, trang 90)

b.2 Những yêu cầu của ngôn ngữ ontology

- Cú pháp được định nghĩa tường minh

  Cú pháp phải rõ ràng để máy có thể xử lý thông tin.

- Hỗ trợ lập luận hiệu quả

  Cho phép kiểm tra các trường hợp nhằm thiết kế các ontology lớn, tích hợp và chia sẽ các ontology từ các nguồn khác

- Một ngữ nghĩa hình thức

Ngữ nghĩa hình thức mô tả chính xác ngữ nghĩa của tri thức. Ngữ nghĩa là điều kiện tiên quyết cho hỗ trợ lập luận, nó cho phép:

- ✓ Kiểm tra tính nhất quán của ontology và tri thức
- ✓ Kiểm tra các mối quan hệ giữa các lớp không được định trước
- ✓ Phân cấp tự động các thực thể trong các lớp

- Đủ khả năng diễn đạt
- Thuận lợi cho sự diễn đạt

b.3 OWL

OWL là một mở rộng của RDF Schema. Về ngữ nghĩa, OWL dùng ngữ nghĩa của các lớp và các thuộc tính của RDF (rdfs:Class, rdfs:subClassOf…) và thêm các từ vựng nền tảng vào để hỗ trợ sự diễn đạt phong phú hơn. Sự mở rộng đó cũng phù hợp với kiến trúc phân tầng của web ngữ nghĩa.

b.3.1 Ba loại OWL

OWL Full

OWL Full sử dụng tất cả các từ vựng nền tảng (primitive) của ngôn ngữ OWL. Nó cho phép kết hợp tùy ý các từ vựng nền tảng với RDF và RDF Schema. Điều này có thể làm thay đổi ngữ nghĩa của các từ vựng nền tảng (RDF hoặc OWL) được định trước bằng cách áp dụng các từ vựng ngôn ngữ (language primitive) vào với nhau.

Ưu điểm của OWL Full: hoàn toàn tương thích từ dưới lên (upward-compatible) với RDF cả về cú pháp lẫn ngữ nghĩa: mọi tài liệu RDF nào hợp lệ thì cũng là tài liệu OWL Full hợp lệ, và mọi kết luận của RDF/RDF Schema nào có giá trị (valid) thì cũng là kết luận của OWL có giá trị (valid).

Nhược điểm của OWL Full: ngôn ngữ trở nên quá mạnh mẽ đến mức là không thể quyết định được (undecidable), ảnh hưởng đến hỗ trợ lập luận đầy đủ hoặc hỗ trợ lập luận hiệu quả.

OWL DL

OWL DL là một ngôn ngữ con của OWL Full, có thể sử dụng các hàm tạo lập (constructor) từ OWL, cung cấp sự diễn đạt tối ưu và đảm bảo tất cả các kết luận là có thể dự tính được và sẽ hoàn thành trong một thời gian nhất định.

Ưu điểm của OWL DL: cho phép hỗ trợ lập luận hiệu quả.

Nhược điểm: mất toàn bộ tính tương thích với RDF. Thông thường, một tài liệu RDF phải được mở rộng theo một số cách và bị giới hạn theo các cách khác trước khi nó là một tài liệu OWL DL hợp lệ. Mọi tài liệu OWL DL hợp lệ là tài liệu RDF hợp lệ.

OWL Lite

Dùng cho việc xây dựng các phân cấp nhóm và các ràng buộc đơn giản.

Nhược điểm của OWL Lite: loại bỏ các lớp được liệt kê, các phát biểu tách biệt (disjointness) và giá trị cardinality (số của giá trị) chỉ được phép là 0 và 1. Rất hạn chế trong sự diễn đạt.

Ưu điểm của OWL Lite: dễ sử dụng và dễ thực thi

b.3.2 Sự lựa chọn ngôn ngữ con

Việc lựa chọn ngôn ngữ con nào phù hợp nhất là phụ thuộc vào nhu cầu của mỗi người. Sự lựa chọn giữa OWL Lite và OWL DL phụ thuộc vào phạm vi người dùng cần cấu trúc diễn đạt. Sự lựa chọn giữa OWL DL và OWL Full chủ yếu phụ thuộc vào phạm vi người dùng cần những tiện ích xây dựng siêu mô hình (metamodeling) của RDF Schema.

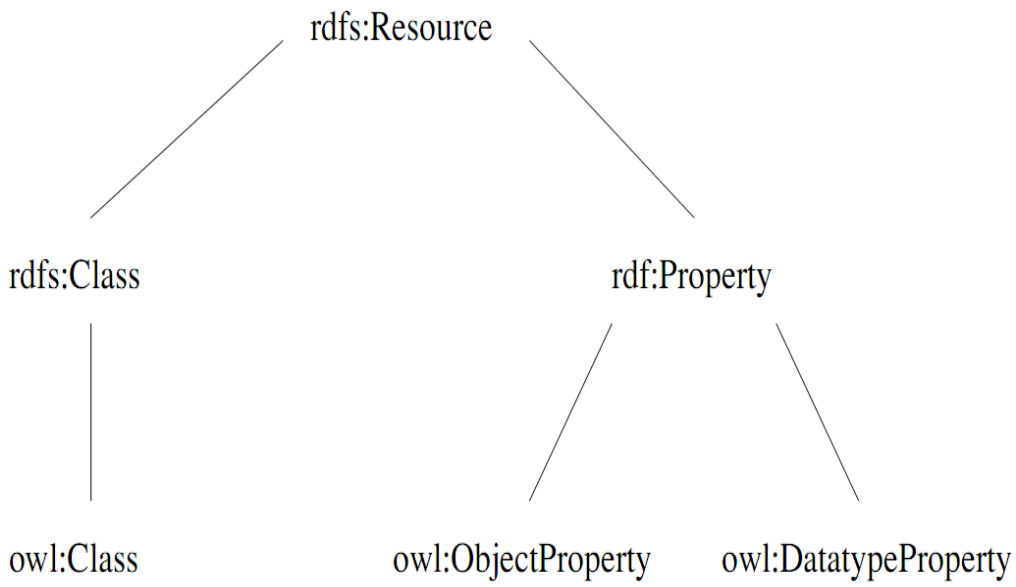
b.3.3 Tính tương thích của ba ngôn ngữ con

- Mọi ontology OWL Lite hợp lệ thì cũng hợp lệ trên ontology OWL DL
- Mọi ontology OWL DL hợp lệ thì cũng hợp lệ trên ontology OWL Full
- Mọi kết luận của OWL Lite có giá trị thì cũng có kết luận có giá trị trên OWL DL
- Mọi kết luận của OWL DL có giá trị thì cũng có kết luận có giá trị trên OWL Full

b.3.4 Mối quan hệ giữa OWL và RDF/RDFS

- Tất cả các loại OWL đều dùng RDF cho cú pháp của chúng
- Các thực thể (instance) được khai báo giống như trong RDF, sử dụng các mô tả của RDF và thông tin định kiểu
- Các hàm tạo lập (constructor) của OWL, chẳng hạn như owl:Class và owl:DatatypeProperty và owl:ObjectProperty là những đặc trưng của các bản sao (counterpart) của OWL

Hình 5 biểu thị các mối quan hệ của lớp con giữa một vài từ vựng mẫu nền tảng của OWL và RDF/RDFS



Hình 2.10 Mối quan hệ của lớp con giữa OWL và RDF/RDFS

(Nguồn từ: A Semantic web Primer (2004), Grigoris Antoniou and Frank van Harmelen, page 115, figure 4.1)

b.3.5 Ngôn ngữ OWL

Cú pháp

OWL xây dựng dựa trên RDF và RDF Schema và sử dụng cú pháp dựa trên XML của RDF:

- Cú pháp dựa trên XML (http://www.w3.org/TR/owl-xmlsyntax/) không tuân theo các quy ước của RDF.
- Cú pháp trừu tượng được sử dụng trong tài liệu định kiểu ngôn ngữ (http://www.w3.org/TR/owl-semantics/) thì súc tích và dễ đọc hơn nhiều so với hoặc là cú pháp XML hoặc là cú pháp RDF/XML.
- Cú pháp lược đồ dựa trên các quy ước của UML (Unified Modeling Language_Ngôn ngữ xây dựng mô hình hợp nhất) được sử dụng rộng rãi, và giúp con người trở nên quen thuộc với OWL.

Tiêu đề (header)

Tài liệu OWL thường được gọi là ontology OWL. Phần tử gốc của ontology OWL là phần tử “rdf:RDF”, và cũng cho biết số lượng không gian tên (namespace):

```
<rdf:RDF
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

Ontology OWL có thể bắt đầu với một bộ xác nhận (assertions). Những xác nhận này được nhóm lại dưới phần tử "owl:Ontology", và chứa các chú thích (comments), sự kiểm tra phiên bản và kể cả các ontology khác. Ví dụ:

```
<owl:Ontology rdf:about="">
    <rdfs:comment>An example OWL ontology</rdfs:comment>
    <owl:priorVersion
        rdf:resource="http://www.mydomain.org/uni-ns-old"/>
    <owl:imports
        rdf:resource="http://www.mydomain.org/persons"/>
    <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

"owl:imports" liệt kê các ontology khác mà nội dung của nó là một phần của ontology hiện tại. Các namespace được sử dụng để hợp nhất, còn các ontology được nhập vào cung cấp các định nghĩa để sử dụng. Thường có một phần tử import cho mỗi namespace được sử dụng, nhưng cũng có thể nhập thêm các ontology khác.

Ngoài ra, "owl:imports" còn là thuộc tính bắc cầu: nếu ontology A nhập vào ontology B, và ontology B nhập vào ontology C, thì ontology A cũng nhập vào ontology C.

Phần tử lớp (class elements)

Các lớp được định nghĩa bằng cách sử dụng phần tử "owl:Class" (lớp con của "rdfs:Class"). Chẳng hạn, chúng ta định nghĩa lớp "associateProfessor" như sau:

```
<owl:Class rdf:ID="associateProfessor">
    <rdfs:subClassOf rdf:resource="#academicStaffMember"/>
</owl:Class>
```

Lớp này là tách biệt (disjoint) với lớp “assistantProfessor” và “professor” khi dùng phần tử “owl:disjointWith”. Những phần tử này có thể nằm trong định nghĩa trước, hoặc được thêm vào bằng cách tham chiếu đến ID thông qua việc sử dụng “rdf : about”. Cơ chế này được kế thừa từ RDF.

```
<owl:Class rdf:about="#associateProfessor">
        <owl:disjointWith rdf:resource="#professor"/>
        <owl:disjointWith rdf:resource="#assistantProfessor"/>
</owl:Class>
```

Tính tương đương (equivalence) của các lớp có thể được định nghĩa bằng cách sử dụng phần tử “equivalentClass”:

```
<owl:Class rdf:ID="faculty">
        <owl:equivalentClass rdf:resource="#academicStaffMember"/>
</owl:Class>
```

Có 2 lớp được định nghĩa trước: “owl:Thing” (chứa mọi thứ) và “owl:Nothing” (lớp rỗng). Mỗi lớp là một lớp con của “owl:Thing” và một lớp cha (superclass) của “owl:Nothing”.

Phần tử thuộc tính

- Thuộc tính đối tượng: liên kết đối tượng với đối tượng. Ví dụ như “isTaughtBy” và “supervises”
- Thuộc tính kiểu dữ liệu: liên kết đối tượng với giá trị kiểu dữ liệu. Ví dụ như “phone”, “title”, “age”…

OWL không yêu cầu định nghĩa trước kiểu dữ liệu và cũng không cung cấp các tiện ích định nghĩa cụ thể.

Ví dụ về thuộc tính kiểu dữ liệu:

```
<owl:DatatypeProperty rdf:ID="age">
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
        #nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Các kiểu dữ liệu do người dùng định nghĩa thường được tập hợp trong XML Schema và sau đó được sử dụng trong ontology OWL.

Ví dụ về thuộc tính đối tượng:

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
    <rdfs:domain rdf:resource="#course"/>
    <rdfs:range rdf:resource="#academicStaffMember"/>
    <rdfs:subPropertyOf rdf:resource="#involves"/>
</owl:ObjectProperty>
```

OWL cho phép liên kết các thuộc tính đảo (inverse properties). Ví dụ như cặp "isTaughtBy" và "teaches":

```
<owl:ObjectProperty rdf:ID="teaches">
    <rdfs:range rdf:resource="#course"/>
    <rdfs:domain rdf:resource="#academicStaffMember"/>
    <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

Hình 6 minh họa mối quan hệ giữa một thuộc tính và sự đảo ngược của thuộc tính đó.

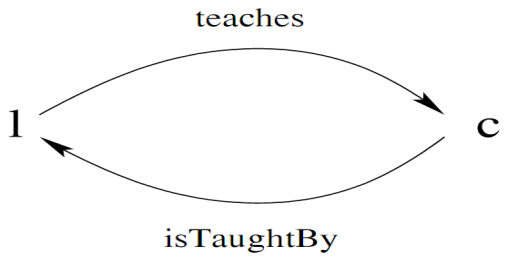


Figure 4.2 Inverse properties

Hình 2.11 Các thuộc tính đảo

(Nguồn từ: A Semantic web Primer (2004), Grigoris Antoniou and Frank van Harmelen, page ?, figure 4.2)

Tính tương đương (equivalence) của các thuộc tính có thể được định nghĩa thông qua việc sử dụng phần tử "owl:equivalentProperty".

```
<owl:ObjectProperty rdf:ID="lecturesIn">
    <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

Giới hạn thuộc tính

Với "rdfs:subClassOf", chúng ta có thể chỉ rõ lớp C là lớp con của lớp C' khác. Khi đó, mỗi thực thể (instance) của C cũng là thực thể của C'.

Phần tử sau yêu cầu các môn học năm đầu tiên (first-year courses) chỉ được dạy bởi giáo sư (professor):

```
<owl:Class rdf:about="#firstYearCourse">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isTaughtBy"/>
            <owl:allValuesFrom rdf:resource="#Professor"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

"owl:allValuesFrom" chỉ rõ lớp của các giá trị mà thuộc tính được chỉ rõ bởi "owl:onProperty" có thể nhận. Nói cách khác, tất cả các giá trị của thuộc tính phải đến từ lớp này. Trong ví dụ trên, chỉ có các giáo sư (professor) được cho phép như các giá trị của thuộc tính "isTaughtBy".

Có thể khai báo môn toán (mathematics courses) được dạy bởi "David Billington" như sau:

```
<owl:Class rdf:about="#mathCourse">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isTaughtBy"/>
            <owl:hasValue rdf:resource="#949352"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

"owl:hasValue" chỉ ra giá trị cụ thể mà thuộc tính được chỉ rõ bởi "owl:onProperty".

Thông thường, phần tử “owl:Restriction” gồm một phần tử “owl:onProperty” và ít nhất một khai báo giới hạn.

Một kiểu khai báo giới hạn khác xác định các giới hạn cardinality. Chẳng hạn như, chúng ta muốn mỗi môn học (course) được dạy bởi ít nhất một ai đó:

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

“1” là literal, được hiểu như nonNegativeInteger. Sử dụng khai báo không gian tên (namespace) xsd ở phần tử “header” để tham chiếu đến tài liệu XML Schema.

Như vậy, “owl:Restriction” xác định lớp ẩn không có ID, không được định nghĩa bởi “owl:Class”, và chỉ có phạm vi cục bộ (chỉ sử dụng tại một nơi mà ở đó sự giới hạn xuất hiện). Khi nói đến các lớp, là nói đến 2 nghĩa: thứ nhất, lớp mà được định nghĩa bởi “owl:Class” với một ID, và lớp ẩn cục bộ (đối tượng thỏa mãn các điều kiện giới hạn nào đó hoặc kết hợp với các lớp khác). Thứ hai, thường được gọi là sự diễn đạt của lớp.

Các thuộc tính đặc biệt

Một vài thuộc tính của các phần tử thuộc tính có thể được định nghĩa trực tiếp:

- “owl:TransitiveProperty”: xác định thuộc tính bắt cầu (transitive property), như là “has better grade than”, “is taller than” hoặc “is ancestor of”.
- “owl:SymmetricProperty”: xác định thuộc tính đối xứng (symmetric property), như là “has same grade as” hoặc “is sibling of”.

- "owl:FunctionalProperty": xác định thuộc tính có nhiều nhất 1 giá trị cho mỗi đối tượng, như là "age", "height" hoặc "directSupervisor".
- "owl:InverseFunctionalProperty": xác định thuộc tính cho 2 đối tượng khác nhau nào không có cùng giá trị, chẳng hạn như thuộc tính "isTheSocialSecurityNumberfor".

Kết hợp kiểu Boolean

Chúng ta có thể kết hợp các kiểu Boolean (union_hợp, intersection_giao, complement_bổ sung) của các lớp lại với nhau.

Ví dụ: chúng ta có thể nói rằng các môn học (course) và các thành viên nhân viên (staff members) là tách biệt (disjoint) như sau:

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

Như vậy, mỗi course là một thực thể (instance) của complement của staff member, tức là không có course nào là staff member. Tuy nhiên, phát biểu này cũng có thể được diễn đạt bằng cách sử dụng "owl:disjointWith".

Sự hợp (union) của các lớp được xây dựng bằng cách sử dụng "owl:unionOf":

```
<owl:Class rdf:ID="peopleAtUni">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#staffMember"/>
        <owl:Class rdf:about="#student"/>
    </owl:unionOf>
</owl:Class>
```

Sự giao (intersection) được phát biểu bằng “owl:intersectionOf”:

```
<owl:Class rdf:ID="facultyInCS">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#faculty"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#belongsTo"/>
            <owl:hasValue rdf:resource="#CSDepartment"/>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

Chúng ta đã xây dựng điểm giao nhau của 2 lớp, một trong số đó được ẩn: lớp của tất cả các đối tượng thuộc về SCDepartment. Lớp này giao với faculty cho biết toàn bộ cán bộ giảng dạy của một khoa (faculty) trong CSDepartment.

Các kết hợp kiểu Boolean có thể được lồng vào nhau một cách tùy ý:

```
<owl:Class rdf:ID="adminStaff">
  <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#staffMember"/>
        <owl:Class>
            <owl:complementOf>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#faculty"/>
                        <owl:Class rdf:about="#techSupportStaff"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:complementOf>
        </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

Sự liệt kê

Một liệt kê là một phần tử "owl:oneOf", được sử dụng để định nghĩa một lớp bằng cách liệt kê tất cả các phần tử của nó:

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

Các thực thể (instances)

Các thực thể (instance) của các lớp được khai báo giống như trong RDF:

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

Hoặc

```
<academicStaffMember rdf:ID="949352"/>
```

Chúng ta có thể viết chi tiết hơn như sau:

```
<academicStaffMember rdf:ID="949352">
    <uni:age rdf:datatype="&xsd;integer">39</uni:age>
</academicStaffMember>
```

OWL không chấp nhận "unique-names assumption" (giả định tên duy nhất) bởi vì 2 thực thể có tên khác nhau hoặc ID không rõ ràng là 2 cá thể (individuals) khác nhau.

OWL cung cấp một chú thích ngắn (shorthand notation) để xác nhận sự khác nhau từng cặp của tất cả các cá thể trong bảng liệt kê có sẵn:

```
<owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
        <lecturer rdf:about="#949318"/>
        <lecturer rdf:about="#949352"/>
        <lecturer rdf:about="#949111"/>
    </owl:distinctMembers>
</owl:AllDifferent>
```

“owl:distinctMembers” chỉ có thể được sử dụng trong sự kết hợp với “owl:allDifferent”.

Các kiểu dữ liệu (data types)

Không phải tất cả các kiểu dữ liệu XML Schema đều có thể được sử dụng trong OWL. Tài liệu tham khảo OWL (OWL reference document) liệt kê tất cả các kiểu dữ liệu XML Schema có thể sử dụng, bao gồm kiểu chuỗi (string), kiểu số nguyên (integer), kiểu luận lý (boolean), kiểu thời gian (time) và kiểu ngày (date).

Thông tin xác định phiên bản

Phát biểu “owl:priorVersion” chỉ rõ phiên bản trước của ontology hiện tại. Thông tin này không có ngữ nghĩa lý thuyết mô hình hình thức, nhưng có thể được các chương trình và người đọc khai thác cho những mục đích quản lý ontology.

Ngoài “owl:priorVersion” ra, OWL còn có 3 phát biểu khác để chỉ rõ thông tin xác định phiên bản:

- “owl:versionInfo”: thường chứa một chuỗi mang thông tin phiên bản hiện tại, chẳng hạn như từ khóa RCS/CVS.
- “owl:backwardCompatibleWith”: chứa một tham chiếu đến ontology khác, nhận biết ontology được chỉ định là phiên bản trước của ontology chứa đựng. Tất cả các trình định danh (identifier) từ phiên bản trước có cùng cách hiểu trong phiên bản mới. Vì vậy, nó cập nhật các khai báo

không gian tên (namespace) và các phát biểu “owl:imports” để tham chiếu đến URL của phiên bản mới.

