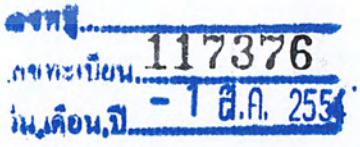
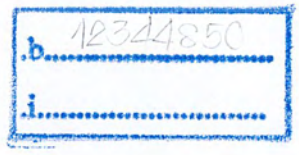
ฐานข้อมูลอาร์ดีเอฟ

RDF DATABASE

จิรวัฒน์ ภูวคีรีวิวัฒน์
ณัฐภัทร โสตถิโสภา
ธัญธิตา แก้วฝั้น





ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2553

ปริญญานิพนธ์ปีการศึกษา 2553

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ฐานข้อมูลอาร์ดีเอฟ

RDF DATABASE

ผู้จัดทำ

1. นายจิรวัฒน์ ภูวคีรีวิวัฒน์ รหัสนักศึกษา 50010235
2. นายณัฐภัทร โสตถิโสภา รหัสนักศึกษา 50010209
3. นางสาวธัญธิตา แก้วฝั้น รหัสนักศึกษา 50011825

อาจารย์ที่ปรึกษา

(รศ.ดร.ศุภมิตร จิตตะยโศธร)

ฐานข้อมูลอาร์ดีเอฟ

นายจิรวัฒน์ ภูวคีรีวิวัฒน์ 50010235
นายณัฐภัทร โสตถิโสภา 50010500
นางสาวธัญธิตา แก้วฝั้น 50010679
รศ.ดร. ศุภมิตร จิตตะยโศธร อาจารย์ที่ปรึกษา
ปีการศึกษา 2553

บทคัดย่อ

เทคโนโลยี XML เป็นเทคโนโลยีที่กำลังได้รับความสนใจในองค์กรสารสนเทศเป็นอย่างมาก เพราะสามารถนำมาใช้เก็บข้อมูลที่สนับสนุนการทำงานแลกเปลี่ยนกันระหว่างองค์กรได้ โดยจะอธิบายข้อมูลในรูปแบบต้นไม้ลำดับชั้น นอกจากนี้ยังได้มีการพัฒนาเทคโนโลยีต่าง ๆ เพื่อเพิ่มขีดความสามารถให้แก่เทคโนโลยี XML ได้ อย่างเช่น DTD, XML Schema, XSL, XSLT, XPath, XLink, XQuery เป็นต้น ซึ่งในหัวข้อโครงงานวิจัยชิ้นนี้ในอันดับแรกชี้ให้เห็นถึงการมอง XML เป็นฐานข้อมูล แล้วจึงศึกษาเกี่ยวกับเทคโนโลยีการเก็บข้อมูลแบบสื่อความหมายอันเป็นพื้นฐานของการพัฒนาเว็บสื่อความหมายที่เรียกว่า RDF ซึ่งเป็นภาษาที่กำหนดให้เก็บข้อมูลในรูปแบบกราฟโดยใช้ไวยากรณ์ภาษา XML ในการบันทึกลงสื่อข้อมูล นอกจากนี้ยังมีรูปแบบอื่น ๆ เช่น N3, N-Triple, Turtle เป็นต้น โดยแต่ละหน่วยข้อมูลจะประกอบขึ้นมาจาก 3 ส่วน ได้แก่ ประธาน ภาคแสดง กรรม และมีการกำหนดรูปแบบในการอธิบายเนื้อหาข้อมูลให้เป็นมาตรฐานสำหรับใช้ในวงกว้างด้วยภาษา RDFS และ OWL ซึ่งภาษาสำหรับสืบค้นข้อมูลเอกสาร RDF นั้นเรียกว่าภาษา SPARQL ที่มีไวยากรณ์ไม่ซับซ้อน งานวิจัยชิ้นนี้มุ่งเน้นการนำเสนอการมองเทคโนโลยี RDF เป็นฐานข้อมูล และการนำฐานข้อมูล RDF ที่มีอยู่บนอินเตอร์เน็ตมาใช้งานผ่านด้วยการสืบค้นเข้ามาในโปรแกรมประยุกต์ แล้วประมวลข้อมูลที่ได้มาจนได้ข้อสรุปใหม่ ซึ่งเป็นการใช้ประโยชน์จากแนวคิดหลักของเว็บสื่อความหมาย ที่มองอินเตอร์เน็ตเป็นฐานข้อมูลขนาดใหญ่ที่รวมกันขึ้นมาจากเอกสาร RDF จากแหล่งข้อมูลต่าง ๆ ที่เปิดให้บุคคลภายนอกเข้ามาสืบค้นข้อมูลนำไปใช้ประโยชน์

RDF DATABASE

Mr. Jirawat	Phoowakheereewiwat	50010235
Mr. Natthapat	Sotthisopha	50010500
Ms. Thunthita	Kaewfun	50010679
Assoc.Prof.Dr.Suphamit	Chittayasothorn	Advisor

Academic Year 2010

ABSTRACT

XML technology has gained considerable attention from many IT organizations by virtue of its support for interoperability amongst organizations. XML describes data in hierarchical tree model. Moreover, there have been many technologies developed to exploit the XML tree model such as DTD, XML Schema, XSL, XSLT, XPath, XLink, XQuery etc. This project aimed to propose, firstly the new perspective of harnessing XML as a database, and secondly study the technology involving storing data as in semantic data, namely RDF which describes data in labeled directed graph model and serialized using XML. Generally, each node is connected to the other node through predicate, where the origin node acts as a subject, and the other acts as an object. Apart from serializing semantic data using XML, there are several other methods available for serialization, to name a few: N3, N-Triple, and Turtle. RDF has some other languages defined for defining standardized form for storing data, namely RDFS and OWL, and the facility for querying data kept in RDF documents is SPARQL which is a simple query language that shares similar structure to SQL with less capabilites. This project also demonstrates how to perform data integration on data scattered on the Internet which follows the main goal of semantic web as seeing the Internet as a gigantic database made up of RDF documents across the Internet which publicly allow outsiders to utilize.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ดีด้วยคำแนะนำ คำปรึกษา ของอาจารย์ที่ปรึกษาที่คอยเอาใจใส่ ให้คำแนะนำ และคอยให้ความช่วยเหลือเสมอมา คือ รศ.ดร.ศุภมิตร จิตตะยโศธร ขอกราบขอบพระคุณเป็นอย่างสูง

ท้ายที่สุดนี้ขอกราบขอบพระคุณขอบคุณ บิดามารดาและครอบครัวอันเป็นที่เคารพรัก ซึ่งได้ให้การอบรมเลี้ยงดูเป็นอย่างดี พร้อมทั้งให้โอกาสทางการศึกษา ให้ความช่วยเหลือและกำลังใจที่ดีเสมอมา

คุณประโยชน์ที่ได้จากปริญญานิพนธ์ฉบับนี้ขอมอบเป็นเครื่องบูชาพระคุณบิดา มารดา ครูอาจารย์ทุกท่านที่ให้การอบรมสั่งสอนสร้างความรู้มา

จิรวัฒน์ ภูวคีรีวิวัฒน์
ณัฐภัทร โสตถิโสภา
ธัญธิตา แก้วฝั้น

สารบัญ

หน้า

บทคัดย่อภาษาไทย I

บทคัดย่อภาษาอังกฤษ II

กิตติกรรมประกาศ III

สารบัญ IV

สารบัญตาราง VI

สารบัญรูป VII

บทที่ 1 บทนำ 1

1.1 ความเป็นมาของปัญหา 1

1.2 วัตถุประสงค์โครงงาน 1

1.3 ประโยชน์ที่คาดว่าจะได้รับ 1

1.4 ขอบเขตของโครงงาน 2

1.5 วิธีการดำเนินงาน 2

บทที่ 2 ทฤษฎีที่เกี่ยวข้องกับเอ็กซ์เอ็มแอล 3

2.1 ทฤษฎีพื้นฐานของภาษา XML 3

2.2 ความถูกต้องของเอกสาร XML 16

2.3 การกำหนดโครสร้างของเอกสาร XML แบบ DTD และแบบ XML Schema 16

2.4 เอ็กซ์พาธ (XPath) 28

2.5 XQuery 36

2.6 แบบจำลองกระบวนการทำงานของ XQuery (XQuery Processing Model) 46

2.7 ประโยชน์ และการประยุกต์ใช้งานของภาษา XML 47

บทที่ 3 ทฤษฎีเกี่ยวข้องกับ RDF 50

3.1 Resource Description Framework (RDF) 50

3.2 RDF Schema (RDFS) 65

3.3 Web Ontology Language (OWL) 71

3.4 RDFlib 107

3.5 SPARQL 113

สารบัญ (ต่อ)

หน้า

3.6 ตัวอย่างการนำ RDF ไปประยุกต์ใช้งาน 134
3.7 การเปลี่ยนรูปแบบจำลองเชิงสัมพันธ์ให้เป็นแบบจำลอง RDF 136
3.8 Open Sesame.....143
3.9 D2R Server.....153
3.10 การแปลงข้อมูลกราฟ RDF จาก Schema.....160

บทที่ 4 การทดลอง 163
4.1 การทดลองที่ 1 สร้างโปรแกรมประยุกต์ที่ใช้งานข้อมูล RDF จากแหล่งข้อมูลต่าง ๆ 163
4.2 การทดลองที่ 2 ทดสอบความสามารถในการเข้าใจออนโทโลยีของภาษาสืบค้น SPARQL.....170
4.3 การทดลองที่ 3 การทดลอง OWL.....173

บทที่ 5 พัฒนาแอปพลิเคชัน RDF และRDFS 178
5.1 การทดลองพัฒนาแอพพลิเคชัน.....178
5.2 จุดประสงค์การทดลอง 180
5.3 ขั้นตอนการทดลอง 180

บทที่ 6 บทวิจารณ์และสรุป 194
6.1 บทสรุป 194
6.2 วิจารณ์สิ่งที่ได้จากโครงงาน.....198
6.3 ปัญหาอุปสรรค และแนวทางการแก้ไข 199
6.4 แนวทางการพัฒนาต่อ 199

บรรณานุกรม 200
ภาคผนวก.....201

สารบัญตาราง

ตาราง		หน้า
2.1	Entity Reference	11
2.2	เครื่องหมายที่ช่วยในการปรับปรุงโครงสร้างของเนื้อหา	22
2.3	รูปแบบการเข้าถึงข้อมูลที่สำคัญ	31
2.4	ตัวอย่างของการเลือกโหนด	31
2.5	ตัวอย่างรูปแบบการเข้าถึงข้อมูลที่มีภาคแสดงเป็นส่วนประกอบ	32
2.6	การเลือกโหนดที่ไม่รู้จักโดยใช้สัญลักษณ์แทนความหมายเอ็กซ์พาธ (XPath wildcard)	32
2.7	ตัวอย่างการเลือกโหนดที่ไม่รู้จักโดยใช้สัญลักษณ์แทนความหมายเอ็กซ์พาธ	33
2.8	ตัวอย่างการเลือกโหนดที่ไม่รู้จักโดยใช้สัญลักษณ์แทนความหมายเอ็กซ์พาธ	33
2.9	แกนของเอ็กซ์พาธ	33
2.10	ตัวอย่างที่อยู่ทางเดิน	34
3.1	SPARQL Unary Operators	119
3.2	SPARQL Binary Operators	120
3.3	SPARQL Trinary Operator	122
3.4	ตัวอย่างเค้าร่างเชิงสัมพันธ์	139
3.5	ข้อมูลตัวอย่างของตาราง Saving Account	139
3.6	ข้อมูลตัวอย่างของตาราง Account	139
3.7	ข้อมูลตัวอย่างของตาราง Customer	139
3.8	เปรียบเทียบประสิทธิภาพของ D2RQ Engine	155

สารบัญรูป

รูป		หน้า
2.1	ตัวอย่างเอกสารที่นิยามด้วยแท็กของภาษา GML	3
2.2	DTDเปรียบเสมือนแม่แบบสำหรับเอกสาร SGML ที่ใช้งาน	4
2.3	ตัวอย่างเอกสาร SGML เก็บข้อมูลติดต่อบุคคล 2 รายการ ใน 1 เอกสาร	4
2.4	ตัวอย่างเอกสาร HTML	5
2.5	โครงสร้างเอกสาร XML	6
2.6	รูปแบบที่สมบูรณ์ของการประกาศเอกสาร XML	7
2.7	ตัวอย่างการบอกประเภทของเอกสาร XML	8
2.8	ตัวอย่างการแทรกข้อความอธิบายอยู่ในแท็ก	8
2.9	ตัวอย่างข้อความส่วนประมวลผล	9
2.10	ตัวอย่างเอกสาร XML ที่มี <PRESIDENTIAL_DATABASE> เป็นรูตอิลิเมนต์	9
2.11	ตัวอย่างแท็กที่เหลื่อมซ้อนกัน	10
2.12	ตัวอย่างการใช้ อัญประกาศและ บุพสัญญา ร่วมกัน	10
2.13	ความสัมพันธ์ระหว่าง 7 มาตรฐานของ XML	12
2.14	การทำงานของ DOM	13
2.15	การทำงานของ SAX	14
2.16	โครงสร้างต้นไม้ของ DTD ของPRESIDENT.xml	14
2.17	ตัวอย่างเอกสาร PRESIDENT.xml	15
2.18	กระบวนการตรวจสอบความถูกต้องของเอกสาร	16
2.19	การประกาศ DTD แบบภายใน (Internal DTD)	18
2.20	การประกาศ DTD แบบภายนอก	19
2.21	รูปแบบการประกาศอิลิเมนต์	19
2.22	รูปแบบการประกาศอิลิเมนต์ที่ภายในประกอบด้วยอิลิเมนต์อื่นๆ	20
2.23	ตัวอย่างอิลิเมนต์ที่ภายในประกอบด้วยอิลิเมนต์อื่นๆ	20
2.24	รูปแบบการประกาศอิลิเมนต์ที่ภายในประกอบด้วยข้อความ	20
2.25	ตัวอย่างการประกาศอิลิเมนต์ที่ภายประกอบด้วยข้อความ	20
2.26	รูปแบบการประกาศอิลิเมนต์ว่าง	21
2.27	ตัวอย่างอิลิเมนต์ว่าง	21
2.28	รูปแบบการประกาศอิลิเมนต์ที่รองรับเนื้อหาทุกรูปแบบ	21

สารบัญรูป (ต่อ)

รูป		หน้า
2.29	ตัวอย่างอิลิเมนต์ที่รองรับเนื้อหาทุกรูปแบบ	21
2.30	ตัวอย่างอิลิเมนต์ที่ภายในอิลิเมนต์จะประกอบด้วยอิลิเมนต์ปะปนอยู่กับข้อความ	21
2.31	ตัวอย่างการประกาศอิลิเมนต์ลูกแบบมีลำดับ	22
2.32	ตัวอย่างการประกาศอิลิเมนต์ลูกแบบตัวเลือก	22
2.33	ตัวอย่างอิลิเมนต์ที่ใช้เครื่องหมายพิเศษในการปรังปรุงโครงสร้างเนื้อหา	23
2.34	รูปแบบการประกาศแอตทริบิวต์	23
2.35	ตัวอย่างการประกาศแอตทริบิวต์	24
2.36	ตัวอย่างการกำหนดโครงสร้างแบบ DTD	24
2.37	ตัวอย่างการกำหนดโครงสร้างแบบ XML Schema	25
2.38	รูปแบบการประกาศอิลิเมนต์แบบธรรมดา	26
2.39	ตัวอย่างการประกาศอิลิเมนต์แบบธรรมดา	26
2.40	รูปแบบการประกาศอิลิเมนต์แบบซับซ้อน	26
2.41	ตัวอย่างการประกาศอิลิเมนต์แบบซับซ้อน	26
2.42	รูปแบบการประกาศแอตทริบิวต์	27
2.43	ตัวอย่างการประกาศแอตทริบิวต์	27
2.44	เอกสาร XML ที่ใช้ประกอบการอธิบายเรื่อง XPath	29
2.45	ตัวอย่างชนิดของโหนด	29
2.46	ตัวอย่างของค่าอะตอมมิก	30
2.47	ตัวอย่างความสัมพันธ์แบบพ่อแม่และลูก	30
2.48	รูปแบบของที่อยู่ทางเดิน	34
2.49	ตัวอย่างการเขียนนิพจน์สำหรับการเข้าถึงข้อมูล Author	36
2.50	ตัวอย่างการใช้ for และการคืนค่าผลลัพธ์	37
2.51	ตัวอย่างการใช้ let และการคืนค่าผลลัพธ์	38
2.52	ตัวอย่างที่ค้นหาชื่อหนังสือจากเอกสาร books.xml	38
2.53	ตัวอย่างใช้ order by	39
2.54	ตัวอย่างใช้ return	39
2.55	ตัวอย่างการใช้ฟังก์ชันการหาค่าสูงสุด	40
2.56	ตัวอย่างการเรียกใช้ ฟังก์ชั่น ในอิลิเมนต์	41

สารบัญรูป (ต่อ)

รูป		หน้า
2.57	ตัวอย่างการเรียกใช้ฟังก์ชั่นในภาคแสดงของรูปแบบการเข้าถึงข้อมูล	41
2.58	ตัวอย่างการเรียกใช้ฟังก์ชั่นใน let cause	41
2.59	ตัวอย่างการกำหนดเงื่อนไขสำหรับการแสดงผลของผลลัพธ์	41
2.60	รูปแบบของการสร้างฟังก์ชั่นขึ้นใช้งานเอง	42
2.61	ตัวอย่างของฟังก์ชั่นที่สร้างขึ้นเอง	42
2.62	ตัวอย่างการกำหนดเงื่อนไขในการจำกัดจำนวน (1)	43
2.63	ตัวอย่างการกำหนดเงื่อนไขในการจำกัดจำนวน (2)	43
2.64	ตัวอย่างการค้นหาข้อมูลภายในเอกสาร XML ที่มีมากกว่า 1 เอกสาร	44
2.65	ตัวอย่างการกำหนดเงื่อนไขด้วยแอตทริบิวต์	45
2.66	ตัวอย่างการแสดงแอตทริบิวต์ในผลลัพธ์	45
2.67	แบบจำลองการทำงานของ XQuery	46
3.1	แสดง URLs เป็นซับเซตของ URIs	51
3.2	ตัวอย่างกราฟ FOAF	53
3.3	การเขียน RDF แบบ N-triple	54
3.4	การเขียน RDF แบบ มาตรฐาน	55
3.5	การเขียน RDF แบบ Notation-3	55
3.6	ข้อมูลกราฟ RDF	57
3.7	แผนภาพกราฟ	57
3.8	RDF/XML แบบที่ 1 ที่ได้จากรูป 3.7	58
3.9	RDF/XML แบบที่ 2 ที่ได้จากรูป 3.7	59
3.10	ตัวอย่างการใช้ RDF Containers	60
3.11	แผนภาพการใช้ RDF Containers	61
3.12	ตัวอย่างการใช้ Alternative	62
3.13	แผนภาพการใช้ Alternative	62
3.14	ตัวอย่างการใช้ Collection	63
3.15	แผนภาพการใช้ Collection	64
3.16	ตัวอย่าง rdf:Description	65
3.17	ตัวอย่างไวยากรณ์โดยย่อ	65

สารบัญรูป (ต่อ)

รูป		หน้า
3.18	ไวยากรณ์และส่วนประกอบของ RDF Schema	66
3.19	การอธิบายโครงสร้างของ เมตะเดตา	66
3.20	การอธิบายความสัมพันธ์ของ แหล่งข้อมูล ในรูปแบบของ RDF Schema	67
3.21	ตัวอย่างข้อมูลในการกำหนดส่วนของ RDF Schema	68
3.22	การอธิบายความสัมพันธ์ของ คลาส 2 คลาส	69
3.23	RDF Document ของกราฟที่อยู่ในรูป 3.22	70
3.24	สถาปัตยกรรมแบบลำดับชั้นของเว็บสื่อความหมาย	75
3.25	ระดับความสามารถของภาษาย่อย	75
3.26	ความสัมพันธ์เชิงคลาสระหว่าง OWL และ RDF/RDFS	77
3.27	ตัวอย่างรูตอิลิเมนส์	78
3.28	ตัวอย่างอิลิเมนส์ owl:Ontology	78
3.29	ตัวอย่างการใช้อิลิเมนส์ owl:Class นิยามคลาส associateProfessor	79
3.30	ตัวอย่างการใช้อิลิเมนส์ owl:disjointWith	79
3.31	ตัวอย่างการใช้อิลิเมนส์ owl:equivalentClass	79
3.32	ตัวอย่างนิยามคุณสมบัติชนิดข้อมูล	80
3.33	ตัวอย่างนิยามคุณสมบัติออบเจ็คต์	80
3.34	ตัวอย่างเชื่อมเข้าหาคุณสมบัติที่เป็นส่วนกลับ	81
3.35	คุณสมบัติที่เป็นส่วนกลับของกันและกัน	81
3.36	ตัวอย่างเชื่อมเข้าหาคุณสมบัติที่เป็นส่วนกลับ	81
3.37	ตัวอย่างการระบุคลาสย่อยนิรนาม	82
3.38	ตัวอย่างกำหนดค่าตายตัวให้แก่คุณสมบัติ	82
3.39	ตัวอย่างการใช้ owl:someValuesFrom	83
3.40	ตัวอย่างการกำหนดจำนวนสมาชิกขั้นต่ำด้วย OWL	84
3.41	ตัวอย่างการกำหนดจำนวนสมาชิกขั้นต่ำ และขั้นสูง	84
3.42	ตัวอย่างการกำหนดคุณสมบัติเพิ่มเติมลงไปในคุณสมบัติ	86
3.43	ตัวอย่างการนิยามคลาสด้วยการรวมกันด้วยบูลีน	86
3.44	ตัวอย่างการนิยามคลาสโดยการยูเนียน	86
3.45	ตัวอย่างการนิยามคลาสโดยการอินเตอร์เซ็คชัน	87

สารบัญรูป (ต่อ)

รูป		หน้า
3.46	ตัวอย่างการรวมกันด้วยบูลีนแบบซ้อน	88
3.47	ตัวอย่างการแจกแจงของคลาส	88
3.48	ตัวอย่างการสร้างอินสแตนซ์วิธีที่ 1	89
3.49	ตัวอย่างการสร้างอินสแตนซ์วิธีที่ 2	89
3.50	การระบุรายละเอียดเพิ่มเติมของอินสแตนซ์ด้วย การใช้คุณสมบัติที่ประกาศไว้ในออนโทโลยี	89
3.51	กำหนดให้คุณสมบัติ isTaughtBy เป็นคุณสมบัติเชิงฟังก์ชัน	89
3.52	สร้างอินสแตนซ์ของรายวิชาหนึ่งขึ้นมา และทำการกำหนดผู้สอนมากกว่า 1 คนให้	90
3.53	การระบุความแตกต่างระหว่างอินสแตนซ์	90
3.54	การระบุความแตกต่างระหว่างอินสแตนซ์ด้วยวิธีที่รวบรัดกว่า	90
3.55	ตัวอย่างการระบุข้อมูลเกี่ยวกับรุ่นของเอกสาร	91
3.56	การนิยามคลาส C1	92
3.57	การนิยามคลาส C2 เพื่อความถูกต้อง	92
3.58	ความสัมพันธ์ระหว่างคลาสภายในออนโทโลยีสัตว์ป่าในทวีปแอฟริกา	93
3.59	สเตทเมนต์ที่กล่าวว่ากิ่งไม้เป็นส่วนหนึ่งของต้นไม้	94
3.60	ออนโทโลยีสัตว์ป่าในทวีปแอฟริกา	94
3.61	ความสัมพันธ์ระหว่างคลาสภายในออนโทโลยีเครื่องพิมพ์	97
3.62	ออนโทโลยีเครื่องพิมพ์	97
3.63	การประกาศเนมสเปสต่าง ๆ ภายในเอกสาร	101
3.64	การนิยามคลาสของทุกคลาสในภาษา OWL	101
3.65	ความสัมพันธ์ระหว่างคลาส Thing และคลาส Nothing	101
3.66	นิยามคลาส Thing และคลาส Nothing	102
3.67	นิยามของ owl:EquivalenceClass, owl:EquivalenceProperty และ owl:disjointWith	102
3.68	นิยามของ owl:sameIndividualAs, owl:differentFrom, และ owl:sameAs	103
3.69	นิยามของ owl:distinctMembers และ owl:AllDifferent	103
3.70	นิยามของ owl:unionOf	103
3.71	นิยามของ owl:complementOf	104
3.72	นิยามของ owl:Restriction	104

สารบัญรูป (ต่อ)

รูป		หน้า
3.73	นิยามของ owl:onProperty, owl:allValuesFrom, owl:someValuesFrom, owl:hasValue, owl:minCardinality, owl:maxCardinality, และ owl:cardinality	105
3.74	นิยามของ owl:ObjectProperty	105
3.75	นิยามของ owl:TransitiveProperty, owl:SymmetricProperty, owl:FunctionalProperty, และ owl:InverseFunctionalProperty	106
3.76	นิยามของ owl:inverseOf	106
3.77	ข้อสังเกตเพิ่มเติมต่าง ๆ ของโครงสร้างต่าง ๆ ภายใน OWL	107
3.78	โค้ดการใช้งาน RDFlib	108
3.79	รูปแบบการเก็บทริปเปิลด้วย RDFlib	108
3.80	โค้ดการสืบค้นด้วยเมธอด triples()	108
3.81	โค้ดการบันทึกข้อมูลกราฟ RDF ลงบนสื่อบันทึกข้อมูล	109
3.82	โค้ดการอ่านข้อมูลกราฟ RDF โดยการอ่านข้อมูลที่บันทึกอยู่ในเครื่อง	109
3.83	โค้ดการกระทำกันระหว่างอ็อบเจ็คต์ชนิด Graph	109
3.84	โค้ดการสร้างอ็อบเจ็คต์ชนิด URIRef	109
3.85	โค้ดสร้างอ็อบเจ็คต์ชนิด Namespace และ URIRef จากเนมสเปสที่ประกาศไว้	110
3.86	โค้ดสร้างอ็อบเจ็คต์ชนิด Namespace จาก URIRefที่นิยามไว้ก่อนแล้วในกราฟข้อมูล RDF	110
3.87	โค้ดสร้างอ็อบเจ็คต์ชนิด Namespace จากURIRef ของคลาส ConjunctiveGraph	110
3.88	โค้ดบันทึกข้อมูลกราฟ RDF ลงฐานข้อมูล	111
3.89	โค้ดอ่านข้อมูลกราฟ RDF จากฐานข้อมูล	112
3.90	โค้ดสืบค้นข้อมูลจากกราฟภายในตัวเก็บทริปเปิล	112
3.91	โค้ดสืบค้นข้อมูลรูปแบบ CONSTRUCT ด้วยไลบรารี RDFLib	113
3.92	ข้อมูลกราฟ RDF เกี่ยวกับข้อมูลภาพยนตร์	114
3.93	ข้อมูลกราฟ RDF ในรูปแบบของกราฟ	116
3.94	ลักษณะเครื่องมือที่ใช้ในการสืบค้นข้อมูลกราฟ RDF ชื่อว่า Twinkle	117
3.95	ประกาศ PREFIX สำหรับใช้ในการสืบค้น	117
3.96	ประกาศ BASE สำหรับใช้ในการสืบค้น	118
3.97	โค้ดภาษา SPARQL	118

สารบัญรูป (ต่อ)