- “owl:incompatibleWith”: cho biết ontology chứa đựng là phiên bản sau của ontology được tham chiếu. Về cơ bản, điều này là thuộc quyền của tác giả tạo ontology.

b.3.6 OWL trong OWL

Không gian tên (namespaces)

```
<?xml version="1.0"?>
<!DOCTYPE owl [
        <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
        <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
        <!ENTITY owl "http://www.w3.org/2002/07/owl#"> ]>
<rdf:RDF
        xml:base ="http://www.w3.org/2002/07/owl"
        xmlns ="&owl;"
        xmlns:owl ="&owl;"
        xmlns:rdf ="&rdf;"
        xmlns:rdfs="&rdfs;"
        xmlns:dc ="http://purl.org/dc/elements/1.1/">
```

URI của tài liệu trên được định nghĩa như là namespace mặc định. Hơn nữa, việc sử dụng các định nghĩa thực thể (entity) cho phép chúng ta viết tắt URL trong các giá trị thuộc tính.

Lớp của lớp (Metaclasses)

Lớp của tất cả các lớp chính là lớp con của lớp của tất cả các lớp của RDF Schema:

```
        <rdfs:Class rdf:ID="Class">
                <rdfs:label>Class</rdfs:label>
                <rdfs:comment>The class of all OWL classes</rdfs:comment>
```

```
        <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
    </rdfs:Class>
```

“Thing’ là lớp đối tượng phổ biến nhất trong OWL, và “Nothing” là lớp đối tượng rỗng.

Tính tương đương của lớp

Tính tương đương của lớp được biểu thị là “owl:EquivalentClass”, cho biết mối quan hệ của lớp con và luôn luôn được định rõ giữa 2 lớp. Điều này là tương tự như “owl:EquivalentProperty”. Các phát biểu disjointness chỉ được định rõ giữa các lớp.

```
<rdf:Property rdf:ID="EquivalentClass">
        <rdfs:label>EquivalentClass</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="&rdfs;subClassOf"/>
        <rdfs:domain rdf:resource="#Class"/>
        <rdfs:range rdf:resource="#Class"/>
</rdf:Property>
<rdf:Property rdf:ID="EquivalentProperty">
        <rdfs:label>EquivalentProperty</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="&rdfs;subPropertyOf"/>
</rdf:Property>
<rdf:Property rdf:ID="disjointWith">
        <rdfs:label>disjointWith</rdfs:label>
        <rdfs:domain rdf:resource="#Class"/>
        <rdfs:range rdf:resource="#Class"/>
</rdf:Property>
```

Tính tương đương (equality) và tính không tương đương (inequality) chỉ có thể được định rõ giữa những thứ tùy ý. Trong OWL Full, phát biểu này cũng có thể được áp dụng cho các lớp. “owl:sameAs” hoàn toàn đồng nghĩa như “owl:sameIndividualAs”.

```
<rdf:Property rdf:ID="sameIndividualAs">
        <rdfs:label>sameIndividualAs</rdfs:label>
```

```
        <rdfs:domain rdf:resource="#Thing"/>
        <rdfs:range rdf:resource="#Thing"/>
    </rdf:Property>
    <rdf:Property rdf:ID="differentFrom">
        <rdfs:label>differentFrom</rdfs:label>
        <rdfs:domain rdf:resource="#Thing"/>
        <rdfs:range rdf:resource="#Thing"/>
    </rdf:Property>
    <rdf:Property rdf:ID="sameAs">
        <rdfs:label>sameAs</rdfs:label>
        <EquivalentProperty rdf:resource="#sameIndividualAs"/>
    </rdf:Property>
```

"owl:distinctMembers" chỉ được dùng cho "owl:AllDifferent":

```
    <rdfs:Class rdf:ID="AllDifferent">
        <rdfs:label>AllDifferent</rdfs:label>
    </rdfs:Class>
    <rdf:Property rdf:ID="distinctMembers">
        <rdfs:label>distinctMembers</rdfs:label>
        <rdfs:domain rdf:resource="#AllDifferent"/>
        <rdfs:range rdf:resource="&rdf;List"/>
    </rdf:Property>
```

Xây dựng lớp từ lớp khác

"owl:unionOf" xây dựng một lớp từ danh sách.

```
    <rdf:Property rdf:ID="unionOf">
        <rdfs:label>unionOf</rdfs:label>
        <rdfs:domain rdf:resource="#Class"/>
        <rdfs:range rdf:resource="&rdf;List"/>
    </rdf:Property>
```

Tương tự với "owl:intersectionOf" và "owl:oneOf".

Giới hạn thuộc tính của các lớp

Các giới hạn trong OWL xác định lớp của các đối tượng đó thỏa mãn một vài điều kiện kèm theo:

```
<rdfs:Class rdf:ID="Restriction">
        <rdfs:label>Restriction</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Class"/>
</rdfs:Class>
```

Tất cả các thuộc tính sau chỉ được phép xuất hiện bên trong một định nghĩa giới hạn (restriction definition), tức là domain của chúng là “owl:Restriction”, nhưng chúng không tuân theo range của chúng:

```
<rdf:Property rdf:ID="onProperty">
        <rdfs:label>onProperty</rdfs:label>
        <rdfs:domain rdf:resource="#Restriction"/>
        <rdfs:range rdf:resource="&rdf;Property"/>
</rdf:Property>
<rdf:Property rdf:ID="allValuesFrom">
        <rdfs:label>allValuesFrom</rdfs:label>
        <rdfs:domain rdf:resource="#Restriction"/>
        <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>
<rdf:Property rdf:ID="hasValue">
        <rdfs:label>hasValue</rdfs:label>
        <rdfs:domain rdf:resource="#Restriction"/>
</rdf:Property>
<rdf:Property rdf:ID="minCardinality">
        <rdfs:label>minCardinality</rdfs:label>
        <rdfs:domain rdf:resource="#Restriction"/>
        <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</rdf:Property>
```

Các thuộc tính

"owl:ObjectProperty" là trường hợp đặc biệt của "rdf:Property".

```
<rdfs:Class rdf:ID="ObjectProperty">
    <rdfs:label>ObjectProperty</rdfs:label>
    <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>
```

Tương tự đối với "owl:DatatypeProperty'. Và "owl:TransitiveProperty" chỉ áp dụng cho các thuộc tính đối tượng:

```
<rdfs:Class rdf:ID="TransitiveProperty">
    <rdfs:label>TransitiveProperty</rdfs:label>
    <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
</rdfs:Class>
```

Và tương tự đối với các thuộc tính hàm đảo, thuộc tính hàm, thuộc tính đối xứng.

b.4 Kết luận

- OWL là một chuẩn dùng cho ontology web, mô tả ngữ nghĩa của tri thức theo cách mà máy có thể sử dụng được.
- OWL dựa trên RDF và RDFS. Các thực thể (instance) được định nghĩa thông qua việc sử dụng mô tả RDF; và hầu hết từ vựng mẫu (primitive) của RDFS được sử dụng.
- Ngữ nghĩa hình thức và lập luận hình thức được cung cấp thông qua việc ánh xạ OWL vào logic. Logic vị từ và logic mô tả được sử dụng cho mục đích này.

2.2.1.3 Logic và suy diễn [2]

A. Giới thiệu

Sự biểu diễn tri thức đã xuất hiện trong lĩnh vực trí tuệ nhân tạo, trong triết học và thậm chí đã có từ thời Hy lạp cổ đại; và Arictotle được xem là cha đẻ của logic.

Logic vẫn còn là nền tảng của biểu diễn tri thức, đặc biệt là trong logic vị từ (logic bậc nhất). Logic rất phổ biến và nó cung cấp một ngôn ngữ bậc cao mà ở đó tri thức được diễn đạt theo cách dễ hiểu và có khả năng diễn đạt cao. Nó có ngữ nghĩa hình thức dễ hiểu, và định rõ ngữ nghĩa tường minh của các phát biểu logic. Như vậy, logic có thể cung cấp các giải thích cho các câu trả lời.

Các ngôn ngữ RDF và OWL (Lite và DL) được xem như là các cụ thể hóa của logic vị từ. Sự phù hợp được minh họa bằng các ngữ nghĩa tiên đề dưới dạng các tiên đề logic. Mặc khác, các luật không thể xác nhận thông tin mà một người hoặc là đàn ông (man) hoặc là đàn bà (woman), trong khi thông tin này được diễn đạt dễ dàng trong OWL bằng cách sử dụng union “disjoint”.

Tuy nhiên, tiên đề của luật có thể không nằm trong khả năng diễn đạt của logic vị từ nên cần có một kiểu hệ thống luật mới. Hệ thống luật ra đời phụ thuộc vào lượng thông tin nhận được (đầy đủ hay không đầy đủ).

B. Các luật đơn điệu

b.1 Ví dụ

Giả sử ta có các vị từ cơ bản về mối quan hệ gia đình sau:

mother(X,Y)	X is the mother of Y
father(X,Y)	X is the father of Y
male(X)	X is male
female(X)	X is female

Từ các vị từ này, chúng ta có thể định nghĩa các mối quan hệ xa hơn nhờ sử dụng các luật thích hợp:

- Định nghĩa một “parent” (ba mẹ): một parent là hoặc father (cha) hoặc mẹ (mother):

mother(X,Y) → parent(X,Y)

father(X,Y) → parent(X,Y)

- Định nghĩa một "brother": (anh em trai) là người đàn ông có chung parent:

  male(X), parent(P,X),parent(P,Y),notSame(X,Y) → brother(X,Y)

  Vị từ "notSame" chỉ rõ sự khác nhau

- "sister" (chị em gái): là người đàn bà có chung parent:

  female(X), parent(P,X), parent(P,Y), notSame(X,Y) → sister(X,Y)

- "Uncle" (cậu): là brother (anh em trai) của "parent" (ba mẹ):

  brother(X,P), parent(P,Y) → uncle(X,Y)

- "Grandmother" (bà): là "mother" (mẹ) của "parent" (ba mẹ):

  mother(X,P), parent(P,Y) → grandmother(X,Y)

- "Ancestor" (ông bà tổ tiên) hoặc là "parent" (ba mẹ) hoặc là "ancestor" (tổ tiên) của "parent":

  parent(X,Y) → ancestor(X,Y)

  ancestor(X,P), parent(P,Y) → ancestor(X,Y)

b.2 Cú pháp của luật đơn điệu

Xét một luật đơn giản: tất cả các khách hàng thường xuyên trên 60 tuổi được hưởng quyền giảm giá đặc biệt:

loyalCustomer(X), age(X) > 60 → discount(X)

Các thành phần của luật:

- variables (biến): X
- constants (hằng): là các giá trị cố định: 60
- predicates (vị từ): liên kết các đối tượng: loyalCustomer, >
- function symbols (ký hiệu hàm): trả lại giá trị cho những đối số nào đó: age

b.2.1 Luật

Một luật r có dạng (ký hiệu là pl(r)):

$$B_1,...,B_n \rightarrow A$$

Ở đây, A, $B_1,\ldots,B_n$ là các công thức nguyên tử. A là "*head*" của luật, và $B_1,\ldots,B_n$ là các tiền đề của luật. Tập $\{B_1,\ldots,B_n\}$ được tham chiếu đến "body" của luật.

Dấu "," trong 'body' của luật được đọc nối tiếp nhau: nếu B_1 và B_2 và … và B_n đúng, thì A cũng đúng.

Thông thường, tất các các biến xuất hiện trong một luật đều có tính ẩn, tính phổ quát và tính định lượng (dùng $\forall X$)

Vì vậy, có thể biểu diễn luật r trên như sau:

$$\forall \forall X_1 ... \forall X_k((B_1 \wedge ... \wedge B_n) \rightarrow A)$$

Hoặc là:

$$\forall \forall X_1 ... \forall X_k (A \vee \neg B_1 \vee ... \vee \neg B_n)$$

Ở đây $X_1,\ldots,X_k$ là tất cả các biến xuất hiện trong A, $B_1,\ldots,B_n$

b.2.2 Các sự kiện (facts)

Sự kiện là công thức nguyên tử. Ví dụ: loyalCustomer(a345678), cho biết khách hàng có ID a345678 là khách hàng thường xuyên. Các biến của sự kiện có tính ẩn, tính phổ quát và tính định lượng.

b.2.3 Các chương trình logic

Chương trình logic P là một tập các sự kiện và các luật có ngôi. Sự tịnh tuyến của logic vị từ *pl(P)* là tập tất cả các biểu diễn của các luật và các sự kiện của logic vị từ trong P.

b.2.4 Các mục tiêu (goals)

Một goal có dạng:

$$B_1,...,B_n \rightarrow$$

Nếu n=0, chúng ta có goal rỗng:

Các goal trong logic vị từ được biểu diễn như sau:

$$\forall X_1 \ldots \forall X_k (\neg B_1 \vee \ldots \vee \neg B_n)$$

Hoặc là:

$$\neg \exists X_1 \ldots \exists X_k (B_1 \wedge \ldots \wedge B_n)$$

Ở đây, $X_1,\ldots,X_k$ là tất cả các biến xuất hiện trong $B_1,\ldots,B_n$.

b.3 Ngữ nghĩa của luật đơn điệu

Để trả lời một truy vấn ta dùng suy diễn của logic vị từ của các luật, các sự kiện và các câu hỏi. Giả sử cho trước một chương trình logic P và một truy vấn:

$$B_1,\ldots, B_n \rightarrow$$

Với các biến $X_1,\ldots,X_k$, chúng ta trả lời chắc chắn khi nào và chỉ khi nào:

$$pl(P) \models \exists X_1 \ldots \exists X_k (B_1 \wedge \ldots \wedge B_n) \quad (1)$$

Hoặc tương đương khi nào:

$$pl(P) \cup \{\neg \exists X_1 \ldots \exists X_k (B_1 \wedge \ldots \wedge B_n)\} \text{ là không thỏa mãn} \quad (2)$$

Chúng ta đưa ra một câu trả lời chắn chắn nếu sự suy diễn của logic vị từ của chương trình P, cùng với sự suy diễn của truy vấn của logic vị từ là không thỏa mãn.

Các lượng từ $\neg, \vee, \wedge, \rightarrow, \forall, \exists$ có nghĩa lần lượt là not (không), or (hoặc), and (và), implies (hàm ý), for all (với mọi), there is (tồn tại).

(1) và (2) cho biết: khi sự suy diễn của logic vị từ P là đúng, thì $\exists X_1 \ldots \exists X_k (B_1 \wedge \ldots \wedge B_n)$ cũng phải đúng. Tức là, có tồn tại các giá trị cho các biến $X_1,\ldots,X_k$ để cho tất cả các công thức nguyên tử trở nên đúng.

C. Luật không đơn điệu

c.1 Giới thiệu

Trong các hệ thống luật không đơn điệu, một luật không được áp dụng cho tất cả các tiền đề bởi vì chúng ta còn phải xét tới các chuỗi lập luận đối lập. Thường chúng ta xét các luật “defeasible” bởi vì chúng sẽ bị các luật khác hủy bỏ. Để chấp nhận các

xung đột giữa các luật, các công thức nguyên tử phủ định có thể xuất hiện trong "head" và "body" của các luật. Chẳng hạn như, chúng ta có thể viết:

$$p(X) \rightarrow q(X)$$
$$r(X) \rightarrow \neg q(X)$$

Để phân biệt giữa các luật "defeasible", luật "standard" và luật đơn điệu, chúng ta dùng một mũi tên như sau:

$$p(X) \Rightarrow q(X)$$
$$r(X) \Rightarrow \neg q(X)$$

Có thể giải quyết sự xung đột bằng cách sử dụng "quyền ưu tiên": một luật nào đó được ưu tiên thì luật đó sẽ được chấp nhận. Các quyền ưu tiên được sử dụng dựa trên những nguyên tắc: tính tin cậy, tính mới và tính cụ thể của luật. Tuy nhiên, quan hệ quyền ưu tiên không có tính chu kỳ (acyclic), nghĩa là không thể có sự lặp lại của hình thức: $r_1 > r_2 > ... > r_n > r_1$

c.2 Định nghĩa cú pháp

Một luật defeasible có dạng:

$$r : L_1,...,L_n => L$$

Ở đây, r là nhãn, $\{L_1 ,...,L_n\}$ là "body" (hoặc các tiền đề), và L là "head" của luật. $L,L_1,...,L_n$ là các literal khẳng định hoặc literal phủ định (một lieral là một công thức nguyên tử $p(t_1,….,t_n)$ hoặc phủ định của nó $\neg p(t_1,...,t_n)$). Đôi khi chúng ta chỉ rõ "head" của một luật là *head(r)*, "body" của nó là *body(r)* và sử dụng nhãn r để tham chiếu đến tất cả các luật.

Một chương trình logic defeasible là một bộ ba (F, R, >) bao gồm một tập F các sự kiện, một tập R có ngôi của các luật defeasible, và một quan hệ nhị phân không chu kỳ > trên R.

D. Biểu diễn các luật đơn điệu trong XML

Mục đích là tạo ra tri thức dưới dạng các luật mà máy có thể sử dụng được.

d.1 Các thuật ngữ

Các thuật ngữ (terms) được biểu diễn bằng cách dùng các thẻ <term>, <function>, <var> và <const>. Chẳng hạn như thuật ngữ sau:

f(X, a, g(b, Y))

được biểu diễn như sau:

```
<term>
    <function>f</function>
    <term>
        <var>X</var>
    </term>
    <term>
        <const>a</const>
    </term>
    <term>
        <function>g</function>
        <term>
            <const>b</const>
        </term>
        <term>
            <var>Y</var>
        </term>
    </term>
</term>
```

d.2 Các công thức nguyên tử

Đối với các công thức nguyên tử, chúng ta sử dụng thêm thẻ <atom> và thẻ <predicate>. Chẳng hạn như, công thức:

p(X, a, f(b, Y))

được biểu diễn như sau:

```
<atom>
```

```
<predicate>p</predicate>
<term>
    <var>X</var>
</term>
<term>
    <const>a</const>
</term>
<term>
    <function>f</function>
    <term>
        <const>b</const>
    </term>
    <term>
        <var>Y</var>
    </term>
</term>
</atom>
```

d.3 Các sự kiện (facts)

Một sự kiện chỉ là một công thức nguyên tử, được bao bọc bởi các thẻ <fact> đóng và mở. Chẳng hạn như, sự kiện p(a) được biểu diễn như sau:

```
<fact>
    <atom>
        <predicate>p</predicate>
        <term>
            <const>a</const>
        </term>
    </atom>
</fact>
```

d.4 Các luật

Một luật gồm có một “head” và một “body”. “head” là một công thức nguyên tử và “body” có thể là một sự phối hợp của các công thức nguyên tử (“body” có thể là một chuỗi rỗng). Chúng ta sử dụng các thẻ <rule>, <head> và <body>. Chẳng hạn như luật:

$$p(X, a),q(Y, b) \rightarrow r(X,Y)$$

được biểu diễn như sau:

```
<rule>
    <head>
        <atom>
            <predicate>r</predicate>
            <term>
                <var>X</var>
            </term>
            <term>
                <var>Y</var>
            </term>
        </atom>
    </head>
    <body>
        <atom>
            <predicate>p</predicate>
            <term>
                <var>X</var>
            </term>
            <term>
                <const>a</const>
            </term>
        </atom>
        <atom>
            <predicate>q</predicate>
```

```
                    <term>
                        <var>Y</var>
                    </term>
                    <term>
                        <const>b</const>
                    </term>
                </atom>
            </body>
        </rule>
```

d.5 Các truy vấn

Các truy vấn được biểu diễn như các "body" của các luật, được bao bọc bởi các thẻ <query>.

E. Biểu diễn các luật không đơn điệu trong trong XML

So với các luật đơn điệu, các luật không đơn điệu có những khác nhau về mặt cú pháp sau:

- Không có các ký hiệu hàm
- Các nguyên tử phủ định có thể xuất hiện trong "head" và "body" của luật
- Mỗi luật có một nhãn (label)
- Ngoài các luật và các sự kiện (facts), một chương trình cũng chứa các phát biểu của quyền ưu tiên

F. Kết luận

- Các luật không đơn điệu rất có ích trong các tình huống mà ở đó thông tin có hiệu lực nhưng không đầy đủ. Chúng là những luật có thể bị các luật khác loại bỏ.
- Các quyền ưu tiên được sử dụng để giải quyết các xung đột giữa các luật không đơn điệu.
- Sự biểu diễn của các luật trong các ngôn ngữ như ngôn ngữ XML là không phức tạp.

2.2.1.4 Logic mô tả [1][6]

A. Giới thiệu

Logic mô tả thuộc họ hình thức biểu diễn tri thức, biểu diễn tri thức của một miền ứng dụng bằng cách, đầu tiên, định nghĩa các khái niệm liên quan của miền đó (thuật ngữ), sau đó dùng những khái niệm này chỉ rõ các thuộc tính của các đối tượng và các cá thể xuất hiện trong miền đó. Logic mô tả được trang bị thêm ngữ nghĩa dựa trên logic và hình thức, đặc biệt là có hỗ trợ lập luận cho phép suy diễn tri thức một cách rõ ràng từ một tri thức nào đó.

Sự hợp nhất giữa tính diễn đạt của logic mô tả và tính phức tạp của những vấn đề lập luận là một trong những vấn đề quan trọng nhất trong nghiên cứu logic mô tả. Tuy nhiên, sự phát triển của logic mô tả dựa trên các tầng xây dựng cú pháp cơ bản là các khái niệm nguyên tử (các vị từ nhất nguyên), các luật nguyên tử (các vị từ nhị phân) và các cá thể nguyên tử (các hằng), khả năng diễn đạt của ngôn ngữ để xây dựng các khái niệm và các luật phức tạp, và sự hỗ trợ của trình suy diễn.

Hệ logic mô tả không chỉ chứa các bộ thuật ngữ (TBox) và các bộ xác nhận (ABox) mà còn giúp các dịch vụ lập luận chúng. Nhiệm vụ lập luận thuật ngữ là xác định liệu một mô tả có thỏa mãn (không có mâu thuẫn) hay không? Hoặc liệu mô tả này có tổng quát hơn mô tả kia hay không? Tức là liệu mô tả này có xếp gộp vào mô tả kia hay không?

B. Cú pháp và ngữ nghĩa của logic mô tả

b.1 Quy tắc ký hiệu

Các khái niệm nguyên tử ký hiệu là các chữ C, D

Các luật ký hiệu là các chữ R, S

Các luật hàm (đặc điểm_features, thuộc tính_attributes) ký hiệu là các chữ f, g

Các số nguyên không âm (trong giới hạn số) ký hiệu là n, m

Các cá thể (individuals) ký hiệu là a, b

Trong tất cả các trường hợp, chúng ta có thể dùng chỉ số dưới dòng (subscripts) (thường dùng khi định nghĩa cú pháp, ngữ nghĩa và trong các ví dụ trừu tượng).

Trong các ví dụ cụ thể, các quy tắc sau thường dùng:

- Tên khái niệm bắt đầu bằng một chữ cái hoa, theo sau là những chữ cái thường (ví dụ như Human, Male).
- Tên luật (cũng như các tên hàm) bắt đầu bằng chữ cái thường (ví dụ như hasChild, marriedTo).
- Tên cá thể là toàn chữ cái hoa (ví dụ như CHARLES, MARY).

b.2 Các mô tả khái niệm và mô tả luật

Các mô tả sơ cấp là các khái niệm nguyên tử (tên khái niệm) và các luật nguyên tử (tên luật). Những mô tả phức tạp được xây dựng từ các mô tả sơ cấp cùng với các constructor khái niệm và constructor luật.

Các mô tả khái niệm trong AL (Attributive Language) được hình thành theo luật cú pháp sau:

(Syntax)		(constructors)
C, D →	A |	Khái niệm nguyên tử
	T |	Khái niệm tổng quát
	⊥ |	Khái niệm bottom
	¬A |	Phủ định nguyên tử
	C ⊓ D |	Giao nhau (intersection)
	∀R.C |	Giới hạn giá trị (value restriction)
	∃R.T	Lượng từ tồn tại giới hạn (limited existential quantification)

Trong AL, phủ định chỉ được áp dụng cho các khái niệm nguyên tử, và chỉ có khái niệm mức đỉnh (top) được áp dụng cho phạm vi số lượng tồn tại theo một luật.

Giả sử Person và Female là những khái niệm nguyên tử, thì Person ⊓ Female và Person ⊓ ¬Female là những khái niệm của AL lần lượt mô tả những người là đàn bà và những người không phải đàn bà. Ngoài ra, nếu giả sử hasChild là một luật nguyên tử thì chúng ta có thể tạo ra những khái niệm Person ⊓ ∃hasChild.T và Person ⊓ ∀hasChild.Female cho biết những người có con nói chung, và những người có toàn con gái. Ta có thể sử dụng khái niệm mức dưới (bottom) để mô tả những người không có con bằng khái niệm Person ⊓ ∀hasChild.⊥

Để định nghĩa ngữ nghĩa hình thức của các khái niệm AL, chúng ta xem biên dịch I gồm một tập hợp không rỗng Δ^I (miền của biên dịch) và một hàm biên dịch, gán cho mỗi khái niệm nguyên tử A một tập $A^I \subseteq \Delta^I$ và gán cho mỗi luật nguyên tử R một quan hệ nhị phân $R^I \subseteq \Delta^I x \Delta^I$. Hàm biên dịch được mở rộng cho các mô tả khái niệm bằng những định nghĩa quy nạp sau:

$$
\begin{aligned}
T^I &= \Delta^I \\
\bot^I &= \emptyset \\
(\neg A)^I &= \Delta^I \setminus A^I \\
(C \sqcap D)^I &= C^I \cap D^I \\
(\forall R.C)^I &= \{\alpha \in \Delta^I \mid \forall b.\ (a,b) \in R^I \rightarrow b \in C^I\} \\
(\exists R.T)^I &= \{\alpha \in \Delta^I \mid \exists b.\ (a,b) \in R^I\}
\end{aligned}
$$

Hai khái niệm C, D tương đương nhau (C ≡ D) nếu $C^I = D^I$ cho tất cả biên dịch I. Ví dụ, hai khái niệm ∀hasChild.Female ⊓ ∀hasChild.Student và ∀hasChild.(Female ⊓ Student) là tương đương nhau.

b.3 Giới hạn cho các biên dịch luật:

Những giới hạn này yêu cầu các biên dịch luật phải thỏa các thuộc tính nào đó, như là tính bắc cầu (transitivity) và tính hàm (functionality). Những thuộc tính khác có thể đối xứng (symmetry) hoặc liên kết (connections) giữa các luật khác nhau.

b.3.1 Các luật hàm

Xét một tập con N_F của tập tên luật N_R, thì các phần tử của nó được gọi là các đặc tính (features). Một biên dịch phải ánh xạ các features f đến các quan hệ nhị phân hàm $f^I \subseteq \Delta^I \times \Delta^I$, nghĩa là các quan hệ thỏa $\forall a,b,c.f^I(a,b) \wedge f^I(a,c) \rightarrow b = c$. Đôi khi các quan hệ hàm được xem như hàm không đầy đủ. Vì vậy, ta viết $f^I(a) = b$ hơn là viết $f^I(a,b)$. AL sử dụng thêm các feature được ký hiệu là AL_f.

b.3.2 Các luật bắc cầu

Xét một tập con N_{R+} của N_R, thì các tên luật $R \in N_{R+}$ được gọi là các luật bắc cầu. Một biên dịch phải ánh xạ các luật bắc cầu $R \in N_{R+}$ vào các quan hệ nhị phân bắc cầu $R^I \subseteq \Delta^I \times \Delta^I$. AL sử dụng thêm các luật bắc cầu được ký hiệu là AL_{R+}.

b.3.3 Các hàm tạo lập (constructor) khái niệm

Các constructor khái niệm đưa ra các mô tả luật, mô tả khái niệm, và chuyển chúng thành các mô tả khái niệm phức tạp hơn.

Cú pháp và ngữ nghĩa của các constructor khái niệm thông dụng của logic mô tả:

Name	Syntax	Semantics	Symbol
Top	$\top$	$\Delta^{\mathcal{I}}$	$\mathcal{AL}$
Bottom	$\bot$	$\emptyset$	$\mathcal{AL}$
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$\mathcal{AL}$
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$\mathcal{U}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\mathcal{C}$
Value restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.\ (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	$\mathcal{AL}$
Existential quant.	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.\ (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	$\mathcal{E}$
Unqualified number restriction	$\geqslant n\,R$ $\leqslant n\,R$ $=n\,R$	$\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\rvert \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\rvert \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\rvert = n\}$	$\mathcal{N}$
Qualified number restriction	$\geqslant n\,R.C$ $\leqslant n\,R.C$ $=n\,R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\rvert \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\rvert \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \lvert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\rvert = n\}$	$\mathcal{Q}$
Role-value-map	$R \subseteq S$ $R = S$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow (a,b) \in S^{\mathcal{I}}\}$ $\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \leftrightarrow (a,b) \in S^{\mathcal{I}}\}$	
Agreement and disagreement	$u_1 \doteq u_2$ $u_1 \neq u_2$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b \in \Delta^{\mathcal{I}}.\ u_1^{\mathcal{I}}(a) = b = u_2^{\mathcal{I}}(a)\}$ $\{a \in \Delta^{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta^{\mathcal{I}}.\ u_1^{\mathcal{I}}(a) = b_1 \neq b_2 = u_2^{\mathcal{I}}(a)\}$	$\mathcal{F}$
Nominal	I	$I^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ with $\lvert I^{\mathcal{I}}\rvert = 1$	$\mathcal{O}$

Bảng 2.1 Cú pháp và ngữ nghĩa của các hàm tạo lập (constructor) thông dụng

(Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider (), THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications, page 497)

Một vài cú pháp tường minh của các constructor khái niệm và chuyển chúng sang cú pháp trừu tượng:

Name	Concrete syntax	Abstract syntax
Top	TOP	$\top$
Bottom	BOTTOM	$\bot$
Intersection	(and $C_1 \cdots C_n$)	$C_1 \sqcap \cdots \sqcap C_n$
Union	(or $C_1 \cdots C_n$)	$C_1 \sqcup \cdots \sqcup C_n$
Negation	(not C)	$\neg C$
Value restriction	(all R C)	$\forall R.C$
Limited existential quantification	(some R)	$\exists R.\top$
Existential quantification	(some R C)	$\exists R.C$
At-least number restriction	(at-least n R)	$\geqslant n\,R$
At-most number restriction	(at-most n R)	$\leqslant n\,R$
Exact number restriction	(exactly n R)	$=n\,R$
Qualified at-least restriction	(at-least n R C)	$\geqslant n\,R.C$
Qualified at-most restriction	(at-most n R C)	$\leqslant n\,R.C$
Qualified exact restriction	(exactly n R C)	$=n\,R.C$
Same-as, agreement	(same-as u_1 u_2)	$u_1 \doteq u_2$
Role-value-map	(subset R_1 R_2)	$R_1 \subseteq R_2$
Role fillers	(fillers R $I_1 \cdots I_n$)	$\exists R.I_1 \sqcap \cdots \sqcap \exists R.I_n$
One-of	(one-of $I_1 \cdots I_n$)	$I_1 \sqcup \cdots \sqcup I_n$

Bảng 2.2 Cú pháp tường minh và trừu tượng của các constructor khái niệm

(Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider (), THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications, page 498)

b.3.4 Các contructor luật

Các constructor luật đưa ra các mô tả luật, mô tả khái niệm, và chuyển chúng thành các mô tả luật phức tạp hơn. Cú pháp và ngữ nghĩa của các constructor luật thông dụng:

Name	Syntax	Semantics	Symbol
Universal role	U	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	U
Intersection	$R \sqcap S$	$R^{\mathcal{I}} \cap S^{\mathcal{I}}$	$\sqcap$
Union	$C \sqcup D$	$R^{\mathcal{I}} \cup S^{\mathcal{I}}$	$\sqcup$
Complement	$\neg R$	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$	$\neg$
Inverse	R^{-}	$\{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}$	-1
Composition	$R \circ S$	$R^{\mathcal{I}} \circ S^{\mathcal{I}}$	$\circ$
Transitive closure	R^{+}	$\bigcup_{n \geq 1} (R^{\mathcal{I}})^n$	$+$
Reflexive-transitive closure	R^{*}	$\bigcup_{n \geq 0} (R^{\mathcal{I}})^n$	$*$
Role restriction	$R\|_C$	$R^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \times C^{\mathcal{I}})$	r
Identity	$id(C)$	$\{(d,d) \mid d \in C^{\mathcal{I}}\}$	id

Bảng 2.3 Cú pháp và ngữ nghĩa của các constructor luật thông dụng

(Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider (), THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications, page 499)

Ký hiệu o biểu thị thành phần thông thường của các quan hệ nhị phân, tức là:

$$R^{\mathcal{I}} \circ S^{\mathcal{I}} = \{(a,c) \mid \exists b.\ (a,b) \in R^{\mathcal{I}} \wedge (b,c) \in S^{\mathcal{I}}\}$$

Thành phần lặp lại được ký hiệu dưới dạng $(R^I)^n$:

$$(R^{\mathcal{I}})^0 = \{(d,d) \mid d \in \Delta^{\mathcal{I}}\} \quad \text{and} \quad (R^{\mathcal{I}})^{n+1} = (R^{\mathcal{I}})^n \circ R^{\mathcal{I}}$$

Cú pháp tường minh của các constructor luật:

Name	Concrete syntax	Abstract syntax
Universal role	`top`	U
Intersection	`(and` $R_1 \cdots R_n$`)`	$R_1 \sqcap \cdots \sqcap R_n$
Union	`(or` $R_1 \cdots R_n$`)`	$R_1 \sqcup \cdots \sqcup R_n$
Complement	`(not R)`	$\neg R$
Inverse	`(inverse R)`	R^-
Composition	`(compose` $R_1 \cdots R_n$`)`	$R_1 \circ \cdots \circ R_n$
Transitive closure	`(transitive-closure R)`	R^+
Reflexive-transitive closure	`(transitive-reflexive-closure R)`	R^*
Role restriction	`(restrict R C)`	$R\|_C$
Identity	`(identity C)`	$id(C)$

Bảng 2.4 Cú pháp tường minh và trừu tượng của các constructor luật

(Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider (), THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications, page 500)

b.4 Họ ngôn ngữ AL

Khi thêm các constructor vào AL ta sẽ được các ngôn ngữ diễn đạt hơn. Sự hợp (union) của các khái niệm (ký hiệu là U) được viết là C ⊔ D, và được phiên dịch như sau: $(C \sqcup D)^I = C^I \sqcup D^I$.

Lượng từ tồn tại đầy đủ (ký hiệu là ϵ) được viết là ∃ R.C, và được phiên dịch như sau: $(\exists R.C)^I = \{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$.

Cú pháp ∃ R.C khác cú pháp ∃ R.T (∃ R.T là những khái niệm tùy ý được phép xuất hiện trong phạm vi lượng từ tồn tại).

Giới hạn số (ký hiệu là N) được viết là ≥ nR (giới hạn tối thiểu) và ≤ nR (giới hạn tối đa) với n là số nguyên (integer) không âm. Các giới hạn số này lần lượt được phiên dịch như sau: $(\geq nR)^I = \{a \in \Delta^I \mid |\{b|(a,b) \in R^I\}| \geq n\}$ và $(\leq nR)^I = \{a \in \Delta^I | |\{b|(a,b) \in R^I\}| \leq n\}$.

Phủ định của các khái niệm tùy ý (ký hiệu là C, cho complement) được viết là ¬C, và được phiên dịch là: $(\neg C)^I = \Delta^I \setminus C^I$.

Với việc sử dụng thêm các constructor, chúng ta có thể mô tả những người có hoặc không hơn 1 con hoặc ít hơn 3 con, một trong 3 người con đó là con gái (female): Person ⊓ (≤1hasChild ⊔ (≥3hasChild ⊓ ∃ hasChild.Female)).

Ta có thể mở rộng AL bằng cách sử dụng thêm các constructor tạo một ngôn ngữ AL đặc biệt. Tên ngôn ngữ mở rộng có dạng: AL[U][Ɛ][N][C]

Với chữ cái theo tên ngôn ngữ tượng trưng cho sự hiện diện của constructor tương ứng. Chẳng hạn như, ALƐN là dạng mở rộng của AL với sự hiện diện của lượng từ tồn tại đầy đủ (Ɛ) và các giới hạn số (N).

Theo quan điểm ngữ nghĩa, không phải tất cả những ngôn ngữ này đều rõ ràng. Lượng từ tồn tại đầy đủ và hợp có thể được diễn đạt bằng cách sử dụng sự phủ định. Tuy nhiên, để phân biệt ngôn ngữ AL với sự phủ định kèm theo và bản sao (counterpart) của nó có lượng từ tồn tại đầy đủ và union, chúng ta sử dụng chữ C thay cho chữ UƐ trong các tên ngôn ngữ. Ví dụ, ta có thể viết ALC thay cho ALUƐ và viết ALCN thay cho ALUƐN.

C. Nền tảng tri thức

Nền tảng tri thức của logic mô tả gồm một bộ tiên đề thuật ngữ (TBox) và một bộ tiên đề xác nhận (ABox). Cú pháp và ngữ nghĩa của bộ tiên đề thuật ngữ và tiên đề xác nhận:

Name	Syntax	Semantics
Concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Concept equality	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Role equality	$R \equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
Concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role assertion	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$

Bảng 2.5 Cú pháp và ngữ nghĩa của các tiên đề xác nhận và thuật ngữ

(Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider (), THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications, page 501)

c.1 TBox (Bộ thuật ngữ)

Chúng ta đã biết cách tạo ra các mô tả phức tạp từ các khái niệm để mô tả các lớp của các đối tượng. Các tiên đề thuật ngữ tạo ra các phát biểu về việc bằng cách nào mà các khái niệm hay các luật liên kết được với nhau. Chúng ta chọn các định nghĩa là các tiên đề cụ thể và xem các thuật ngữ như một bộ định nghĩa mà trong đó, các khái niệm nguyên tử là các chữ viết tắt hay các tên để tạo ra các khái niệm phức tạp. Nếu các định nghĩa trong thuật ngữ chứa các cycle, thì chúng ta phải chấp nhận ngữ nghĩa fixpoint để khiến chúng được rõ ràng.

c.1.1 Các tiên đề thuật ngữ

Thông thường, các tiên đề thuật ngữ có dạng:

$$C \sqsubseteq D \ (R \sqsubseteq S) \text{ hoặc } C \equiv D \ (R \equiv S)$$

Với C, D là những khái niệm (và R, S là những luật). Các tiên đề loại thứ nhất được gọi là tiên đề bao hàm (inclusions), trong khi đó các tiên đề loại thứ hai được gọi là tiền đề tương đương (equalities). Một biên dịch I thỏa inclusion $C \sqsubseteq D$ nếu $C^I \sqsubseteq D^I$, và nó thỏa equality $C \equiv D$ nếu $C^I = D^I$. Nếu I thỏa một tiên đề thì chúng ta nói rằng đó là một mô hình (model) của tiên đề này. Hai tiên đề tương đương nhau nếu chúng có cùng model nhau.

c.1.2 Các định nghĩa

Một tương đương (ở bên trái) là một khái niệm nguyên tử là một định nghĩa. Các định nghĩa được sử dụng để cung cấp các tên biểu tượng cho những mô tả phức tạp. Chẳng hạn như tiên đề Mother ≡ Woman ⊓ ∃ hasChild.Person định nghĩa mẹ (Mother) là phụ nữ có con. Hay tiên đề Parent ≡ Mother ⊔ Father, định nghĩa bố mẹ (Parent) là người phụ nữ có con hoặc là người đàn ông có con.

Bộ định nghĩa phải rõ ràng. Bộ định nghĩa có ngôi T là một bộ thuật ngữ (TBox) nếu không có tên biểu tượng được định nghĩa hơn một lần, nghĩa là nếu đối với mỗi khái niệm nguyên tử A, có tối đa một tiền đề trong T mà phần bên tay phải là A. Các thuật ngữ (TBox) với các khái niệm liên quan đến các mối quan hệ gia đình:

Woman	≡	Person ⊓ Female
Man	≡	Person ⊓ ¬Woman
Mother	≡	Woman ⊓ ∃ hasChild.Person
Father	≡	Man ⊓ ∃ hasChild.Person
Parent	≡	Father ⊓ Mother
Grandmother	≡	Mother ⊓ ∃ hasChild.Parent
MotherWithManyChildren	≡	Mother ≥ 3hasChild
MotherWithoutDaughter	≡	Mother ⊓ ∀ hasChild.¬Woman
Wife	≡	Woman ⊓ ∃ hasHusband.Man

Các khái niệm nguyên tử thuộc T được chia thành 2 tập hợp, một là các ký hiệu tên N_T, nằm phía bên tay trái của tiên đề và tập hợp còn lại là các ký hiệu nền tảng B_T, chỉ xuất hiện ở bên tay phải của tiên đề. Ký hiệu tên là các khái niệm được định nghĩa và các ký hiệu nền tảng là các khái niệm nguyên thủy.

Nếu chúng ta biết các ký hiệu nền tảng đại diện cho cái gì và T là rõ ràng thì ngữ nghĩa của các ký hiệu tên hoàn toàn được xác định. Như vậy, nếu một thuật ngữ là rõ ràng thì mỗi thuật ngữ tương đương cũng rõ ràng.

Liệu có hay không có các định nghĩa lặp lại (cyclic)? Hãy xem một thuật ngữ có sự lặp lại: Human' ≡ Animal ⊓ ∃ ∀ hasParent.Human'. Human' là một ký hiệu tên, Animal và hasParent là các ký hiệu nền tảng. Human' có thể là tập hợp của tất cả động vật, một vài loài hoặc bất kỳ một tập hợp các động vật khác có đặc điểm là với mỗi động vật, nó cũng có tổ tiên của nó.