รูป	หน้า
3.98 ผลลัพธ์จากการสืบค้น	118
3.99 โค้ดภาษา SPARQL	119
3.100 ผลลัพธ์จากการสืบค้น	119
3.101 โค้ดภาษา SPARQL	123
3.102 ผลลัพธ์จากการสืบค้น	123
3.103 โค้ดภาษา SPARQL	124
3.104 ผลลัพธ์จากการสืบค้น	124
3.105 โค้ดภาษา SPARQL	124
3.106 ผลลัพธ์จากการสืบค้น	125
3.107 โค้ดภาษา SPARQL	125
3.108 ผลลัพธ์จากการสืบค้น	125
3.109 โค้ดภาษา SPARQL	126
3.110 ผลลัพธ์จากการสืบค้น	126
3.111 โค้ดภาษา SPARQL	127
3.112 ผลลัพธ์จากการสืบค้น	127
3.113 โค้ดภาษา SPARQL	128
3.114 ผลลัพธ์จากการค้นหา	128
3.115 โค้ดภาษา SPARQL	129
3.116 ผลลัพธ์จากการสืบค้น	130
3.117 โค้ดภาษา SPARQL	130
3.118 โค้ดภาษา SPARQL	130
3.119 โค้ดภาษา SPARQL	131
3.120 ผลลัพธ์จากการสืบค้น	131
3.121 RDF Dataset หรือ Background graph	131
3.122 RDF Dataset หรือ http://example.org/foaf/aliceFoaf graph	132
3.123 RDF Dataset หรือ http://example.org/foaf/bobFoaf	132
3.124 โค้ดภาษา SPARQL	132

สารบัญรูป (ต่อ)

รูป หน้า

3.125 โค้ดภาษา SPARQL 133

3.126 ผลลัพธ์จากการสืบค้น....133

3.127 ตัวรับฟีดชื่อ GreatNews 134

3.128 ตัวอย่างเอกสาร RSS....135

3.129 ส่วนประกอบต่าง ๆ ภายใน Sesame และการขึ้นต่อกันของแต่ละส่วนประกอบ....144

3.130 ดีพลอยเว็บแอปพลิเคชันที่แนบมากับ Sesame....145

3.131 OpenRDF Sesame Server.... 145

3.132 OpenRDF Workbench....146

3.133 นิยามคลาส RemoteGraph สำหรับเชื่อมต่อไปยังที่เก็บข้อมูลกราฟ RDF ใน Open RDF SesameServe 147

3.134 โค้ดตัวอย่างเชื่อมต่อไปยังกราฟข้อมูล RDF จำลองที่สร้างไว้บน D2R Server 152

3.135 การให้บริการของ D2R Server แก่โปรแกรมประยุกต์ชนิดต่าง ๆ 154

3.136 พารามิเตอร์ที่ generate-mapping รับ....156

3.137 พารามิเตอร์ที่ dump-rdf รับ 156

3.138 เว็บไซต์ที่ d2r-server สร้างขึ้นมาให้โดยอัตโนมัติ....157

3.139 หน้าเว็บไซต์ที่เปิดรับคำสั่งสืบค้นภาษา SPARQL 157

3.140 เค้าร่างฐานข้อมูลเชิงสัมพันธ์ของการประชุมวิชาการศาสนา....158

3.141 อธิบายการเชื่อมต่อเข้าสู่ระบบฐานข้อมูลเชิงสัมพันธ์ และค่าตั้งต้นของ D2R Server....158

3.142 ตัวอย่างคำสั่งภาษา D2RQ ที่ทำงานกับเอนทิตี religion ในเค้าร่างฐานข้อมูลเชิงสัมพันธ์....159

3.143 ตัวอย่างคำสั่งสืบค้นด้วย Construct....160

3.144 ตัวอย่างการสร้าง Blank node 161

3.145 กราฟข้อมูลนักศึกษาและอาจารย์ ต้นแบบที่จะถูกแปลง....161

3.146 กราฟข้อมูลนักศึกษาและอาจารย์ ผลลัพธ์ที่ถูกแปลงแล้ว....162

สารบัญรูป (ต่อ)

รูป		หน้า
4.1	แผนภาพลำดับการสืบค้นข้อมูลจากแหล่งข้อมูล	163
4.2	โค้ดประกาศเนมสเปส และฟังก์ชันที่ทำงานกับโปรโตคอล HTTP	164
4.3	โค้ดฟังก์ชันที่ทำหน้าที่สืบค้นข้อมูลจาก DBpedia	165
4.4	โค้ดฟังก์ชันที่ทำหน้าที่สืบค้นข้อมูลวงดนตรีจาก Freebase	166
4.5	โค้ดฟังก์ชันที่ทำหน้าที่สืบค้นรายชื่ออัลบั้ม และบทวิจารณ์ของอัลบั้มต่าง ๆ	167
4.6	โค้ดเรียกใช้ฟังก์ชันต่าง ๆ ที่นิยามเอาไว้ เพื่อสืบค้นข้อมูลจากแหล่งต่าง ๆ	168
4.7	ผลลัพธ์จากการเรียกโปรแกรมประยุกต์	168
4.8	คลาส อินสแตนส์ และความสัมพันธ์ระหว่างอินสแตนส์ภายในออนโทโลยี	170
4.9	โค้ดอธิบายคลาส ภาคแสดง และอินสแตนส์ของออนโทโลยีสำหรับใช้ในการทดลอง	171
4.10	คำสั่งสืบค้น	172
4.11	โค้ดโปรแกรมที่สืบค้นข้อมูลแบบเดียวกัน	172
4.12	ผลลัพธ์การสืบค้นด้วยเครื่องมือ Twinkle	172
4.13	ผลลัพธ์จากการสืบค้นด้วยโปรแกรมที่ใช้ไลบรารี RDFlib	173
4.14	การกำหนด คลาส ของ Ontology	174
4.15	ตัวอย่างการกำหนด Restriction ของคลาส President	174
4.16	การใส่ Object Property	175
4.17	การใส่ชนิดข้อมูล Property	175
4.18	ตัวอย่าง instance ของคลาส President	176
4.19	การใส่รายละเอียดย่อยของ แต่ละ instance ตาม Restriction ที่กำหนดไว้	176
4.20	ภาพโปรแกรมโดยรวม	177
4.21	มี Reasoning check ความถูกต้องของ ontology โดย Pellet 1.5.2	177
4.22	ความสัมพันธ์ระหว่างกราฟออนโทโลยี President	178
4.23	source code ที่ Protégé สามารถ generate ได้	178
5.1	ทิศทางการสืบค้นข้อมูลของแอพพลิเคชันกลาง	180
5.2	RDF Schema สำหรับเก็บข้อมูลสมาชิกเว็บไซต์คอนโดมีเนียม	181
5.3	RDF Schema สำหรับเก็บข้อมูลที่จำเป็นสำหรับเก็บโฆษณา	182
5.4	หน้าหลักของเว็บไซต์	183
5.5	หน้าค้นหาโฆษณาของเว็บไซต์	183

สารบัญรูป (ต่อ)

รูป		หน้า
5.6	โค้ดสำหรับค้นหาโฆษณาภายในฐานข้อมูล RDF โดยสร้างคำสั่งสืบค้น SPARQL	184
5.7	แผนภาพ ER อธิบายข้อมูลโฆษณาคอนโดมีเนียม	186
5.8	เค้าร่างฐานข้อมูลเชิงสัมพันธ์สำหรับเก็บข้อมูลโฆษณาคอนโดมีเนียม	187
5.9	หน้าเว็บไซต์หลักของเว็บไซต์คอนโดมีเนียมที่พัฒนาด้วยภาษา PHP	187
5.10	RDF Schema ที่ได้จากการแปลงเค้าร่างฐานข้อมูลเชิงสัมพันธ์	188
5.11	หน้าเว็บที่ได้จากการตั้งเซิร์ฟเวอร์	189
5.12	แผนภาพ ER อธิบายข้อมูลโฆษณาอพาร์ตเมนต์	189
5.13	เค้าร่างฐานข้อมูลเชิงสัมพันธ์สำหรับเก็บข้อมูลโฆษณาอพาร์ตเมนต์	190
5.14	หน้าหลักของเว็บไซต์ที่พัฒนาขึ้นมา	190
5.15	RDF Schema ที่ได้มาจากการแปลงเค้าร่างฐานข้อมูลเชิงสัมพันธ์	191
5.16	หน้าเว็บที่ได้จากการตั้งเซิร์ฟเวอร์	192
5.17	RDF Schema สำหรับเว็บเอปพลิเคชันกลาง	193
5.18	หน้าหลักของเว็บแอปพลิเคชันกลาง	193
ก.1	หน้าเว็บสมัครสมาชิกใหม่	201
ก.2	หน้าเว็บสำหรับแก้ไขประวัติส่วนตัว	202
ก.3	แสดงรายการโฆษณาคอนโดมีเนียมของสมาชิก	202
ก.4	หน้าเว็บสำหรับแก้ไขโฆษณา	203
ก.5	หน้าเว็บให้ผู้ใช้เลือกคอนโดมีเนียมที่ต้องการลงโฆษณา	203
ก.6	หน้าเว็บกรอกรายละเอียดคอนโดมีเนียมแห่งใหม่ที่สมาชิกต้องการลงประกาศโฆษณา	204
ก.7	หน้าเว็บไซต์สำหรับกรอกรายละเอียดโฆษณาห้องพัก	204
ก.8	หน้าเว็บหลัก	205
ก.9	หน้าเว็บสมัครสมาชิกใหม่	206
ก.10	หน้าเว็บสำหรับแก้ไขประวัติส่วนตัว	206
ก.11	รายการโฆษณาคอนโดมีเนียมของสมาชิก	207

สารบัญรูป (ต่อ)

รูป		หน้า
ก.12	หน้าเว็บให้ผู้ใช้เลือกคอนโดมีเนียมที่ต้องการลงโฆษณา	207
ก.13	หน้าเว็บรายละเอียดคอนโดมีเนียมแห่งหนึ่งที่สมาชิกต้องการลงประกาศโฆษณา	208
ก.14	หน้าเว็บไซต์สำหรับกรอกรายละเอียดโฆษณาห้องพัก	208
ก.15	แถบเมนูที่ใช้สำหรับสืบค้นข้อมูล	209
ก.16	หน้าเว็บสมัครสมาชิกใหม่	209
ก.17	สมาชิกลงชื่อเข้าใช้งานระบบเพื่อประกาศโฆษณาอพาร์ทเมนต์	210
ก.18	รายการโฆษณาอพาร์ทเมนต์ของสมาชิก	210
ก.19	หน้าเว็บรายละเอียดอพาร์ทเมนต์แห่งหนึ่งที่สมาชิกต้องการลงประกาศโฆษณา	211
ก.20	แถบเมนูที่ใช้สำหรับสืบค้นข้อมูล	211
ก.21	หน้าหลักของเว็บแอปพลิเคชันกลาง	212
ก.22	ผลลัพธ์จากการสืบค้นข้อมูลของเว็บเอปพลิเคชันกลาง	212

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

เนื่องจากปัจจุบันอินเตอร์เน็ตเข้ามามีบทบาทในชีวิตประจำวันของมนุษย์มากขึ้น เวิลด์ไวด์เว็บ (World Wide Web) คือพื้นที่ที่เก็บข้อมูลข่าวสารที่เชื่อมต่อกันทาง ถ้าหากเรามองเป็นฐานข้อมูลขนาดใหญ่เราก็จะสามารถดึงข้อมูลที่เป็นประโยชน์มาได้เป็นจำนวนมากมาย ซึ่งในการแลกเปลี่ยนข้อมูลนั้นจำเป็นต้องหาสิ่งที่ทำให้เกิดการแลกเปลี่ยนข้อมูลจำนวนมากนี้ ที่เกิดจากหลายๆที่รวมกันรวบรวมข้อมูลให้เกิดประสิทธิภาพในบริษัทหรือองค์กรต่างๆ มีการสื่อสารกันโดยใช้อินเตอร์เน็ต และยังมีเทคโนโลยีใหม่ๆ เช่น ภาษา XML, เทคโนโลยี RDF และ OWL เกิดขึ้น โดยเราสามารถใช้เทคโนโลยีเหล่านี้เป็นสื่อกลางในการอธิบายข้อมูลที่มีอยู่ได้ในรูปแบบเอกสาร ทำให้เกิดความคิดที่จะมองเอกสารที่ใช้เทคโนโลยีดังกล่าวในรูปแบบของฐานข้อมูลกันเป็นอย่างมากในวงการวิชาการ

จะเป็นประโยชน์อย่างมากหากเราต้องการจะทราบผลข้อมูลใดๆ แล้วเราสามารถที่จะสืบค้นเอกสาร XML หรือ RDF ที่มีจำนวนมากมายในอินเตอร์เน็ต แล้วได้ผลลัพธ์ของการสืบค้นออกมาเป็นเอกสาร XML หรือ RDF ฉบับเดียวที่มีเนื้อหาครบถ้วน หรืออาจใช้ออนโทโลยีเพื่อเป็นตัวช่วยในการสืบค้นข้อมูลได้ง่ายขึ้น

1.2 วัตถุประสงค์โครงงาน

1) เพื่อศึกษา XML และ RDF ในฐานะฐานข้อมูล
2) เพื่อศึกษาโครงสร้างและมุมมองของเอกสาร RDF ในเชิงฐานข้อมูล
3) เพื่อศึกษาภาษา SPARQL ที่ใช้ในการสืบค้นข้อมูล
4) เพื่อศึกษาออนโทโลยีของข้อมูล และวิธีสร้างออนโทโลยี (Ontology)
5) เพื่อศึกษา ความสามารถของภาษา SPARQL ในการสืบค้นเอกสาร RDF
6) พัฒนาแอพพลิเคชัน RDF และ RDFS

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้รับความรู้ ความเข้าใจเกี่ยวกับเอกสาร XML และ RDF ในฐานะฐานข้อมูล
2) ได้รับความรู้ เข้าใจลักษณะเอกสาร RDF
3) ได้รับความรู้ ความเข้าใจเกี่ยวกับภาษา SPARQL ในการสืบค้นเอกสาร RDF และ OWL
4) ได้รับความรู้เกี่ยวกับออนโทโลยีของข้อมูล และวิธีสร้างออนโทโลยี
5) ได้รับความรู้เกี่ยวกับ ภาษา SPARQL ในการสืบค้นเอกสาร OWL

6) สามารถจัดการไฟล์เอกสาร XML และ RDF ได้

7) เพิ่มศักยภาพในทางวิจัย โดยดำเนินการค้นคว้าวรรณกรรมที่เกี่ยวข้องด้วยตนเอง

8) ได้โปรแกรมประยุกต์ที่มีศักยภาพในการใช้งานได้จริงในธุรกิจและอุตสาหกรรม

1.4 ขอบเขตของโครงงาน

ศึกษาเทคโนโลยีของ RDF ประโยชน์และการนำไปใช้ในอินเตอร์เน็ต ว่ามีการใช้งานเอกสาร RDF ในลักษณะใดและมาใช้ในงานฐานข้อมูลอย่างไร อีกทั้งต้องเข้าใจเทคโนโลยีภาษา XML ในการเขียนโครงสร้าง ไวยากรณ์ และสามารถสร้างไฟล์เอกสาร XML ได้ ซึ่งนำมาใช้ในการเข้าใจเอกสาร RDF ซึ่งใช้ ภาษา XML เป็นไวยากรณ์สำหรับอธิบายกราฟข้อมูล

ต่อจากนั้นจึงศึกษาโครงสร้างเอกสาร RDF ลักษณะการเขียนต่างๆ และภาษา SPARQL ในการสืบค้นข้อมูล จากนั้นทดลองสร้างไฟล์ RDF ขึ้นมาเพื่อศึกษาลักษณะของเอกสาร ความสามารถที่เพิ่มมาจาก XML

จากนั้นจึงศึกษาออนโทโลยีของเอกสาร RDF แล้วทดลองสร้างออนโทโลยีขึ้นมา ทดสอบภาษา SPARQL ในการทดลองสืบค้นข้อมูลในเอกสารออนโทโลยีนั้นๆ

1.5 วิธีการดำเนินงาน

1) ศึกษาค้นคว้าและรวบรวมข้อมูลเบื้องต้นที่เกี่ยวข้องกับ RDF

2) ศึกษาเกี่ยวกับ จุดประสงค์ และลักษณะการนำไปใช้ของเอกสาร RDF

3) ศึกษาภาษา XML ในฐานะฐานข้อมูล

4) ทดลองเขียนและสร้างไฟล์เอกสาร XML

5) ศึกษา RDF ในฐานะฐานข้อมูล

6) ศึกษาภาษา SPARQL แล้วความสามารถในการสืบค้นข้อมูลในลักษณะกราฟ

7) ศึกษาภาษา OWL (Web Ontology Language)

8) ทดลองสร้างออนโทโลยีและทดลองการใช้ SPARQL กับ ontology

9) สร้างเว็บแอปพลิเคชันต้นแบบที่ใช้งาน RDF ในฐานะฐานข้อมูล

10) สร้างเว็บแอปพลิเคชันต้นแบบที่ใช้งานระบบฐานข้อมูลเชิงสัมพันธ์ แล้วนำเครื่องมือสร้างกราฟข้อมูลเชิงสัมพันธ์ แล้วนำเครื่องมือมาสร้างกราฟข้อมูล RDF จำลอง

บทที่ 2
ทฤษฎีที่เกี่ยวข้องกับ XML

2.1 ทฤษฎีพื้นฐานของภาษา XML

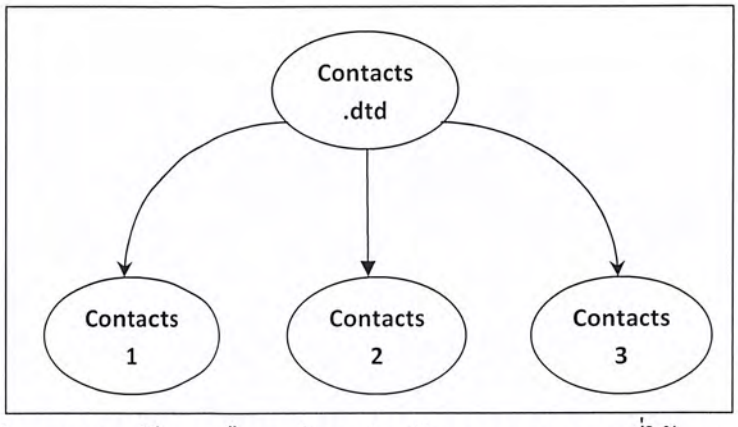
2.1.1 ประวัติความเป็นมาของภาษา XML

ภาษา XML (XML Extensible Markup Language) เป็นภาษาที่ถูกเผยแพร่โดยสมาคม W3C (World Wide Web Consortium) จุดประสงค์เพื่อสร้างภาษาที่สามารถนิยามข้อมูลได้ (data definition) ภาษา XML จัดเป็นภาษาที่อยู่ในตระกูลของภาษาที่ใช้อธิบายและ/หรือให้คำจำกัดความของข้อมูล ภาษา XML มีรากฐานมาจากภาษา GML (Generalized Markup Lange) ซึ่งบริษัทไอบีเอ็ม (IBM) ได้คิดค้นขึ้นในปี ค.ศ. 1969 เพื่อใช้ระบบประมวลผลข้อความ โดยแนวคิดของภาษานี้คือเพื่อให้นักพัฒนาสามารถคิดกลุ่มของแท็ก (Tag) ขึ้นเอง เพื่อใช้ในการอธิบายเอกสาร

```
:h1.Example of GML
:p.Here is the first paragraph
and here is an ordered list
:ol
:li.Item A
:li.Item B
:li.Item C
:eol.
```

รูป 2.1 ตัวอย่างเอกสารที่นิยามด้วยแท็กของภาษา GML

ถ้าสังเกตดูจะเห็นว่าเอกสารของภาษา GML มีส่วนคล้ายกับภาษา HTML (HTML) เช่นกัน ตัวอย่างเช่น :h1. ในภาษา GML มีลักษณะคล้ายกับ <h1> ในภาษา HTML หรือ :P. ก็คล้ายกับ <p> เป็นต้น หลังจากที่ภาษา GML เกิดขึ้นมาแล้ว จุดอ่อนของภาษา GML คือ ไม่มีความเป็นมาตรฐาน ทำให้เป็นอุปสรรคในการนำไปใช้งาน จึงเป็นเหตุให้มีความพยายามคิดค้นภาษามาร์คอัพ (markup) ตัวใหม่ ที่ยึดพื้นฐานมาจากภาษา GML เพื่อสร้างให้เป็นมาตรฐาน ภาษาใหม่ที่ได้ก็คือ SGML (Standard Generalized Markup Language) โดยมีการเปิดตัวครั้งแรกในปี ค.ศ. 1980 และได้ผ่านการปรับปรุงแก้ไขหลายครั้งจนกระทั่งปี ค.ศ. 1986 ก็ได้ภาษา SGML ที่สมบูรณ์เป็นมาตรฐาน โดยมีรหัสเป็นทางการคือ ISO 8879:1986 ภาษา SGML จะต้องมีเอกสารที่เรียกว่า DTD (Document Type Definition) เพื่อนิยามโครงสร้างของเอกสาร



รูป 2.2 DTDเปรียบเสมือนแม่แบบสำหรับเอกสาร SGML ที่ใช้งาน

```
01: <!DOCTYPE contacts SYSTEM “contacts.dtd”>
02: <Contacts>
03:     <Contact>
04:         <Name>Brad Pitt</Name>
05:         <Phone>0814567234</Phone>
06:         <Email>bradpitt@hollywood.com</Email>
07:     </Contact>
08:     <Contact>
09:         <Name>Britney Spears</Name>
10:         <Phone>0814567234</Phone>
11:         <Email>britneyspears@hollywood.com</Email>
12:     </Contact>
13: </Contacts>
```

รูป 2.3 ตัวอย่างเอกสาร SGML เก็บข้อมูลติดต่อบุคคล 2 รายการ ใน 1 เอกสาร

จากรูป 2.3 จะเห็นได้ว่าบรรทัดที่ 1 มีการอ้างอิงถึงเอกสาร contacts.dtd เพราะเป็นต้นแบบเอกสารที่ต้องยึดถือไว้ เนื้อหาในไฟล์ contacts.dtd นั้นระบุข้อกำหนดต่างๆ ของโครงสร้างเอกสาร ประโยชน์ของภาษา SGML ในยุคนั้น คือการนำไปใช้ในแวดวงการพิมพ์ของต่างประเทศ แต่เมื่อเวลาผ่านไปมีข้อจำกัดเรื่องค่าใช้จ่ายในการสร้างและบำรุงรักษา ภาษานี้จึงไม่ค่อยได้รับความนิยม แต่ภาษา SGML นับว่าเป็นจุดเริ่มต้นที่สำคัญของภาษาใหม่ๆ หนึ่งในนั้นคือภาษา HTML ซึ่งได้นำพื้นฐานมาจากภาษา SGML โดยการสร้างกลุ่มของแท็กหรืออิลิเมนต์เพื่อใช้ในการแสดงผลบนเว็บผ่านโปรแกรมเว็บเบราเซอร์

```
01: <html>
02: <head><title>Contacts</title></head>
03: <body>
04:     <h1>Contacts</h1>
05:     <b>Brad Pitt</b><br>0814567234</br>
06:     <i>bradpitt@hollywood.com</i><br></br>
07:     <b>Britney
                Spears</b><br>0814567234</br>
08:     <i> britneyspears
                @hollywood.com</i><br></br>
09:      <a href="other.html">Other
                                    Contacts</a>
10: </body>
```

รูป 2.4 ตัวอย่างเอกสาร HTML

ภาษา HTML ช่วยให้เราสามารถสร้างสรรค์เว็บเพจได้สวยงามและหลากหลาย แต่ความสามารถแท้ๆ ของภาษา HTML ก็แป็นเพียงแค่การแสดงผลทางเว็บเท่านั้น ข้อมูลที่นิยามด้วยแท็ก HTML ไม่พร้อมที่จะนำไปใช้ทำอย่างอื่นเลย ต่อมาได้มีการนำข้อดีของภาษา SGML และภาษา HTML เข้าไว้ด้วยกันนั่นคือได้นำความสามารถในการนิยามข้อมูลของภาษา SGML และความสามารถนำมาใช้งานผ่านระบบอินเทอร์เน็ตของภาษา HTML มาพัฒนาเป็นภาษา XML ซึ่งเป็นเทคโนโลยีที่เกี่ยวพันกับการกำหนดลักษณะ และโครงสร้างข้อมูล ซึ่งเป็นข้อมูลที่สามารถอธิบายความหมายของตัวข้อมูลเองได้ (Self-Described Data) โดยอาศัยแท็กที่ผู้ใช้กำหนดขึ้นเพื่อนิยามข้อมูล อีกทั้งเอกสาร XML จะมีการอ้างถึงการกำหนดโครสร้างของเอกสาร XML แบบ DTD หรือ การกำหนดโครสร้างของเอกสาร XML แบบ XML Schema (XML Schema) ซึ่งมีประโยชน์สำหรับการนำมาวางรูปแบบเอกสาร ทำให้สามารถนำมาเป็นรูปแบบในการแลกเปลี่ยนเอกสารระหว่างหน่วยงานหรือองค์กรได้โดยไม่กังวลเรื่องความแตกต่างของรูปแบบข้อมูล จุดเด่นของภาษา XML ที่ได้รับความนิยมและแพร่หลายในปัจจุบันคือเป็นภาษาที่แสดงผลได้หลายแพลตฟอร์ม (Platform independent)

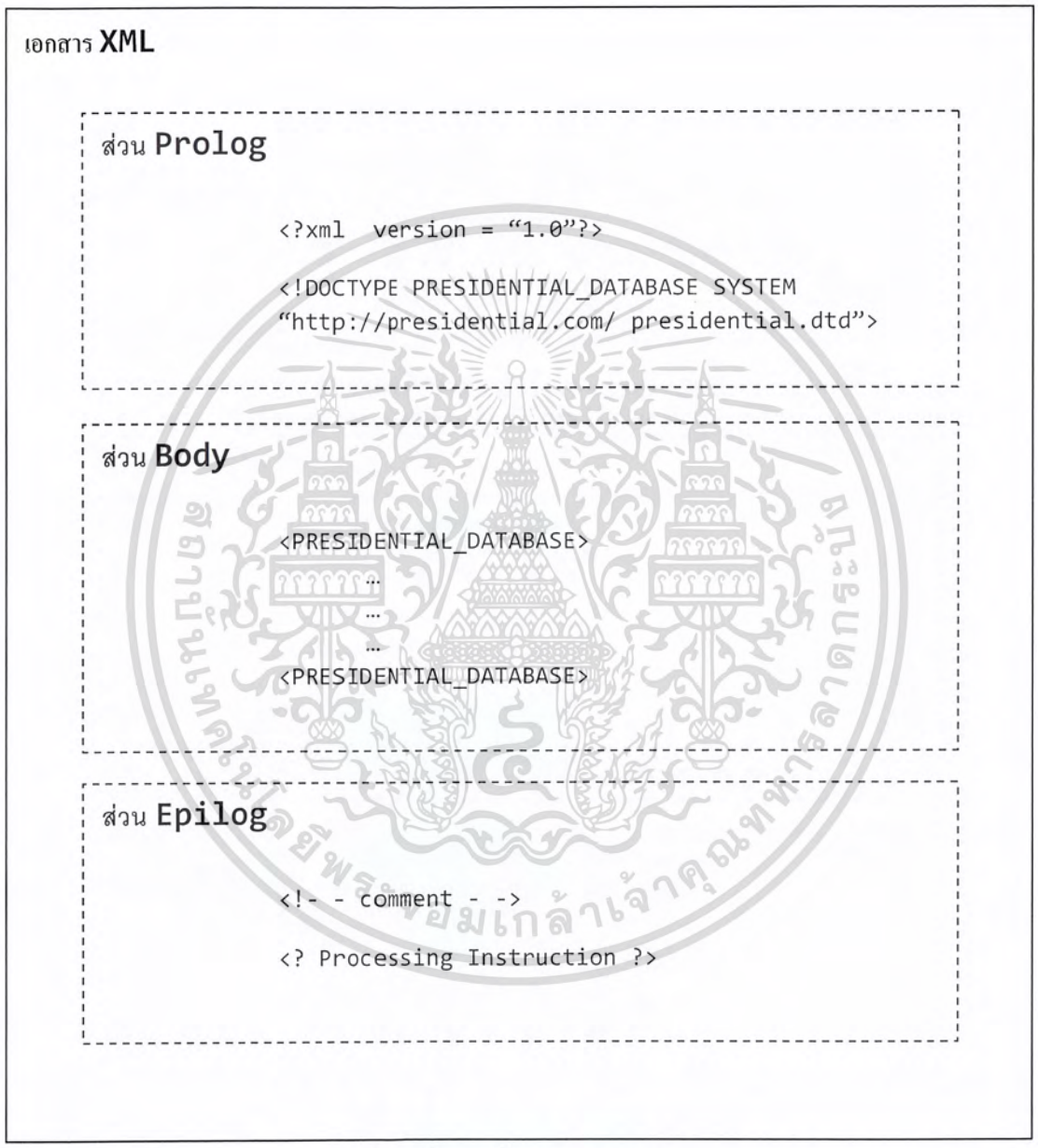
2.1.2 โครงสร้างเอกสาร XML

เอกสาร XML มีโครงสร้างประกอบด้วย 3 ส่วนหลัก ดังนี้

1) ส่วนโปรล็อก (Prolog) ส่วนนี้ประกอบด้วย 2 ส่วนย่อย คือ การประกาศเอกสาร XML (XML declaration) และชนิดของเอกสาร XML (document type declaration)