Còn nếu thuật ngữ T là không lặp (acyclic) thì nó rõ ràng. Ta có thể mở rộng thuật ngữ T bằng cách thay thế mỗi sự xuất hiện tên ở phía bên tay phải của các định nghĩa bằng các khái niệm tương ứng. T' là ký hiệu của sự mở rộng của T. Như vậy, nếu T là thuật ngữ không lặp và T' là sự mở rộng của T thì T và T' có các ký hiệu tên và ký hiệu nền tảng giống nhau, T và T' tương đương nhau và cả T lẫn T' đều rõ ràng như nhau. Hãy xem sự mở rộng của họ TBox sau:

Woman ≡ Person ⊓ Female

Man ≡ Person ⊓ ¬(Person ⊓Female)

Mother ≡ (Person ⊓ Female) ⊓ ∃ hasChild.Person

Father ≡ (Person ⊓ ¬(Person ⊓Female)) ⊓ ∃ hasChild.Person

Parent ≡ ((Person ⊓ ¬(Person ⊓Female)) ⊓ ∃ hasChild.Person) ⊓
((Person ⊓Female) ⊓ ∃ hasChild.Person)

Grandmother ≡ ((Person ⊓ Female) ⊓ ∃ hasChild.Person) ⊓
∃ hasChild.(((Person ⊓ ¬(Person ⊓ Female)) ⊓
∃ hasChild.Person ⊓ ((Person ⊓Female) ⊓ ∃ hasChild.Person))

MotherWithManyChildren ≡ ((Person ⊓ Female) ⊓ ∃ hasChild.Person) ≥
3 hasChild

MotherWithoutDaughter ≡ ((Person ⊓Female) ⊓ ∃ hasChild.Person) ⊓
∀ hasChild.(¬(Person ⊓ Female))

Wife ≡ (Person ⊓Female) ⊓ ∃ hasHusband.(Person ⊓ ¬(Person ⊓Female))

c.1.3 Các thuật ngữ với các tiên đề bao hàm (inclusion)

Đôi khi có một số khái niệm nào đó mà chúng ta không thể định nghĩa một cách đầy đủ, nhưng chúng ta có thể nêu ra những điều kiện cần thiết cho khái niệm bằng cách sử dụng một bao hàm hay một cụ thể hóa (⊑). Ví dụ như: Woman ⊑ Person.

Một bộ tiên đề là một bộ thuật ngữ tổng quát T nếu mỗi khái niệm nguyên tử nằm bên trái của tiên đề có tối đa một tiên đề.

Có thể chuyển bộ thuật ngữ tổng quát thành bộ thuật ngữ bình thường chỉ chứa các định nghĩa bằng cách chọn một ký hiệu nền tảng mới Ā và thay sự cụ thể hóa A ⊑ C bằng định nghĩa A ≡ Ā ⊓ C. Ví dụ, ta có thể viết lại Woman ⊑ Person như sau:

$$\text{Woman} \equiv \overline{\text{Woman}} \sqcap \text{Person}$$

c.2 ABox (bộ xác nhận)

Thành phần thứ 2 của nền tảng tri thức là ABox (còn gọi là world description). Trong ABox, có thể mô tả một tình trạng cụ thể của một miền ứng dụng dưới dạng các khái niệm và các luật. Một vài các nguyên tử khái niệm và các nguyên tử luật trong ABox có thể được định nghĩa các tên của TBox. Trong ABox, có thể diễn tả các cá thể (individuals) bằng cách đặt cho chúng các tên, và xác nhận các thuộc tính của những cá thể này. Chúng ta ký hiệu tên cá thể là a, b, c. Sử dụng các khái niệm C và R, người ta có thể tạo ra các xác nhận (assertions) của 2 loại sau trong ABox:

C (a) và R (b,c)

C(a) là các xác nhận khái niệm (concept assertions) cho biết a thuộc về C (a là C). R(b,c) là các xác nhận luật (luật assertions) cho biết c là R của b. Ví dụ, nếu PETER, PAUL và MARY là các tên cá thể, thì Father (PETER) có nghĩa là PETER là cha (father), và hasChild(MARY,PAUL) có nghĩa là PAUL là con của MARY. Ví dụ về ABox:

MotherWithoutDaughter(MARY)	Father(PETER)
hasChild(MARY; PETER)	hasChild(PETER;HARRY)
hasChild(MARY; PAUL)	

Chúng ta có thể cung cấp ngữ nghĩa cho các ABox bằng cách mở rộng các biên dịch đến các tên cá thể. Biên dịch $I = (\Delta^I, \cdot^I)$ không chỉ ánh xạ các khái niệm nguyên tử và các luật nguyên tử đến các tập hợp và các quan hệ, mà còn ánh xạ mỗi tên cá thể a đến phần tử $a^I \in \Delta^I$.

Một biên dịch thỏa ABox A nếu nó thỏa mỗi xác nhận trong A: biên dịch I thỏa C(a) nếu $a^I \in C^I$, và biên dịch I thỏa R(a,b) nếu $(a^I, b^I) \in R^I$. Như vậy, I là một mô hình (model) của xác nhận hoặc mô hình của ABox. Do đó, một mô hình của A và T là một sự trừu tượng của thế giới thực (concrete world) mà ở đó các khái niệm được hiểu như các tập con của miền.

D. Suy diễn

Hệ biểu diễn tri thức dựa trên logic mô tả có thể trình diễn các loại lập luận cụ thể và có thể lưu trữ các xác nhận khái niệm và các định nghĩa khái niệm. Nền tảng tri

thức (TBox và ABox) có một ngữ nghĩa tương đương với bộ tiên đề trong logic vị từ bậc nhất (first-order predicate logic). Nền tảng tri thức có thể tạo ra các tri thức tường minh thông qua các suy diễn.

d.1 Nhiệm vụ lập luận các khái niệm

Một khái niệm sinh ra ngữ nghĩa nếu một vài biên dịch thỏa các tiên đề T (mô hình T) để khái niệm biểu thị một tập hợp không rỗng trong biên dịch đó. Một khái niệm với thuộc tính này có thể thỏa đối với T và có thể không thỏa đối với cái khác.

Kiểm tra tính thỏa mãn của các khái niệm là một suy diễn quan trọng. Sự thỏa mãn này có liên quan đến vấn đề xếp gộp (subsumption). Khái niệm C xếp gộp vào khái niệm D nếu trong mỗi mô hình T, tập hợp do C bao hàm là một tập hợp con của tập do D bao hàm. Các thuật toán kiểm tra sự xếp gộp cũng được sử dụng để sắp xếp có hệ thống các khái niệm của một TBox theo một nguyên tắc chung. Các mối quan hệ giữa các khái niệm đáng quan tâm là sự tương đương (equivalence) và tách biệt (disjointness). Giả sử T là một TBox, thì:

- Tính thỏa mãn (satisfiability): khái niệm C thỏa đối với T nếu tồn tại một mô hình I của T để C^I không rỗng. Như vậy, I là một model của C.
- Sự xếp gộp (subsumption): khái niệm C xếp gộp vào khái niệm D đối với T nếu $C^I \sqsubseteq D^I$ cho mỗi model I của T. Như vậy, ta viết là $C \sqsubseteq_T D$ hoặc $T \models C \sqsubseteq D$.
- Sự tương đương (equivalence): hai khái niệm C và D tương đương nhau đối với T nếu $C^I = D^I$ cho mỗi model I của T. Như vậy, ta viết $C \equiv_T D$ hoặc $T \models C \equiv D$.
- Tách biệt (disjointness): hai khái niệm C và D là tách biệt (disjoint) đối với T nếu $C^I \cap D^I = \varnothing$ cho mỗi model I của T.

Nếu TBox T là rõ ràng, thì chúng ta có thể bỏ qua lượng từ “with respect to T”. Và chúng ta cũng bỏ qua lượng từ trong trường hợp TBox là rỗng, và chúng ta viết $\models C \sqsubseteq D$ nếu C xếp gộp vào D, và $\models C \equiv D$ nếu C và D tương đương nhau.

Giảm xuống sự xếp gộp: với C, D là các khái niệm, thì:

- ✓ C không thỏa <=> C được xếp gộp vào ⊥
- ✓ C và D tương đương nhau <=> C được xếp gộp vào D và D được xếp gộp vào C
- ✓ C và D là disjoint <=> C ⊓ D được xếp gộp vào ⊥
- ✓ Các phát biểu cũng đúng đối với TBox

Giảm xuống tính không thỏa mãn:

- ✓ C được xếp gộp vào D <=> C ⊓ ¬D là không thỏa mãn
- ✓ C và D tương đương nhau <=> cả (C ⊓ ¬D) và (¬C ⊓ D) đều không thỏa mãn
- ✓ C và D là những disjoint <=> C ⊓ D là không thỏa mãn
- ✓ Các phát biểu cũng đúng đối với TBox

Giảm bớt tính không thỏa mãn (Reducing Unsatisfiability): cho C là một khái niệm:

- ✓ C là không thỏa mãn <=>
- ✓ C được xếp gộp vào ⊥ <=>
- ✓ C và ⊥ tương đương nhau <=> C
- ✓ T là disjoint
- ✓ Các phát biểu cũng đúng đối với TBox

d.2 Khử TBox

Nếu T là một TBox không lặp (acyclic) thì chúng ta có thể chuyển các vấn đề lập luận đối với T sang các vấn đề đối với TBox rỗng. Trong sự mở rộng (expansion), mỗi định nghĩa thuộc trong những dạng A ≡ D để D chỉ chứa các ký hiệu nền tảng nhưng không chứa các ký hiệu tên. Đối với mỗi khái niệm C, chúng ta định nghĩa sự khai triển của C đối với T là khái niệm C’ bằng cách thay thế mỗi sự xuất hiện của ký hiệu tên A trong C bằng khái niệm D (ở đó A ≡ D là định nghĩa của A trong T’) – một sự mở rộng của T. Ví dụ như, chúng ta mở rộng khái niệm Woman ⊓ Man đối với TBox.

Khi thay thế Woman là Person ⊓ Female và Man là Person ⊓ ¬(Person ⊓ Female) ta sẽ được khái niệm: Person ⊓ Female ⊓ Person ⊓ ¬(Person ⊓ Female).

Cho C là khái niệm và C' là mở rộng của C, ta có $C \equiv_T C'$: C thỏa đối với T nếu và chỉ nếu C' thỏa đối với T. Tuy nhiên, C' không chứa các tên được định nghĩa, và C' thỏa đối với T nếu và chỉ nếu nó thỏa.

Nếu D là một khái niệm khác thì chúng ta cũng có $D \equiv_T D'$. Vì vậy, $C \sqsubseteq_T D$ nếu và chỉ nếu $C' \equiv_T D'$, và $C \equiv_T D$ nếu và chỉ yếu $C' \equiv_T D'$. Mặc khác, vì C' và D' chỉ chứa các ký hiệu nền tảng nên điều này ngụ ý rằng:

$T \models C \sqsubseteq D$ nếu và chỉ nếu $\models C' \sqsubseteq D'$

$T \models C \equiv D$ nếu và chỉ nếu $\models C' \equiv D'$

Với các đối số (arguments) giống nhau, ta có thể biểu diễn: C và D là tách biệt (disjoint) đối với T nếu và chỉ yếu C' và D' là tách biệt (disjoint).

Việc khai triển các khái niệm đối với TBox không lặp sẽ làm gia tăng TBox trong vấn đề lập luận. Tuy nhiên, trong trường hợp xấu nhất, kích cỡ của T' tăng lũy thừa theo kích cỡ của T, nên C' lớn hơn C thông qua một nhân tố (factor) gia tăng lũy thừa theo kích cỡ của T.

d.3 Nhiệm vụ lập luận các ABox

Sau khi xây dựng xong bộ thuật ngữ và sử dụng các dịch vụ lập luận của logic mô tả để kiểm tra sự thỏa mãn của tất cả các khái niệm và các mối quan hệ xếp gộp, thì ABox được hoàn thiện bởi các xác nhận (assertions) về các cá thể (individuals). ABox gồm có C(a) (xác nhận khái niệm) và R(a,b) (xác nhận luật). Tuy nhiên, sự biểu diễn tri thức phải nhất quán với nhau bởi vì theo quan điểm logic, nếu không thì sẽ rút ra những kết luận tùy ý. Chẳng hạn như, nếu ABox chứa các xác nhận Mother(MARY) và Father(MARY), thì hệ thống có thể xác nhận rằng phát biểu này là không nhất quán.

Theo ngữ nghĩa lý thuyết mô hình, chúng ta có thể cung cấp định nghĩa hình thức nhất quán. Một ABox A nhất quán đối với TBox T nếu có một biên dịch là mô hình

của cả A và T. Như vậy, A nhất quán nếu nó nhất quán đối với TBox rỗng. Ví dụ, một bộ xác nhận {Mother(MARY), Father(MARY)} nhất quán (đối với TBox rỗng) bởi vì, không có bất kỳ giới hạn nào về biên dịch Mother và Father, hai khái niệm được hiểu giống nhau vì chúng có cùng một phần tử. Tuy nhiên, các xác nhận thì không nhất quán đối với TBox bởi vì trong mỗi mô hình của nó, Mother và Father là những tập hợp disjoint.

Tương tự đối với các khái niệm (TBox), việc kiểm tra tính nhất quán của ABox đối với TBox không lặp (acyclic) sẽ làm giảm bớt việc kiểm tra sự mở rộng của ABox. Định nghĩa sự mở rộng của A đối với T là ABox A’ bằng cách thay thế mỗi xác nhận khái niệm C(a) trong A bằng xác nhận C’(a), với C’ là sự mở rộng của C đối với T. Trong mỗi mô hình của T, khái niệm C và sự mở rộng của nó C’ được hiểu giống nhau. Vì vậy, A’ nhất quan đối với T nếu và chỉ nếu sự mở rộng A’ nhất quán.

Trong trường hợp tính nhất quán, nhiệm vụ lập luận đối với ABox đối với TBox không lặp có thể được chuyển sang lập luận về các ABox được mở rộng.

ABox A có thể đưa ra các truy vấn về các mối quan hệ giữa các khái niệm, các luật và các cá thể. Suy diễn của ABox nguyên mẫu mà các truy vấn đó dựa vào là sự kiểm tra thực thể (instance check) hoặc kiểm tra liệu một xác nhận có được kế thừa từ một ABox hay không? Khi xác nhận a được kế thừa từ A và ta viết A |= a nếu mỗi biên dịch thỏa A, nghĩa là mỗi mô hình của A cũng thỏa a. Nếu a là một xác nhận luật, thì sự kiểm tra thực thể là dễ dàng bởi vì ngôn ngữ mô tả không chứa các constructor để tạo ra các luật phức tạp. Nếu a là dạng C(a) thì chúng ta có thể chuyển sự kiểm tra thực thể thành vấn đề nhất quán đối với ABox bởi vì có sự liên kết:

$$A \models C(a) \text{ nếu và chỉ nếu } A \cup \{\neg C(a)\} \text{ không nhất quán}$$

Như vậy, lập luận các khái niệm có thể làm giảm bớt sự kiểm tra tính nhất quán. Tương tự, tính thỏa mãn của khái niệm có thể làm giảm bớt tính nhất quán của ABox bởi vì đối với mỗi khái niệm C, chúng ta có: C thỏa mãn nếu và chỉ nếu {C(a)} nhất quán (với a là tên cá thể tùy ý).

E. Luật

Nền tảng tri thức bao gồm TBox T và ABox A. Chúng ta biểu thị tri thức đó bằng một cặp K = (T, A).

Trong hệ logic mô tả như CLASSIC hoặc LOOM, ngoài các thuật ngữ và các mô tả thế giới (world description) còn có sử dụng các luật (rules) để diễn đạt tri thức. Sự khác nhau cơ bản nhất của các luật đó là những diễn đạt dạng: C => D, với C và D là các khái niệm. Luật này nói rằng "nếu một cá thể (individual) được chứng minh là một thực thể (instance) C thì sinh ra một thực thể (instance) D". Các luật đó thường được gọi là luật xuôi (trigger rule_luật một chiều xuôi).

Ngữ nghĩa của tập hợp có ngôi R của các luật xuôi có thể được mô tả bằng quá trình lập luận xuôi.

Có thể sử dụng ngữ nghĩa khai báo của các tiên đề bao hàm (inslusion) như ngữ nghĩa cho các luật. Tuy nhiên, sự khác biệt quan trọng giữa luật xuôi C => D và tiên đề bao hàm C $\sqsubseteq$ D là luật xuôi không tương đương với sự phủ định của nó ¬C => ¬D. Hơn nữa, khi áp dụng các luật xuôi, không thực hiện phân tích trường hợp. Chẳng hạn như, các bao hàm (inclusions) C $\sqsubseteq$ D và ¬C $\sqsubseteq$ D hàm ý rằng mỗi đối tượng thuộc D, trong khi đó không có luật xuôi nào C => D và ¬C => D áp dụng cho cá thể (individual) a mà cả C(a) và ¬C(a) đều không được chứng minh.

Để đạt được ngữ nghĩa của các luật xuôi theo cách khai báo, chúng ta phải tăng cường logic mô tả bằng toán tử tri thức K, mà không tham chiếu đến các đối tượng trong miền, nhưng tham chiếu đến nền tảng tri thức biết về miền nào. Đầu tiên, chúng ta thêm một trường hợp vào luật cú pháp để tạo ra các khái niệm tri thức:

C, D → KC (khái niệm tri thức)

Khái niệm KC cho biết các đối tượng đó là các thực thể (instances) của C. Kế đến, dùng K để chuyển các luật xuôi C => D thành các tiên đề bao hàm: KC $\sqsubseteq$ D.

Toán tử K đứng trước khái niệm C có tác dụng là tiên đề chỉ áp dụng cho các cá thể (individuals) xuất hiện trong ABox và TBox, và ngụ ý chúng là các thực thể (instances) của C. Vì thế, đã ngăn cản tiên đề bao hàm ảnh hưởng đến tính thỏa mãn

hoặc các mối quan hệ xếp gộp giữa các khái niệm. Nhờ đó, chúng ta sẽ định nghĩa ngữ nghĩa hình thức cho toán tử K có hiệu quả rất cao.

Nền tảng tri thức luật là một bộ ba K = (T, A, R), với T là TBox, A là ABox và R là bộ luật, được viết như các tiên đề dạng KC ⊑ D. Sự mở rộng thủ tục của bộ ba đó là nền tảng tri thức $\bar{\mathcal{K}}$ = (T, Ā) có được từ (T, A) bằng cách áp dụng các luật xuôi.

Chúng ta định nghĩa biên dịch tri thức là một cặp (I, W), với I là biên dịch bậc nhất và W là tập con của biên dịch bậc nhất. Mỗi biên dịch bậc nhất sinh ra một đơn ánh $\cdot^{I,W}$ liên kết các khái niệm và các luật lần lượt với các tập con Δ và ΔxΔ. Đối với khái niệm mức đỉnh (top) (T), khái niệm mức dưới (bottom) (⊥), đối với các khái niệm nguyên tử phủ định và các luật nguyên tử, thì $\cdot^{I,W}$ thỏa $\cdot^{I}$. Đối với các giao nhau (intersections), các giới hạn giá trị (value restrictions) và các lượng từ tồn tại (existential quantifications), thì định nghĩa tương tự như một trong số $\cdot^{I}$:

$$(C \sqcap D)^{I,W} = C^{I,W} \cap D^{I,W}$$

$$(\forall R.C)^{I,W} = \{a \in \Delta \mid \forall b.\ (a,b) \in R^{I,W} \rightarrow b \in C^{I,W}\}$$

$$(\exists R.T)^{I,W} = \{a \in \Delta \mid \exists b.\ (a,b) \in R^{I,W}\}$$

Đối với các constructor khác, $\cdot^{I,W}$ được định nghĩa tương tự. Tuy nhiên, đối với các khái niệm C mà không có sự xuất hiện K, thì các tập $C^{I,W}$ và C^{I} là giống nhau. Tập biên dịch W trở nên quan trọng khi chúng ta định nghĩa ngữ nghĩa của toán tử tri thức:

$$(\mathbf{K}C)^{\mathcal{I},\mathcal{W}} = \bigcap_{\mathcal{J} \in \mathcal{W}} C^{\mathcal{J},\mathcal{W}}$$

Nó cũng cho phép toán tử K xuất hiện trước các luật và định nghĩa ngữ nghĩa của các diễn đạt luật dạng KR một cách tương tự.

Một biên dịch tri thức (I,W) thỏa một bao hàm C ⊑ D nếu $C^{I,W} \sqsubseteq D^{I,W}$, và một tương đương C ≡ D nếu $C^{I,W} = D^{I,W}$. Nó thỏa một xác nhận C(a) nếu $a^{I,W} = \gamma^{I,W} \in C^{I,W}$, và một xác nhận R(a,b) nếu $(a^{I,W}, b^{I,W}) = (\gamma(a), \gamma(b)) \in R^{I,W}$. Nó thỏa một nền tảng tri thức luật K(T, A, R) nếu nó thỏa mọi tiên đề trong T, mọi xác nhận trong A và mọi luật trong R.

Một mô hình tri thức (epistemic model) cho một luật hoặc một nền tảng tri thức luật K là một tập không rỗng lớn nhất W của các biên dịch bậc nhất để mà đối với mỗi I $\in$W, thì biên dịch tri thức (I,W) thỏa K.

Nếu (T,A) thỏa bậc nhất, thì tập các mô hình bậc nhất của (T,A) chỉ là mô hình trí thức của nền tảng tri thức luật K = (T, A, Ø) mà bộ luật của nó là rỗng.

Cho K = (T,A,R) là nền tảng tri thức luật sao cho (T,A) là thỏa bậc nhất. Khi đó, K là một mô hình trí thức duy nhất. Ví dụ, cho R gồm luật Kstudent $\sqsubseteq \forall$eats.JunkFood. Luật này cho biết "những cá thể (individuals) đó là những sinh viên (students) chỉ ăn thức ăn muối (junk food)".

Tuy nhiên, về mặt ngữ nghĩa trí thức, có thể các luật không cho phép lập luận tương phản. Chúng ta hãy xem nền tảng tri thức luật sau:

$$A2 = \{\neg \forall \text{eats.JunkFood(PETER)}\}$$

Trong trường hợp này, ={$\neg \forall$eats.JunkFood(PETER) là đúng trong mọi biên dịch bậc nhất của mô hình trí thức W. Tuy nhiên, vì tính tối đa của W, nên có ít nhất một biên dịch trong W mà Peter là một student và một người ai khác mà ở đó Peter không phải là student. Vì vậy, Peter không được biết là một student.

Như vậy, sự mở rộng thủ tục của một nền tảng tri thức luật K chỉ chứa các xác nhận phải được thỏa bởi mô hình trí thức của K. Cho K = (T,A,R) là nền tảng tri thức luật. Nếu (T,A) là thỏa bậc nhất, thì mô hình trí thức của K chứa chính xác các mô hình bậc nhất của sự mở rộng thủ tục $\mathcal{K}$ = (T, Ā).

2.2.2 Phương pháp phân tích câu tiếng Việt bằng ngôn ngữ BNF/EBNF

2.2.2.1 Ngôn ngữ BNF [4]

A. Giới thiệu

BNF viết tắt từ Backus-Naur Form, là một trong những ngôn ngữ phân tích cấu trúc văn phạm thông dụng nhất. BNF là một dạng toán học hình thức (formal

mathematical way) dùng để mô tả ngôn ngữ, được John Backus phát triển và tiếp tục được Peter Naur cải tiến thêm để mô tả cú pháp của ngôn ngữ lập trình (Algol 60).

B. Cú pháp của BNF

Cú pháp của ngôn ngữ lập trình bao gồm các ký hiệu (symbols) và các luật (rules), và có thể được định rõ bằng cách sử dụng giản đồ dòng (flow diagrams) hoặc bằng BNF. Tuy nhiên, người ta thường sử dụng BNF hơn. BNF mô tả những yêu cầu vào (input) và ra (output) đối với những dự án lập trình.

b.1 Các ký hiệu (symbols) và thuật ngữ (terminology):

Production: phát biểu được viết trong BNF.

<>: cho biết từ (word) đang được định nghĩa hoặc được định nghĩa. Các từ và các ký hiệu xuất hiện mà không có dấu ngoặc nhọn <> là những nguyên thể (literal) và xuất hiện trong ngôn ngữ đang được định nghĩa một cách chính xác khi chúng được định rõ.

::= (:=): là một thực thể của (“is an instance of”).

|: có nghĩa là “hoặc” (or).

b.2 Ví dụ

- “Một id có thể là a hoặc b”. Ta có thể biểu diễn câu này trong BNF như sau: **<id> ::= a|b**
- “Một id có thể gồm 1 đến 3 chữ cái”. Ta có thể biểu diễn câu này trong BNF như sau:

  <id> ::= <letter>|<letter><letter>|<letter><letter><letter>

  <letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|

  A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
- “Một id có thể gồm một hay nhiều chữ cái”. Ta có thể biểu diễn câu này trong BNF như sau:

  <id> ::= <letter>|<letter><id>

  <letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|

A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

- "Một tên gồm có tên (first name), tên lót (middle initial) và họ (last name)". Ta có thể biểu diễn câu này trong BNF như sau:

  <name> ::= <firstname><blank><middleinit>.<blank><lastname>
  <firstname> ::= <capletter><lowerletter>|<firstname><lowerletter>
  <capletter> ::=
  A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
  <lowerletter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
  <blank> ::= ' '
  <middleinit> ::= <capletter>
  <lastname> ::= <firstname>

2.2.2.2 Ngôn ngữ EBNF [5]

A. Giới thiệu

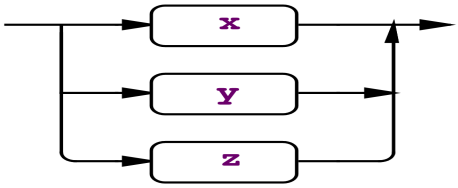
Ngôn ngữ BNF rất cồng kềnh khi chỉ rõ những thứ có tính lặp đi lặp lại nhiều lần bởi vì chúng ta phải sử dụng đệ quy. Trong khi EBNF (Extended BNF) có hỗ trợ thêm toán tử tùy chọn (optional operators) và toán tử lặp (repetition operators) cũng như có thêm các luật con được đặt trong dấu ngoặc đơn () để hỗ trợ việc mô tả văn phạm thêm ngắn gọn và súc tích hơn. Tất cả các khối phần tử văn phạm được đặt trong dấu ngoặc đơn () được xem như là những luật con (subrules).

Ngữ pháp EBNF là ngữ pháp phi ngữ cảnh (context-free grammars) bởi vì chúng ta không giới hạn các luật đối với ngữ cảnh nào đó. Chẳng hạn như, chúng ta không thể ràng buộc một luật diễn đạt cho những tình huống mà bộ nhận diện (recognizer) đã so khớp (match) hoặc sẽ so khớp luật khác.

B. Cú pháp của EBNF

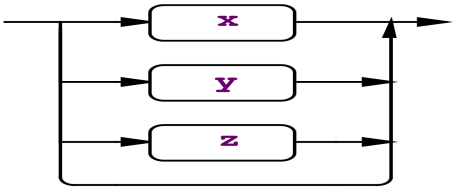
Các phần tử của văn phạm (chẳng hạn như x, y, z…) có thể kèm theo sau bởi các toán tử hoặc không kèm theo các toán tử (?, *, +…), hoặc chúng có thể được đặt trong dấu ngoặc đơn () để hình thành nên các luật con.

(«x»|«y»|«z»): cho biết chỉ thỏa hoặc x hoặc y hoặc z trong luật con mà thôi.

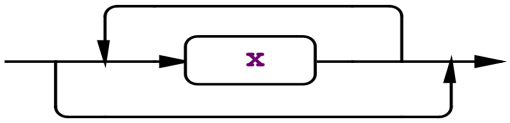


x?: cho biết phần tử x là tùy ý (x không xuất hiện hoặc x xuất hiện một lần).

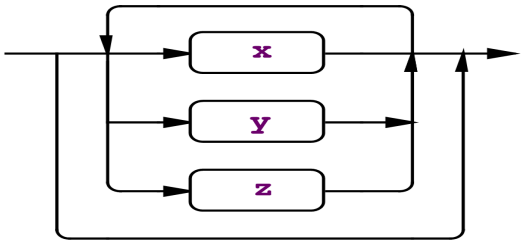
(«x»|«y»|«z»)?: giống như cú pháp x?, tức là cho biết luật con («x»|«y»|«z») không xuất hiện hoặc xuất hiện một lần.



x*: cho biết phần tử x có thể không xuất hiện hoặc có lặp lại (từ một lần trở lên).

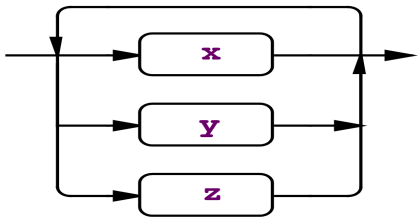


(«x»|«y»|«z»)*: giống như cú pháp x*, tức là cho biết luật con («x»|«y»|«z») không xuất hiện hoặc có lặp lại (từ 1 lần trở lên).



x+: cho biết phần tử x xuất hiện một lần hoặc nhiều lần

(«x»|«y»|«z»)+: giống như cú pháp x+, tức là cho biết luật con («x»|«y»|«z») xuất hiện một lần hoặc nhiều lần



* Ký hiệu:

«x»: biểu thị một đoạn văn phạm x

«y»: biểu thị một đoạn văn phạm y

«z»: biểu thị một đoạn văn phạm z

(): biểu thị một luật con

?: cho biết phần tử có thể xuất hiện một lần hoặc không xuất hiện (tùy ý)

*: cho biết phần tử không xuất hiện hoặc có sự lặp lại

+: cho biết phần tử có thể xuất hiện 1 hoặc nhiều lần

2.2.2.3 So sánh ngôn ngữ BNF và EBNF

EBNF không mạnh hơn BNF trong vấn đề ngôn ngữ mà nó chỉ thuận tiện hơn trong việc viết cú pháp (cú pháp gọn và súc tích hơn) bằng cách có hỗ trợ thêm một vài toán tử, nên EBNF thường được sử dụng hơn.

2.2.2.4 Một số chương trình xử lý ngôn ngữ BNF/EBNF

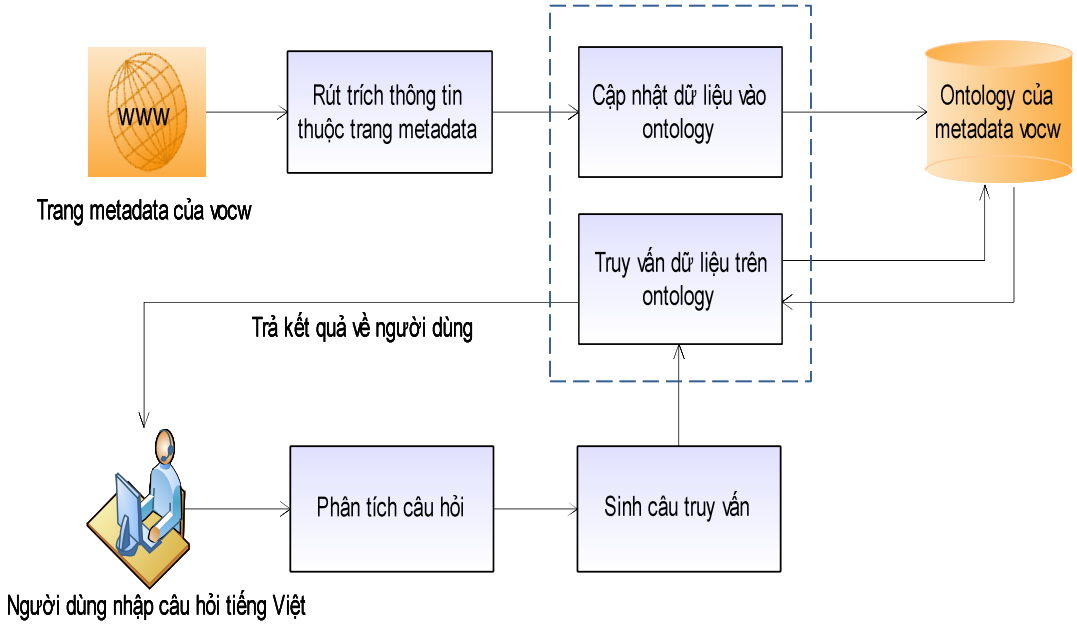
YACC (Yet Another Compiler Compiler).

ANTLR (ANother Tool for Language Recognition) (chẳng hạn như ANTLRWorks…).

CHƯƠNG III - XÂY DỰNG CHƯƠNG TRÌNH

3.1 Mô hình hệ thống

Hệ thống bao gồm bốn thành phần theo hai luồng chính:



Hình 3.1 Mô hình hệ thống

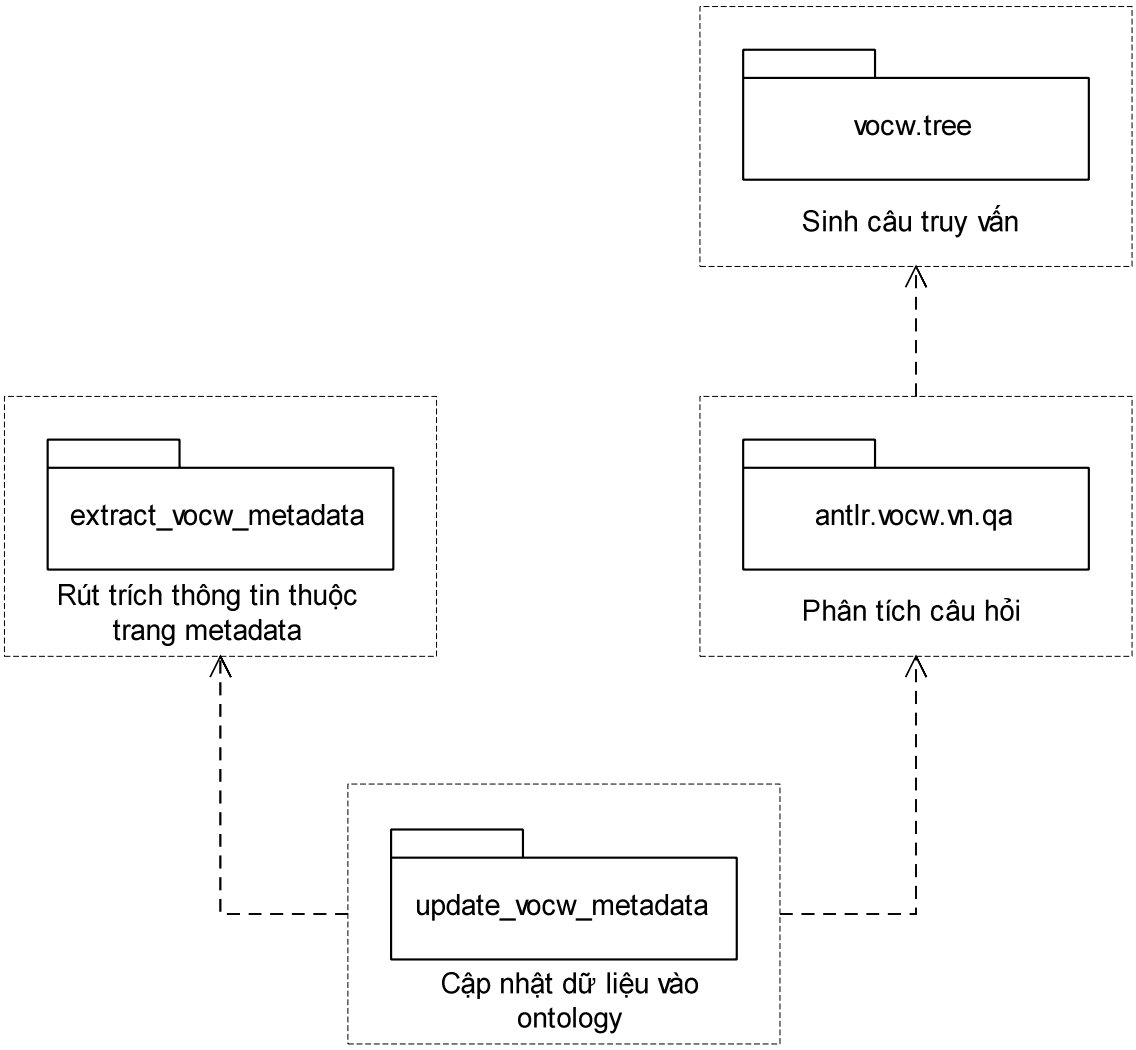
3.1.1 Bốn thành phần của hệ thống

- Phân tích câu hỏi
- Sinh câu truy vấn
- Cập nhật và truy vấn dữ liệu trên ontology
- Rút trích thông tin thuộc trang metadata

3.1.2 Hai luồng dữ liệu chính của hệ thống

- Luồng dữ liệu từ việc lấy thông tin từ trang metadata của kho học liệu mở (vocw) và cập nhật dữ vào ontology:
  - Để thuận tiện cho việc xây dựng thử nghiệm chương trình, toàn bộ trang web vocw được lưu về máy cục bộ. Nội dung trang web được cập nhật định kỳ về máy cục bộ.
  - Thành phần rút trích thông tin tìm kiếm các trang metadata, lấy thông tin của các trường dữ liệu và cập nhật dữ liệu vào ontology của metadata vocw.
- Luồng dữ liệu từ câu hỏi người dùng nhập liệu và kết quả hệ thống trả về:
  - Sau khi người dùng đưa câu hỏi vào, thành phần phân tích câu hỏi của hệ thống thực hiện phân tích câu hỏi của người dùng.
  - Hệ thống định nghĩa trước một danh sách các dạng cú pháp của câu hỏi. Nếu câu hỏi người dùng không thuộc danh sách này, hệ thống trả về kết quả không phân tích được và dừng chương trình. Ngược lại, câu hỏi sau khi được phân tích sẽ sinh ra cây cú pháp tương ứng.
  - Cây cú pháp này được chuyển sang thành phần sinh câu truy vấn. Tại đây, cây cú pháp chuyển thành câu truy vấn SPARQL. Câu truy vấn này sẽ lấy dữ liệu từ ontology của metadata vocw để trả về cho người dùng.

3.2 Các thành phần của hệ thống



Hình 3.2 Các thành phần của hệ thống

Các thành phần được đóng gói trong hệ thống:

- Phân tích câu hỏi: antlr.vocw.vn.qa
- Sinh câu truy vấn: vocw.tree
- Cập nhật và truy vấn dữ liệu trên ontology: update_vocw_metadata
- Rút trích thông tin thuộc trang metadata: extract_vocw_metadata

3.2.1 Thành phần phân tích câu hỏi

3.2.1.1 Dữ liệu vào và ra của thành phần

- Dữ liệu vào: câu hỏi tiếng Việt
- Dữ liệu ra: cây sinh mã

3.2.1.2 Các từ khoá

A. Câu hỏi tiếng Việt

Trong thành phần này, câu hỏi tiếng Việt được chuyển sang dạng mã Unicode. Miền xác định là câu hỏi người dùng chỉ gồm các vấn đề liên quan đến khóa học như đã đề cập ở phần 1.2.1.

Ví dụ 1: Ai đã có viết sách “Toan” và sách “Van” trong năm 2009?

Dạng mã Unicode:

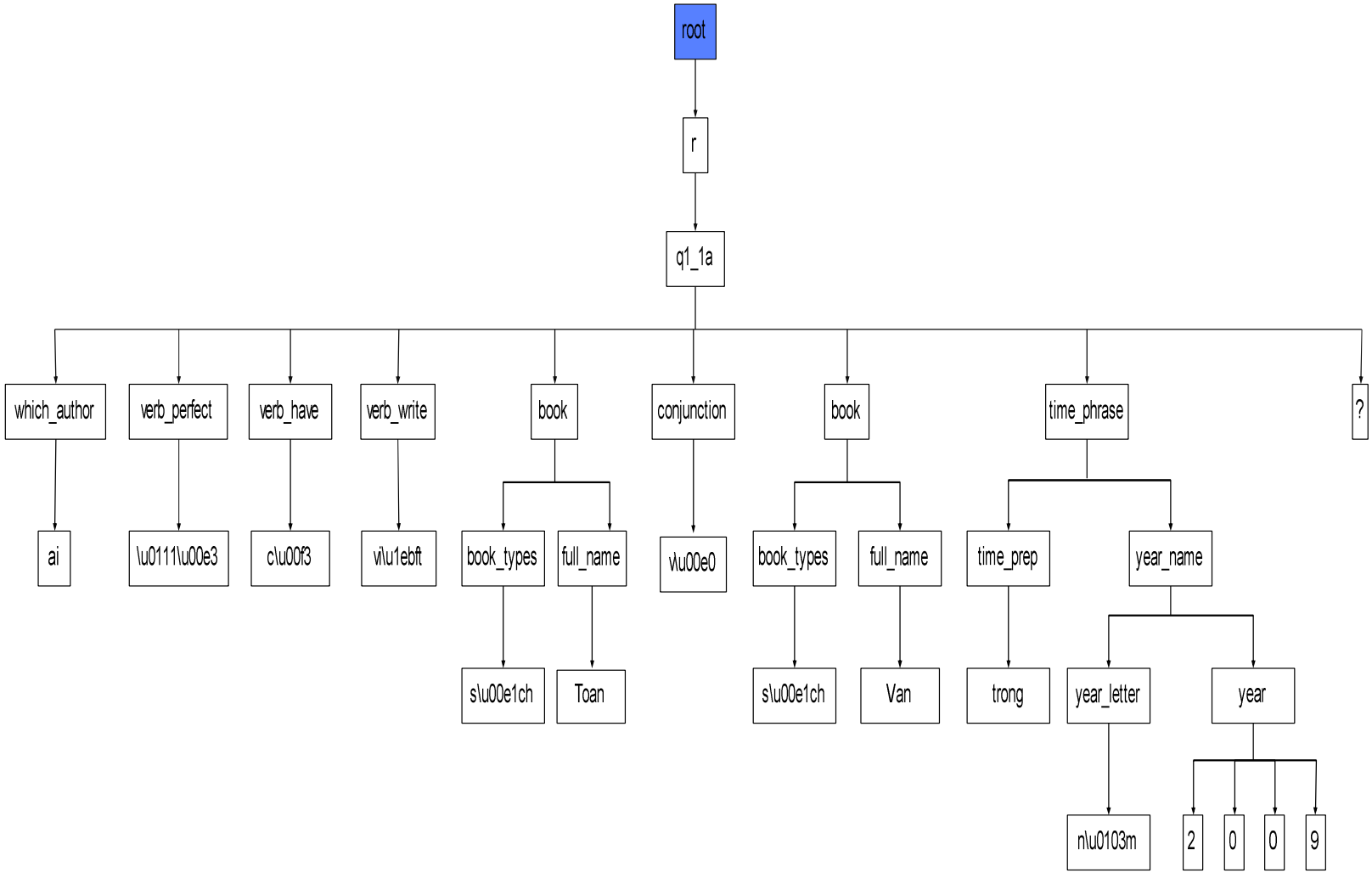
ai \u0111\u00e3 c\u00f3 vi\u1ebft s\u00e1ch "Toan" v\u00e0 s\u00e1ch "Van" trong n\u0103m 2009 ?