2) ส่วนตัวเอกสาร(Body) เป็นส่วนของเนื้อหาเอกสารประกอบ ด้วยแท็กที่นิยามข้อความหรือข้อมูล และข้อมูลภายในแท็ก

3) ส่วนเอพิล็อก (Epilog)เป็นส่วนที่เป็นข้อความอธิบาย (comment) และ ส่วนประมวลผล (Processing Instruction) โดยทั่วไปส่วนของเอพิล็อกจะแทรกกระจายไปส่วนต่างๆของเอกสาร แต่ในรูป 2.5 จะแสดงไว้ส่วนท้ายเพื่อแสดงให้เห็นโครงสร้างเอกสาร XML ได้อย่างชัดเจน



รูป 2.5 โครงสร้างเอกสาร XML

โครงสร้างของเอกสาร XML มีข้อมูลที่ต้องทำความเข้าใจอยู่ 4 ประเภท ได้แก่ การประกาศเอกสาร XML , การบอกประเภทเอกสาร XML , ข้อความอธิบาย และส่วนประมวลผล

2.1.2.1 การประกาศเอกสาร XML (XML declaration)

การประกาศเอกสาร XML เป็นการประกาศให้ทราบว่าเอกสารนี้เป็นเอกสาร XML ไม่ใช้ไฟล์ข้อความธรรมดา นอกจากเอกสาร XML แล้ว เอกสารอื่นๆ ที่มีรากฐานมาจากภาษา XML ก็ต้องมีการประกาศความเป็นภาษานั้นๆ ด้วย เช่น ภาษา XHTML, ภาษา WML ที่ใช้กับภาษา WAP (Wireless Application Protocol)

```
<?xml version= "1.0" encoding= "ชุดรหัสตัวอักษร" standalone="yes/no"?>
```

รูป 2.6 รูปแบบที่สมบูรณ์ของการประกาศเอกสาร XML

จากรูปที่ 2.6 ในการประกาศเอกสารอักษรทุกตัวของแต่ละแอตทริบิวต์ จะต้องเป็นตัวอักษรพิมพ์เล็ก (lower case) ห้ามใช้ตัวอักษรพิมพ์ใหญ่ (uppercase) และต้องเรียงลำดับตามที่แสดงเท่านั้น โดยความหมายของแต่ละแอตทริบิวต์ มีดังนี้

1) version เป็นแอตทริบิวต์ที่ต้องระบุไว้เสมอ เพื่อบ่งบอกเวอร์ชั่นของ XML ซึ่งมีการเปลี่ยนแปลงได้ในอนาคต

2) encoding เป็นแอตทริบิวต์ตัวเลือก (Optional) ที่ระบุเมื่อจำเป็นต้องใช้เท่านั้น โดยบ่งบอกชุดรหัสอักษรที่ใช้เอกสาร XML เพื่อให้ตัวแปลเอกสาร (XML parser)เช่น UTF-8

3) standalone เป็นแอตทริบิวต์ตัวเลือก (Optional) ที่ระบุเมื่อจำเป็นต้องใช้เท่านั้น แอตทริบิวต์นี้จะบอกให้รู้ว่าเอกสาร XML ขึ้นอยู่กับเอกสารอื่นหรือไม่ ถ้ามีค่าเป็น yes หมายถึง เอกสาร XML นั้นไม่ขึ้นกับเอกสารอื่น แต่ถ้า no หมายถึง เอกสาร XML นั้นขึ้นกับเอกสารอื่น เช่น เอกสารการกำหนดโครงสร้างเอกสาร XML แบบ DTDหรือ การกำหนดโครสร้างของเอกสาร XML แบบ XML Schema

2.1.2.2 การบอกประเภทของเอกสาร XML (Document type declaration)

การบอกประเภทของเอกสาร XML คือ การประกาศชนิดของเอกสาร XML รวมถึงไฟล์ DTDที่กำหนดโครงสร้างของเอกสาร

```
<!DOCTYPE PRESIDENTIAL_DATABASE SYSTEM
"http://presidential.com/ presidential.dtd">
```

รูป 2.7 ตัวอย่างการบอกประเภทของเอกสาร XML

ตัวอย่างจากรูป 2.7 เป็นการประกาศให้ทราบว่าเอกสาร XML นี้มีอิลิเมนต์ PRESIDENTIAL_DATABASE เป็นรูตอิลิเมนต์ (Root element) และมีไฟล์ DTD ชื่อ presidential.dtd อยู่ที่ http://presidential.com/ presidential.dtd สำหรับคีย์เวิร์ด SYSTEM ต้องระบุไว้เมื่อมีการอ้างถึงตำแหน่งไฟล์ด้วย URI (Uniform Resource Identifier) ในกรณีที่อ้างถึงไฟล์แบบโลคอล (Local) ต้องใช้คีย์เวิร์ด PUBLIC แทน เช่น ไฟล์ที่อยู่บนคอมพิวเตอร์เครื่องเดียวกัน

การประกาศประเภทของเอกสาร XML สามารถประกาศได้ 2 แบบ

1) การประกาศแบบภายใน (Internal DTD) เป็นการประกาศโดยแทรก DTD ไว้ในส่วนโปรล็อกของเอกสาร XML
2) การประกาศแบบภายนอก (External DTD) เป็นการประกาศโดยการแยกไฟล์DTD ออกมาต่างหาก

2.1.2.3 ข้อความอธิบาย (Comment)

การเขียนข้อความอธิบายของเอกสาร XML ข้อความอธิบายจะอยู่ภายในเครื่องหมาย <!- - และ - -> โดยข้อความอธิบายจะปรากฏอยู่ที่ส่วนใดของเอกสารก็ได้ แต่จะมีข้อห้ามดังต่อไปนี้

1) ห้ามเขียนข้อความอธิบายลงในส่วนบนสุดของเอกสาร XML เพราะสงวนไว้สำหรับการประกาศเอกสาร XML
2) ห้ามแทรกข้อความอธิบายอยู่ในแท็ก

```
<Elemant <!- - comment - -> Value </Element>
```

รูป 2.8 ตัวอย่างการแทรกข้อความอธิบายอยู่ในแท็ก

3) ห้ามมีเครื่องหมายลบ 2 ตัวติดกัน(--) ซึ่งจะทำให้มีลักษณะเหมือนกับเครื่องหมายเปิดปิดข้อความอธิบาย เพราะจะทำให้ตัวแปลภาษา XML ทำงานสับสนทำงานสับสนและเกิดข้อผิดพลาดได้

ประโยชน์ของข้อความอธิบายโดยพื้นฐานก็คือ เป็นข้อความสำหรับให้ใครอ่านก็ได้ อาจเป็นข้อความเพื่อเน้นย้ำ แนะนำ หรือเตือนความจำ เป็นต้น โดยที่ตัวแปรภาษา XML จะละเว้นข้อความอธิบายไว้ ไม่ถูกแปลเหมือนข้อความอื่นๆในเอกสาร XML และสามารถอาศัย

คุณสมบัติของข้อความอธิบายมาช่วยตัดงานบางส่วนออกไปชั่วคราวโดยไม่ต้องลบทิ้ง เพื่อไม่ให้ตัวแปลภาษา XML แปลงานส่วนนั้น

2.1.2.4 ส่วนประมวลผล (Processing Instruction)

การแทรกส่วนประมวลผล เป็นส่วนที่ตัวแปลภาษาจะต้องประมวลผล ข้อความส่วนที่เป็นส่วนประมวลผลจะขึ้นต้นด้วยเครื่องหมาย <? และลงท้ายด้วย ?>

```
<?xml-stylesheet type= “text/xsl” href= “mystyle.xsl”?>
```

รูป 2.9 ตัวอย่างข้อความส่วนประมวลผล

จากตัวอย่างในรูปที่ 2.9 เป็นการบอกให้ตัวแปลภาษา XML นำไฟล์ mystyle.xsl มากำหนดรูปแบบการแสดงผลของเอกสาร XML ทางเบราเซอร์

2.1.3 กฎพื้นฐานของภาษา XML

กฎพื้นฐานของภาษา XML ได้แก่ กฎไวยากรณ์ที่เกี่ยวข้องกับอิลิเมนต์ โดยอิลิเมต์นั้นมีความหมายครอบคุมตั้งแต่แท็กและข้อมูลภายในแท็ก กฎไวยากรณ์ที่เกี่ยวข้องกับอิลิเมนต์ มีดังนี้

1) เอกสาร XML หนึ่งๆ จะมีรูตอิลิเมนต์ได้เพียงหนึ่งเดียวเท่านั้น ซึ่งทำหน้าที่คลุมอิลิเมนต์อื่นๆ ทั้งหมด

```
<PRESIDENTIAL_DATABASE>
     <PRESIDENT PRES_NAME="Washington_G">
          <PROFILE>
               <BIRTH_YR>1732</BIRTH_YR>
               <PARTY>Federalist</PARTY>
          </PROFILE>
     </PRESIDENT>
     <PRESIDENT PRES_NAME="Adams_J">
          <PROFILE>
               <BIRTH_YR>1735</BIRTH_YR>
               <PARTY>Federalist</PARTY>
          </PROFILE>
     </PRESIDENT>
</PRESIDENTIAL_DATABASE>
```

รูป 2.10 ตัวอย่างเอกสาร XML ที่มี <PRESIDENTIAL_DATABASE> เป็นรูตอิลิเมนต์

2) แท็กเปิดและแท็กปิดต้องเหมือนกัน ต่างกันเพียงในแท็กปิดต้องมีเครื่องหมายทับ (slash) นำหน้าชื่อแท็กนั้น เช่น <PARTY>Federalist</PARTY>

3) ห้ามระบุแท็กเหลื่อมซ้อนกัน (Overlap) คือแท็กที่เปิดก่อนต้องปิดทีหลัง

```
<BIRTH_YR><PARTY>1735Federalist</BIRTH_YR></PARTY>
```

รูป 2.11 ตัวอย่างแท็กที่เหลื่อมซ้อนกัน

4) ชื่อแท็กที่พิมพ์ด้วยตัวอักษรพิมพ์ใหญ่ หรือตัวอักษรพิมพ์เล็ก ถือว่าแตกต่างกัน (case-sensitive) เช่น แท็ก <party>, <Party>, <PARTY> เหล่านี้ถือว่าเป็นคนละแท็กกัน

5) แท็กว่าง (Empty tag) คือแท็กที่ไม่มีข้อมูลข้างใน มีวิธีเขียน 2 แบบ แบบที่หนึ่งคือ <PARTY/> หรืออีกแบบหนึ่งคือ <PARTY></PARTY> แต่ส่วนมากนิยมใช้แบบแรก และแท็กว่างสามารถมีแอตทริบิวต์ได้เหมือนอิลิเมนต์หรือแท็กปกติทุกอย่าง

6) ค่าของแอตทริบิวต์ ต้องอยู่ในเครื่องหมายคำพูดแบบอัญประกาศ ("...") หรือ บุพสัญญา ('...') อย่างใดอย่างหนึ่งเท่านั้น ในกรณีที่ค่าของแอตทริบิวต์ต้องอยู่ร่วมกันหลายค่า ต้องหลีกเลี่ยงโดยการใช้เครื่องหมายที่แตกต่างกัน

```
<link onClick="location.href='main.xml' ">
                    หรือ
<link onClick="location.href='main.xml' ">
```

รูป 2.12 ตัวอย่างการใช้ อัญประกาศและ บุพสัญญา ร่วมกัน

7) ในภาษา XML มีอักขระที่สงวนไว้ 5 ตัว ได้แก่ <, >, &, " และ ' เพราะต้องใช้เป็นส่วนประกอบตามโครงสร้างภาษาซึ่งไม่สามารถนำไปตั้งเป็นชื่ออิลิเมนต์,ชื่อ แอตทริบิวต์ หรือเนื้อความภายในเอกสารได้ โดยอักษรพิเศษชุดนี้เรียกว่า เอนทิตี้อ้างอิง (Entity Reference) และอักขระพิเศษสามารถแทนด้วยชุดตัวอักษรพิเศษ ดังตาราง 2.1 ตัวอย่างเช่น หากต้องการส่งข้อความ "president ที่มีปีเกิด < ค.ศ.1798" สามารถเขียนภาษา XML ได้ดังนี้ <text>president ที่มีปีเกิด &lt ค.ศ.1798</text> เป็นต้น

ตาราง 2.1 Entity Reference

อักขระพิเศษ	ชุดอักษรพิเศษ
<	<
>	&
&	>
“	"
‘	'

8) การตั้งชื่อแท็ก มีหลักเกณฑ์ที่ควรจำ คือ ชื่อแท็กต้องขึ้นต้นด้วยอักษร หรือเครื่องหมายขีดล่าง (Underscore) เท่านั้น ตัวถัดไปต้องเป็นตัวอักษร, ตัวเลข, เครื่องหมายจุด (.) , เครื่องหมายยัติภังค์ (-) , เครื่องหมายขีดล่าง(_) หรือเครื่องหมายโคลอน(:) เท่านั้นชื่อแท็กมีคุณสมบัติตัวอักษรพิมพ์ใหญ่และตัวอักษรพิมพ์เล็กถือว่าแตกต่างกัน อักษรสามตัวแรกของชื่อแท็กห้ามเป็นคำว่า“xml”ไม่ว่าจะตัวอักษรพิมพ์ใหญ่หรือพิมพ์เล็ก เพื่อเป็นการสงวนไว้ใช้ในอนาคต

2.1.4 มาตรฐานที่เกี่ยวข้องกับภาษา XML

มาตรฐานเสริมศักยภาพของการทำงานของภาษา XML ที่ถูกเสนอต่อ W3C มีอยู่จำนวนมาก ซึ่งเป็นตัวช่วยเสริมสร้างศักยภาพแก่ภาษา XML ตัวอย่างดังต่อไปนี้

2.1.4.1 เนมสเปส (Namespace)

เนมสเปสทำหน้าที่ช่วยในการกำหนดขอบเขตให้กับชื่อ อิลิเมนต์และแอตทริบิวต์เพื่อหลีกเลี่ยงการถูกเรียกใช้ซ้ำกันระหว่างเอกสาร XML ที่เกิดจากแหล่งกำเนิดต่างกัน เนมสเปสแต่ละชุดเป็นการรวมของชื่อต่างๆ ของอิลิเมนต์และแอตทริบิวต์ของ XML ที่ถูกระบุพร้อมกันกับURI ที่เป็นแหล่งกำเนิดของเอกสาร XML นั้น

2.1.4.2 XSL (Extensible Style Sheet Language)

XSL เป็นภาษาในการกำหนดสไตล์ชีทให้กับข้อมูล XML สำหรับการนำเสนอในเว็บเบราเซอร์หรือสื่ออื่นๆ XSL จะมีศักยภาพสูงกว่า ซีเอสเอส (Cascading Style Sheet, CSS) ที่ใช้สำหรับ HTML

2.1.4.3 XLink

XLink มีพื้นฐานการทำงานมาจาก HyTime และ XLink เป็นการปรับปรุงการเชื่อมโยงใน HTML โดยสนับสนุนการเชื่อมโยงแบบสองทาง (bi-direction) แบบหนึ่งต่อกลุ่ม (one-to-many) และแบบแยกตามประเภท (typed links) เช่น การอ้างอิงเชิงอรรถ การอภิธานศัพท์

2.1.4.4 XPointer

XPointer มีพื้นฐานการทำงานมาจาก TEI (Text Encoding Initiative TEI อนุญาตให้สามารถชี้ไปที่จุดหรือตำแหน่งใดก็ได้ในเอกสารเป้าหมาย โดยไม่ต้องกำหนด การเชื่อมโยงแบบ HTML ไว้ล่วงหน้า

2.1.4.5 RDF (Resource Description Framwork)

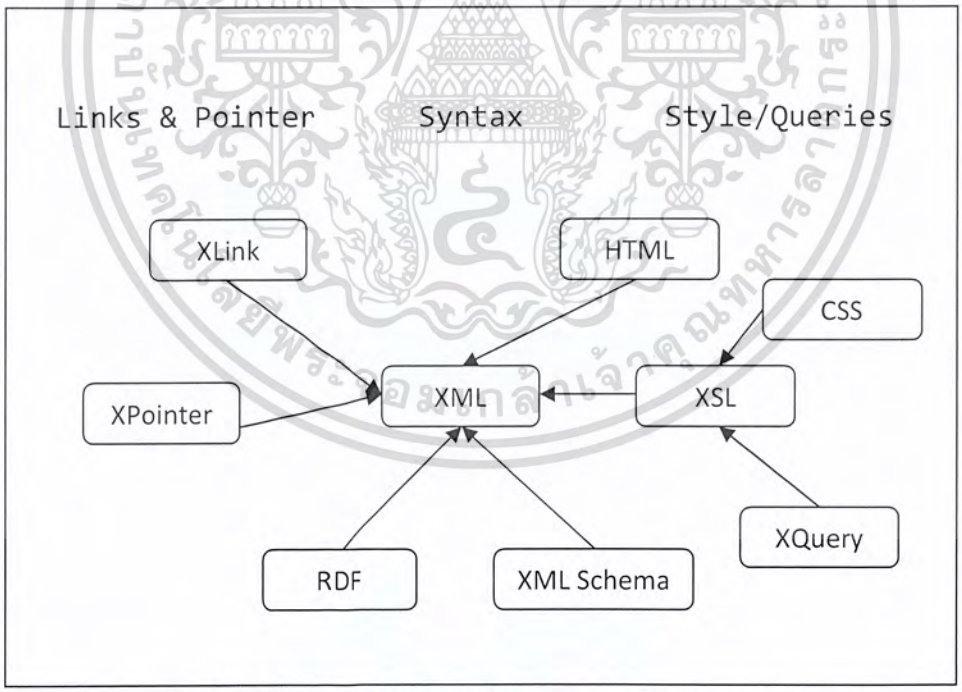
RDFเป็นเมตะเดตาสำหรับเอกสาร XML คล้ายกับเมตะเดตาแท็ก ของ HTML

2.1.4.6 XML Schema

XML Schema เข้ามาแทนที่ DTD สำหรับการกำหนดประเภทของข้อมูลของ XML โดย XML Schema เป็นการประยุกต์หลายเทคโนโลยีเข้าด้วยกัน ได้แก่ XData, SOX, DCD และ DDML

2.1.4.7 XQuery

XQuery เป็นส่วนขยายเพิ่มเติมของXSLสำหรับใช้อ้างถึงและกลั่นกรองส่วนที่เป็นอิลิเมนต์และแท็กของเอกสาร XML โดย XQuery ช่วยกำหนดรูปแบบที่ชัดเจน เข้าใจง่ายในการเข้าถึงอิลิเมนต์ที่เฉพาะเจาะจง และสำหรับการค้นหาโหนด ที่มีคุณลักษณะพิเศษต่างๆ



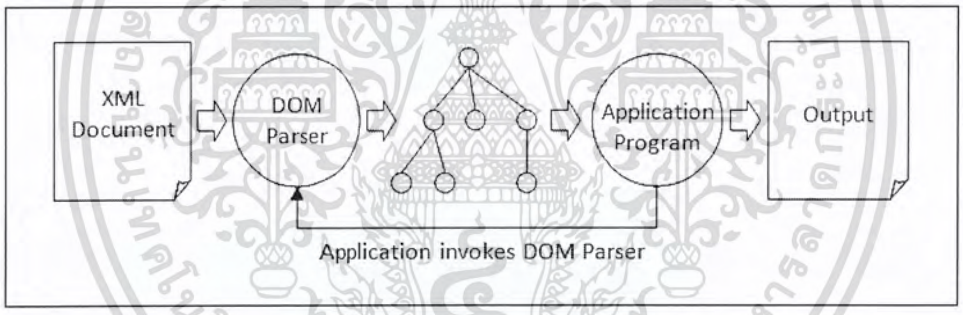
รูป 2.13 ความสัมพันธ์ระหว่าง 7 มาตรฐานของ XML

นอกจากมาตรฐานทั้ง 7 ของ XML ที่กล่าวมานั้น ยังมีการกำหนดมาตรฐานด้านเอพีไอ (Application Programming Interface, API) เพื่อที่จะให้แอพพลิเคชันสามารถเข้าถึงเอกสาร

ได้ทันที เมื่อเอกสาร XML เข้าสู่กระบวนการแปลเอกสาร (parsing) โดยมี API 2 กลุ่มที่ได้รับความนิยม ได้แก่

2.1.4.8 DOM (Document Object Model , DOM)

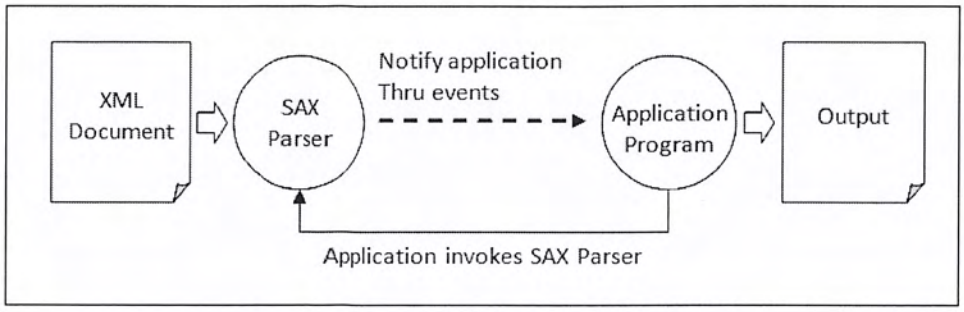
DOM เป็น โครงสร้างต้นไม้ (Tree-based API) ที่ทำการประมวลผลโครงสร้างเอกสาร XML ให้เป็นโครงสร้างแบบต้นไม้ที่ประกอบไปด้วยลำต้น กิ่งก้านสาขา และใบ เพื่อให้แอพลิเคชันสามารถเข้าถึงจุดต่างๆ ของโครงสร้างต้นไม้นั้นได้ โดยที่ขอบเขตจะถูกจำกัดโดยหน่วยความจำที่เรียกใช้ได้ในขณะนั่น นั่นคือ DOM เป็นกระบวนการดึงข้อมูลในเอกสาร XML มาใช้งานในแอพพลิเคชันโดยมีหลักการคือ จะนำข้อมูลจากเอกสาร XML ทั้งเอกสารมาวางเป็นโครงสร้างลักษณะลำดับชั้นต้นไม้ในหน่วยความจำของเครื่องคอมพิวเตอร์ แล้วใช้วิธีท่องไปในต้นไม้เพื่อดึงข้อมูลมา ซึ่งวิธีการโหลดข้อมูลทั้งเอกสาร XML เข้ามาไว้ในหน่วยความจำนั้นทำให้เกิดเป็นข้อเสียอย่างหนึ่งของ DOM เพราะจะกลายเป็นข้อจำกัดในเรื่องหน่วยความจำในกรณีที่เอกสาร XML มีขนาดใหญ่มากๆ ก็จะต้องใช้หน่วยความจำมาก และอาจไม่มีเนื้อที่เหลือพอที่จะทำงาน



รูป 2.14 การทำงานของ DOM

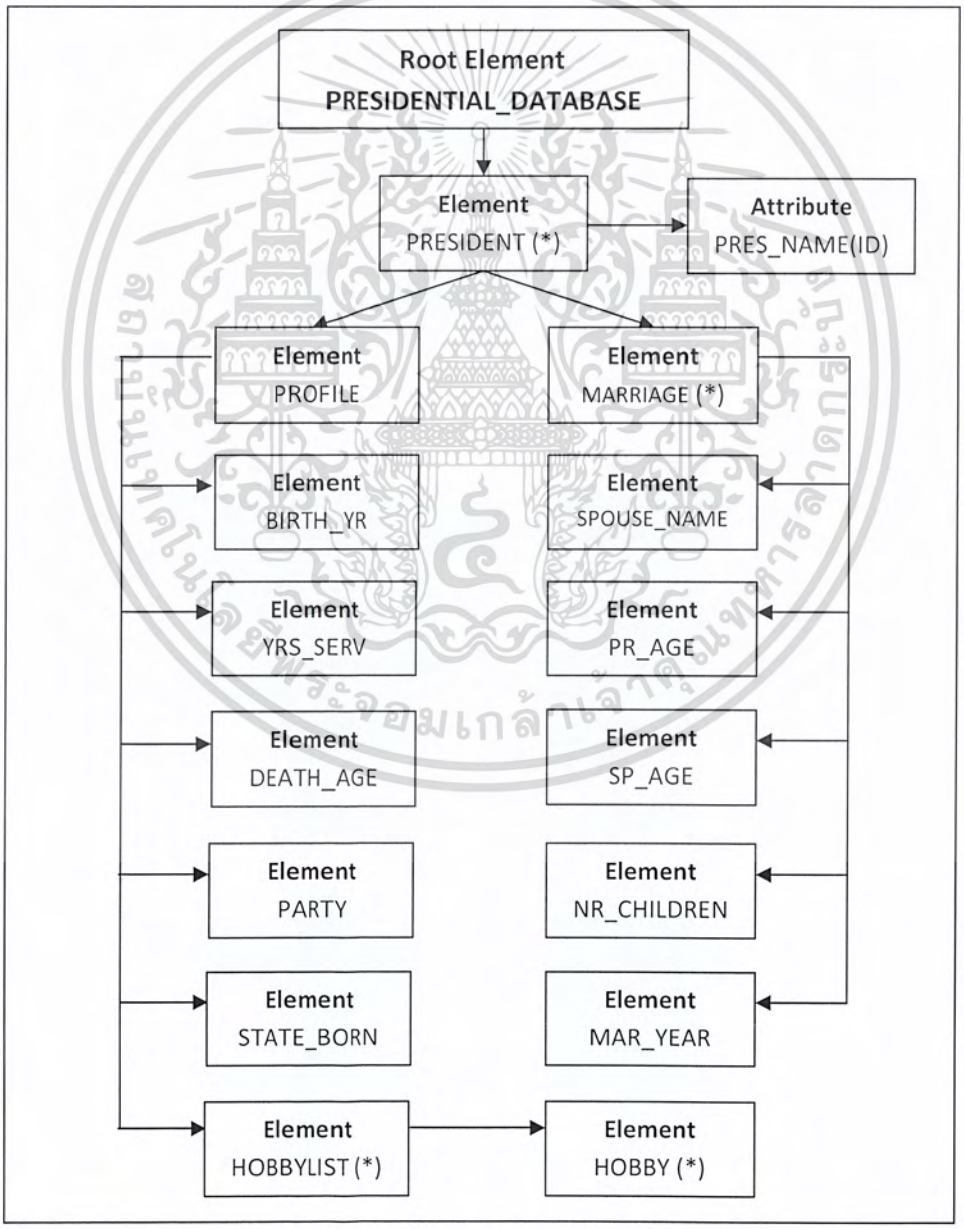
2.1.4.9 SAX (Simple API for XML, SAX)