Miền xác định câu hỏi: tác giả, sách, năm xuất bản.

B. Cây cú pháp

Cây cú pháp được tạo ra trong quá trình phân tích luật (cú pháp) EBNF. Trong đó nút gốc là tên luật. Các nút còn lại chứa luật con.

Trở lại ví dụ 1: biểu thức EBNF cho những dạng câu hỏi như ví dụ 1:



Hình 3.3 Cây cú pháp sau khi phân tích ví dụ 1

C. Cây sinh mã truy vấn

Cây sinh mã truy vấn được tạo ra bằng ánh xạ lên một số nút của cây cú pháp.

c.1 Đặc điểm của ngôn ngữ truy vấn SPARQL

- Xác định thành phần truy vấn (select)
- Xác định địa chỉ lưu trữ ontology (from)
- Xác định điều kiện (quan hệ) truy vấn

c.2 Quá trình ánh xạ

- Thông tin truy vấn: các từ nghi vấn trong câu hỏi tiếng Việt.
- Địa chỉ lưu trữ ontology: chỉ đến một trang ontology chính chứa tham chiếu đến tất cả các trang ontology khác.
- Điều kiện truy vấn dựa vào mối quan hệ ngữ nghĩa giữa các danh từ trong câu hỏi tiếng Việt. Mối quan hệ có thể là động từ hay giới từ . Mỗi mối quan hệ là một câu thuộc điều kiện truy vấn. Các câu này có thể liên kết bằng liên kết "và" hoặc "hoặc".

c.3 Xét ví dụ trên

Ai đã có viết sách "Toan" và sách "Van" trong năm 2009?

- Thông tin truy vấn: ai
- Địa chỉ lưu trữ ontology: http://localhost/owl_test/vocw_full.owl
- Điều kiện truy vấn:
  - ✓ Ai viết sách: mối quan hệ "viết" giữa danh từ "ai" và "sách"
  - ✓ Sách "Toan": mối quan hệ "tên là" giữa danh từ "sách" và "Toan"
  - ✓ Sách "Van": mối quan hệ "tên là" giữa danh từ "sách" và "Van"
  - ✓ (Ai viết) sách trong năm: mối liên hệ "được viết" giữa danh từ "sách" và "năm"

Do địa chỉ lưu trữ không thay đổi nên chỉ xét các nút của cây sinh mã để phân loại vào thành phần truy vấn và điều kiện truy vấn. Các nút này thuộc một trong ba loại lưu trữ sau:

- Nút gốc chỉ lưu: thông tin truy vấn
- Nút con lưu mối quan hệ hai ngôi:
  - ✓ Quan hệ thuộc tính đối tượng (Object property): là quan hệ hai ngôi giữa hai đối tượng .
  - ✓ Quan hệ thuộc tính dữ liệu (Datatype property): là quan hệ hai ngôi giữa đối tượng và giá trị thực (có kiểu dữ liệu cơ bản).
- Nút lá lưu mối quan hệ hai ngôi:
  - ✓ Quan hệ thuộc tính đối tượng (Object property): là quan hệ hai ngôi giữa hai đối tượng .
  - ✓ Quan hệ thuộc tính dữ liệu (Datatype property): là quan hệ hai ngôi giữa đối tượng và giá trị thực (có kiểu dữ liệu cơ bản).

Quá trình phân tích cú pháp còn thực hiện ánh xạ tại mỗi nút trên cây cú pháp cho ra cây sinh mã truy vấn.

- Ánh xạ ontology:

  f1: nút → {từ để hỏi, từ vựng của ontology}

- Ví dụ:

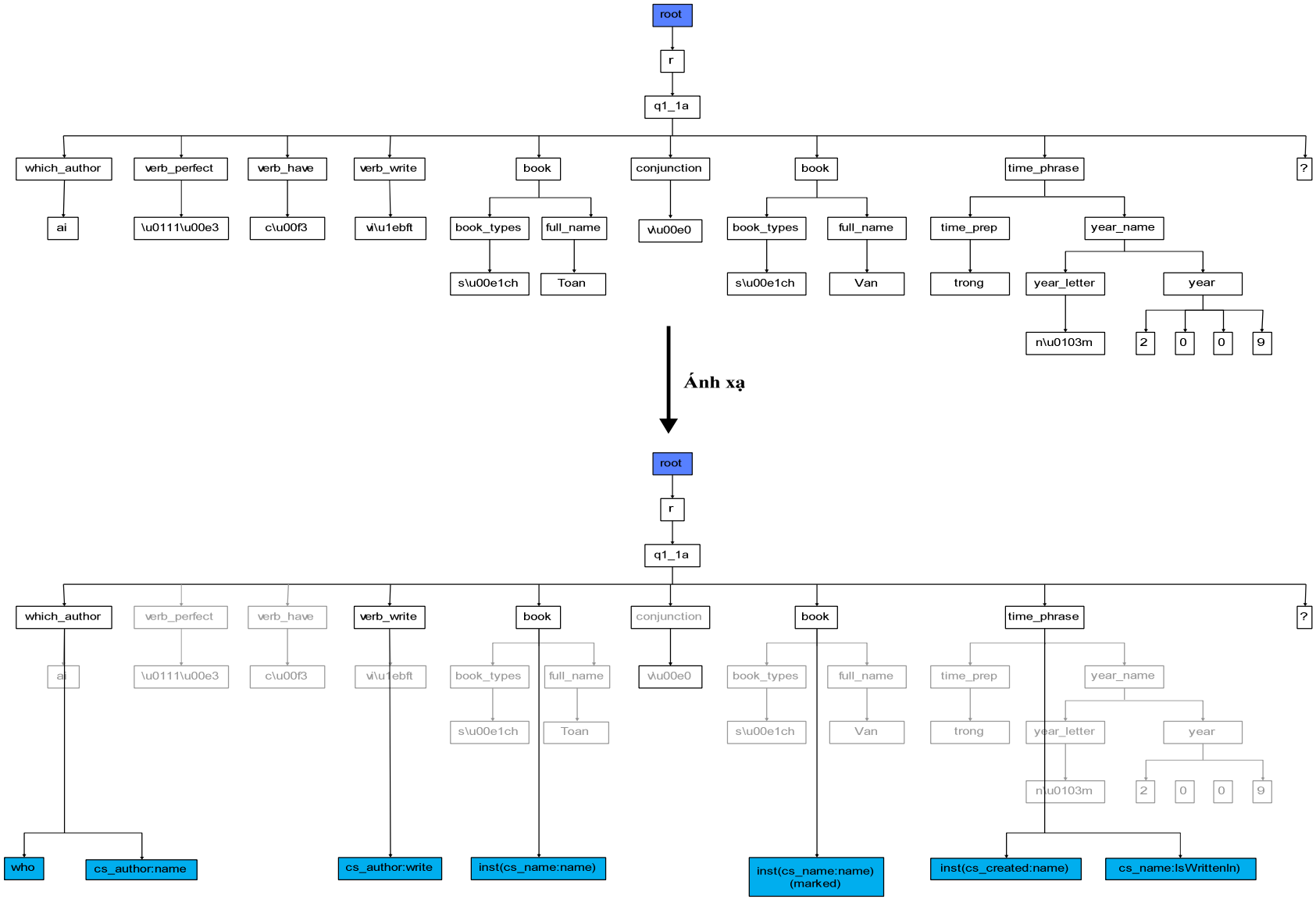
  which_author → {who, cs_author:name}

  verb_write → cs_author:write

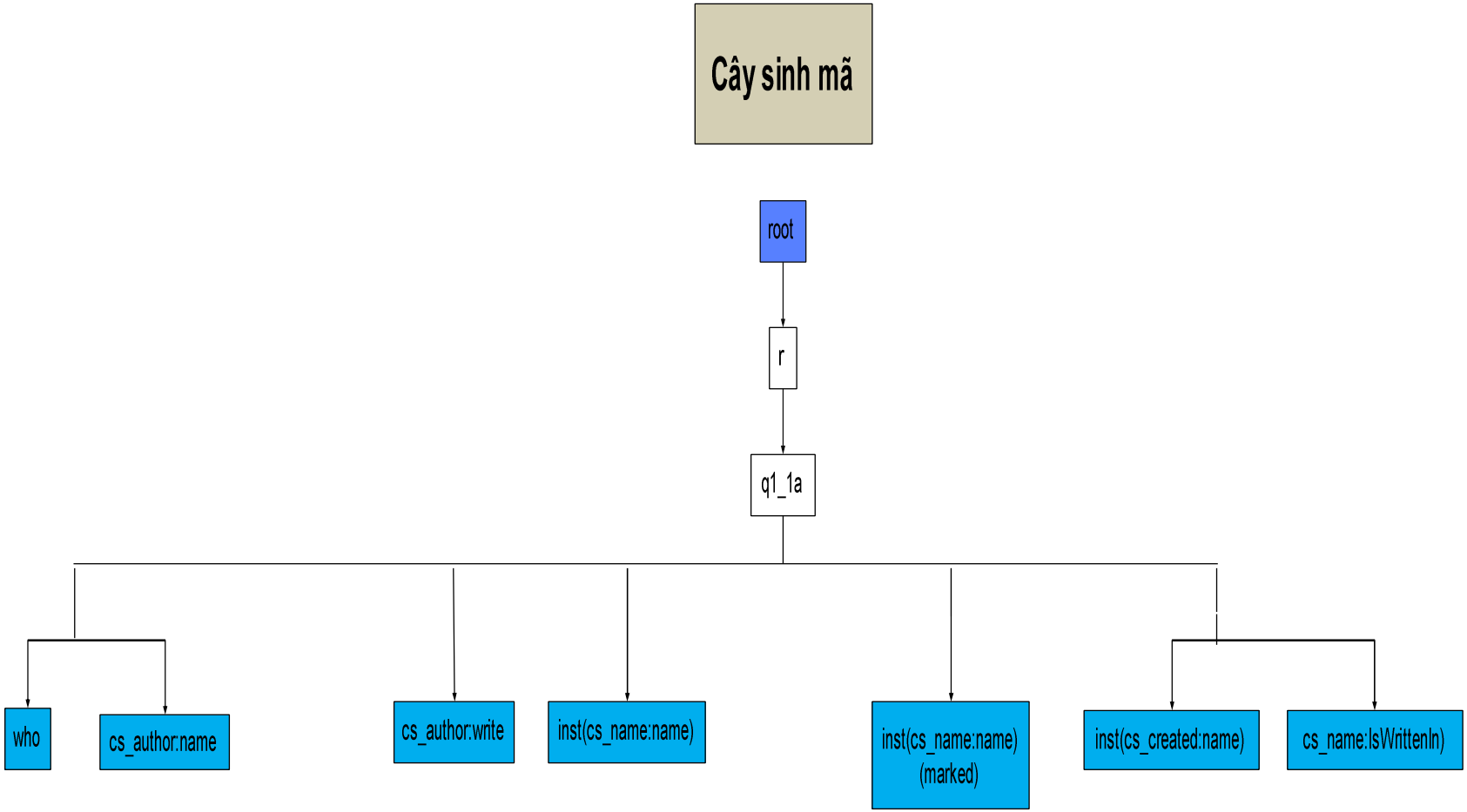
  book → inst(cs_name:name)

  year_name → {inst(cs_created:name), cs_name:isWrittenIn }

- Từ cây cú pháp ở hình 3.3, mô phỏng ánh xạ lên cây cú pháp để tạo cây sinh mã.



Hình 3.4 Thực hiện ánh xạ lên nút thuộc cây cú pháp



Hình 3.5 Cây sinh mã truy vấn

c.4 Định dạng XML của cây sinh mã

*** Nút:**

- **Trường hợp 1:** mô tả cho một bộ ba (triples) của ontology

```
<item key="">
        <sentence option="intersection">
                <domain var="1">{từ vựng}</domain>
                <range var="1" select="0">{từ vựng}</range>
                <relation>{mối liên hệ (object property) giữa hai từ vựng}</relation>
        </sentence>
</item>
```

Ví dụ: tác giả viết sách

```
<item key="">
        <sentence option="intersection">
                <domain var="1">author</domain>
                <range var="1" select="0">course</range>
                <relation>cs_author:write</relation>
        </sentence>
</item>
```

- **Trường hợp 2**: mô tả cho một bộ ba trong đó một từ vựng nhận giá trị thực

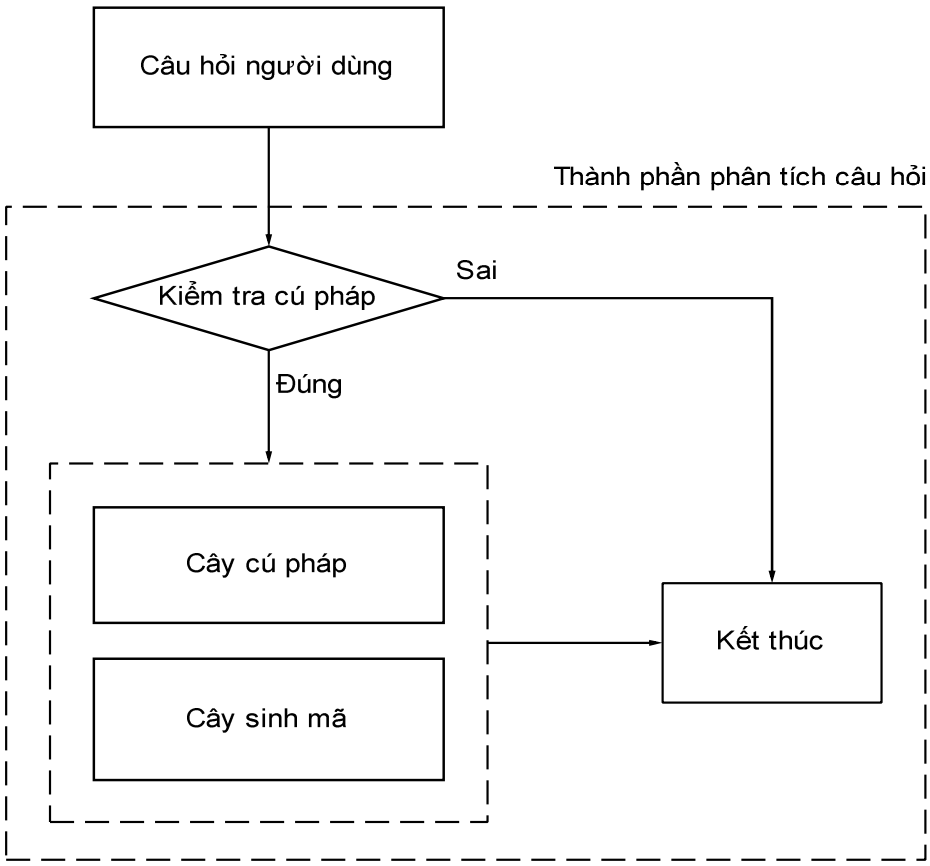
```
<item key="">
        <sentence option="intersection">
                <domain var="1">{từ vựng}</domain>
                <range var="1" select="0">{từ vựng}</range>
                <relation>{mối liên hệ (data property) hai từ vựng}</relation>
        </sentence>
        <search key="" value="dữ liệu (invidual) của từ vựng" attr="0">
                <var>{từ vựng}</var>
                <value>{dữ liệu (invidual) của từ vựng}</value>
                <neg>0</neg>
        </search>
</item>
```

Ví dụ: sách “A”

```
<item key="">
        <sentence option="intersection">
                <domain var="1">course</domain>
```

```
        <range var="1" select="0">coursename</range>
        <relation>cs_name:content</relation>
 </sentence>
 <search key="" value="A" attr="0">
        <var>coursename</var>
        <value>A</value>
        <neg>0</neg>
</search>
 </item>
```

3.2.1.3 Luồng dữ liệu



Hình 3.6 Luồng dữ liệu của thành phần phân tích câu hỏi

- Kiểm tra cú pháp: dựa vào danh sách các EBNF cho câu hỏi, dữ liệu vào được kiểm tra thuộc kiểu EBNF nào? Nếu dữ liệu vào thuộc một kiểu EBNF thì ta có cây cú pháp và cây sinh mã tương ứng.

3.2.1.4 Sơ đồ các lớp chính

A. Thành phần phân tích câu hỏi

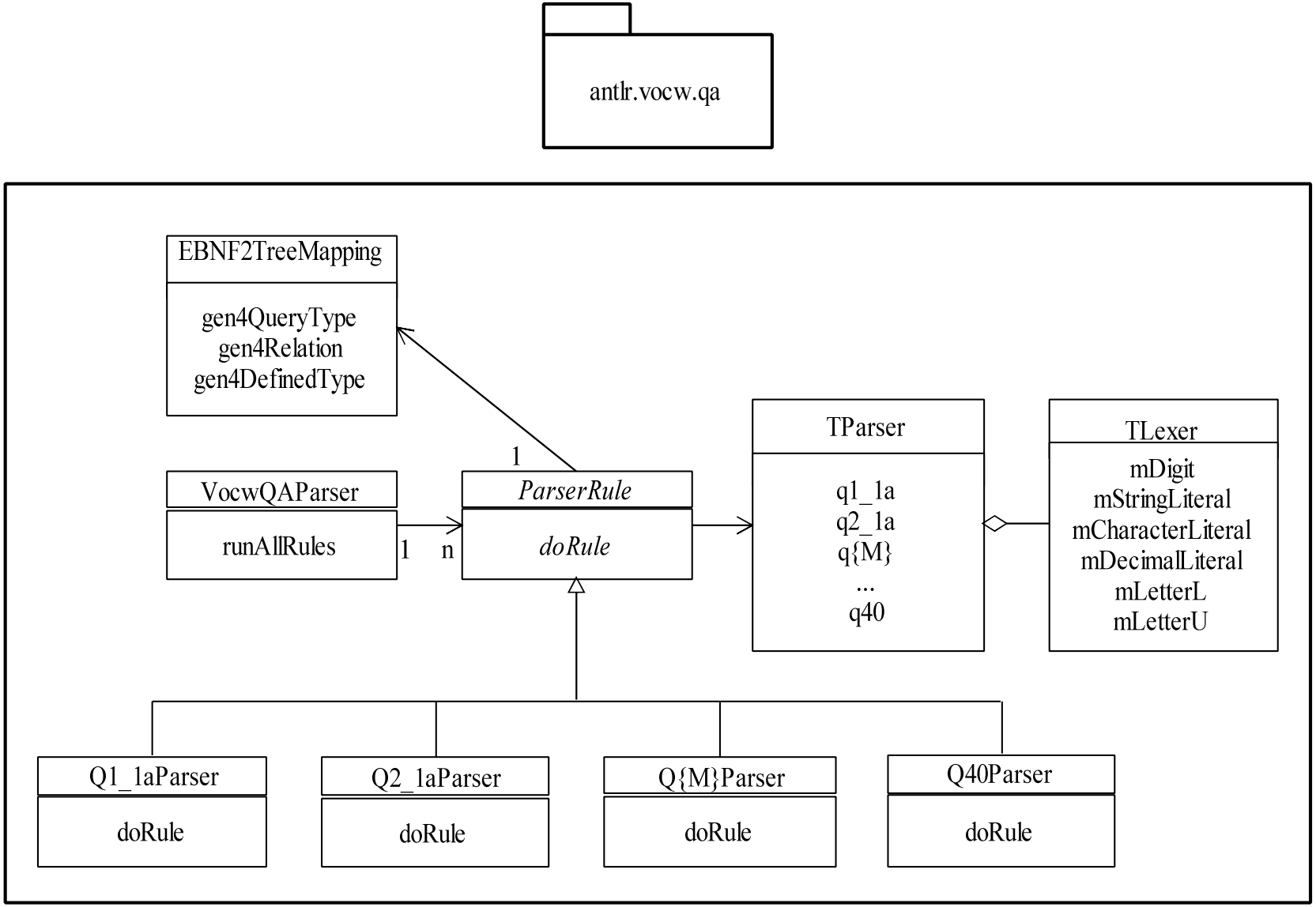
Thành phần phân tích câu hỏi: antlr.vocw.qa

Các lớp chính:

- TLexer: định nghĩa và kiểm tra các từ.
- TParser: định nghĩa và kiểm tra luật (bằng cú pháp EBNF).
- ParserRule: định nghĩa trừu tượng việc kiểm tra một luật.
- Q{*parser_number*}Parser: kiểm tra luật xác định (được đánh số *parser_number*). Ví dụ: lớp Q1_1aParser: kiểm tra luật q1_1a.
- VocwQAParser: kiểm tra cú pháp một câu hỏi đưa vào với các luật đã được định nghĩa trong Tparser.
- EBNF2TreeMapping: thực hiện ánh xạ từ cây cú pháp sang cây sinh mã. Lớp này gọi các đối tượng thuộc thành phần vocw.tree hỗ trợ thực hiện ánh xạ.

Lớp TLexer và TParser được viết dựa trên mã mở ANTLR. Việc định nghĩa luật cú pháp EBNF và ANTLR sẽ sinh lớp TLexer và TParser để kiểm tra cú pháp.

Khi thêm luật mới, chỉ cần định nghĩa thêm luật và ANTLR sẽ sinh thêm thành phần kiểm tra và thêm lớp Q{*new_parser_number*}Parser, nên quá trình này gần như không can thiệp vào mã chương trình đã có. Vì vậy giảm thiểu rủi ro của việc sửa và thêm mới.