SAX เป็น Event-based API ที่รายงานตามเหตุการณ์ของกระบวนการแปลเอกสาร เช่น จุดเริ่มต้นและจุดสิ้นสุดของอิลิเมนต์ต่างๆ ไปให้แอพลิเคชันผ่านทางการร้องขอ โดยปกติจะไม่มีการสร้างโครงสร้างต้นไม้ขึ้นมา เนื่องจากการทำงานเป็นแบบ Event-based API การเข้าถึงเอกสารจึงทำได้ง่ายไม่ซับซ้อน และที่สำคัญผู้ใช้สามารถทำกระบวนการแปลเอกสาร เอกสารที่มีขนาดใหญ่มากกว่าหน่วยความจำที่เรียกใช้ได้ และผู้ใช้สามารถสร้างโครงสร้างข้อมูลของเอกสาร XML ให้อยู่ในรูปแบบที่ตนเองต้องการได้



รูป 2.15 การทำงานของ SAX

2.1.5 ตัวอย่างเอกสาร XML



รูป 2.16 โครงสร้างต้นไม้ของ DTD ของPRESIDENT.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PRESIDENTIAL_DATABASE [
      <!ELEMENT PRESIDENTIAL_DATABASE (PRESIDENT)*>
      <!ELEMENT PRESIDENT (PROFILE, MARRIAGE*)>
      <!ATTLIST PRESIDENT PRES_NAME ID #REQUIRED>
      <!ELEMENT PROFILE (BIRTH_YR, YRS_SERV, DEATH_AGE?, PARTY, STATE_BORN,
HOBBYLIST*)>
      <!ELEMENT MARRIAGE (SPOUSE_NAME, PR_AGE, SP_AGE, NR_CHILDREN, MAR_YEAR)>
      <!ELEMENT BIRTH_YR (#PCDATA)>
      <!ELEMENT YRS_SERV (#PCDATA)>
      <!ELEMENT DEATH_AGE (#PCDATA)>
      <!ELEMENT PARTY (#PCDATA)>
      <!ELEMENT STATE_BORN (#PCDATA)>
      <!ELEMENT HOBBYLIST (HOBBY)*>
      <!ELEMENT HOBBY (#PCDATA)>
      <!ELEMENT SPOUSE_NAME (#PCDATA)>
      <!ELEMENT PR_AGE (#PCDATA)>
      <!ELEMENT SP_AGE (#PCDATA)>
      <!ELEMENT NR_CHILDREN (#PCDATA)>
      <!ELEMENT MAR_YEAR (#PCDATA)>
     ] >
<PRESIDENTIAL_DATABASE>
     <PRESIDENT PRES_NAME="Washington_G">
          <PROFILE>
               <BIRTH_YR>1732</BIRTH_YR>
               <YRS_SERV>7</YRS_SERV>
               <DEATH_AGE>67</DEATH_AGE>
               <PARTY>Federalist</PARTY>
               <STATE_BORN>Virginia</STATE_BORN>
               <HOBBYLIST>
                    <HOBBY>Fishing</HOBBY>
                    <HOBBY>Riding</HOBBY>
               </HOBBYLIST>
          </PROFILE>
          <MARRIAGE>
               <SPOUSE_NAME>Custis_M_D</SPOUSE_NAME>
               <PR_AGE>26</PR_AGE>
               <SP_AGE>27</SP_AGE>
               <NR_CHILDREN>0</NR_CHILDREN>
               <MAR_YEAR>1759</MAR_YEAR>
          </MARRIAGE>
     </PRESIDENT>
     <PRESIDENT PRES_NAME="Adams_J">
          <PROFILE>
               <BIRTH_YR>1735</BIRTH_YR>
               <YRS_SERV>4</YRS_SERV>
               <DEATH_AGE>90</DEATH_AGE>
               <PARTY>Federalist</PARTY>
               <STATE_BORN>Massachusetts</STATE_BORN>
          </PROFILE>
          <MARRIAGE>
               <SPOUSE_NAME>Smith_A</SPOUSE_NAME>
               <PR_AGE>28</PR_AGE>
               <SP_AGE>19</SP_AGE>
               <NR_CHILDREN>5</NR_CHILDREN>
               <MAR_YEAR>1764</MAR_YEAR>
          </MARRIAGE>
     </PRESIDENT>
</PRESIDENTIAL_DATABASE>
```

รูป 2.17 ตัวอย่างเอกสาร PRESIDENT.xml

2.2 ความถูกต้องของเอกสาร XML

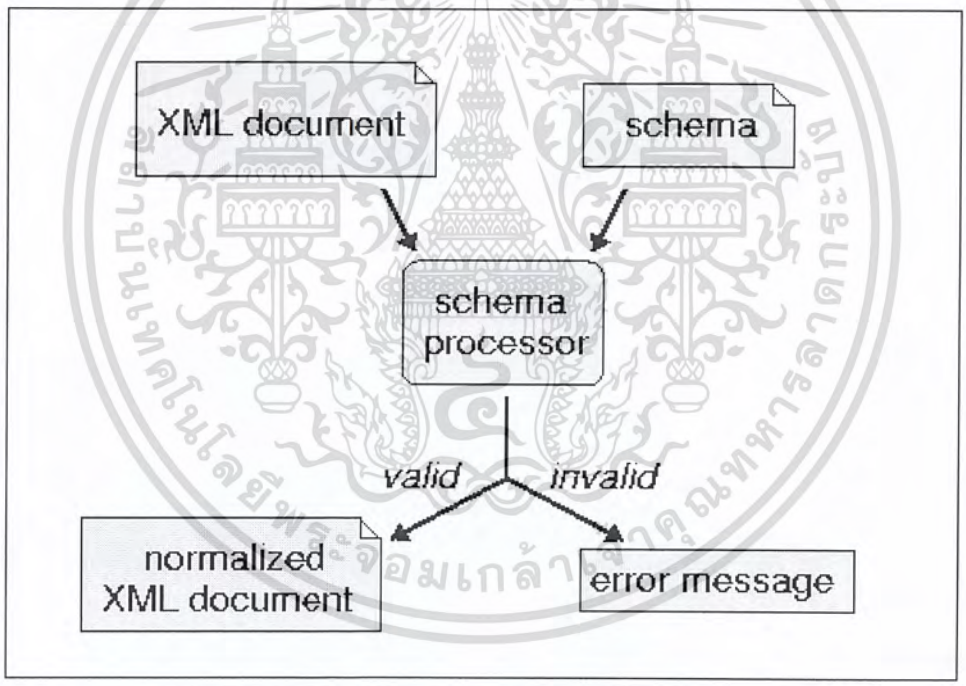
เนื่องจากเอกสาร XML ยอมให้สร้างแท็กขึ้นมาเองได้ ดังนั้น จึงจำเป็นต้องมีมาตรฐานที่ใช้วัดความถูกต้องของเอกสาร XML ขึ้นมา มี 2 ประเภท ได้แก่

2.2.1 เอกสารถูกต้อง (Well-formed XML)

คือเอกสาร XML ที่มีโครงสร้างถูกต้อง โดยมีลักษณะที่ถูกต้องตามกฎพื้นฐานของภาษา XML (หัวข้อ 2.1.3) ซึ่งจะเห็นว่าความถูกต้องในระดับถูกต้อง นั้นรับประกันความถูกต้องแค่ความถูกต้องของไวยากรณ์พื้นฐานของภาษา XML เท่านั้น

2.2.2 เอกสารถูกต้องสมบูรณ์ (Valid XML)

คือเอกสาร XML ที่มีโครงสร้างข้อมูลถูกต้องและครบถ้วน ซึ่งอาศัย DTD หรือ XML Schema ในการอธิบายตัวข้อมูล เช่น องค์ประกอบของข้อมูล, ชนิดของข้อมูล และอื่นๆ เพื่อนำมาทำการตรวจสอบความถูกต้อง (Validation) ของเอกสาร XML



รูป 2.18 กระบวนการตรวจสอบความถูกต้องของเอกสาร

2.3 การกำหนดโครสร้างของเอกสาร XML แบบ DTD และแบบ XML Schema

การกำหนดโครงสร้างของเอกสาร XML นั้นสามารถทำได้หลายวิธี เช่น DCD (Document Content Description), RELAX/TREX, BizTalk, XDR (XML-Data Reduced), DTD, XML Schema เป็นต้น แต่ที่นิยมในปัจจุบันได้แก่ DTD และ XML Schema โครงสร้างของเอกสาร XML นั้นมีประโยชน์สำคัญ 2 ประการ คือ

1) เป็นการกำหนดรูปแบบโครงสร้างของเอกสาร XML หรือเรียกว่าแบบจำลองข้อมูล (Data model) ซึ่งเป็นการกำหนดว่าเอกสาร XML มีแท็กอะไรบ้าง มีแอตทริบิวต์อะไรบ้าง ค่าของแอตทริบิวต์เป็นอะไรบ้าง เป็นต้น
2) เป็นสัญญาระหว่างองค์กรที่ใช้เอกสาร XML ในการแลกเปลี่ยนข้อมูล เพื่อให้รูปแบบของเอกสาร XML ที่ใช้มีรูปแบบที่ตรงกัน และเพื่อความถูกต้องในการนำเอกสาร XML ไปใช้

2.3.1 การกำหนดโครงสร้างเอกสาร XML แบบ DTD

การวางโครงสร้างของเอกสาร XML แบบ DTD นั้นสืบทอดมาจากภาษา SGML ปัจจุบันมีแนวโน้มของการใช้งานน้อยลง แต่ก็ควรศึกษาเพื่อเป็นพื้นฐานในการออกแบบโครงสร้างของเอกสาร XML เพราะมีบางเทคโนโลยียังใช้งาน DTD อยู่บ้าง เช่น เทคโนโลยี WAP (Wireless Application Protocol) ซึ่งเป็นการเรียกดูข้อมูลบนอินเทอร์เน็ตผ่านมือถือ ซึ่งถูกสร้างด้วยภาษา WML (Wireless Markup Language) เป็นภาษาที่อยู่ในตระกูลของภาษา XML และมีการกำหนดโครงสร้างเอกสารแบบ DTD

2.3.1.1 การประกาศโครงสร้างเอกสารแบบ DTD

มีลักษณะการประกาศอยู่ 2 รูปแบบได้แก่

1) การประกาศแบบภายใน (Internal DTD) การใช้งาน DTD แบบภายใน จะประกาศไว้ในการบอกประเภทเอกสาร XML โดยจะแทรก DTD ไว้ในตัวเอกสาร XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PRESIDENTIAL_DATABASE [
      <!ELEMENT PRESIDENTIAL_DATABASE (PRESIDENT)*>
      <!ELEMENT PRESIDENT (PROFILE)>
      <!ATTLIST PRESIDENT PRES_NAME ID #REQUIRED>
      <!ELEMENT PROFILE (BIRTH_YR, YRS_SERV, DEATH_AGE?,PARTY,
                         STATE_BORN, HOBBYLIST*)>
      <!ELEMENT BIRTH_YR (#PCDATA)>
      <!ELEMENT YRS_SERV (#PCDATA)>
      <!ELEMENT DEATH_AGE (#PCDATA)>
      <!ELEMENT PARTY (#PCDATA)>
      <!ELEMENT STATE_BORN (#PCDATA)>
      <!ELEMENT HOBBYLIST (HOBBY)*>
      <!ELEMENT HOBBY (#PCDATA)>
     ] >
<PRESIDENTIAL_DATABASE>
     <PRESIDENT PRES_NAME="Washington_G">
          <PROFILE>
               <BIRTH_YR>1732</BIRTH_YR>
               <YRS_SERV>7</YRS_SERV>
               <DEATH_AGE>67</DEATH_AGE>
               <PARTY>Federalist</PARTY>
               <STATE_BORN>Virginia</STATE_BORN>
               <HOBBYLIST>
                    <HOBBY>Fishing</HOBBY>
                    <HOBBY>Riding</HOBBY>
               </HOBBYLIST>
          </PROFILE>
     </PRESIDENT>
</PRESIDENTIAL_DATABASE>
```

รูป 2.19 การประกาศ DTD แบบภายใน (Internal DTD)

2) การประกาศแบบภายนอก (External DTD) การประกาศ DTD แบบภายนอกนั้น จะมีการแยกส่วนของ DTD ออกจากไฟล์เอกสาร XML โดยมีการอ้างอิงถึงไฟล์โครงสร้างจะมีนามสกุล .dtd ลักษณะการเรียกใช้ไฟล์ DTD แสดงดังรูปที่ 2.20

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PRESIDENTIAL_DATABASE SYSTEM
            "http://presidential.com/ presidential.dtd">
<PRESIDENTIAL_DATABASE>
     <PRESIDENT PRES_NAME="Washington_G">
          <PROFILE>
               <BIRTH_YR>1732</BIRTH_YR>
               <YRS_SERV>7</YRS_SERV>
               <DEATH_AGE>67</DEATH_AGE>
               <PARTY>Federalist</PARTY>
               <STATE_BORN>Virginia</STATE_BORN>
               <HOBBYLIST>
                    <HOBBY>Fishing</HOBBY>
                    <HOBBY>Riding</HOBBY>
               </HOBBYLIST>
          </PROFILE>
     </PRESIDENT>
</PRESIDENTIAL_DATABASE>
```

รูป 2.20 การประกาศ DTD แบบภายนอก

2.3.1.2 การประกาศอิลิเมนต์

ในเอกสารที่เป็นเอกสารที่ถูกต้องสมบูรณ์ (Valid XML) จะต้องประกาศชนิดของทุกๆอิลิเมนต์ที่ใช้ในเอกสาร โดยจะต้องแสดงชื่อของชนิดอิลิเมนต์ และข้อมูลประเภทตัวอักษรของอิลิเมนต์ที่ถูกต้อง การประกาศประเภทของอิลิเมนต์ใน DTD มีลักษณะเหมือนฐานข้อมูล และต้องตรงกับรูปแบบลักษณะโครงสร้างทั้งหมดของเอกสาร นั่นคือ DTD จะแสดงชนิดของอิลิเมนต์ที่อยู่ในเอกสาร คำสั่งของอิลิเมนต์และข้อมูลที่กำหนดในอิลิเมนต์

1) รูปแบบของการประกาศประเภทของอิลิเมนต์ (Element Type Declaration) ภายในเอกสาร XML จะประกอบด้วยอิลิเมนต์จำนวนมากตั้งแต่รูตอิลิเมนต์และอิลิเมนต์ต่างๆ ประกอบกันเป็นเอกสาร XML ซึ่งการประกาศอิลิเมนต์จึงถือว่าเป็นสิ่งสำคัญที่จะต้องกล่าวถึง

```
<!ELEMENT  ชื่ออิลิเมนต์  (เนื้อหาของอิลิเมนต์)>
```

รูป 2.21 รูปแบบการประกาศอิลิเมนต์

ชื่ออิลิเมนต์นั้นมีข้อกำหนดตามที่กล่าวไว้ในหัวข้อกฎพื้นฐานของภาษา XML (หัวข้อ 2.1.3) ส่วนเนื้อหาของ XML มีโอกาสเป็นไปได้คือ เนื้อหาของอิลิเมนต์ประกอบด้วย

อิลิเมนต์อื่นๆ, เนื้อหาอิลิเมนต์ประกอบด้วยข้อความปกติ และเนื้อหาของอิลิเมนต์ไม่มีอะไรอยู่เลย (เป็นแท็กว่าง) หรือในอิลิเมนต์จะประกอบด้วยอิลิเมนต์ปะปนอยู่กับข้อความ

```
<!ELEMENT ชื่ออิลิเมนต์ (อิลิเมนต1, อิลิเมนต์2, อิลิเมนต์3,..., อิลิเมนต์n)>
```

รูป 2.22 รูปแบบการประกาศอิลิเมนต์ที่ภายในประกอบด้วยอิลิเมนต์อื่นๆ

```
<PROFILE>
        <BIRTH_YR>1732</BIRTH_YR>
        <YRS_SERV>7</YRS_SERV>
        <DEATH_AGE>67</DEATH_AGE>
        <PARTY>Federalist</PARTY>
        <STATE_BORN>Virginia</STATE_BORN>
        <HOBBYLIST>
            <HOBBY>Fishing</HOBBY>
            <HOBBY>Riding</HOBBY>
        </HOBBYLIST>
</PROFILE>
```

รูป 2.23 ตัวอย่างอิลิเมนต์ที่ภายในประกอบด้วยอิลิเมนต์อื่นๆ

ตัวอย่างในรูป 2.23 แสดงให้เห็นว่าอิลิเมนต์ PROFILE จะประกอบด้วยอิลิเมนต์อื่นๆภายใน ได้แก่ BIRTH_YR, YRS_SERV, DEATH_AGE, PARTY, STATE_BORN, HOBBYLIST และHOBBY

```
<!ELEMENT ชื่ออิลิเมนต์ (#PCDATA)>
```

รูป 2.24 รูปแบบการประกาศอิลิเมนต์ที่ภายในประกอบด้วยข้อความ

ตัวอย่างในรูป 2.24 แสดงรูปแบบการประกาศอิลิเมนต์ที่ภายในประกอบด้วยข้อความธรรมดา โดยที่ PCDATA (Parsed Character Data) เป็นชนิดของข้อมูลโครงสร้างของเอกสาร XML แบบ DTD

```
<STATE_BORN>Virginia</STATE_BORN>
```

รูป 2.25 ตัวอย่างการประกาศอิลิเมนต์ที่ภายประกอบด้วยข้อความ

```
<!ELEMENT    ชื่ออิลิเมนต์    EMPTY>
```

รูป 2.26 รูปแบบการประกาศอิลิเมนต์ว่าง

```
<PRESIDENT PRES_NAME="Washington_G">
```

รูป 2.27 ตัวอย่างอิลิเมนต์ว่าง

ตัวอย่างในรูป 2.27 แสดงอิลิเมนต์ที่ภายในไม่มีข้อความ ไม่มีอิลิเมนต์ มีเพียงแอตทริบิวต์ อิลิเมนต์ลักษณะนี้เป็นอิลิเมนต์ว่าง

```
<!ELEMENT    ชื่ออิลิเมนต์    ANY>
```

รูป 2.28 รูปแบบการประกาศอิลิเมนต์ที่รองรับเนื้อหาทุกรูปแบบ

```
<!ELEMENT    HOBBY    ANY>
```

รูป 2.29 ตัวอย่างอิลิเมนต์ที่รองรับเนื้อหาทุกรูปแบบ

ตัวอย่างในรูป 2.29 แสดงการใช้คำหลักแอนนี่ (ANY Content) เมื่อต้องการให้อิลิเมนต์สามารถบรรจุข้อมูลได้ทุกชนิดนั่นคืออิลิเมนต์ชนิดนี้ สามารถบรรจุค่าว่างหรืออิลิเมนต์ลูกได้ในทุกคำสั่งหรือมีจำนวนซ้ำกัน และอาจไม่มีข้อมูลประเภทตัวอักษร (Character Data) การประกาศแบบนี้เป็นการประกาศที่หละหลวม และเป็นการสร้างชนิดของอิลิเมนต์ที่ไม่มีการบังคับชนิดของข้อมูล

```
<PROFILE>
                ประวัติของผู้ดำรงตำแหน่งประธานาธิบดี
                <BIRTH_YR>1732</BIRTH_YR>
<PROFILE>
```

รูป 2.30 ตัวอย่างอิลิเมนต์ที่ภายในอิลิเมนต์จะประกอบด้วยอิลิเมนต์ปะปนอยู่กับข้อความ

2) การกำหนดเนื้อหาของอิลิเมนต์ (Element Content) ถ้าอิลิเมนต์มีการกำหนดเนื้อหาของอิลิเมนต์ด้วยอิลิเมนต์ลูกจะไม่สามารถ บรรจุข้อมูลประเภทตัวอักษรได้ โครงสร้างของเนื้อหา (Content Model) มีได้ 2 รูปแบบได้แก่ แบบมีลำดับ (Sequence) และ แบบตัวเลือก (Choice)

```
<!ELEMENT MARRIAGE (SPOUSE_NAME, PR_AGE, SP_AGE)>
      <!ELEMENT SPOUSE_NAME (#PCDATA)>
      <!ELEMENT PR_AGE (#PCDATA)>
      <!ELEMENT SP_AGE (#PCDATA)>
```

รูป 2.31 ตัวอย่างการประกาศอิลิเมนต์ลูกแบบมีลำดับ

ตัวอย่างในรูป 2.31 แสดงรูปแบบที่มีลำดับของอิลิเมนต์ลูก ซึ่งสามารถแยกชื่อของอิลิเมนต์ด้วยเครื่องหมายจุลภาค (,) และไม่สามารถประกาศอิลิเมนต์ภายในเอกสาร XMLสลับตำแหน่งกันได้

```
<!ELEMENT MARRIAGE (SPOUSE_NAME| PR_AGE| SP_AGE)>
      <!ELEMENT SPOUSE_NAME (#PCDATA)>
      <!ELEMENT PR_AGE (#PCDATA)>
      <!ELEMENT SP_AGE (#PCDATA)>
```

รูป 2.32 ตัวอย่างการประกาศอิลิเมนต์ลูกแบบตัวเลือก

ตัวอย่างในรูป 2.32 แสดงรูปแบบของอิลิเมนต์ลูกที่สามารถเลือกได้เพียงหนึ่งอิลิเมนต์เท่านั้นตามที่ได้ประกาศไว้ ซึ่งแยกด้วยเครื่องหมายไปป์ (|)

นอกจากนี้สามารถปรับปรุงการกำหนดโครงสร้างของเนื้อหา โดยใช้เครื่องหมาย ? + และ * ซึ่งอธิบายความหมายตามตาราง 3.2

ตาราง 2.2 เครื่องหมายที่ช่วยในการปรับปรุงโครงสร้างของเนื้อหา

ตัวอักษร	ความหมาย
?	มีอิลิเมนต์ลูกได้ 1 ตัวหรือไม่มีเลย
+	มีอิลิเมนต์ลูกได้ 1 ตัวขึ้นไป
*	มีอิลิเมนต์ลูกกี่ตัวก็ได้หรือไม่มีเลย

```
<!ELEMENT MARRIAGE (SPOUSE_NAME| PR_AGE| SP_AGE)+>
```

รูป 2.33 ตัวอย่างอิลิเมนต์ที่ใช้เครื่องหมายพิเศษในการปรังปรุงโครงสร้างเนื้อหา

ตัวอย่างในรูป 2.33 สามารถอธิบายได้ดังนี้ เมื่ออิลิเมนต์ MARRIAGE มีอิลิเมนต์ลูกที่สามารถเลือกได้ตัวเดียว คือ SPOUSE_NAME หรือ PR_AGE หรือ SP_AGE แต่เมื่อเพิ่มเครื่องหมาย + ไว้ด้านหลังเป็นการกำหนดให้ต้องมีอิลิเมนต์ลูกอย่างน้อย 1 ตัวขึ้นไปและสามารถเลือกอิลิเมนต์ลูกตัวใดก็ได้มามาได้ที่ละตัว

2.3.1.3 การประกาศแอตทริบิวต์

การประกาศแอตทริบิวต์ มีความสำคัญไม่น้อยไปกว่าการประกาศอิลิเมนต์ เนื่องจากเอกสาร XML โดยทั่วไปจะประกอบไปด้วยข้อมูลที่ถูกเก็บอยู่ภายในอิลิเมนต์และข้อมูลที่บรรจุอยู่ภายในแอตทริบิวต์ ขึ้นอยู่กับผู้ออกแบบโครงสร้างของเอกสารจะเป็นผู้กำหนด การกำหนดแอตทริบิวต์จะอยู่ในส่วนของ DTD ซึ่งเรียกว่าการประกาศรายชื่อของแอตทริบิวต์ (Attribute-list Declaration) การประกาศค่าแอตทริบิวต์ทำได้ดังนี้

1) กำหนดชื่อของแอตทริบิวต์ร่วมกับอิลิเมนต์ในเอกสารที่ถูกต้องสมบูรณ์ โดยสามารถกำหนดได้เฉพาะในแท็กเริ่มต้นและแอตทริบิวต์เหล่านั้น เพื่อมานิยามอิลิเมนต์ในเอกสาร
2) กำหนดชนิดข้อมูลในแอตทริบิวต์แต่ละตัว
3) ข้อกำหนดรายละเอียดสำหรับแต่ละแอตทริบิวต์คือ ต้องมีการกำหนดค่าของแอตทริบิวต์ ถ้าไม่มีกำหนดการประกาศแอตทริบิวต์จะเป็นตัวชี้ตัวประมวลผล

```
<!ATTLIST ชื่ออิลิเมนต์ ชื่อแอตทริบิวต์ ชนิดข้อมูลของแอตทริบิวต์
(#REQUIRED|#IMPILIED|#FIXED ค่าปกติของแอตทริบิวต์)>
```

รูป 2.34 รูปแบบการประกาศแอตทริบิวต์

รูป 2.34 แสดงรูปแบบของแอตทริบิวต์ โดยชื่ออิลิเมนต์ หมายถึง ชื่อของอิลิเมนต์ที่ปรากฏอยู่ ชื่อแอตทริบิวต์หมายถึงชื่อของแอตทริบิวต์ที่ปรากฏ ชนิดข้อมูลของแอตทริบิวต์หมายถึงชนิดของข้อมูลที่กำหนด ซึ่ง W3C กำหนดไว้ได้แก่ #REQUIRED หมายถึงแอตทริบิวต์นั้นต้องมีการกำหนดค่าทุกครั้ง #IMPILE หมายถึงแอตทริบิวต์นั้นจะระบุค่าหรือไม่ก็ได้ #FIXED ค่าปกติของแอตทริบิวต์ หมายถึงเมื่อมีการกำหนดคำหลักนี้จะต้องระบุค่าปกติต่อท้ายเสมอ

```
<!ATTLIST PRESIDENT PRES_NAME ID #REQUIRED>
```

รูป 2.35 ตัวอย่างการประกาศแอตทริบิวต์

ตัวอย่างในรูป 2.36 แสดงการประกาศแอตทริบิวต์ โดยมี PRESIDENT เป็นชื่ออิลิเมนต์ PRES_NAME เป็นชื่อแอตทริบิวต์ ID เป็นชนิดข้อมูลของแอตทริบิวต์หมายความว่าค่าของแอตทริบิวต์ในแต่ละอิลิเมนต์ต้องไม่ซ้ำกัน #REQUIRED หมายถึงแอตทริบิวต์นั้นต้องมีการกำหนดค่าทุกครั้ง

2.3.2 การกำหนดโครงสร้างเอกสาร XML แบบ XML Schema

การกำหนดโครงสร้างของเอกสาร XML แบบ DTD ที่กล่าวมาแล้วนั้นเป็นรูปแบบของการกำหนดโครงสร้างของเอกสารของภาษา SGML ยังมีข้อด้อยที่ไม่เหมาะสมกับภาษา XML หลายประการเช่น มีชนิดของข้อมูลน้อย และมีรูปแบบการเขียนที่ไม่เหมือนกับภาษา XML เป็นต้น ทำให้ต้องใช้ตัวแปรภาษาคนละตัวกับภาษา XML ด้วยเหตุผลที่กล่าวมาข้างต้นทางสมาคม W3C จึงได้ออกมาตรฐานการกำหนดโครงสร้างของเอกสาร XML แบบ XML Schema มาตรฐานของ XML Schema แบ่งออกเป็น 3 ส่วนคือ

1) ข้อมูลเบื้องต้น (Primer)

2) โครงสร้าง (Structure)

3) ชนิดของข้อมูล (Data type)

2.3.2.1 ลักษณะของ XML Schema

เพื่อให้เห็นภาพลักษณ์ของ XML Schema ที่ชัดเจน จะอธิบายการกำหนดโครงสร้างแบบ DTD ควบคู่ไปกับ XML Schema

```
<!ELEMENT PRESIDENT (MARRIAGE*)>
      <!ELEMENT MARRIAGE (SPOUSE_NAME, SP_AGE)>
            <!ELEMENT SPOUSE_NAME (#PCDATA)>
            <!ELEMENT SP_AGE (#PCDATA)>
```

รูป 2.36 ตัวอย่างการกำหนดโครงสร้างแบบ DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema   xmlns:xsd = http://www.w3.org/2001/XMLSchema
                    targetName = http://presidential.com
                     xmlns = http://presidential.com
               elementDefault = “qualified”>
      <xsd:element name = “PRESIDENT”>
           <xsd:complexType>
                <xsd:sequence>
            <xsd:element ref = “MARRIAGE”
                         minOccurs = “1” maxOccurs =  “unbounded”/>
                </xsd:sequence>
           </xsd:complexType>
      </xsd:element>
      <xsd:element name = “MARRIAGE”>
           <xsd:complexType>
                <xsd:sequence>
            <xsd:element ref = “SPOUSE_NAME”
                         minOccurs = “1” maxOccurs =  “unbounded”/>
            <xsd:element ref = “SP_AGE”
                         minOccurs = “1” maxOccurs =  “unbounded”/>
                </xsd:sequence>
           </xsd:complexType>
      </xsd:element>
      <xsd:element name = “SPOUSE_NAME” type = “xsd:string”>
      <xsd:element name = “SP_AGE” type = “xsd:string”>
</xsd:schema>
```

รูป 2.37 ตัวอย่างการกำหนดโครงสร้างแบบ XML Schema

เมื่อเปรียบเทียบการกำหนดโครงสร้างของเอกสารแบบ DTD (รูป 2.36) กับการกำหนดโครงสร้างแบบ XML Schema (รูป 2.37) แล้วจะเห็นได้ว่าการกำหนดโครงสร้างแบบ XML Schema นั้นมีความซับซ้อนกว่าแบบ DTD พอสมควร

2.3.2.2 ประเภทอิลิเมนต์ของโครงสร้างแบบ XML Schema

การกำหนดโครงสร้างของเอกสาร XML แบบ XML Schema ที่แสดงในรูป 2.37 จะประกอบด้วยการกำหนดอิลิเมนต์ต่างๆจำนวนมาก อิลิเมนต์ที่กล่าวถึงใน XML Schema แบ่งออกเป็น 2 ประเภทดังนี้

1) อิลิเมนต์แบบธรรมดา (Simple Type) คืออิลิเมนต์ที่มีข้อมูลภายในเป็นสตริง หรือตัวเลข หรือข้อมูลพื้นฐานอื่นๆ ชนิดข้อมูลของอิลิเมนต์แบบธรรมดาถูกกำหนดในมาตรฐาน XML Schema

```
<xsd:element name = "ชื่ออิลิเมนต์" type = "ชื่อชนิดข้อมูล" >
```

รูป 2.38 รูปแบบการประกาศอิลิเมนต์แบบธรรมดา

```
<xsd:element name = "SPOUSE_NAME" type = "xsd:string">
```

รูป 2.39 ตัวอย่างการประกาศอิลิเมนต์แบบธรรมดา

2) อิลิเมนต์แบบซับซ้อน (Complex Type) คืออิลิเมนต์ที่มีข้อมูล ภายในเป็นอิลิเมนต์ หรืออิลิเมนต์ที่มีแอตทริบิวต์ลูกอยู่ภายใน ดังนั้นการประกาศอิลิเมนต์แบบซับซ้อนก็คือการแจกแจงว่ามีแอตทริบิวต์อะไร และมีอิลิเมนต์ลูกอะไรอยู่ภายในนั่นเอง

```
<xsd:element name = "ชื่ออิลิเมนต์">
      <xsd:complexType>
            [<xsd:sequence>,<xsd:choice>]
           < element 1>
           < element 2>
           < element 3>
           .
           .
           .
           < element n>
              [</xsd:sequence>,</xsd:choice>]
      </xsd:complexType>
</xsd:element>
```

รูป 2.40 รูปแบบการประกาศอิลิเมนต์แบบซับซ้อน

```
<xsd:element name = "MARRIAGE">
       <xsd:complexType>
              <xsd:sequence>
            <xsd:element ref = "SPOUSE_NAME" minOccurs = "1"
            maxOccurs =  "unbounded"/>
            <xsd:element ref = "SP_AGE" minOccurs = "1"
            maxOccurs =  "unbounded"/>
              </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

รูป 2.41 ตัวอย่างการประกาศอิลิเมนต์แบบซับซ้อน

คีย์เวิร์ด “<xsd:sequence>” xsd (XML Schema Definition) หมายถึงการบ่งบอกว่าเป็นเอกสาร XML Schema และไฟล์ข้อมูลของ XML Schema มีนามสกุล (.xsd) ส่วน sequence หมายถึง การเรียงลำดับของอิลิเมนต์ลูกที่อยู่ภายในอิลิเมนต์ที่ซับซ้อน และอิลิเมนต์ที่ปรากฏในเอกสาร XML ที่ต้องอ้างถึงต้องเรียงลำดับตามกำหนด และคีย์เวิร์ด “<xsd:choice>” หมายถึงการระบุว่าจะมีอิลิเมนต์ลูกตัวใดตัวหนึ่งเท่านั้นที่ปรากฏอยู่ในเอกสาร XML ที่อ้างถึง

การกำหนดโครงสร้างของเอกสาร XML แบบ XML Schema สามารถกำหนดจำนวนอิลิเมนต์ที่สามารถได้ในเอกสาร XML ด้วยการใช้แอตทริบิวต์ “minOccurs” เป็นแอตทริบิวต์ที่ระบุจำนวนต่ำสุดที่จำนวนอิลิเมนต์ในเอกสาร XML สามารถปรากฏได้ ค่าที่ระบุส่วนมากได้แก่ 0 กับ 1 และแอตทริบิวต์ “maxOccurs” เป็นแอตทริบิวต์ที่ระบุจำนวนสูงสุดที่จำนวนอิลิเมนต์ XML สามารถปรากฏได้ ค่าที่ระบุสามารถระบุได้หลายค่าตั้งแต่ 1 จนถึง “unbounded” การระบุค่าสูงสุดเป็น “unbounded” หมายความว่าสามารถระบุค่าของอิลิเมนต์นั้นได้โดยไม่จำกัดจำนวน ในกรณีที่ไม่ได้ระบุแอตทริบิวต์ “minOccurs” และ “maxOccurs” ค่าปกติของของจำนวนอิลิเมนต์คือ 1 หมายความว่าจำนวนอิลิเมนต์ที่ปรากฏในเอกสาร XML ได้ต่ำสุด 1 ครั้ง และมากสุด 1 ครั้ง

2.3.2.3 ประเภทแอตทริบิวต์ของโครงสร้างแบบ XML Schema

รูปแบบการประกาศแอตทริบิวต์ ของการกำหนดโครงสร้างของเอกสาร XML แบบ XML Schema มีรูปแบบที่เหมือนกับ รูปแบบการประกาศอิลิเมนต์แบบธรรมดาเพราะแอตทริบิวต์จัดอยู่ในอิลิเมนต์แบบธรรมดา

```
<xsd:attribute name = “ชื่อแอตทริบิวต์” type = “ชื่อชนิดข้อมูล” [use default fixed ค่าปกติ]/>
```

รูป 2.42 รูปแบบการประกาศแอตทริบิวต์

จากรูป 2.42 ชื่อแอตทริบิวต์ หมายถึงชื่อของแอตทริบิวต์ ชื่อชนิดข้อมูล หมายถึงชื่อชนิดของข้อมูล เช่น ข้อความ ตัวเลข และวันที่ เป็นต้น นอกจากนี้ยังมีแอตทริบิวต์ทางเลือกให้สามารถเลือกใช้ ได้แก่ แอตทริบิวต์ “use” หมายถึงการระบุระดับความจำเป็นของแอตทริบิวต์นี้ว่าต้องมีหรือไม่ ค่าของแอตทริบิวต์นี้มี 3 ค่า คือ จำเป็นต้องมี (reqired), มีหรือไม่มีก็ได้ (option), และไม่ต้องมี (prohibited) หรือแอตทริบิวต์ “default” ใช้กำหนดค่าปกติของแอตทริบิวต์ หรือแอตทริบิวต์ “fixed” ใช้กำหนดค่าปกติให้แก่แอตทริบิวต์ ทั้งนี้แอตทริบิวต์ทั้ง 3 ที่กล่าวมานั้นจะใช้หรือไม่ใช้ก็ได้เนื่องจากเป็นแอตทริบิวต์ทางเลือก

```
<xsd:element name = “SPOUSE_NAME” type = “xsd:string”>
```

รูป 2.43 ตัวอย่างการประกาศแอตทริบิวต์

2.3.3 ความแตกต่างระหว่าง รูปแบบการกำหนดโครงสร้าง เอกสาร XML แบบ DTD กับ รูปแบบการกำหนดโครงสร้างเอกสาร XML แบบ XML Schema

1) การกำหนดโครงสร้างเอกสาร XML แบบ XML Schema มีรูปแบบการเขียนเหมือนเอกสาร XML ให้ใช้ตัวแปลภาษาตัวเดียวกันได้แต่รูปแบบการกำหนดโครงสร้างเอกสาร XML แบบ DTD มีรูปแบบการเขียนที่ต่างจากเอกสาร XML ทำให้ต้องมีตัวแปลภาษาสำหรับ DTD แยกต่างหากทำให้เกิดความสิ้นเปลืองในการใช้งาน
2) การกำหนดโครงสร้างเอกสาร XML แบบ XML Schema สนับสนุนการใช้เนมสเปสให้สามารถใช้แท็กที่มีชื่อเหมือนกันในเอกสารเดียวกัน แต่มีความหมายแตกต่างกัน สำหรับรูปแบบการกำหนดโครงสร้างเอกสาร XML แบบ DTD นั้นไม่สามารถทำได้
3) การกำหนดโครงสร้างเอกสาร XML แบบ XML Schema มีชนิดของข้อมูลมากกว่ารูปแบบการกำหนดโครงสร้าง เอกสาร XML แบบ DTD และชื่อชนิดของข้อมูลมีชื่อเรียกที่ถูกใช้โดยทั่วไปในการเขียนโปรแกรม สำหรับรูปแบบการกำหนดโครงสร้าง XML แบบ DTD มีชื่อชนิดข้อมูลที่ไม่คุ้นเคย เช่น PCDATA, NMTOKEN เป็นต้น
4) การกำหนดโครงสร้างเอกสาร XML แบบ XML Schema สามารถกำหนดลำดับของอิลิเมนต์ที่ปรากฏในเอกสาร XML ได้
5) การกำหนดโครงสร้างเอกสาร XML แบบ XML Schema สามารถกำหนดรูปแบบชนิดของข้อมูลตามที่ผู้ใช้ต้องการได้ เช่น เบอร์โทรศัพท์กำหนดให้มีรูปแบบ “xx-xxx-xxxx” หมายถึงรูปแบบของเบอร์โทรศัพท์ที่มีตัวเลขข้างหน้า 2 หลักตามด้วยขีดและตัวเลขอีก 3 หลักตามด้วยขีด และตัวเลขอีก 4 หลักเป็นต้น

2.4 เอ็กซ์พาธ (XPath)

เอ็กซ์พาธคือภาษาที่ใช้ค้นหาข้อมูลภายในเอกสาร XML โดยใช้เอ็กซ์พาธเพื่อระบุอิลิเมนต์หรือแอตทริบิวต์ในเอกสาร XML ที่ต้องเดินทางไป XPath ใช้รูปแบบการเข้าถึงข้อมูล (Path expression) ในการเลือกโหนดหรือเซตของโหนดในเอกสาร XML ซึ่งรูปแบบการเข้าถึงข้อมูลเหล่านี้มีรูปแบบการใช้เหมือนกับรูปแบบการเข้าถึงข้อมูลในระบบไฟล์ของคอมพิวเตอร์ นอกจากนี้ XPath ยังมีฟังก์ชันสำเร็จรูป (Build-in function) ให้เลือกใช้กว่า 100 ฟังก์ชัน ได้แก่ ฟังก์ชันเกี่ยวกับตัวอักษร ตัวเลข การเปรียบเทียบวันเวลา เป็นต้น นอกจากนี้ XPath ยังเป็นมาตรฐานของ W3C อีกด้วย

2.4.1 คำศัพท์ต่างๆ ใน XPath

1) โหนด (Node)ใน XPath มีโหนดทั้งหมด 7 ชนิด คือ อิลิเมนต์, แอตทริบิวต์, ข้อความ, เนมสเปส, คำสั่งในการดำเนินการ, ส่วนประมวลผล, ข้อความอธิบาย และรูตโหนด

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PRESIDENTIAL_DATABASE SYSTEM
            “http://presidential.com/ presidential.dtd”>
<PRESIDENTIAL_DATABASE>
     <PRESIDENT PRES_NAME="Washington_G">
          <PROFILE>
               <BIRTH_YR>1732</BIRTH_YR>
               <YRS_SERV>7</YRS_SERV>
               <DEATH_AGE>67</DEATH_AGE>
               <PARTY>Federalist</PARTY>
               <STATE_BORN>Virginia</STATE_BORN>
          </PROFILE>
     </PRESIDENT>
     <PRESIDENT PRES_NAME="Adams_J">
          <PROFILE>
               <BIRTH_YR>1735</BIRTH_YR>
               <YRS_SERV>4</YRS_SERV>
               <DEATH_AGE>90</DEATH_AGE>
               <PARTY>Federalist</PARTY>
               <STATE_BORN>Massachusetts</STATE_BORN>
          </PROFILE>
     </PRESIDENT>
     <PRESIDENT PRES_NAME="Jefferson_T">
          <PROFILE>
               <BIRTH_YR>1743</BIRTH_YR>
               <YRS_SERV>8</YRS_SERV>
               <DEATH_AGE>83</DEATH_AGE>
               <PARTY>Demo-Rep</PARTY>
               <STATE_BORN>Virginia</STATE_BORN>
          </PROFILE>
     </PRESIDENT>
</PRESIDENTIAL_DATABASE>
```

รูป 2.44 เอกสาร XML ที่ใช้ประกอบการอธิบายเรื่อง XPath

```
<PRESIDENTIAL_DATABASE> ----> Document nodeหรือRoot element node

<STATE_BORN>Virginia</STATE_BORN> ----> element node

PRES_NAME="Washington_G" ----> attribute node
```

รูป 2.45 ตัวอย่างชนิดของโหนด

2) ค่าอะตอมมิก (Atomic Value) ค่าอะตอมมิก คือโหนดที่ไม่มีลูกหรือพ่อแม่

```
Virginia

"Washington_G"
```

รูป 2.46 ตัวอย่างของค่าอะตอมมิก

3) ไอเท็ม (Item) ไอเท็ม คือ ค่าอะตอมมิก หรือ โหนด

2.4.2 ความสัมพันธ์ระหว่างโหนด

1) พ่อแม่ (Parent) อิลิเมนต์และแอตทริบิวต์แต่ละตัวมีพ่อแม่ได้แค่ตัวเดียว

2) ลูก (Children) อิลิเมนต์โหนดหนึ่งๆ อาจมีลูกได้ตั้งแต่ศูนย์ตัวขึ้นไป

3) พี่น้อง (Sibling) โหนดที่มีพ่อแม่เดียวกัน

4) บรรพบุรุษ (Ancestor) โหนดที่เป็นพ่อแม่ของโหนดนั้นๆ รวมไปถึงปู่ย่าตายายขึ้นๆ ไป

5) ผู้สืบสกุล (Descendent) โหนดที่เป็นลูกๆของโหนดนั้นๆ รวมไปถึงลูกหลานขึ้นๆ ไป

```
<PRESIDENT PRES_NAME="Washington_G">
        <PROFILE>

            <BIRTH_YR>1732</BIRTH_YR>
            <YRS_SERV>7</YRS_SERV>
            <DEATH_AGE>67</DEATH_AGE>
            <PARTY>Federalist</PARTY>
            <STATE_BORN>Virginia</STATE_BORN>

        </PROFILE>
    </PRESIDENT>
```

รูป 2.47 ตัวอย่างความสัมพันธ์แบบพ่อแม่และลูก

ตัวอย่างในรูป 3.55 อิลิเมนต์ “PROFILE” เป็นพ่อแม่ของอิลิเมนต์ “BIRTH_YR”, “YRS_SERV”, “DEATH_AGE”, “PARTY”, “STATE_BORN” และ อิลิเมนต์ “BIRTH_YR”, “YRS_SERV”, “DEATH_AGE”, “PARTY”, “STATE_BORN” เป็นพี่น้องกันและมีบรรพบุรุษคือ อิลิเมนต์ “PROFILE”, “PRESIDENT” ในอีกทางโหนดที่เป็นผู้สืบสกุลของอิลิเมนต์ “PRESIDENT” คืออิลิเมนต์ “PROFILE”, “BIRTH_YR”, “YRS_SERV”, “DEATH_AGE”, “PARTY”, “STATE_BORN”

2.4.3 การเลือกโหนด

เอ็กซ์พาธใช้รูปแบบการเข้าถึงข้อมูล ในการเลือกโหนดหรือเซตของโหนดในเอกสาร XML โดยการเลือกโหนดจะเลือกตั้งแต่รูตโหนดเป็นลำดับลงไป

ตาราง 2.3 รูปแบบการเข้าถึงข้อมูล ที่สำคัญ

Expression	คำอธิบาย
ชื่อโหนด	เลือกโหนดลูกทั้งหมดที่มีชื่อตามที่ระบุ
/	เลือกโหนดจากรูตโหนด
//	เลือกโหนดในเอกสารจากโหนดปัจจุบันที่ตรงกับที่ระบุ โดยไม่สนใจตำแหน่งที่อยู่ของโหนดนั้น
.	เลือกโหนดปัจจุบัน
..	เลือกพ่อแม่ของโหนดปัจจุบัน
@	เลือกแอตทริบิวต์

ตาราง 2.4 ตัวอย่างของการเลือกโหนด

Path Expression	คำอธิบาย
PRESIDENTIAL_DATABASE	เลือกโหนดลูกทั้งหมดของอิลิเมนต์ PRESIDENTIAL_DATABASE
/ PRESIDENTIAL_DATABASE	เลือกโหนดรูตโหนดที่ชื่อว่า PRESIDENTIAL_DATABASE
PRESIDENTIAL_DATABASE /PRESIDENT	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT ทั้งหมดที่เป็นลูกของ PRESIDENTIAL_DATABASE
//PRESIDENT	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT ทั้งหมดโดยไม่สนใจว่าจะมีอิลิเมนต์นี้ปรากฏอยู่ที่ใดของเอกสาร
PRESIDENTIAL_DATABASE //PRESIDENT	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT ทั้งหมดที่เป็นผู้สืบสกุลของอิลิเมนต์ที่ชื่อ PRESIDENTIAL_DATABASE โดยไม่สนใจว่าจะมี

ตาราง 2.4 ตัวอย่างของการเลือกโหนด(ต่อ)

	อิลิเมนต์ PRESIDENT ปรากฏอยู่ที่ใดของเอกสาร
//@ PRES_NAME	เลือกแอตทริบิวต์ PRES_NAME ทั้งหมด

2.4.3.1 ภาคแสดง (Predicate)

ภาคแสดงใช้เพื่อหาโหนดที่ระบุเฉพาะเจาะจงลงไป โดยภาคแสดงจะระบุอยู่ในวงเล็บก้ามปูเสมอ

ตาราง 2.5 ตัวอย่างรูปแบบการเข้าถึงข้อมูล ที่มีภาคแสดงเป็นส่วนประกอบ

Path Expression	ผลที่ได้
PRESIDENTIAL_DATABASE /PRESIDENT[0]	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT ตัวแรกที่เป็นลูกของ PRESIDENTIAL_DATABASE
PRESIDENTIAL_DATABASE /PRESIDENT[last()]	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT ตัวสุดท้ายที่เป็นลูกของ PRESIDENTIAL_DATABASE
PRESIDENTIAL_DATABASE /PRESIDENT[position()<3]	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT สองตัวแรกที่เป็นลูกของ PRESIDENTIAL_DATABASE
//PRESIDENT[@ PRES_NAME]	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT ทั้งหมด ที่มีแอตทริบิวต์ชื่อ PRES_NAME
//PRESIDENT[@ PRES_NAME= "Washington_G"]	เลือกอิลิเมนต์ที่ชื่อ PRESIDENT ทั้งหมด ที่มีแอตทริบิวต์ชื่อ PRES_NAME ที่มีค่า"Washington_G"
PRESIDENTIAL_DATABASE / PROFILE [BIRTH_YR >1730]/PARTY	เลือกอิลิเมนต์ที่ชื่อ PARTY ทั้งหมดที่เป็นลูกของ PROFILE ซึ่งเป็นลูกของ PRESIDENTIAL_DATABASE และมีค่าของอิลิเมนต์ BIRTH_YR มากกว่า 1730

2.4.3.2 การเลือกโหนดที่ไม่รู้จัก

ตาราง 2.6 การเลือกโหนดที่ไม่รู้จักโดยใช้ สัญลักษณ์แทนความหมายเอ็กซ์พาธ (XPath wildcard)

wildcard	คำอธิบาย
*	ใช้เลือกโหนดอิลิเมนต์ใดๆ
@*	ใช้เลือกโหนดแอตทริบิวต์ใดๆ
Node ()	ใช้เลือกโหนดชนิดใดๆก็ได้

ตาราง 2.7 ตัวอย่างการเลือกโหนดที่ไม่รู้จักโดยใช้ สัญลักษณ์แทนความหมายเอ็กซ์พาธ

Path Expression	ผลที่ได้
/ PROFILE /*	เลือกอิลิเมนต์ทั้งหมดที่เป็นลูกของอิลิเมนต์ที่ชื่อ PROFILE
//*	เลือกแอตทริบิวต์ทั้งหมดในเอกสาร
//PARTY[@*]	เลือกอิลิเมนต์ที่ชื่อ PARTY ทั้งหมดที่มีแอตทริบิวต์ค่าใดๆ

2.4.3.3 การเลือกหลายเส้นทาง

หากต้องการเลือกหลายเส้นทางในรูปแบบการเข้าถึงข้อมูล สามารถใช้สัญลักษณ์ "|" เป็นตัวคั่นแต่ละเส้นทาง

ตาราง 2.8 ตัวอย่างการเลือกโหนดที่ไม่รู้จักโดยใช้ สัญลักษณ์แทนความหมายเอ็กซ์พาธ

Path Expression	ผลที่ได้
// PROFILE /PARTY|// PROFILE / BIRTH_YR	เลือกอิลิเมนต์ที่ชื่อ PARTY และ BIRTH_YR ทั้งหมดที่เป็นลูกของ PROFILE
// PARTY | // BIRTH_YR	เลือกอิลิเมนต์ที่ชื่อ PARTY และ BIRTH_YR ทั้งหมดทั้งหมดในเอกสาร
/PRESIDENT/PROFILE/PARTY | // BIRTH_YR	เลือกอิลิเมนต์ที่ชื่อ PARTY ทั้งหมดที่เป็นลูกของอิลิเมนต์ PROFILE ซึ่งเป็นลูกของอิลิเมนต์ PRESIDENT และเลือกอิลิเมนต์ที่ชื่อ BIRTH_YR ทั้งหมดในเอกสาร

2.4.4 แกนของเอ็กซ์พาธ

แกนของเอ็กซ์พาธใช้กำหนดความสัมพันธ์ของเซตของโหนดกับโหนดปัจจุบัน

ตาราง 2.9 แกนของเอ็กซ์พาธ

ชื่อแกน	คำอธิบาย
Ancestor	เลือกบรรพบุรุษทั้งหมดของโหนดปัจจุบัน
Ancestor-or-self	เลือกบรรพบุรุษทั้งหมดของโหนดปัจจุบันและเลือกโหนดปัจจุบันด้วย
Attribute	เลือกแอตทริบิวต์ทั้งหมดของโหนดปัจจุบัน
Child	เลือกลูกทั้งหมดของโหนดปัจจุบัน

ตาราง 2.9 แกนของเอ็กซ์พาธ (ต่อ)

Descendant	เลือกผู้สืบสกุลทั้งหมดของโหนดปัจจุบัน
Descendant-or-self	เลือกผู้สืบสกุลทั้งหมดของโหนดปัจจุบันและเลือกโหนดปัจจุบันด้วย
Following	เลือกทุกอย่างภายในเอกสารที่อยู่หลังแท็กปิดของโหนดปัจจุบัน
Following-sibling	เลือกพี่น้องทั้งหมดที่อยู่หลังแท็กปิดของโหนดปัจจุบัน
Namespace	เลือกแนมสเปสโหนดทั้งหมดของโหนดปัจจุบัน
Parent	เลือกพ่อแม่ของโหนดปัจจุบัน
Preceding	เลือกทุกอย่างภายในเอกสารที่อยู่ก่อนแท็กเปิดของโหนดปัจจุบัน
Preceding-sibling	เลือกพี่น้องทั้งหมดที่อยู่ก่อนหน้าแท็กเปิดของโหนดปัจจุบัน

2.4.5 ที่อยู่ของรูปแบบการเข้าถึงข้อมูล

ที่อยู่ของทางเดินสามารถเป็นได้ 2 แบบ คือ สัมบูรณ์ (Absolute) หรือ สัมพัทธ์ (Relative) โดยที่อยู่ของทางเดินแบบสัมบูรณ์จะเริ่มต้นด้วยเครื่องหมาย "/" แต่ที่อยู่ของทางเดินแบบสัมพัทธ์จะไม่เริ่มต้นด้วยเครื่องหมาย "/" และทั้ง 2 กรณีนี้จะประกอบด้วยขั้นตอนตั้งแต่หนึ่งขั้นตอนขึ้นไป แต่ละขั้นตอนจะหาค่ากับโหนดต่างๆ ในเซตของโหนดปัจจุบัน ซึ่งขั้นตอนมีดังนี้

1) แกน คือ กำหนดความสัมพันธ์ของต้นไม้ระหว่างโหนดที่เลือกกับโหนกปัจจุบัน
2) โหนดทดสอบ (Node test) คือการระบุโหนดข้างในแกน
3) ภาคแสดง ตั้งแต่ศูนย์ตัวขึ้นไป คือการระบุเซตของโหนดที่เลือกมาแล้วให้เฉพาะเจาะจงลงไป

```
ชื่อแกน::nodetest[predicate]
```

รูป 2.48 รูปแบบของที่อยู่ทางเดิน

ตาราง 2.10 ตัวอย่างที่อยู่ทางเดิน

ชื่อแกน	คำอธิบาย
Child::PROFILE	เลือกโหนดที่ชื่อว่า PROFILE ทั้งหมดที่เป็นลูกของโหนด

ตาราง 2.10 ตัวอย่างที่อยู่ทางเดิน(ต่อ)