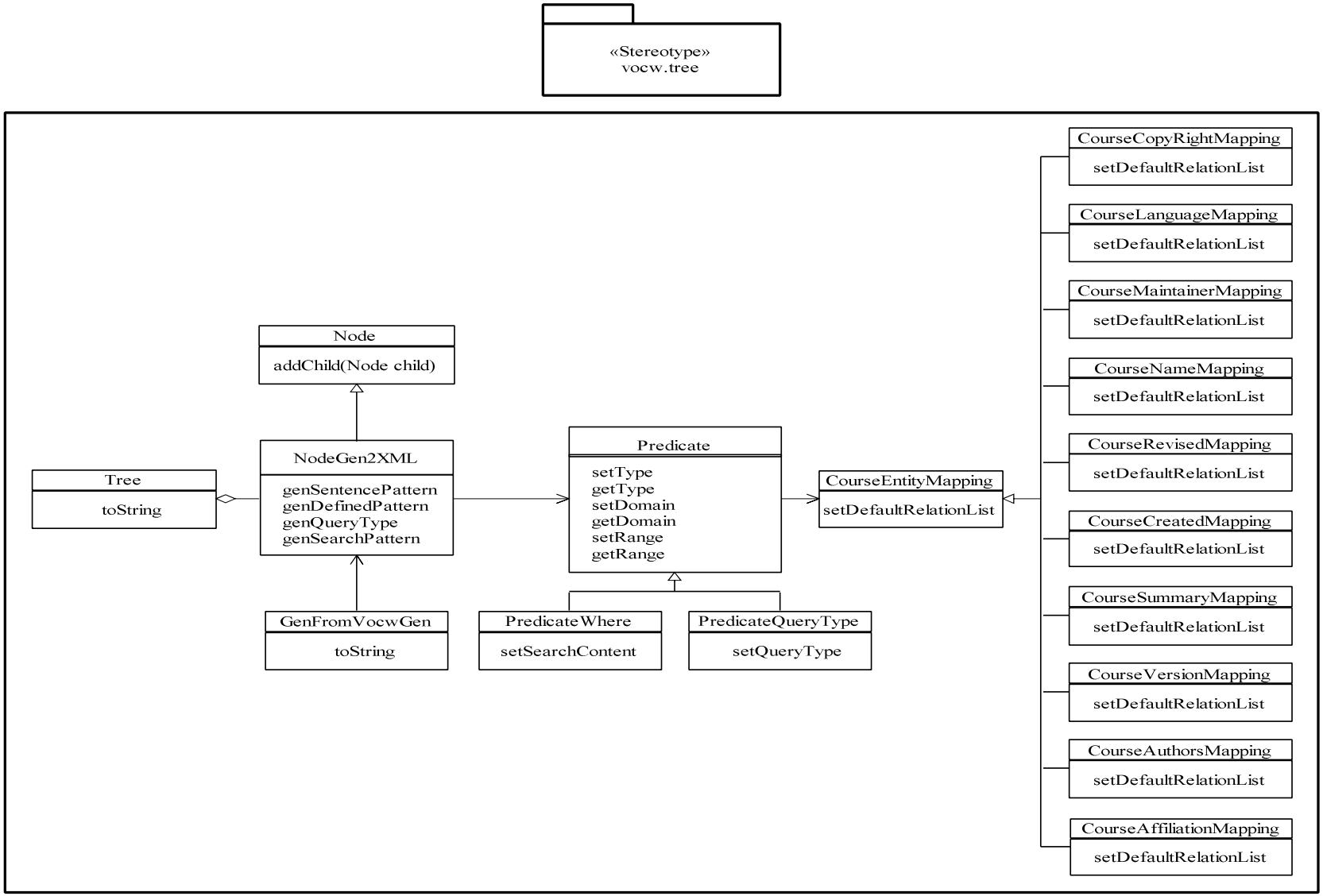
Hình 3.7 Sơ đồ thành phần phân tích câu hỏi

B. Thành phần ánh xạ ontology

Thành phần ánh xạ ontology: vocw.tree

Dựa vào mô tả ontology ở phần 3.2.4, tổ chức các lớp chính như sau:

- Course{*name*}Mapping: thực hiện ánh xạ lớp xác định trong ontology. Ví dụ: CourseAuthorMapping: ánh xạ lớp cs_author trong ontology.
- CourseEntityMapping: định nghĩa trừu tượng việc ánh xạ với một lớp trong ontology.
- PredicateWhere: ánh xạ các quan hệ trong câu hỏi với các bộ ba trong ontology.
- PredicateQueryType: lưu thông tin từ nghi vấn trong câu hỏi.
- Node: định nghĩa trừu tượng nút của cây sinh mã.
- NodeGen2XML: sinh mã xml cho đối tượng thuộc lớp Node.
- Tree: cây sinh mã (đối tượng thuộc lớp NodeGen2XML).
- GenFromVocwGen: duyệt cây sinh mã cho ra dạng XML.



Hình 3.8 Sơ đồ thành phần ánh xạ ontology

3.2.2 Thành phần sinh mã SPARQL

3.2.2.1 Dữ liệu vào và ra của thành phần

- Dữ liệu vào: cây sinh mã dạng XML
- Dữ liệu ra: câu truy vấn SPARQL

3.2.2.2 Giới hạn xem xét câu hỏi tiếng Việt

- Câu hỏi đơn.
- Câu hỏi có thể chứa một hay nhiều từ liên kết từ đơn là "và" và "hoặc".

✓ **Xét trường hợp "hoặc"**

SPARQL hỗ trợ từ khoá UNION dùng cho truy vấn trường hợp một hay nhiều khả năng.

✓ **Xét trường hợp "và"**

Ví dụ: Ai đã có viết sách "Toan" và sách "Van" trong năm 2009?
Hiểu theo cách khác là:
{Ai đã có viết sách "Toan" trong năm 2009 mà
{người này cũng viết sách "Van" trong năm 2009}
?}

Trường hợp này được xử lý là dạng SPARQL lồng nhau. Do mỗi nút của cây ánh xạ có thể là nút gốc của một cây con khác, cây sinh mã có thể biểu diễn truy vấn lồng nhau để tham gia vào quá trình sinh mã SPARQL.

Chúng ta thấy rằng trường hợp này có dạng tổng quát như sau:

- Xét các từ vựng trong 2.1.
- Đặt các từ vựng là tập hợp: A_i
- Đặt các dữ liệu cụ thể của từ vựng là phần tử. x_i

Chúng ta biểu diễn ví dụ trên như sau:

Author Name[Toan,Van] Created[2009] ?

Dạng tổng quát:

$A_1\ A_2 \ldots A_i[xi_1, xi_2, \ldots, xi_n] \ldots A_k[xk_1] \ldots A_n?$

Mỗi câu hỏi chỉ có một thành phần từ vựng được hỏi với nhiều dữ liệu cụ thể bằng từ liên kết "và" và "hoặc". Và các thành phần từ vựng còn lại có thể được hỏi tối đa một dữ liệu cụ thể.

Trở lại ví dụ:

{Ai đã có viết sách "Toan" trong năm 2009 mà (*)

{người này cũng viết sách "Van" trong năm 2009}(**)

?}

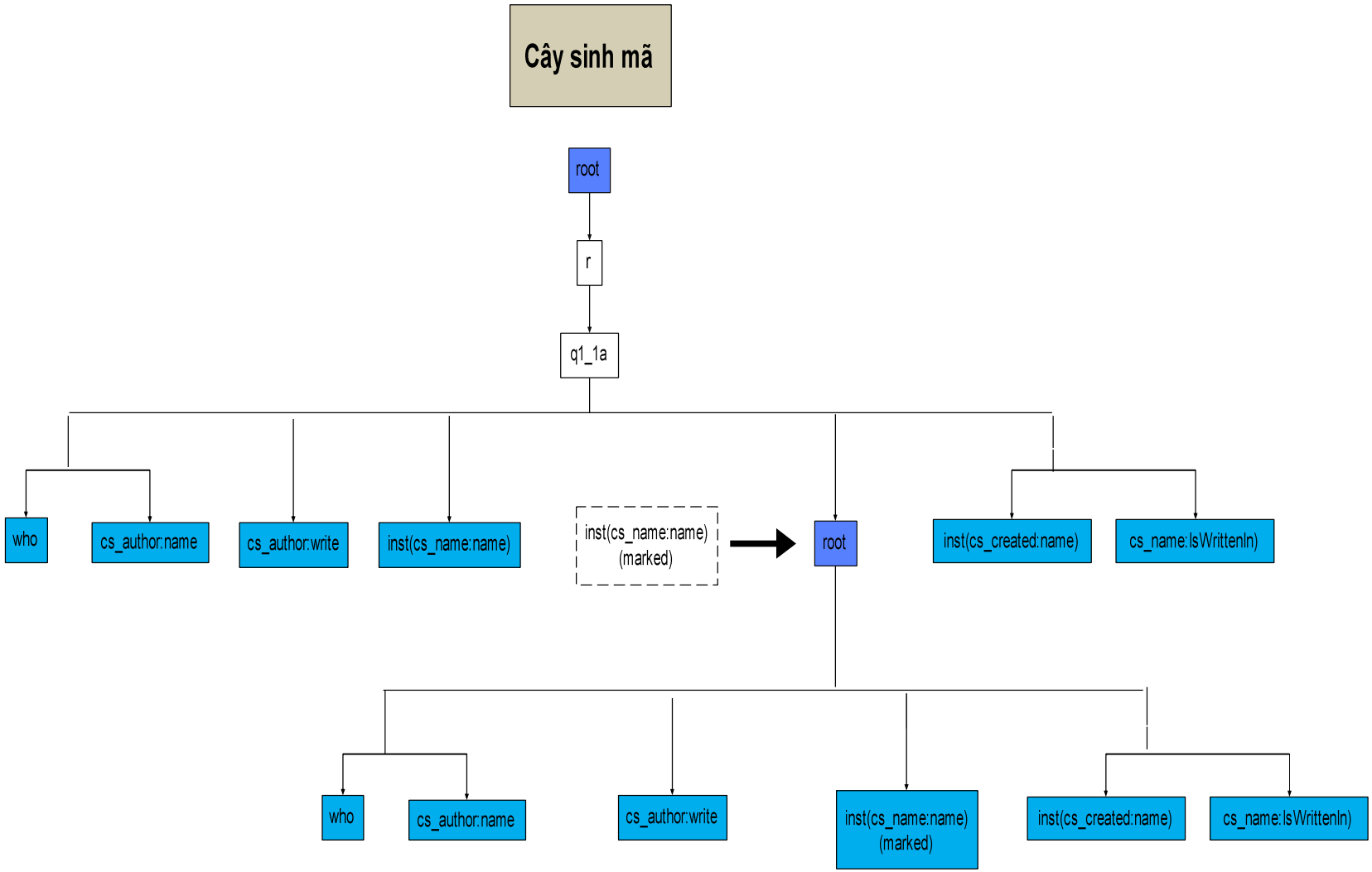
Chúng ta thấy rằng câu con (**) chỉ khác câu cha (*) ở thành phần sách "Van".

Dạng tổng quát

$$A_1\ A_2 \ldots A_i[xi_1, xi_2, \ldots, xi_n] \ldots A_k[xk_1] \ldots A_n?$$

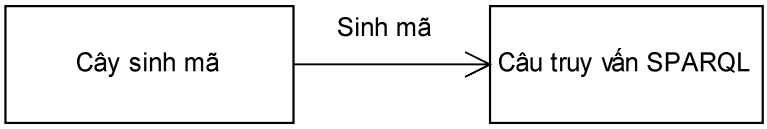
Có thể phân tích thành n câu, các câu khác nhau ở thành phần thứ i. Trên cây cú pháp, mỗi nút xi_k là nút gốc chứa cây con là tất cả các nút của cây ngoại trừ các nút xi_k.

Cây cú pháp trong trường hợp ví dụ trên:



Hình 3.9 Cây sinh mã

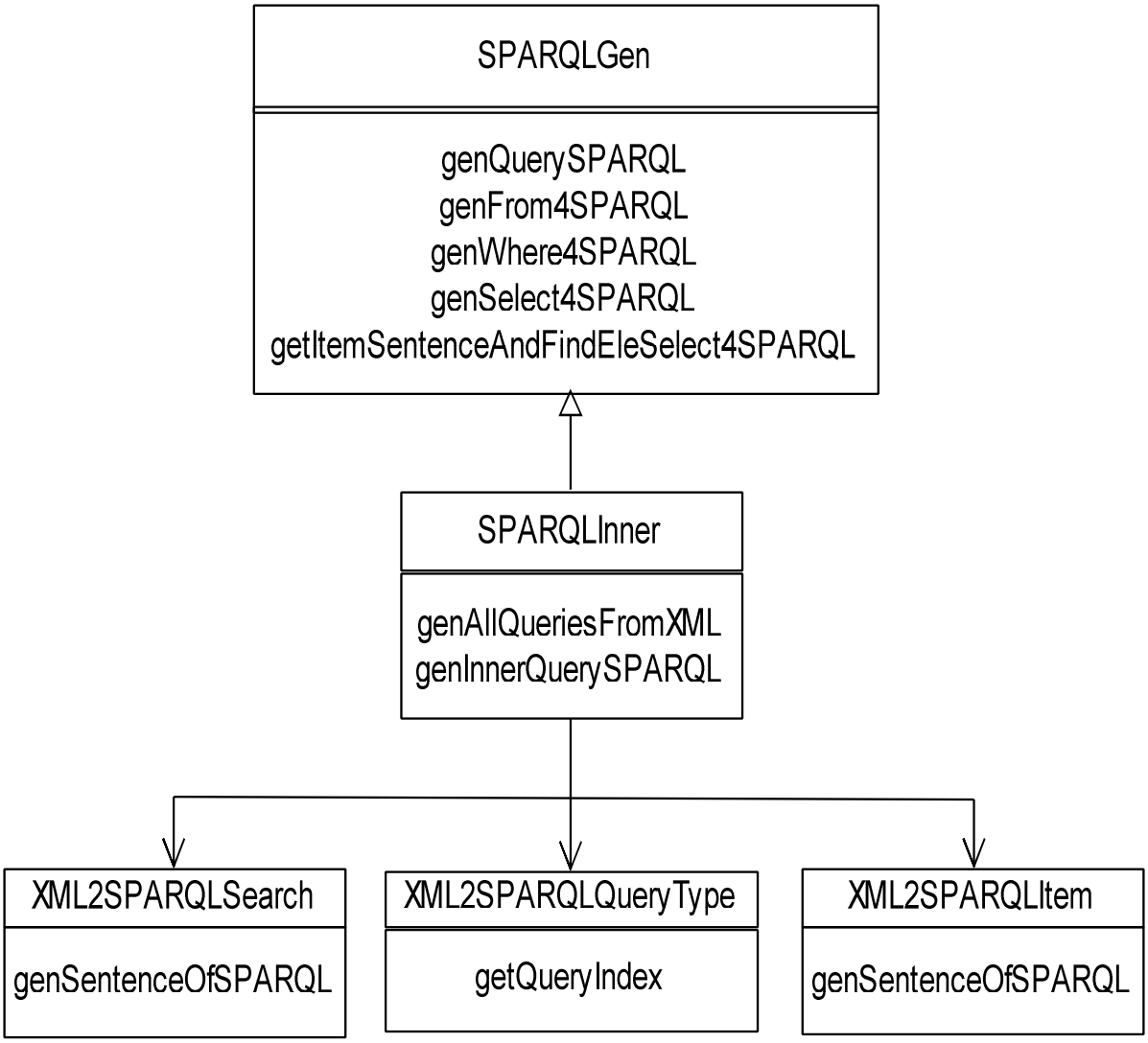
3.2.2.3 Luồng dữ liệu



Quá trình sinh truy vấn từ dạng XML của cây sinh mã là quá trình duyệt các nút của cây theo chiều sâu.

.

3.2.2.4 Sơ đồ các lớp chính mô tả



Hình 3.10 Các lớp thuộc thành phần sinh mã

3.2.3 Thành phần xây dựng ontology và truy vấn

3.2.3.1 Dữ liệu vào và ra của thành phần

- Dữ liệu vào: ontology mô tả thô
- Dữ liệu ra: ontology chứa dữ liệu metadata của trang học liệu mở Việt Nam

3.2.3.2 Cấu trúc ontology

A. Các lớp ontology

- Lớp metadata của khóa học (chứa toàn bộ thông tin khóa học)
- Lớp tên khóa học
- Lớp mã khóa học
- Lớp ngôn ngữ khóa học
- Lớp tóm tắt khóa học
- Lớp từ khóa khóa học
- Lớp loại tài liệu khóa học
- Lớp bản quyền khóa học
- Lớp tác giả khóa học
- Lớp chịu trách nhiệm xuất bản khóa học
- Lớp chỉnh lý khóa học
- Lớp phiên bản khóa học
- Lớp ngày tạo khóa học
- Lớp ngày chỉnh sửa khóa học

B. Mô tả thông tin từng lớp

- **Lớp metadata chứa các thuộc tính sau:**

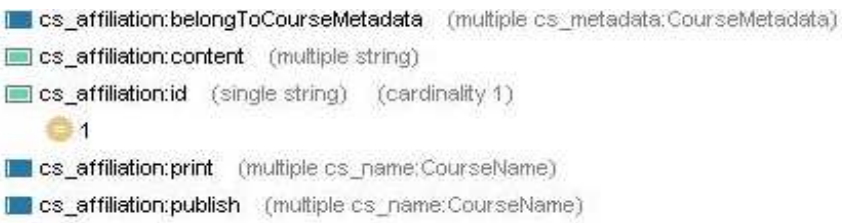
- **Các lớp còn lại đều chứa các thuộc tính chung sau**

+ cs_{field_name}:id

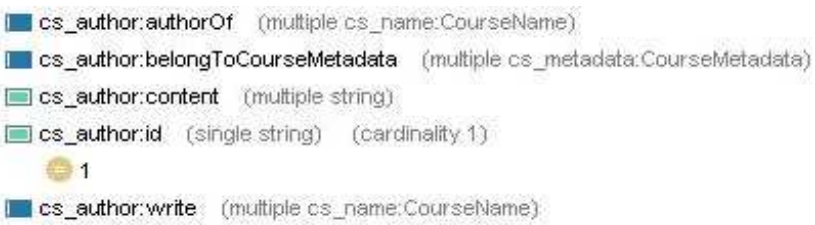
+ cs_{field_name}:content

+ cs_{field_name}:belongToCourseMetadata

- **Lớp affiliation**



- **Lớp author**



- **Lớp name (tên khoá học)**

3.2.3.3 Luồng dữ liệu

- Bước 1: Thêm dữ liệu metadata vào vocw ontology
- Bước 2: Kiểm tra tính nhất quán của vocw ontology
- Bước 3: Thực hiện lấy dữ liệu từ các suy diễn của vocw ontology
- Bước 4: Khi có truy vấn đến, hệ thống rút trích thông tin trả về.

Các bước trên được thực hiện tuần tự. Nếu một trong các bước trên sai thì kết thúc chương trình.

Thuận lợi:

- Cập nhật dữ liệu mới dễ dàng.
- Thêm suy diễn cho vocw ontology dễ dàng.