Attribute::PRES_NAME	เลือกแอตทริบิวต์ที่ชื่อว่า PROFILE ทั้งหมดทของโหนดปัจจุบัน
Child::*	เลือกลูกทั้งหมดของโหนดปัจจุบัน
Attribute::*	เลือกแอตทริบิวต์ทั้งหมดของโหนดปัจจุบัน
Child::text()	เลือกโหนดลูกที่เป็นข้อความของโหนดปัจจุบันทั้งหมด
Child::node()	เลือกโหนดลูกที่ของโหนดปัจจุบันทั้งหมด
Descendant:: PROFILE	เลือกอิลิเมนต์ PROFILE ที่เป็นผู้สืบสกุลของโหนดปัจจุบัน
Ancestor:: PROFILE	เลือกอิลิเมนต์ PROFILE ที่เป็นบรรพบุรุษของโหนดปัจจุบัน
Ancestor-or-self:: PROFILE	เลือกอิลิเมนต์ PROFILE ที่เป็นบรรพบุรุษของโหนดปัจจุบันทั้งหมดและถ้าโหนดปัจจุบันเป็น PROFILE เลือกโหนดปัจจุบันด้วย
Child::*/ Child::PROFILE	เลือก PROFILE ที่เป็นของหลานของโหนดปัจจุบันทั้งหมด

2.4.6 ตัวดำเนินการเอ็กซ์พาธ (XPath Operator)

ตาราง 2.11 ตัวดำเนินการเอ็กซ์พาธ

ตัวดำเนินการ	คำอธิบาย	ตัวอย่าง	ค่าที่ Return
|	คำนวณ 2 เซตของโหนด	//PARTY|// BIRTH_YR	เซตของโหนดที่เป็นอิลิเมนต์ที่ชื่อ PARTY และ BIRTH_YR ทั้งหมด
+	การบวก	8 + 2	10
-	การลบ	8 - 2	6
*	การคูณ	8 * 2	16
Div	การหาร	8 div 2	4
=	เท่ากับ	BIRTH_YR =1750	จริง ถ้า BIRTH_YR เท่ากับ 1750
!=	ไม่เท่ากับ	BIRTH_YR != 1750	จริง ถ้า BIRTH_YR ไม่เท่ากับ 1750
<	น้อยกว่า	BIRTH_YR < 1750	จริง ถ้า BIRTH_YR น้อยกว่า 1750
<=	น้อยกว่าหรือเท่ากับ	BIRTH_YR <=1750	จริง ถ้า BIRTH_YR น้อยกว่าเท่ากับ 1750
>	มากกว่า	BIRTH_YR > 1750	จริง ถ้า BIRTH_YR มากกว่า 1750

ตาราง 2.11 ตัวดำเนินการเอ็กซ์พาธ(ต่อ)

>=	มากกว่าหรือเท่ากับ	BIRTH_YR >= 1750	จริง ถ้า BIRTH_YR มากกว่าเท่ากับ 1750
Or	หรือ	BIRTH_YR > 1700 or BIRTH_YR < 1800	จริง ถ้าจริง ถ้า BIRTH_YR มากกว่า 1700 หรือน้อยกว่า 1800
And	และ	BIRTH_YR > 1700 and BIRTH_YR < 1800	จริง ถ้าจริง ถ้า BIRTH_YR มากกว่า 1700 และน้อยกว่า 1800
Mod	หารเอาเศษ	5 mod 2	1

2.5 XQuery

XQuery คือ ภาษาที่ใช้ค้นหาข้อมูลภายในเอกสาร XML ถูกคิดค้นและพัฒนาโดยองค์กร W3C เพื่อให้เป็นมาตรฐานสำหรับการค้นหาข้อมูลภายในเอกสาร XML โดย XQuery เป็นภาษาสำเร็จรูป (Functional Language) ที่ผู้ใช้สามารถอ่านและเข้าใจ (Human-readable) และผลลัพธ์จะยังคงอยู่ในรูปแบบตามโครงสร้างของเอกสาร XML ด้วยเช่นกัน สำหรับการเขียนชุดคำสั่งหนึ่งๆ ในการค้นหาข้อมูลนั้นจะอยู่ในรูปของนิพจน์ซึ่งนิพจน์ที่เป็นส่วนประกอบใน XQuery มีดังนี้ รูปแบบการเข้าถึงข้อมูล, ไวยากรณ์ FLWOR, การใช้ตัวดำเนินการและฟังก์ชั่นสำเร็จรูป, การใช้เงื่อนไขในการแสดงผลลัพธ์(Condition Expression) และการใช้เงื่อนไขในการจำกัดจำนวนข้อมูล(Quantified Expression) พร้อมกันนี้ XQuery ยังมีความสามารถในการค้นหาข้อมูลจากเอกสาร XML มากกว่า 1 เอกสารในการค้นหาแต่ละครั้งโดยที่เอกสาร XML เหล่านั้นไม่จำเป็นต้องถูกเก็บไว้ในที่เก็บข้อมูลที่เดียวกัน และความสามารถในการสร้างฟังก์ชั่นใช้งานเองที่ได้ที่นอกเหนือจากฟังก์ชั่นสำเร็จรูป

2.5.1 รูปแบบการเข้าถึงข้อมูล

XQuery มีวิธีการชี้ตำแหน่งของส่วนต่างๆในเอสาร XML โดยอิงตามหลักไวยากรณ์ของ Xpath ซึ่งยังคงมองเอกสาร XML ในลักษณะของลำดับชั้นต้นไม้ดังตัวอย่างต่อไปนี้

```
/BookStore/Book/Author
```

คำอธิบาย : เข้าถึงข้อมูลของ Author Element ทั้งหมดที่อยู่ภายใต้ Book Element และ Book Element ต้องอยู่ภายใต้ BookStore Element

รูป 2.49 ตัวอย่างการเขียนนิพจน์สำหรับการเข้าถึงข้อมูล Author

2.5.2 ไวยากรณ์ FLWOR

ไวยากรณ์ที่ใช้ในการค้นหาข้อมูลของ Xquery นั้นพยายามลอกเลียนแบบการทำงานมาจากภาษา SQL ให้มากที่สุด เพื่อให้สามารถเรียนรู้ได้ง่าย ไวยากรณ์ที่กล่าวมานี้เรียกว่า FLWOR ซึ่งย่อมาจาก for, let, where, order by และ return สำหรับการค้นหาข้อมูลในเอกสาร XML นั้น for และ/หรือ let จะใช้ในการกำหนดเอกสาร XML ที่ต้องการค้นหาข้อมูล และเป็นการชี้ตำแหน่งเริ่มต้น where จะใช้สำหรับการกรองข้อมูลที่ต้องการค้นหา order by จะใช้สำหรับกำหนดการเรียงลำดับของข้อมูลผลลัพธ์ และ return จะใช้ในการกำหนดรูปแบบของผลลัพธ์ตามโครงสร้างของ XML ใน XQuery นั้นให้ความสำคัญกับตัวอักษรตัวเล็กในทุกทุกคำสั่งของ XQuery

2.5.2.1 for และ let

for ใช้สำหรับกำหนดตัวแปร และ/หรือชุดของตัวแปร ซึ่งในแต่ละชุดคำสั่งของ for นั้นจะคืนค่าออกมาเป็นกลุ่มของโหนด และค่าผลลัพธ์ในแต่ละโหนด (รวมถึงโหนดลูกหลานของโหนดดังกล่าวด้วย) ที่ได้มาก็จะถูกเก็บไว้ในตัวแปรที่กำหนด และมีการวนซ้ำในการคืนค่าของผลลัพธ์ด้วย

let ใช้สำหรับกำหนดตัวแปร และ/หรือชุดของตัวแปรเหมือนกับ for พร้อมกับการคืนค่าข้อมูลก็เหมือนกับ for ด้วยเช่นกัน ซึ่งถ้ามองดูแบบผิวเผินแล้วการทำงานของ letนั้นจะคล้ายกับการทำงานของ for แต่ในความเป็นจริงแล้วทั้งสองยังคงมีสิ่งที่แตกต่างกันอยู่คือ let ไม่มีการวนซ้ำในการคืนค่าผลลัพธ์

ถึงแม้ว่าการทำงานของ for และ let จะคล้ายคลึงกันมาก แต่อย่างไรแล้วทั้งสองยังคงมีความแตกต่างกันในเรื่องของการคืนค่าผลลัพธ์ เพื่อให้เข้าใจถึงความแตกต่างของ for และ let จะแสดงในตัวอย่างต่อไปนี้

```
for $s in (<one/>,<two/>,<three/>)

return <out> {$s} </out>

ผลลัพธ์ที่ได้คือ
<out>
      <one/>
</out>

<out>
       <two/>
</out>

<out>
       <three/>
</out>
```

รูป 2.50 ตัวอย่างการใช้ for และการคืนค่าผลลัพธ์

```
let $s :=(<one/>,<two/>,<three/>)

return <out> {$s} </out>

ผลลัพธ์ที่ได้คือ
<out>

      <one/>

      <two/>

      <three/>

</out>
```

รูป 2.51 ตัวอย่างการใช้ let และการคืนค่าผลลัพธ์

2.5.2.2 where

where ใช้สำหรับกำหนดเงื่อนไขการกรองข้อมูลที่ต้องการจากที่ได้กำหนดไว้ในตัวแปร(จากการใช้ for และ/หรือ let) สำหรับการคืนค่าของการใช้ where นี้จะเป็นในลักษณะของบูลีน การทำงานในส่วนนี้เปรียบเทียบได้กับการใช้ where ใน SQL นั่นเอง

```
For $b in doc("books.xml")   //book

where $b/pubinfo/publisher="Harper and Row"

Return $b/title

ผลลัพธ์ที่ได้คือ

<title>Harold and the Purple Crayon</title>

<title>Harold's Fairy Tale</title>
```

รูป 2.52 ตัวอย่างที่ค้นหาชื่อหนังสือจากเอกสาร books.xml

2.5.2.3 order by

order by ใช้ในการกำหนดการเรียงลำดับของข้อมูล และ/หรือชุดของข้อมูล และการใช้งานของ order by นั้นจะต้องถูกกำหนดไว้ก่อนการใช้ return การทำงานในส่วนนี้เปรียบเทียบได้กับการใช้ order by ใน SQL แต่ความพิเศษของ order by ใน XQuery คือ สามารถเรียงลำดับข้อมูลที่ไม่มีอยู่ในผลลัพธ์ได้

```
for $b in doc("books.xml") //book
where $b/pubinfo/publisher = "Harper and Row"
order by $b/publishinfo/year
return $b/title
ผลลัพธ์ที่ได้คือ
<title>Harold and The Purple Crayon</title>
<title>Harold's Fairy Tale</title>
```

รูป 2.53 ตัวอย่างใช้ order by

ตัวอย่างการใช้ order by ในรูป 2.53 มีการใช้ order by ในการเรียงลำดับปีที่พิมพ์หนังสือซึ่งไม่แสดงในส่วนของผลลัพธ์ แต่ผลลัพธ์ที่ได้นั้นมีการจัดเรียงลำดับปีที่พิมพ์

2.5.2.4 return

return ใช้ในการกำหนดรูปแบบการแสดงผลของผลลัพธ์ ซึ่งอิลิเมนต์ที่ปรากฏในก่อนหน้านั้นไม่จำเป็นต้องเป็นอิลิเมนต์ที่มีอยู่ในเอกสาร XML ที่ทำการค้นหาก็ได้ และถ้าก่อนหน้า return ไม่ปรากฏ order by อยู่ก่อน การเรียงลำดับของข้อมูลในผลลัพธ์จะเป็นไปตามลำดับเอกสาร XML ที่ใช้ในการค้นหา หรือเป็นไปตามลำดับที่ได้ถูกกำหนดไว้จากการชุดคำสั่ง for

```
for $b in doc("books.xml") //book
where $b/pubinfo/publisher="Harper and Row"
return <HR_book> {$b/title}<HR_book>
ผลลัพธ์ที่ได้คือ
<HR_book>
     <title>Harold and the Purple Crayon</title>
</HR_book>
<HR_book>
    <title>Harold's Fairly Tale</title>
</HR_book>
```

รูป 2.54 ตัวอย่างใช้ return

ตัวอย่างในรูป 2.54 ค้นหาชื่อหนังสือจากเอกสาร book.xml โดยหนังสือเล่มนั้นจะต้องมาจากสำนักพิมพ์ Harper and Row และมีการกำหนดโครงสร้างของผลลัพธ์ โดยสร้างอิลิเมนต์ใหม่ที่ไม่มีอยู่ในเอกสาร books.xml

2.5.2.5 ตัวดำเนินการและฟังก์ชั่นสำเร็จรูป

XQuery มีการกำหนดตัวดำเนินการให้ใช้งาน ได้แก่

1) ตัวดำเนินการทางคณิตศาสตร์ เช่น เครื่องหมายบวก ลบ

2) ตัวดำเนินการสำหรับการเปรียบเทียบ เช่น เครื่องหมายมากกว่า น้อยกว่า

3) ตัวดำเนินการในเชิงตรรกะ เช่น and, or

สำหรับฟังก์ชั่นสำเร็จรูปที่มีให้ใช้ใน XQuery มีมากกว่า 100 ฟังก์ชั่น ซึ่งฟังก์ชั่นเหล่านี้เป็นฟังก์ชั่นเกี่ยวกับ สตริง, ตัวเลข, การเปรียบเทียบวันที่และเวลาและอื่นๆ โดยค่าปกติของเนมสเปสของฟังก์ชั่นคือ fn: ดังนั้นในการเรียกใช้ฟังก์ชั่น เราไม่จำเป็นต้องใส่คำนำหน้า fn: นำหน้าก็ได้ตัวอย่างของฟังก์ชั่นสำเร็จรูป ตัวอย่างเช่น

1) ฟังก์ชันการหาค่าเฉลี่ย(avg)

2) ฟังก์ชันการหาผลรวม(sum)

3) ฟังก์ชันการนับจำนวน(count)

4) ฟังก์ชันการหาค่าสูงสุด(max)

5) ฟังก์ชันการหาค่าต่ำสุด(min)

6) ฟังก์ชันสำหรับการอ้างถึงเอกสาร XML ที่ต้องการค้นหาข้อมูล(doc)

7) ฟังก์ชันสำหรับการนำเสดงเฉพาะข้อมูลภายในแท็ก(text)

8) ฟังก์ชันการตรวจสอบค่าว่าง(empty)

9) ฟังก์ชันการตรวจสอบค่าที่มีอยู่(exists)

10) ฟังก์ชั่นการตรวจสอบค่าที่อยู่(exists)

11) ฟังก์ชันการหาวันที่จากวันเวลา(day-from-dateTime)

```
let $b :=doc("data/books.xml") //book

return <HR_book> {max($b/pubinfo/price) } </HR_book>

ผลลัพธ์ที่ได้คือ

<HR_book>15.45</HR_book>
```

รูป 2.55 ตัวอย่างการใช้ฟังก์ชันการหาค่าสูงสุด

```
<name> {uppercase{$booktitle}}</name>
```

รูป 2.56 ตัวอย่างการเรียกใช้ ฟังก์ชั่น ในอิลิเมนต์

```
doc{"books.mxl"}/bookstore/book[substring{title,1,5}='Harry']
```

รูป 2.57 ตัวอย่างการเรียกใช้ฟังก์ชั่นในภาคแสดงของรูปแบบการเข้าถึงข้อมูล

```
let $name := {substring($booktitle,1,4))
```

รูป 2.58 ตัวอย่างการเรียกใช้ฟังก์ชั่นใน let cause

2.5.3 การกำหนดเงื่อนไขสำหรับการแสดงผลของผลลัพธ์

การกำหนดเงื่อนไขสำหรับการแสดงผลของผลลัพธ์จะถูกใช้ภายใน return เพื่อให้กำหนดเงื่อนไขในการแสดงค่าผลลัพธ์ ซึ่งจะต้องอยู่ในรูปแบบของ if…then…else

```
for $b in doc("books.mxl") //book
return <HR_book> {
    $b/title,
    if($b/pubinfo/year>"1995") then $b/pubinfo/year
    else $b/pubinfo/price
    }</HR_book>
ผลลัพธ์ที่ได้คือ
<HR_book>
   <tiltle> Harold and the Purple Crayon</title>
    <price>4.76</price>
<HR_book>
   <title>Harold's Fairy Tale</title>
    <year>1956</year>
</HR_book>
<HR_book>
    <title> Rise Up Singing</title>
    <year>1988</year>
</HR_boook>
```

รูป 2.59 ตัวอย่างการกำหนดเงื่อนไขสำหรับการแสดงผลของผลลัพธ์

ตัวอย่างในรูป 2.59 เป็นการค้นหาชื่อหนังสือจากเอกสาร books.xml โดยในผลลัพธ์จะแสดงปีที่พิมพ์หนังสือถ้า หนังสือเล่มนั้นพิมพ์หลังจากปี 1995 แต่จะแสดงราคาหนังสือถ้าหนังสือเล่มนั้นพิมพ์ก่อน หรือพิมพ์ในปี 1995

2.5.4 การสร้างฟังก์ชัน

นอกเหนือจากฟังก์ชันสำเร็จรูปแล้ว XQuery ยังเปิดให้ผู้ใช้สามารถสร้างฟังก์ชันขึ้นใช้งานเองได้โดยในการสร้างฟังก์ชั่นหนึ่งๆนั้นจะต้องเป็นไปตามรูปแบบ ดังนี้

```
Declare function prefix:function_name($parameter AS datatype)
  AS returnDataType
{
(: … function code here …)
};
```

รูป 2.60 รูปแบบของการสร้างฟังก์ชั่นขึ้นใช้งานเอง

ในการสร้างฟังก์ชันที่สร้างขึ้นเองนั้นต้องใช้คำหลักว่า “declare function” ชื่อของฟังก์ชั่นต้องมีคำนำหน้า โดยส่วนใหญ่แล้ว ชนิดข้อมูลของตัวแปร จะเหมือนกับชนิดข้อมูลที่มีการระบุแล้วใน XML Schema ส่วนเนื้อหาของฟังก์ชั่นต้องอยู่โดยวงเล็บปีกกาเปิดและปิด

```
declare function local:minPrice{
   $price as xs:decimal?,
   $discount as xs:decimal?)
   AS xs:decimal?
}
let disc := ($price * $discount) div 100
return ($price - $disc)
};
(: Below is an example of how to call the function above : )
<minPrice> {local:minPrice($book/price,$book/discount)}</minprice>
```

รูป 2.61 ตัวอย่างของฟังก์ชั่นที่สร้างขึ้นเอง

2.5.5 การกำหนดเงื่อนไขในการกำจัดจำนวนข้อมูล

การใช้เงื่อนไขประเภทนี้ใช้สำหรับการค้นหาข้อมูลในลักษณะดังนี้ คัดเลือกข้อมูลที่มีเพียงบางอิลิเมนต์ ตรงกับเงื่อนไขที่ต้องการ หรือคัดเลือกข้อมูลที่ทุกๆ อิลิเมนต์จะต้องตรงกับเงื่อนไขที่ต้องการ โดยใช้คำสั่ง some หรือ every ตามลำดับ สำหรับการใช้งานด้วยคำสั่งทั้งสองนั้นจะต้องตามด้วยคำสั่ง satisfies

```
let $b in doc("books.xml")//book
where some $y in $b//year satisfies $y> "1970"
return $b/title
ผลลัพธ์ได้คือ
<title> Harold and the Purple Crayon</title>
<title> Harold's Fairy Tale</title>
<title>Rise Up Singing</title>
```

รูป 2.62 ตัวอย่างการกำหนดเงื่อนไขในการจำกัดจำนวน (1)

ตัวอย่างในรูป 2.62 เป็นการค้นหาชื่อหนังสือจากเอกสาร book.xml โดยที่จะต้องมีหนังสืออย่างน้อย 2 เล่มที่พิมพ์หลังจากปี 1970

```
let $bin doc ("books.xml") //book
where every $y in $b //year satisfies $y> "1995"
return $b/title
ผลลัพธ์ที่ได้คือ ไม่มีคำตอบสำหรับการค้นหานี้
```

รูป 2.63 ตัวอย่างการกำหนดเงื่อนไขในการจำกัดจำนวน (2)

ตัวอย่างในรูป 2.63 เป็นการค้นหาชื่อหนังสือจากเอกสาร book.xml โดยที่หนังสือทุกๆ เล่ม จะต้องพิมพ์หลังจากปี 1995

2.5.6 การค้นหาข้อมูลภายในเอกสาร XML ที่มีมากกว่า 1 เอกสาร

ในกรณีที่ต้องทำการค้นหาข้อมูลจากเอกสาร XML 2 เอกสารขึ้นไป เพื่อนำผลลัพธ์ที่ได้จากทั้งสองเอกสารมารวมกันเป็นเอกสารผลลัพธ์เดียว โดยการรวมผลลัพธ์จากทั้งสองเอกสารนั้นจะต้องมีการกำหนดเงื่อนไขเพื่อให้เอกสารทั้งสองอ้างอิงถึงในสิ่งเดียวกัน ซึ่งสามารถเปรียบเทียบได้กับการค้นหาข้อมูลจาก 2 ตารางขึ้นไป (Join) ใน SQL นั่นเองดังแสดงในตัวอย่างต่อไปนี้

```
for $b in doc ("books.xml") //book,
     $r in doc("reviews.xml") //book
where $b/title=$r/title
return <HR_book> {
            $b/title,
             $b/author.
             $r/review
       } </HR_book>
ผลลัพธ์ที่ได้คือ
<HR_book>
    <title> Rise Up Singing</title>
    <author>
         <lastname>Blood</lastname>
         <firstname>Peter</firstname>
     </author>
     <author>
          <lastname>Patterson</lastname>
          <firstname>Annie</firstname>
     </author>
     <review> This is great!</review>
<HR_book>
```

รูป 2.64 ตัวอย่างการค้นหาข้อมูลภายในเอกสาร XML ที่มีมากกว่า 1 เอกสาร

ตัวอย่างในรูป 2.64 เป็นการค้นหาชื่อหนังสือ และชื่อผู้แต่งจากเอกสาร books.xml รวมกับการค้นหาคำวิจารณ์หนังสือเล่มนั้นจากเอกสาร reviews.xml

2.5.7 การทำงานกับแอตทริบิวต์

ในกรณีที่เอกสาร XML ที่ต้องการค้นหาแอตทริบิวต์ เป็นส่วนประกอบหนึ่งในเอกสารด้วยนั้น สามารถนำมาใช้เป็นข้อมูลหนึ่งในการกำหนดเงื่อนไข และการแสดงผลได้ดังนี้

2.5.7.1 การกำหนดเงื่อนไขด้วยแอตทริบิวต์

XQuery สามารถค้นหาข้อมูลโดยมีการกำหนดเงื่อนไขในการกรองข้อมูลด้วยแอตทริบิวต์

```
for $b in doc("books.xml") //book
where $b/@number>"2000"
return <HR_book> {$b/title} </HR_book>
ผลลัพธ์ที่ได้คือ
<HR_bool>
      <title>Rise Up Singing</title>
<HR_book>
```

รูป 2.65 ตัวอย่างการกำหนดเงื่อนไขด้วยแอตทริบิวต์

ตัวอย่างที่ในรูป 2.65 เป็นการค้นหาชื่อหนังสือจากเอกสาร books.xml โดยหนังสือเล่มนั้นจะต้องมีแอตทริบิวต์ number มากกว่า 2000

2.5.7.2 การแสดงแอตทริบิวต์ในผลลัพธ์

Xquery สามารถแสดงแอตทริบิวต์ในอิลิเมนต์ที่สร้างใหม่ของการแสดงผลลัพธ์ได้

```
for $b in doc("books.xml") //book
return
<HR_book> {$b/@number}
      {$b/title}
</HR_book>
ผลลัพธ์ที่ได้คือ
<HR_book number ="1001">
      <title>Harold and the Purple Crayon</title>
</HR_book>
<HR_book number="2001">
    <title> Rise Up Singing</title>
</HR_book>
```

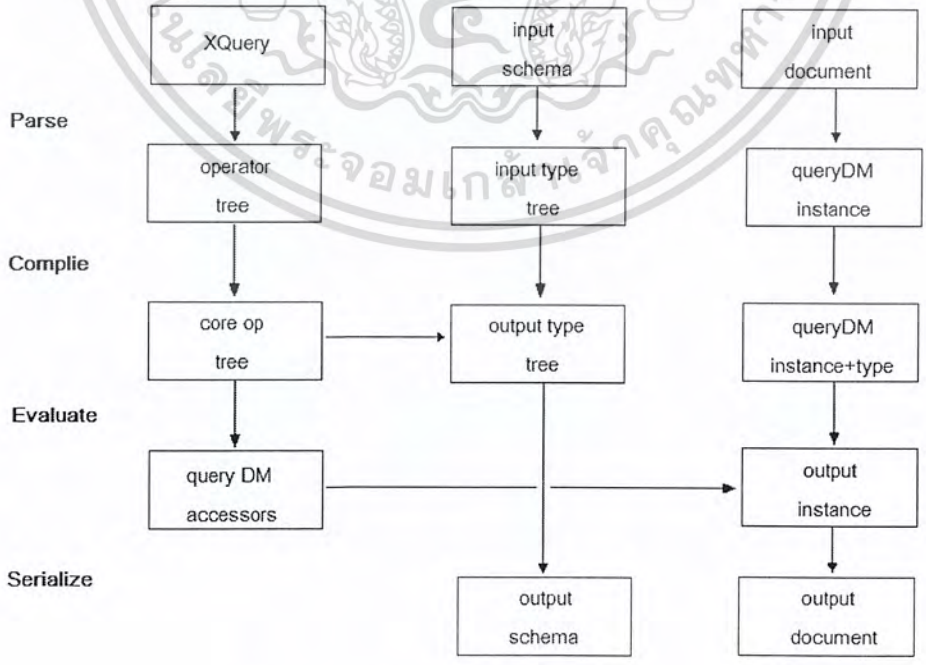
รูป 2.66 ตัวอย่างการแสดงแอตทริบิวต์ในผลลัพธ์

ตัวอย่างในรูป 2.66 ค้นหาชื่อหนังสือจากเอกสาร books.xml โดยกำหนดโครงสร้างของผลลัพธ์โดยสร้างอิลิเมนต์ใหม่ที่ไม่มีอยู่ในเอกสาร และภายในอิลิเมนต์ใหม่นี้แสดงข้อมูลของแอตทริบิวต์ number ด้วย

2.6 แบบจำลองกระบวนการทำงานของ XQuery (XQuery Processing Model)

XQuery Processing Model ประกอบด้วย 4 ขั้นตอน ซึ่งในแต่ละขั้นตอนจะนำผลลัพธ์ที่ได้จากขั้นตอนก่อนหน้ามาเป็นอินพุต เพื่อนำมาเข้าสู่กระบวนการทำงานเพื่อให้ได้เอาท์พุตไปทำงานในขั้นตอนต่อไป ซึ่งรายละเอียดในแต่ละขั้นตอนมีดังนี้

1) Parse มีหน้าที่ในการตรวจสอบความถูกต้องของ XQuery Expression, XML Schema และ XML Document หลังจากตรวจสอบความถูกต้องเรียบร้อยแล้ว จะสร้างโครงสร้างตัวดำเนินการ ในรูปแบบต้นไม้ สร้างโครงสร้างเอกสารในรูปแบบต้นไม้และสร้างโครงสร้างข้อมูลเอกสารในรูปแบบต้นไม้ตามลำดับ
2) Compile มีหน้าที่แปลง XQuery Expression ให้อยู่ในรูปของ XQuery Core และนำโครงสร้างเอกสารกับเอกสาร XML มาตรวจสอบความถูกต้องร่วมกัน
3) Evaluate มีหน้าที่นำ XQuery Core มาประมวลผลเพื่อค้นหาข้อมูลผลลัพธ์ และสร้างโครงสร้างผลลัพธ์
4) Serialize มีหน้าที่นำข้อมูลผลลัพธ์ และโครงสร้างผลลัพธ์มาสร้างเป็นเอกสาร XML



รูป 2.67 แบบจำลองการทำงานของ XQuery

2.7 ประโยชน์ และการประยุกต์ใช้งานของภาษา XML

2.7.1 ประโยชน์ของภาษา XML

1) ใช้สำหรับสร้างข้อมูลที่สามารถอธิบายความหมายของตัวเองได้ (Self-describe data) จากความสามารถในการสร้างแท็กขึ้นมาอธิบายข้อมูลที่อยู่ภายในแท็ก ทำให้ข้อมูลนั้นมีความหมายอยู่ในตัวมันเอง เป็นข้อมูลที่สามารถเขียนโปรแกรมดึงข้อมูลไปใช้งานได้โดยง่าย อีกทั้งมนุษย์ก็สามารถอ่านและเข้าใจได้โดยเช่นกัน ดังนั้นจึงสามารถกล่าวได้ว่าเอกสาร XML มีคุณสมบัติครบทั้งแบบเครื่องจักรสามารถเข้าใจได้ (machine readable) และแบบมนุษย์สามารถเข้าใจได้ (human readable)

2) ใช้สำหรับการแลกเปลี่ยนข้อมูล (data exchange) เนื่องด้วยเอกสาร XML มีลักษณะเป็นไฟล์ข้อความธรรมดา ดังนั้นจึงทำให้เอกสาร XML เป็นภาษากลางที่สามารถใช้ได้ในทุกแพลตฟอร์ม เช่น วินโดวส์, ยูนิกซ์ หรืออื่นๆ ดังนั้นจึงทำให้เอกสาร XML มีความสามารถในการแลกเปลี่ยนเอกสารข้ามแพลตฟอร์มกันได้

3) เป็นรูปแบบข้อความที่ใช้ในการสื่อสาร (messaging format) ระหว่างแอพพลิเคชันหรือโปรแกรม เป็นแนวคิดที่กำลังได้รับความนิยมเป็นอย่างมากสำหรับประโยชน์ของภาษา XML ข้อนี้ เนื่องด้วยตั้งแต่ปี ค.ศ. 2000 เป็นต้นมา แนวคิดนี้เป็นแนวคิดของเว็บเซอร์วิสที่บริษัทไมโครซอฟต์เรียกว่า .NET ซึ่งแนวความคิดนี้ก็คือการเตรียมซอฟแวร์สำหรับให้บริการทำงานบางอย่างอยู่ในเซิร์ฟเวอร์เครื่องใดเครื่องหนึ่ง ภายในเครือข่ายอินเตอร์เน็ต โดยผู้ใช้งานทั่วไปสามารถเรียกใช้บริการนั้นได้

4) ประโยชน์ในเชิงเทคโนโลยีอินเตอร์เน็ตและการพัฒนาเว็บประโยชน์ในเชิงเทคโนโลยีอินเตอร์เน็ตและการพัฒนาเว็บในเมืองไทยยังไม่เห็นประโยชน์มากนัก เพราะลักษณะงานยังไม่มีความจำเป็นให้นำภาษา XML ไปพัฒนา แต่งานบางอย่างเช่น การพัฒนาเว็บสามารถที่จะนำภาษา XML มาช่วยเพิ่มประสิทธิภาพได้

5) เป็นรากฐานของภาษาใหม่ ในการพัฒนาเว็บ ภาษาใหม่ ในการพัฒนาเว็บ เช่น ภาษา XHTML, MathML, VML, WML เป็นต้น

6) ใช้ในแวดวงธุรกิจแบบ B2B (Business to Business) ในกรณีนี้จะต้องใช้ภาษาเฉพาะอย่างเช่น cXML (Commerce XML), xCML (XML Common Business Language) เป็นต้น และในต่างประเทศมีโครงการนำ XML มาใช้แทนระบบ EDI (Electronic Data Interchange) ซึ่งเป็นการแลกเปลี่ยนเอกสารอิเล็กทรอนิกส์ระหว่างองค์กร

2.7.2 การประยุกต์ใช้งานภาษา XML

1) ใช้สำหรับแยกข้อมูลออกจากภาษา HTML การใช้งานภาษา HTML แสดงผลข้อมูลบนบราวเซอร์ ข้อมูลที่แสดงจะถูกเก็บในภาษา HTML แต่เราสามารถใช้ภาษา XML ร่วมกับ HTML ได้โดยใช้ HTML มีบทบาทสำคัญในการแสดงผลและจัดตำแหน่งการ

แสดงผลส่วนภาษา XML จะเป็นส่วนเก็บข้อมูลที่จะแสดงแยกเก็บในไฟล์ (.xml) ต่างหาก ข้อมูลXML จะมักปรากฏเป็นกลุ่มๆ ในภาษาHTMLหรือเรียกว่า "Data Islands"

2) ใช้สำหรับแลกเปลี่ยนข้อมูลระหว่างแอปพลิเคชั่น การเก็บข้อมูลในระบบคอมพิวเตอร์และฐานข้อมูลในปัจจุบันมีหลากหลายรูปแบบ จากปัญหานี้ จึงต้องสร้างรูปแบบของข้อมูลที่ไม่ขึ้นกับแพลตฟอร์ม และสามารถอ่านได้หลายๆ แอปพลิเคชั่น ซึ่งตรงกับคุณสมบัติของภาษา XML เพราะข้อมูลในเอกสาร XML มีลักษณะเป็นไฟล์ข้อมูลธรรมดาที่อ่านได้ทั้งมนุษย์และโปรแกรมแอปพลิเคชั่น

3) ใช้สำหรับเก็บข้อมูล ภาษา XML สามารถนำไปประยุกต์ใช้ในการเก็บข้อมูลทั้งในรูปของไฟล์ข้อมูล และการเก็บข้อมูลลงฐานข้อมูล นอกจากนี้ยังมีแอปพลิเคชั่นที่สามารถสืบค้นข้อมูล XML มาแสดงผลในแพลตฟอร์มต่างๆ ได้

4) ใช้สำหรับการสร้างภาษใหม่ เนื่องจากภาษา XML เป็นภาษาที่สามารถกำหนดโครงสร้างขึ้นเองได้ จึงมีการนำภาษา XML มาเป็นแม่แบบในการสร้างภาษาใหม่ขึ้นมาใช้งานกับแพลตฟอร์มต่างๆ เช่น WML เป็นภาษาที่ใช้สำหรับอุปกรณ์แบบพกพา เช่น โทรศัพท์มือถือ

5) ใช้สำหรับส่งข้อมูลระหว่างองค์กรธุรกิจ (B2B) ภาษา XML เป็นภาษาที่สามารถกำหนดโครงสร้างของเอกสารที่สร้างขึ้นได้ ดังนั้นเมื่อองค์กร ต้องการแลกเปลี่ยนข้อมูลกันจึงต้องมีการกำหนดโครงสร้างที่เหมือนกันจึงทำให้เอกสารที่แลกเปลี่ยนกันนั้นมีความถูกต้องและความน่าเชื่อถือสูง

2.7.3 การใช้งานภาษา XML ในประเทศไทย

หน่วยงานของรัฐและเอกชนหลายแห่งในประเทศไทย ได้มีการนำภาษา XML ไปใช้งานแล้ว โดยส่วนมากจะเป็นในแง่ของการแลกเปลี่ยนเอกสารในรูปแบบของเอกสาร XML (XML as data exchange) ตัวอย่างเช่น

1) ธนาคารแห่งประเทศไทย ได้กำหนด DTD สำหรับโครงสร้างของเอกสาร XML ที่ใช้นิยามข้อมูลทางการเงิน ที่สถาบันการเงินต่างๆ ต้องส่งรายงานมายังธนาคารแห่งประเทศไทย

2) มหาวิทยาลัยของรัฐแห่งหนึ่ง ได้กำหนดให้ส่งข้อมูลเกี่ยวกับการจ่ายค่าหน่วยกิตจากธนาคารมายังมหาวิทยาลัยในรูปแบบ XML

3) โครงการแลกเปลี่ยนข้อมูลภาครัฐของ NECTEC ซึ่งได้มีการวิจัยการเก็บข้อมูลพื้นฐานของภาครัฐด้วย XML เพื่อให้สามารถนำไปใช้แลกเปลี่ยนระหว่างหน่วยงานได้ง่ายขึ้น

4) โครงการของกรมศุลกากรในการพัฒนาระบบงานใหม่ไปสู่ระบบ Paperless Customs โดยหนึ่งในนั้นใช้เทคโนโลยี XML และ ebXML ในการแลกเปลี่ยนข้อมูลทางธุรกิจ

5) เว็บไซต์ thaisarn.com และ rssthai.com ได้รวบรวมข่าวจากสื่อต่างๆ มาเก็บในรูปแบบ RSS ให้เจ้าของเว็บไซต์ต่างๆ นำข่าวไปติดที่เว็บไซต์

บทที่ 3
ทฤษฎีเกี่ยวข้องกับ RDF

3.1 Resource Description Framework (RDF)

W3C ได้สร้างมาตรฐาน RDF ขึ้นมาเพื่อให้เป็นมาตรฐานในการสร้างเมตะเดตา (Metadata) ให้กับทรัพยากรที่มีอยู่มากมายบน World Wide Web เพราะอินเตอร์เน็ตในปัจจุบันมีการเจริญเติบโตที่รวดเร็ว ทำให้ข้อมูลที่มีอยู่นั้นมากมายจนเกินกำลังของมนุษย์จะเข้าไปค้นหาข้อมูลที่ต้องการท่ามกลางที่มีอยู่มากมายนั้นได้ ซึ่ง RDF จะทำให้สิ่งที่มีอยู่มากมายนั้นมีคำอธิบายไปในตัว ทำให้สามารถจัดการได้อย่างอัตโนมัติไม่ต้องให้คนมาจัดการกับทรัพยากรเหล่านี้ด้วยตนเอง

ใน XML ที่กล่าวมานั้นอธิบายเกี่ยวกับมาให้คำอธิบายข้อมูลด้วย tag ซึ่งจะมี เครื่องมือ และ ตัวจัดการในการแลกเปลี่ยนข้อมูลซึ่งกันระหว่างโปรแกรมประยุกต์ แต่ XML นั้นจะไม่พูดเรื่องความหมาย (Semantics) หรือความหมายของข้อมูลนั่นเอง ซึ่ง RDF ก็จะเปรียบเสมือน แบบจำลองข้อมูล (Data model) ที่ประกอบไปด้วย ทริปเปิล (Triple) มองในลักษณะของกราฟ RDF จะใช้ XML เป็นไวยากรณ์ และมี RDF Schema เป็นกลุ่มคำศัพท์ในการอธิบายคุณสมบัติ (Property) ที่ประกอบได้ด้วยกรรม (Object) และค่า รวมถึงความสัมพันธ์ระหว่างกรรมด้วย

3.1.1 ประวัติของ RDF

ในเดือนตุลาคม ปี 1997 ได้เริ่มมีการร่างข้อกำหนดของ RDF ขึ้นมา

ในเดือนพฤศจิกายน ปี 1997 ได้มีการแนะนำ RDF เมตะเดตา

ในเดือนกุมภาพันธ์ ปี 1999 ข้อกำหนดของ RDF ในเรื่องของโครงสร้าง และไวยากรณ์ ก็ได้ออกมา

ในเดือนสิงหาคม ปี 1999 RDF Interest Group ได้ก่อตั้งขึ้นมา

ในเดือนมีนาคม ปี 2000 RDF Schema Specification 1.0 ได้ถูกเผยแพร่สู่สาธารณะ

3.1.2 เมตะเดตา คืออะไร

เมตะเดตา คือ ข้อมูลที่ใช้อธิบายข้อมูลหนึ่ง ซึ่งในที่นี้ก็เจาะจงกับเนื้อหาที่อยู่บน Web นั้นก็คือ link หรือ URL ต่างต่างนั่นเอง ขอบเขตของ เมตะเดตา กับข้อมูล นั่นไม่อาจแบ่งแยกได้อย่างชัดเจน ขึ้นอยู่กับแอพพลิเคชันที่ใช้งานว่าจะมองเห็นอะไร ซึ่งบางกรณีก็อาจจะมองเป็นได้ทั้งสองอย่างพร้อมกัน

3.1.3 แนวคิดในการออกแบบ RDF

RDF ถูกสร้างขึ้นมาโดยมีแนวคิด ดังนี้

1) มีโครงสร้างที่ไม่ซับซ้อน

2) มีความหมายที่เป็นทางการและมีและอ้างอิงที่พิสูจน์ได้

3) สามารถใช้ไวยากรณ์ของ extensible URL-based ได้

4) ใช้ไวยากรณ์ ของ XML เป็นพื้นฐาน

5) รองรับการใช้งานประเภทแบบข้อมูลแบบ XML schema

6) สามารถให้ใครก็ได้สร้าง ประโยค (statement) ให้กับทรัพยากรใดก็ได้

3.1.4 RDF ประกอบด้วยอะไร

1) ทรัพยากร (Resource) ทรัพยากรนั้นจะเปรียบเสมือนอ็อบเจ็ตของสิ่งหนึ่งสิ่งใด เช่น ผู้แต่ง หนังสือ สถานที่ บุคคล ซึ่งทุกทรัพยากรจะมี URI (Uniform Resource Identifier) ซึ่ง URI อาจจะเป็น URL (Uniform Resource Locator) หรือจะเป็นอะไรที่สามารถระบุทรัพยากรได้มาอ้างมาจากที่ใด อาจจะกล่าวได้ว่า URI เป็นสิ่งที่ระบุทรัพยากรเว็บ

รูป 3.1 แสดง URLs เป็นซับเซตของ URIs

2) คุณสมบัติ คุณสมบัติคือ ประเภทพิเศษของทรัพยากรที่ไว้อธิบายความสัมพันธ์ระหว่างทรัพยากร เช่น “written by” ถูกเขียนโดยใคร หรือ “age” มีอายุเท่าไหร่ ซึ่งถูกระบุภายใต้ URIs เหมือนกัน เพื่อบอกความสัมพันธ์ระหว่างสิ่งที่ระบุถึง

3) ประโยค ประโยค คือสิ่งที่เสริมคุณสมบัติ ของทรัพยากร ซึ่ง ประโยค นี้จะประกอบด้วยค่าของ แอตทริบิวต์ของนั้นๆ ซึ่งทริปเปิลจะรวมทั้ง ทรัพยากร คุณสมบัติ และ ค่าคุณสมบัติ มาประกอบกัน

3.1.5 ประเภทข้อมูลที่มีอยู่ใน RDF

ประเภทข้อมูลที่ใช้กันอยู่ มี 2 อย่างหลักๆ ได้แก่

1) ทรัพยากร (Resource) ซึ่งก็คือ URI ต่างๆนั้นเอง

2) ค่าตามตัวอักษร (Literal) ก็คือ ชนิดข้อมูล (Data type) ต่างๆ เช่น ข้อความ ค่าความจริง หรือตัวเลข ซึ่งจะถูกกำหนดตาม XML schema

3.1.6 ประโยชน์ของ RDF

1) ทำให้สามารถจัดการทรัพยากรที่มีอยู่ได้ง่ายขึ้น โดยมารถใช้ระบบอัตโนมัติในการจัดการทรัพยากรแทนมนุษย์ได้ เช่น แทนที่จะใช้คนมานั่งเปิดเว็บดูแล้วก็ค่อยจัดกลุ่มเว็บ ก็เปลี่ยนมาใช้โปรแกรมมาอ่านข้อมูลส่วนที่เป็น RDF แล้ววิเคราะห์ว่าเว็บไซต์นี้เป็นแบบใด

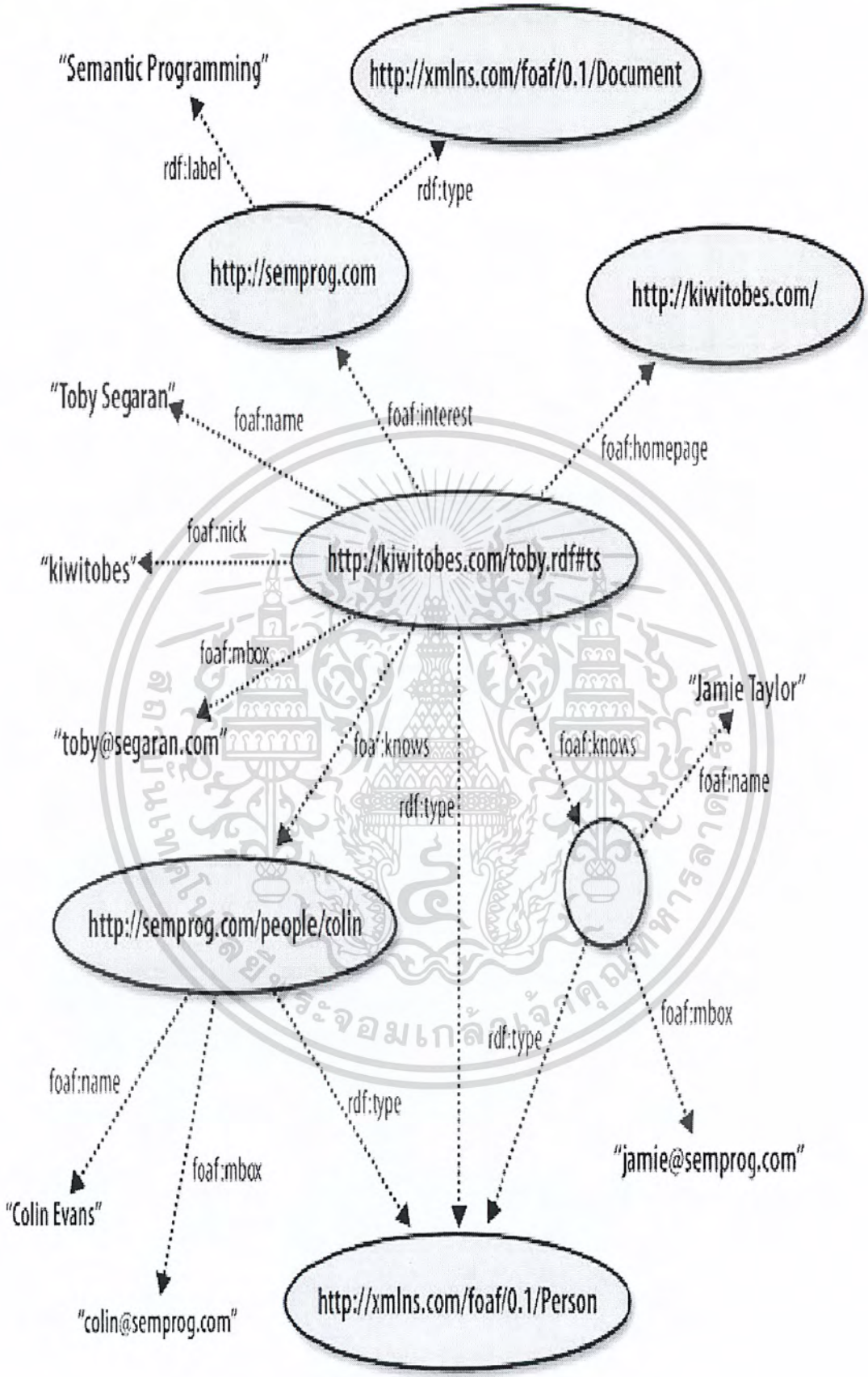
2) ทำให้สามารถใช้ทรัพยากรร่วมกันระหว่างโปรแกรมประยุกต์ได้เพราะ RDF ได้มีการกำหนดมาตรฐานในการจัดการกับข้อมูลในแต่ละด้านขึ้นเช่น Dublin Core เป็นรูปแบบที่ใช้ในการกระทำ เมตะเดตา ในกับหนังสือ ซึ่งจะทำให้โปรแกรมค้นหาหนังสือต่างต่าง สามารถทำงานกับฐานข้อมูลร่วมกันได้ หรือมาตรฐาน RSS (RDF site Summary) ที่เป็นมาตรฐานในการแลกเปลี่ยนข่าวสารระหว่างกัน ซึ่งในปัจจุบันก็มีโปรแกรมที่สามารถอ่าน RSS ได้เป็นจำนวนมาก เป็นต้น นอกจากนี้ เรายังสามารถสร้างรูปแบบของ RDF ขึ้นมาได้เองอีกด้วย

3) เป็นมาตรฐานที่ถูกสร้างขึ้นมาเพื่อสนับสนุนแนวคิดของ Semantic Web ให้เป็นจริง

3.1.7 RDF Serialization Formats

แม้ว่าแบบจำลองข้อมูล RDF จะถูกใช้โดยง่าย Serialization จะเติมเต็มให้สมบูรณ์ เช่น เมื่อกราฟ RDF ถูกบันทึกลงไฟล์หรือมีการส่งข้ามเครือข่าย เพราะว่าวิธีการที่ต่างไปจะมีผลกระทบต่อข้อมูล

เพื่อที่จะเป็นการง่ายในการแยกแยะความแตกต่างของ Serialization format อันดับแรกคือการสร้าง กราฟอย่างง่าย (Simple Graph) ซึ่งจะบอกรายละเอียดข้อมูลรวมถึงความสัมพันธ์ อาจจะเป็นการอธิบายข้อมูลแบบทอดทอดออกไป ดังตัวอย่างกราฟ FOAF (Friend of a Friend)



รูป 3.2 ตัวอย่างกราฟ FOAF

serialization format ก็จะมี notation ต่างๆ เช่น N-Triples , RDF/XML , N3

1) N-Triple จะง่ายต่อการจัดการข้อมูลและตรวจสอบข้อบกพร่องของโปรแกรมประยุกต์

```
<http://kiwitobes.com/toby.rdf#ts>
<http://xmlns.com/foaf/0.1/homepage>
<http://kiwitobes.com/>.
<http://kiwitobes.com/toby.rdf#ts>
<http://xmlns.com/foaf/0.1/nick> "kiwitobes".
<http://kiwitobes.com/toby.rdf#ts>
<http://xmlns.com/foaf/0.1/name> "Toby Segaran".
<http://kiwitobes.com/toby.rdf#ts>
<http://xmlns.com/foaf/0.1/mbox>
<mailto:toby@segaran.com>.
<http://kiwitobes.com/toby.rdf#ts>
<http://xmlns.com/foaf/0.1/interest>
<http://semprog.com>.
<http://kiwitobes.com/toby.rdf#ts>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person>.
<http://kiwitobes.com/toby.rdf#ts>
<http://xmlns.com/foaf/0.1/knows> _:jamie .
<http://kiwitobes.com/toby.rdf#ts>
<http://xmlns.com/foaf/0.1/knows>
<http://semprog.com/people/colin>.
_:jamie <http://xmlns.com/foaf/0.1/name> "Jamie Taylor".
_:Jamie <http://xmlns.com/foaf/0.1/mbox>
<mailto:jamie@semprog.com>.
_:jamie <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person>.
<http://semprog.com/people/colin>
<http://xmlns.com/foaf/0.1/name> "Colin Evans".
<http://semprog.com/people/colin>
<http://xmlns.com/foaf/0.1/mbox>
<mailto:colin@semprog.com>.
<http://semprog.com/people/colin>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person>.
<http://semprog.com>
<http://www.w3.org/2000/01/rdfschema#label>
"Semantic Programming".
<http://semprog.com>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Document>.
```

รูป 3.3 การเขียน RDF แบบ N-triple

2) RDF/XML หรือแบบมาตรฐาน เป็นแบบที่ใช้พื้นฐานมาจาก XML

```
<rdf:RDF
  xmlns:FOAF="http://xmlns.com/foaf/0.1/"
   xmlns:dc= "http://purl.org/dc/elements/1.1/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#'
   xmlns:rev=" http://amk.ca/xml/review/1.0#"
   <!-Implies rdf:type propertyis rev:Review ->
   <rev:Review rdf:about=" http://example.com/rev1">
        <rev:subject rdf:resource="urn:isbn: 1930110111"/>
   </rev:Review>
  <rdf:Description rdf:about=" http://example.com/author/0042
  <FOAF:firstName>Bob</FOAF:firstName>
  <FOAF:homepage rdf:resource=" http://www.snee.com/bob/"/>
  <FOAF:pastProject rdf:resource="urn:isbn1930110111"/>
  <FOAF:surname>DuCharme</FOAF:surname>
 <rdf:Description>
</rdf:RDF http://xmlns.com/foaf/0.1/Document>.
```

รูป 3.4 การเขียน RDF แบบ มาตรฐาน

3) Notation-3 หรือ N3 ซึ่งจะอ่านและเขียนง่ายกว่าแบบมาตรฐาน RDF/XML

```
@prefix rev:< http:///amk.ca/xml/review/1.0#>
@prefix dc:< http://purl.org/dc/elements/1.1/>
@prefix          FOAF:<          http://xmlns.com/foaf/0.1/>
< http://example.com/author/0042>
  FOAF:firstname"Bob";
  FOAF:surname"DuCharme";
  FOAF:homepage< http://www.snee.com/bob/>;
  FOAF:pastProject<urn:isbn:1930110111>;
< http://example.com/rev1>rev:subject[
=<urn:isbn: 1930110111>;
dc:title"XSLT quickly";
dc:creator< http://example.com/author/0042>;
dc:publisher"Manning"].rdf:RDF
http://xmlns.com/foaf/0.1/Document>.
```

รูป 3.5 การเขียน RDF แบบ Notation-3

ในการเขียน RDF แต่ละครั้งจะต้อง โหลด XML เนมสเปสจาก http://www.w3.org/1999/02/22-rdf-syntax-ns# ทุกครั้งโดยเนมสเปสที่จะต้องการใช้งาน ซึ่งในปัจจุบันก็มีเนมสเปสหลายตัวเลือกให้ใช้ ได้แก่

1) Dublin Core ใช้กันมากในงานห้องสมุด

Namespace url: http://purl.org/dc/elements/1.1/

Describes book

Properties: title, creator, publisher, subject, identifier

2) FOAF (Friend-of-a-friend)

Namespace url: http://xmlns.com/foaf/0.1/

Describes people

Classes: Person

Properties: name, interest, mbox, schoolHomepage, workplaceHmepage

3) DOAP (Description of a Project)

Namespace url: http://usefulinc.com/ns/doap#

Describes open source projects

Classes: Project, Repository

Properties: name, homepage, mailing-list, license, maintainer

เนมสเปส เหล่านี้มีการใช้งานจริงอยู่ตามอินเทอร์เน็ต ซึ่งทำให้เมื่อใช้งานเนมสเปสเหล่านี้แล้วแอพพลิเคชันที่สร้างมาสำหรับใช้กับ เนมสเปสเหล่านี้ก็จะใช้ได้กับงานของเราได้ด้วย และสามารถที่จะทำให้แหล่งข้อมูล (Resource) ที่เรามีอยู่สามารถใช้งานร่วมกันผู้อื่นได้อีกด้วย แต่ถ้าไม่พอใจ เนมสเปสที่มีอยู่แล้ว ต้องการสร้างใหม่ขึ้นมาเองก็สามารถที่จะทำได้

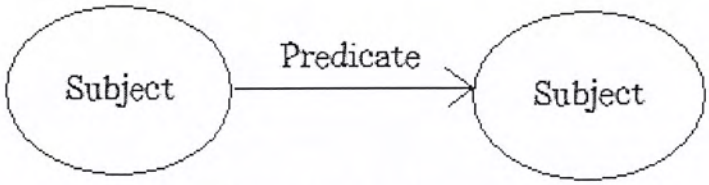
3.1.8 ข้อมูลกราฟ RDF คืออะไร

คือ แผนภาพที่ใช้แสดงโครงสร้างของ RDF ที่ใช้งานอยู่ โดยส่วนประกอบของข้อมูลกราฟ RDF มีอยู่ 3 อย่างคือ