3.2.3.4 Sơ đồ các lớp chính (Hình 3.11)

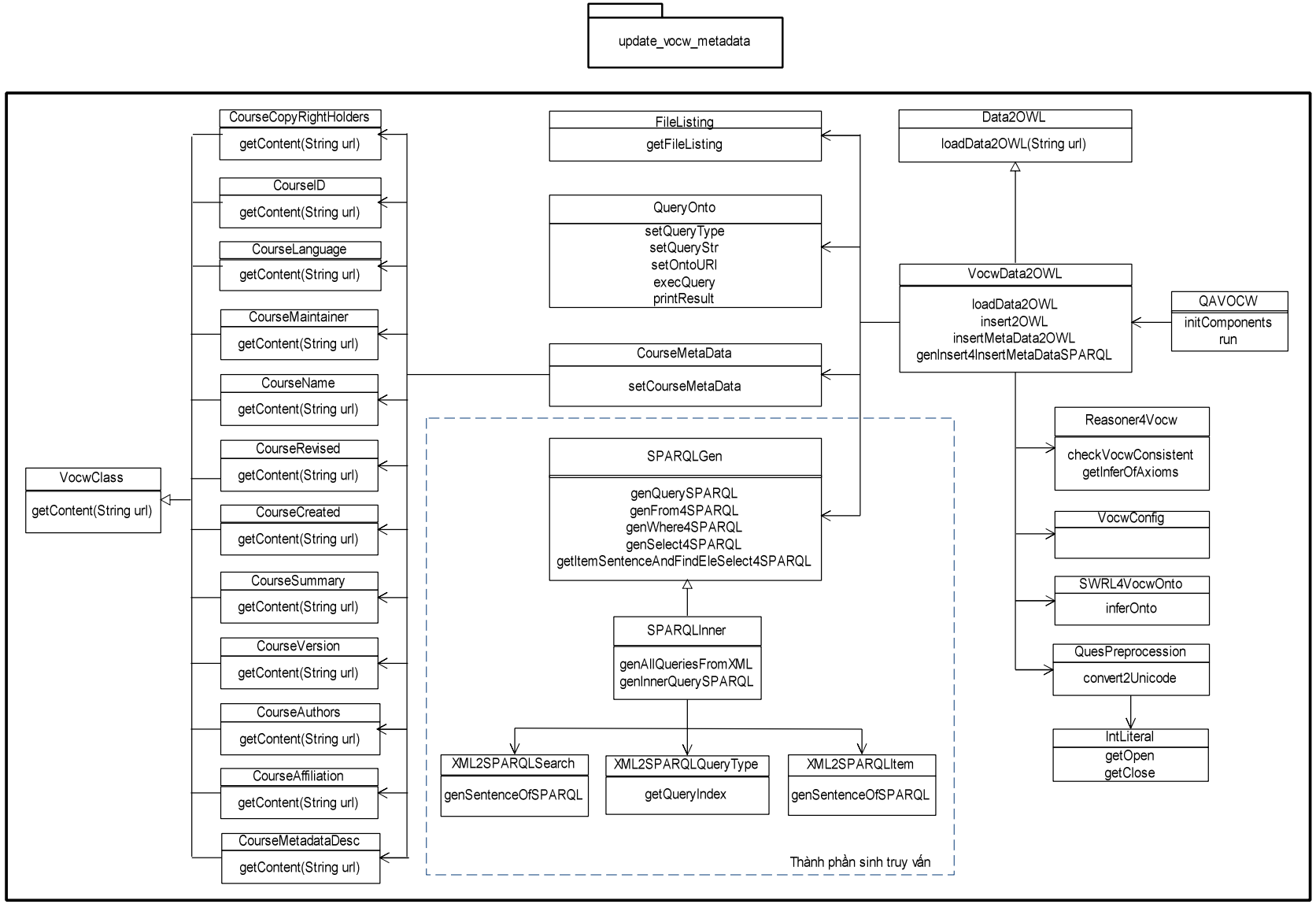
Thành phần xây dựng ontology và truy vấn: update_vocw_metadata

Các lớp chính:

- VocwClass: định nghĩa trừu tượng việc lấy nội dung của một trường thuộc dữ liệu metadata.
- Course{*name*}: lọc lại nội dung thuộc một trường xác định sau khi dữ liệu được rút trích từ trang metadata. Ví dụ: CourseAuthors: lấy nội dung thuộc trường author thuộc dữ liệu metadata.
- FileListing: lấy đường dẫn của tất cả các file metadata thuộc vocw.

- QueryOnto: thực hiện truy vấn lấy dữ liệu trên ontology.
- SPARQLGen: sinh truy vấn từ cây sinh mã .
- SPARQLInner: sinh truy vấn từ cây sinh mã trong trường hợp cây này chứa cây con.
- XML2SPARQLSearching: sinh truy vấn tương ứng với nút thuộc trường hợp 2 của mục c4 (phần 3.2.1/3.2.1.2/C/c4) của cây sinh mã.
- XML2SPARQLQueryType:sinh truy vấn tương ứng với nút gốc của cây sinh mã.
- XML2SPARQLItem: sinh truy vấn tương ứng với nút thuộc trường hợp 1 của mục c4 (phần 3.2.1/3.2.1.2/C/c4) của cây sinh mã.
- Data2OWL: định nghĩa trừu tượng việc đưa dữ liệu từ kho bất kì vào ontology.
- VocwData2OWL: đưa dữ liệu metadata từ kho vocw vào ontology.
- QAVOCW: lớp giao diện để người dùng nhập liệu và nhận kết quả trả về của hệ thống.
- Reasoner4Vocw: lớp kiểm tra tính nhất quán của ontology và lấy dữ liệu từ các suy diễn ánh xạ ngược. Lớp này sử dụng các hàm API hỗ trợ của Pellet.
- VocwConfig: chứa thông tin về đường dẫn của vocw.
- SWRL4Onto: lớp này lấy dữ liệu từ các suy diễn được định nghĩa trong ontology. Lớp này sử dụng các hàm API hỗ trợ của Jess.
- QuesPreprocess: chuyển đổi nội dung sang mã Unicode trước khi xử lý.
- IntLiteral: đánh dấu nội dung trong dấu ngoặc kép “” sang mã Unicode.

  (Nếu giá trị thực của một lớp có tên trùng với từ của luật thì giá trị thực này được đặt vào dấu ngoặc kép)



Hình 3.11 Sơ đồ thành phần xây dựng ontology và truy vấn

3.2.4 Thành phần rút trích thông tin

3.2.4.1 Dữ liệu vào và ra của thành phần

- Dữ liệu vào: danh sách các trang metadata của trang học liệu mở Việt Nam
- Dữ liệu ra: thông tin rút trích từng trường thuộc trang metadata

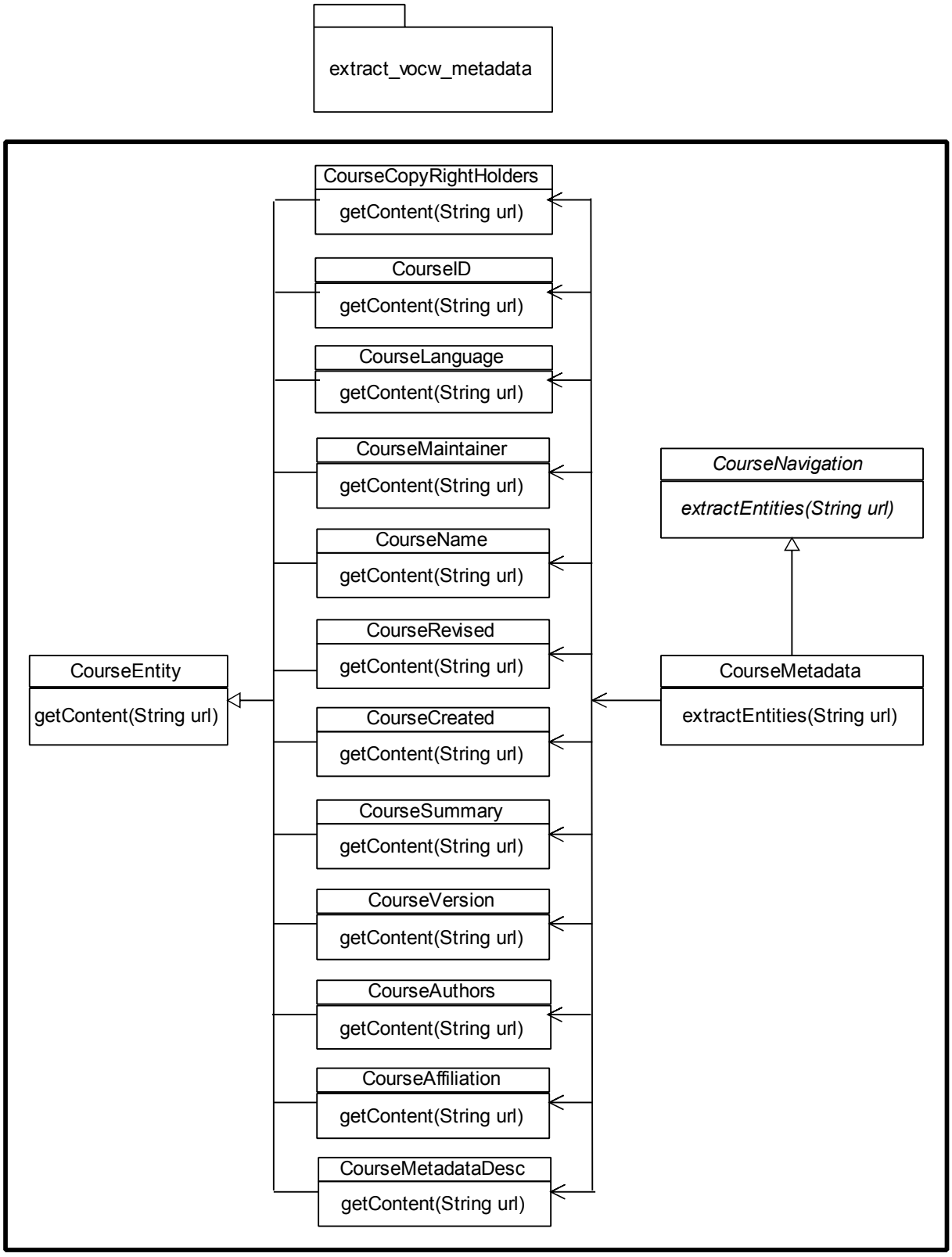
Trang metadata là một trang html. Trang html được lưu dưới dạng cây tự do. Việc lấy thông tin của trường dữ liệu là quá trình truy xuất một nút từ cây.

3.2.4.2 Sơ đồ các lớp chính

Thành phần xây dựng ontology và truy vấn: extract_vocw_metadata

Các lớp chính:

- CourseEntity: định nghĩa trừu tượng việc lấy nội dung từ trường (field) bất kì
- Course{*name*}: rút trích nội dung thuộc một trường xác định từ trang metadata.
- CourseMetadata: chứa nội dung của tất cả các trường từ trang metadata.
- CourseNavigation: định nghĩa trừu tượng việc lấy thông tin từ kho bất kì.

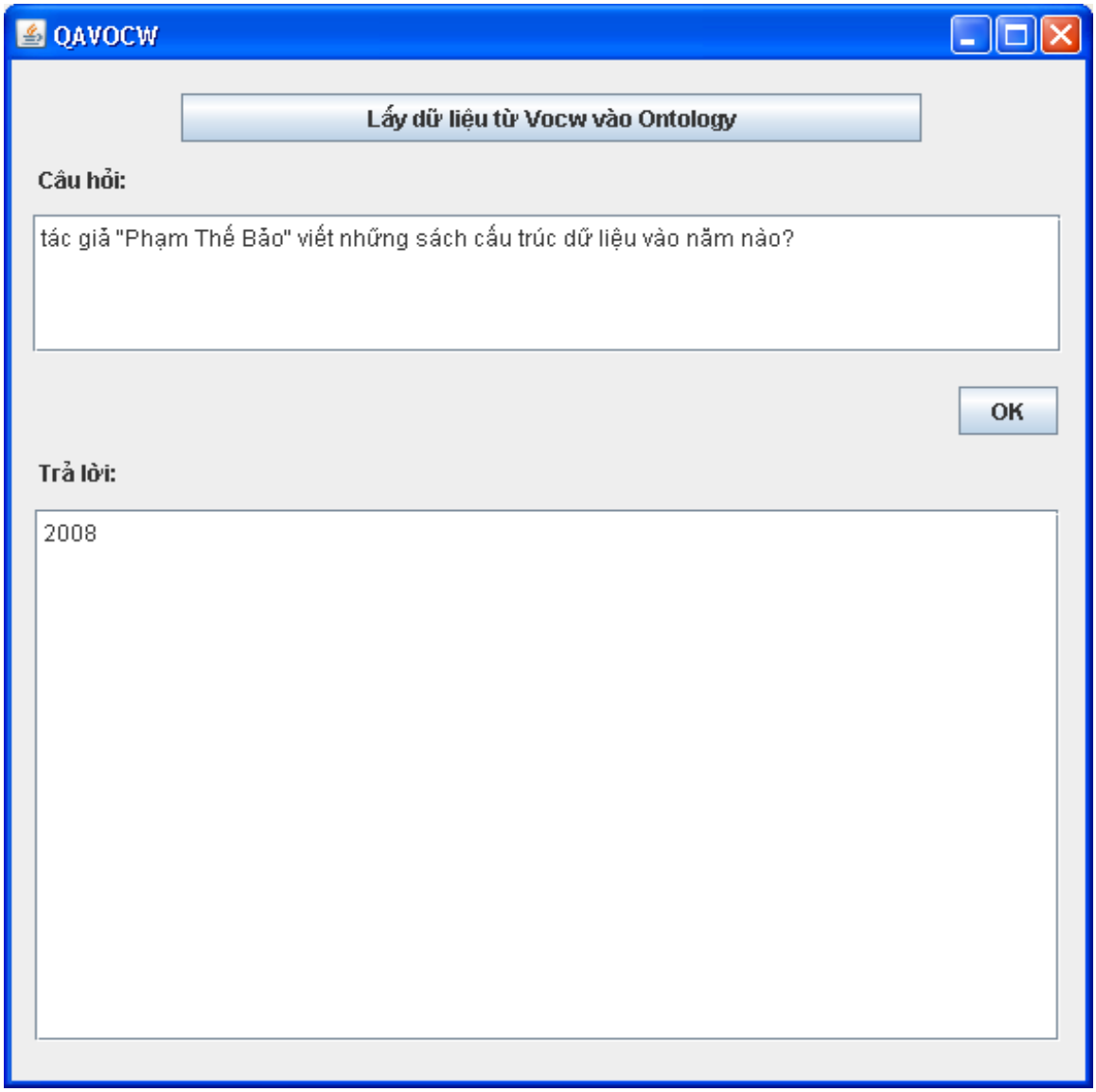


Hình 3.12 Sơ đồ các lớp chính

CHƯƠNG IV - THỬ NGHIỆM

4.1 Cài đặt

- Hệ điều hành WindowXP SP2, version 2002
- Phần mềm Teleport v1.55
- Tải toàn bộ nội dung vocw về máy bằng Teleport v1.55:
  - ✓ Mở Teleport
  - ✓ File->New Project Wizard
    - Chọn Create a browsable
    - Nhấn Next
  - ✓ Điền http://www.vocw.edu.vn/ vào hộp url và nhấn Next
  - ✓ Nhấn Next
  - ✓ Chọn nơi lưu nội dung ở máy
  - ✓ Nhấn Finish
- Tạo địa chỉ url cho nội dung vocw ở máy
  - ✓ Cài đặt Apache
  - ✓ Đặt alias directory nơi lưu vocw ở máy với url: http://localhost/vocw2/
- Cài đặt jdk1.6, jre6
- Chép file vocwqa.rar
- Giao diện chương trình và các bước sử dụng chương trình (xem hình dưới):
  - ✓ Bước 1: Nhấn nút "Lấy dữ liệu từ Vocw vào ontology" để lấy dữ liệu metadata từ kho học liệu mở.
  - ✓ Bước 2: Nhập câu hỏi tiếng Việt vào ô "Câu hỏi". Sau đó nhấn nút "OK".
  - ✓ Bước 3: Kết quả trả về từ hệ thống sẽ hiện ra trong ô "Trả lời".



Hình 4.1 Giao diện chương trình

4.2 Thử nghiệm dữ liệu

- Cấu hình máy chạy kiểm tra dữ liệu (laptop Lenovo): Intel(R) Core(TM)2 Duo CPU, T5750 @ 2.00GHz, 997MHz, 0.99GB of RAM
- Tập câu hỏi kiểm tra
  - ✓ Tổng số luật cú pháp đã được xây dựng (bằng EBNF): 40
  - ✓ Tổng số câu hỏi kiểm tra chuẩn chắc chắn thuộc 40 luật cú pháp (chủ quan): 40

- ✓ Tổng số câu hỏi kiểm tra ngẫu nhiên: 91

➢ Kết quả kiểm tra

- ✓ Tập 40 câu hỏi kiểm tra chuẩn (chủ quan):

  Tỷ lệ đúng: 40/40 (100%)

  Thời gian chạy: 49.13 giây / 40 câu

- ✓ Tập 91 câu hỏi kiểm tra ngẫu nhiên:

  Tỷ lệ câu đúng: 77/91 (84,62%)

  Thời gian chạy: 2.535 phút / 91 câu

4.3 Đánh giá hệ thống

➢ **Tính ổn định và tính chính xác**

Với 40 câu hỏi chuẩn, hệ thống kiểm tra cú pháp đúng hoàn toàn (100%), không phát sinh bất kỳ lỗi nào nên hệ thống chạy ổn định và cho kết quả chính xác.

➢ **Thời gian xử lý câu hỏi**

- ✓ Thời gian chạy thử nghiệm 40 câu hỏi chuẩn: 49.13 giây/40 câu
- ✓ Thời gian chạy thử nghiệm 91 câu hỏi ngẫu nhiên: 2.535 phút/91câu
- ✓ Thời gian trung bình chạy thử nghiệm 1 câu: 1.6 giây/câu

Như vậy với thời gian này (1.6 giây/câu) là chấp nhận được.

➢ **Độ bao phủ**

Với 91 câu hỏi ngẫu nhiên, hệ thống kiểm tra cú pháp đúng 84,62% (<100%). Điều này chứng tỏ luật (câu hỏi) mà người dùng nhập vào nằm ngoài phạm vi 40 luật cú pháp mà chúng tôi đã xây dựng. Nói cách khác, tập 40 luật đó chưa đủ bao quát toàn bộ các câu hỏi (luật) mà người dùng nhập vào. Tuy nhiên, việc cập nhập các luật (câu hỏi) mới này rất dễ dàng, không can thiệp vào mã chương trình đã có, nên giảm thiểu rủi ro của việc chỉnh sửa và thêm mới.

CHƯƠNG V - KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết quả đạt được và những đóng góp chính trong luận văn

Với kết quả chạy thử nghiệm ở trên, chứng tỏ hệ thống tìm kiếm mà chúng tôi đã xây dựng chạy ổn định và cho kết quả chính xác. Vì vậy, hoàn toàn có thể sử dụng hệ thống tìm kiếm này để áp dụng vào việc tìm kiếm những vấn đề liên quan đến sách.

Tuy nhiên, tập luật cú pháp mà chúng tôi xây dựng chưa đủ lớn và cần phải được tiếp tục bổ sung thêm. Việc bổ sung thêm các luật cú pháp mới này hoàn toàn dễ dàng, không can thiệp vào mã của chương trình nên giảm thiểu rủi ro lỗi xảy ra.

5.2 Những khó khăn và hạn chế

Hệ thống đã được thử nghiệm và đánh giá. Các kết quả đạt được là khá tốt với các dữ liệu và phương pháp được mô tả ở trên. Tuy nhiên, đây chỉ mới là một kết quả đầu tiên. Việc xử lý các câu truy vấn tiếng Việt phức tạp vẫn còn là một thách thức rất lớn.

5.3 Các hướng phát triển

Chúng tôi đặt ra vấn đề cần có bộ phân tích cú pháp mạnh hơn cho tiếng Việt cung cấp khả năng nhận biết ngữ nghĩa giúp việc ánh xạ với mô hình ontology tốt hơn. Phân loại câu tiếng Việt giúp tăng tốc quá trình phân tích cú pháp.

Vấn đề phát triển mô hình biểu diễn nghĩa cho nhiều loại đối tượng liên quan đến tìm kiếm tài liệu học tập như tìm trên mục lục, những mô tả tóm tắt nội dung sách … giúp mở rộng không gian tìm kiếm.

TÀI LIỆU THAM KHẢO

Tài liệu nước ngoài:

[1] Dragan Gasˇevic´, Dragan Djuric´, Vladan Devedzˇic' (2006), Model Driven Architecture and Ontology Development, page 58-68, 91-108, p.107-108

[2] Grigoris Antoniou và Frank van Harmelen (2004), A Semantic web Primer, The MIT Press Cambridge, Massachusett, London, England, p.31-33

[3] Natalya F. Noy và Deborah L. McGuinness (2001), Ontology Development 101: A Guide to Creating Your First Ontology

[4] Terence Parr, The Pragmatic Bookshelf, Raleigh, North Carolina Dallas, Texas (2007), The Definitive ANTLR Reference, Building Domain-Specific Languages, page 98-99

[5] Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider, THE DESCRIPTION LOGIC HANDBOOK: Theory, implementation, and applications, page 47-100, page 495-505

[6] Dang Tuan Nguyen, Tuyen Thi-Thanh Do, "E-Library Searching by Natural Language Question-Answering System", Proceedings of the Fifth International Conference on Information Technology in Education and Training (IT@EDU2008), pages: 71-76, Ho Chi Minh and Vung Tau, Vietnam, December 15-16, 2008.

[7] Dang Tuan Nguyen, Tuyen Thi-Thanh Do, "e-Document Retrieval by Question Answering System", International Conference on Communication Technology, Penang, Malaysia, February 25-27, 2009. Proceedings of World Academy of Science, Engineering and Technology, Volume 38, 2009, pages: 395-398, ISSN: 2070-3740.

[8] Dang Tuan Nguyen, Tuyen Thi-Thanh Do, "Natural Language Question Answering Model Applied To Document Retrieval System", International Conference on Computer Science and Technology, Hongkong, China, March 23-25, 2009. Proceedings of World

Academy of Science, Engineering and Technology, Volume 39, 2009, pages: 36-39, ISBN: 2070-3740.

[9] Dang Tuan Nguyen, Tuyen Thi-Thanh Do, "Document Retrieval Based on Question Answering System", Proceedings of the Second International Conference on Information and Computing Science, pages: 183-186, Manchester, UK, May 21-22, 2009. ISBN: 978-0-7695-3634-7. Editions IEEE.

Tài liệu tiếng Việt:

[10] TS. Đỗ Phúc và cộng sự (2004), Phát triển một hệ thống S.E hỗ trợ tìm kiếm thông tin, thuộc lãnh vực CNTT trên Internet qua từ khóa bằng tiếng Việt.

Tài liệu trên Internet:

[11] http://otal.umd.edu/drweb/c++tutorial/lessons/BNF.HTM

[12] Protégé (http://protege.stanford.edu/)

[13] Jess (http://www.jessrules.com/)

[14] Pellet (http://clarkparsia.com/pellet/)

[15] Jena (http://jena.sourceforge.net/)