1) ประธาน (Subject) คือ หัวข้อหรือ แหล่งข้อมูล ที่ต้องการจะอธิบายโดย แหล่งข้อมูลที่ต้องการจะอธิบายจะต้องเป็น URI เท่านั้น เช่น http://www.ku.ac.th/ หรือจะเป็นโหนดว่าง (Blank node) ก็ได้

2) ภาคแสดง (Predicate) หรือ คุณสมบัติ (Property) คือ คุณลักษณะของ แหล่งข้อมูล เช่น ที่อยู่ รหัสไปรษณีย์

3) กรรม (Object) คือ ค่าของ คุณสมบัติ เช่น ที่อยู่ก็จะมีค่าเป็น Kasetsart U. เป็นต้น หรือ อาจจะเป็น แหล่งข้อมูล ที่เป็น URI ก็ได้และอาจจะเป็นโหนดว่างก็ได้



รูป 3.6 ข้อมูลกราฟ RDF

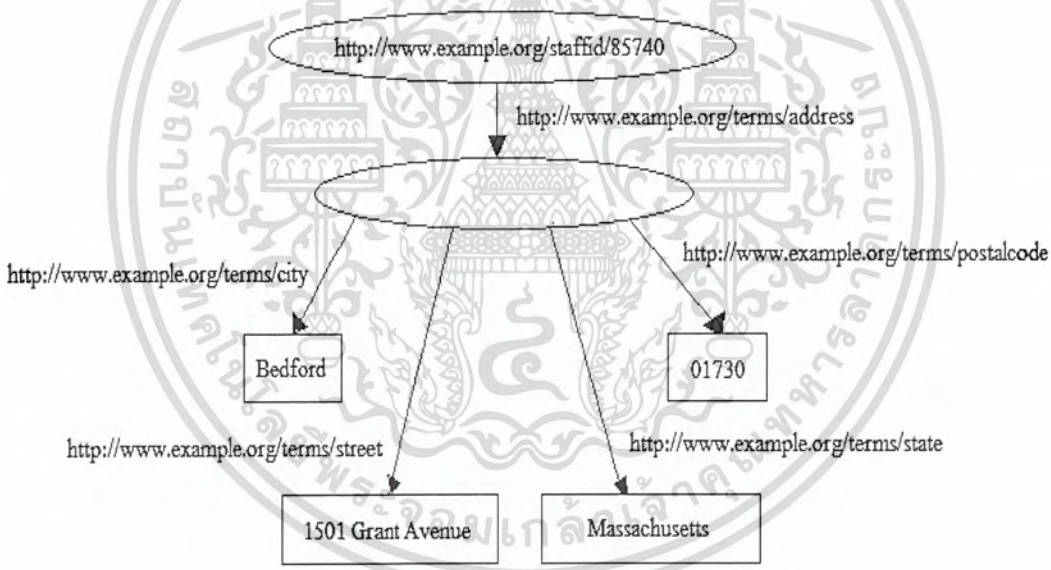
3.1.9 การเขียนกราฟ

1) ภาคแสดง เป็นลูกศรที่ชี้จากประธานไปยังกรรมที่ต้องการดังรูปข้างต้น

2) ชนิดข้อมูล (Data type) แบบ แหล่งข้อมูล หรือ URI จะใช้วงรี

3) ส่วนชนิดข้อมูล แบบ Literal จะใช้รูปสี่เหลี่ยมผืนผ้า

4) ฉะนั้นประธานทุกๆอันจะต้องใช้รูปวงรีอย่างแน่นอน เพราะประธานต้องเป็น URI เท่านั้น



รูป 3.7 แผนภาพกราฟ

จากรูป 3.7 ข้างบนนี้สามารถอธิบายได้ดังนี้

1) http://www.example.org/staffid/85740 เป็น ประธานที่ต้องการอธิบาย

2) http://www.example.org/staffid/85740 มีคุณสมบัติที่ได้กำหนดไว้ที่ http://www.example.org/term/address

3) http://www.example.org/term/address นั้นมี คุณสมบัติ แยกย่อยไปอีก 4 อัน คือ

3.1) http://www.example.org/term/city เป็นชื่อเมืองจากรูปก็ จะได้กรรมเป็น Bedford

3.2) http://www.example.org/term/street ซึ่งเป็นชื่อของถนน จะได้ กรรม เป็น 1501 GrantAvenue

3.3) http://www.example.org/term/state ซึ่งเป็นชื่อของรัฐซึ่งจากรูปก็จะได้ กรรมเป็นMassachusett

3.4) http://www.example.org/term/postalcode ซึ่งเป็นรหัสไปรษณีย์ จากรูปก็จะได้ กรรม เป็น 01730

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:staff=" http://www.example.org/terms/">
  <rdf:Description rdf:about=
          "http://www.example.orgstaffid/85740">
    <staff address rdf:NodeID="address1"/>
  </rdf:Description>
  <rdf:Description rdf:NodeID="address1"/>
     <staff:city>Bedford</staff:city>
     <staff:street>1501 Grant Avenue</staff:street>
     <staff:state>Massachusetts</staff:state>
     <staff:postalcode>01730</staff:postalcode>
   </rdf:Description>
</rdf:RDF>
```

รูป 3.8 RDF/XML แบบที่ 1 ที่ได้จากรูป 3.7

```
<?xml version=”1.0”?>
<rdf:RDF
   xmlns:rdf= “http://www.w3.org/1999/02/22-rdf-syntax-ns#”
   xmlns:staff=” http://www.example.org/terms/”>
   <rdf:Description rdf:about=
                    “http://www.example.orgstaffid/85740”>
      <staff:address>
        <staff:city> Bedford</staff:city>
        <staff:street>1501 Grant Avenue</staff:street>
        <staff:state>Massachusetts</staff:state>
        <staff:postalcode>01730</staff:postalcode>
      </staff:address>
   </rdf:Description>
</rdf:RDF>
```

รูป 3.9 RDF/XML แบบที่ 2 ที่ได้จากรูป 3.7

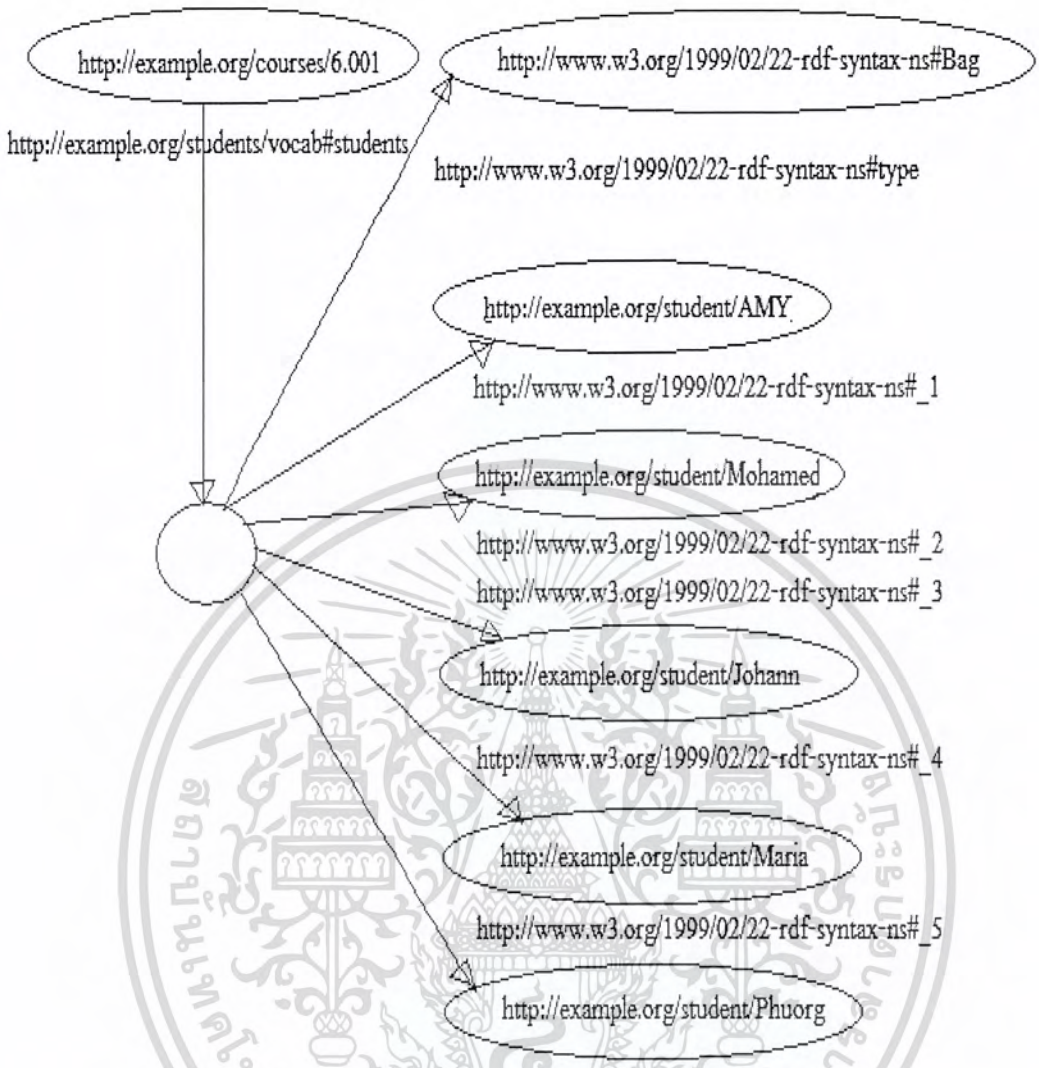
3.1.10 Container

RDF Containers คือ แหล่งข้อมูล ที่มีการรวมหลายๆ กรรม เข้าด้วยกันให้กลายเป็นกลุ่มๆหนึ่ง เช่น กลุ่มของ นักเรียน มีอยู่ 3 ประเภทด้วยกันคือ

1) Bag (rdf:Bag) เป็นการจัดกลุ่มแบบธรรมดา โดยไม่มีการเรียงลำดับและอาจจะมีการซ้ำซ้อนกันก็ได้

```
<?xml version="1.0"?>

<rdf:RDF

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

   xmlns:s=" http://example.org/students/vocab#">

   <rdf:Description rdf:about=
                        "http://example.org/courses/6.001">

     <s:students>

       <rdf:Bag>

         <rdf:li rdf:resource=" http://
                                example.org/students/AMY"/>

         <rdf:li rdf:resource=" http://
                              example.org/students/Mohamed"/>

         <rdf:li rdf:resource=" http://
                              example.org/students/Johann"/>

       </rdf:Bag>

     </s:students>

   </rdf:Description>

</rdf:RDF>
```

รูป 3.10 ตัวอย่างการใช้ RDF Containers



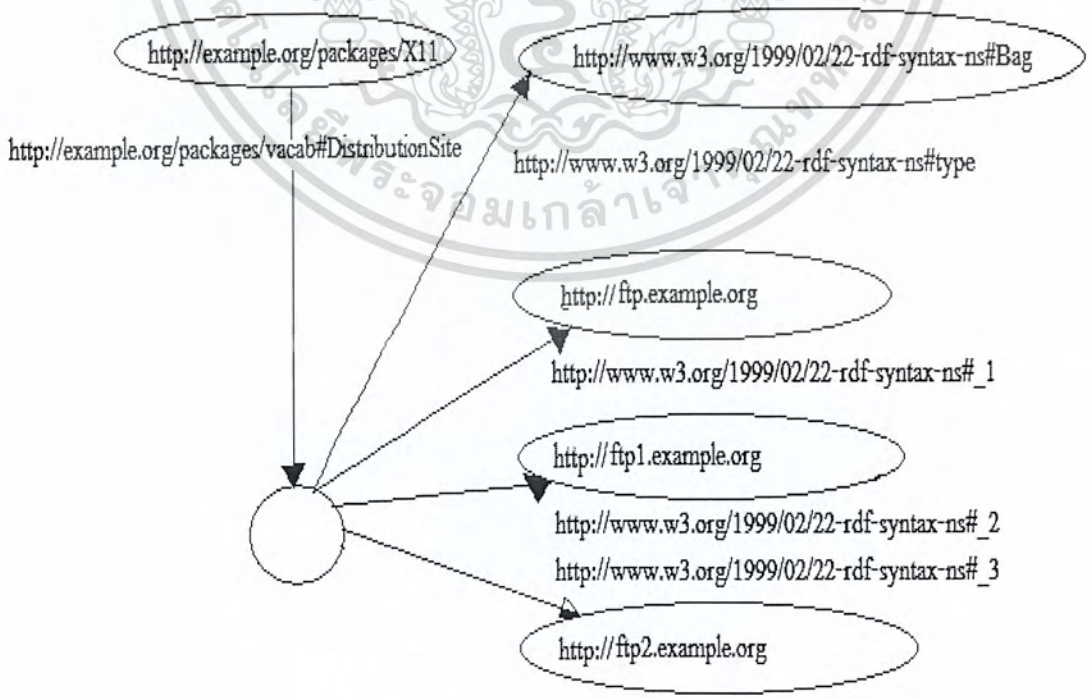
รูป 3.11 แผนภาพการใช้ RDF Containers

2) Sequential (rdf:Seq) เป็นการรวมกลุ่มที่มีลักษณะคล้ายกับ Bag รวมไปถึงการใช้งานก็จะเหมือน Bag เพียงแต่ภายในกลุ่มใน rdf:Seq จะมีการจัดเรียงค่าด้วย

3) Alternative (rdf:Seq) เป็นรูปแบบของ Container ที่ค่าในตัวจะไม่มีค่าใดเลยที่ซ้ำกัน

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf=" http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s=" http://example.org/students/vocab#">
  <rdf:Description rdf:about=
                        "http://example.org/packages/X11">
    <s: DescriptionSite>
       <rdf:Alt>
            <rdf:li rdf:resource=" ftp://ftp.example.org"/>
            <rdf:li rdf:resource=" ftp://ftp1.example.org"/>
            <rdf:li rdf:resource=" ftp://ftp2.example.org"/>
       </rdf:Alt>
    </s: DescriptionSite>
  </rdf:Description>
</rdf:RDF>
```

รูป 3.12 ตัวอย่างการใช้ Alternative



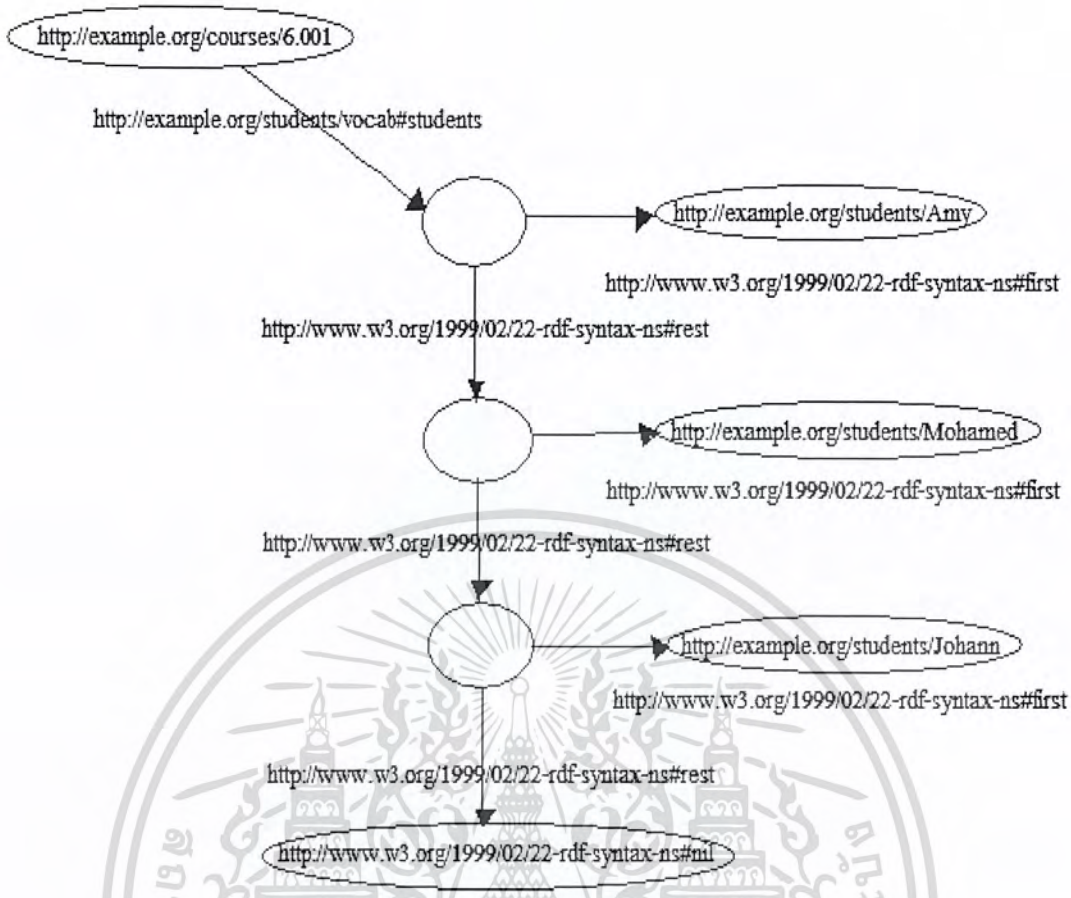
รูป 3.13 แผนภาพการใช้ Alternative

3.1.11 Collection

นอกจากจะมี Container ให้ใช้แล้ว RDF ยังมี Collection ให้ใช้โดย Collection จะมีข้อแตกต่างจาก Collection จะมีสมาชิกที่เลือกเท่านั้น ขณะที่ Collection นั้นจะสนแต่ว่า ตัวมันมีอะไรบ้างเท่านั้น ไม่ได้นึกถึงว่าสมาชิกของมันจะมีอะไรหรือไม่

```
<?xml version=”1.0”?>
<rdf:RDF
   xmlns:rdf=” http://www.w3.org/1999/02/22-rdf-syntax-ns#”
   xmlns:s=” http://example.org/students/vocab#”>
   <rdf:Description rdf:about=
                        “http:// example.org/courses/6.001”>
     <s:students rdf:parse Type=”Collection”>
       <rdf:li rdf:resource rdf:about= “http://
                              example.org/students/AMY”/>
       <rdf:li rdf:resource rdf:about= “http://
                             example.org/students/Mohamed”/>
       <rdf:li rdf:resource rdf:about= “http://
                             example.org/students/Johann”/>
      </s:students>
    </rdf:Description>
</rdf:RDF>
```

รูป 3.14 ตัวอย่างการใช้ Collection



รูป 3.15 แผนภาพการใช้ Collection

3.1.12 GSS (Graph Style Sheet)

เป็นการสร้างรูปแบบของกราฟ โดยมีจุดประสงค์เพื่อทำให้กราฟดูมีความชัดเจนมากขึ้น มีความคล้ายคลึงกับ CSS ที่ใช้ใน HTML โดยสามารถใส่สี เปลี่ยนแปลงรูปแบบตัวอักษร เปลี่ยนสีของเส้น เปลี่ยนลักษณะของเส้น ทำการรวมหลายๆ คุณสมบัติ กลายเป็นตาราง เปลี่ยนลักษณะของกรอบและใส่รูปแบบ Bitmap ลงไป จะเห็นได้ว่าเมื่อมีการใช้ GSS เข้ามาช่วยสามารถทำให้กราฟที่ได้นั้นดูง่ายขึ้น จากเดิมที่มีแต่เส้นสีเขียวลากไปลากมาเหมือนกันหมด ก็สามารถทำเป็นเส้นประเพื่อเน้นถึงว่าเป็นการอ้างอิงไปสู่โหนดอื่นๆ หรือ คุณสมบัติ ที่มีเยอะแยะ ก็สามารถรวมเป็นตารางหนึ่งตาราง และนอกจากนี้ยังมีการใช้ภาพมาแทนกรอบธรรมดาทำให้สามารถสื่อความหมายได้ดีขึ้น

3.1.13 ไวยากรณ์โดยย่อ (Abbreviated Syntax)

เราสามารถย่อไวยากรณ์ของเอกสาร RDF ได้ โดย องค์ประกอบ (element) ของคุณสมบัติที่ไม่มีลูก (Childless property) ภายในคำอธิบายองค์ประกอบ (Description element) อาจถูกแทนที่ด้วย XML แอตทริบิวต์ ในคำอธิบายองค์ประกอบสามารถใช้ชื่อที่ระบุ rdf:type อิลิเมนต์แทน rdf:Description

```
<rdf : Description  rdf : ID="CITT1111" >
   <rdf : type  rdf : resource = "&uni;course"/>
   <uni : courseName>Discrete Mathemetics</uni : courseName>
   <uni : isTaughtBy rdf : resource="#949318"/>
</rdf : Description>
```

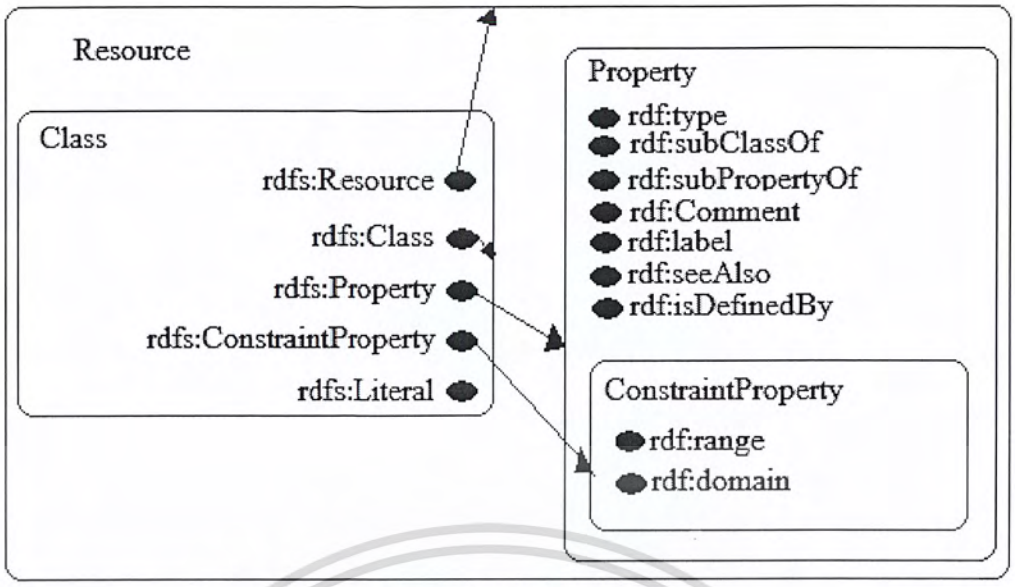
รูป 3.16 ตัวอย่าง rdf:Description

```
<uni : course rdf : ID="CIT1111"
    uni : courseName = "Discrete Mathematics">
   <uni : isTaughtBy rdf : resource = "#949318"/>
 </uni : course>
```

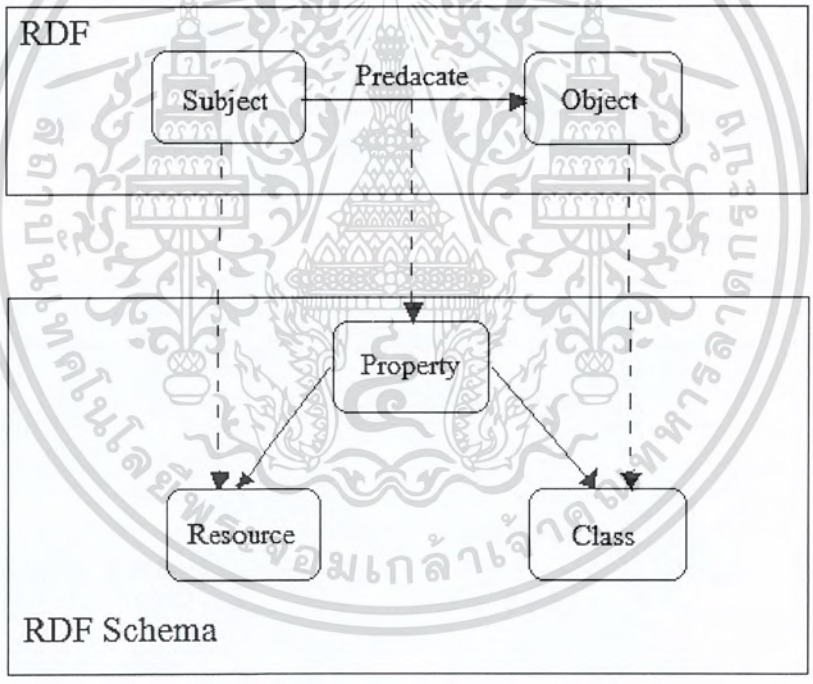
รูป 3.17 ตัวอย่างไวยากรณ์โดยย่อ

3.2 RDF Schema (RDFS)

RDF Schema เป็นการอธิบายโครงสร้างของเมตะเดตา ในการอธิบาย แหล่งข้อมูล โดยกำหนดบนคำศัพท์ เพื่ออธิบายโครงสร้างเมตะเดตา ซึ่งประกอบไปด้วย คุณสมบัติและค่าของคุณสมบัติ โดยอธิบายขยายให้อยู่ในรูปของโหนดที่เป็น คุณสมบัติ และ คลาส โดยสามารถสืบทอดเป็นคุณสมบัติย่อย และ คลาสย่อย ได้ ตามลำดับ โดยส่วนประกอบของ RDFS ประกอบด้วยส่วนที่ใช้ในการนิยามคลาส และส่วนที่ใช้ในการนิยามคุณลักษณะ และส่วนที่เป็นข้อจำกัดของข้อมูล สามารถอธิบายได้ดังรูป 3.18 และในการอธิบายโครงสร้างของ เมตะเดตา จะสามารถอธิบายได้ดังรูป 3.19



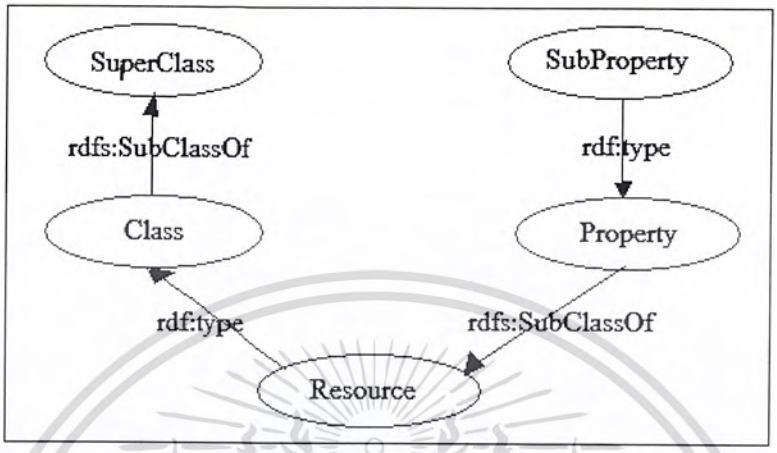
รูป 3.18 ไวยากรณ์และส่วนประกอบของ RDF Schema



รูป 3.19 การอธิบายโครงสร้างของ เมตะเดตา

จากรูปที่ 3.19 เป็นการนำโครงสร้าง เมตะเดตา ในการอธิบายข้อมูล ซึ่งประกอบไปด้วย คุณสมบัติและค่าของคุณสมบัติมากำหนดให้เป็นโหนดเพื่ออธิบาย แหล่งข้อมูล โดยส่วน ภาคแสดง ซึ่งเป็นคุณสมบัติจะถูกกำหนดมาเป็นโหนดคุณสมบัติ และส่วนของกรรม ซึ่งเป็นค่าของคุณสมบัติ จะถูกกำหนดให้เป็นโหนดที่เป็นคลาส โดยทั้งโหนดคุณสมบัติ และคลาสจะสามารถสืบทอดเป็น คุณสมบัติย่อย และ คลาสย่อย ตามลำดับ

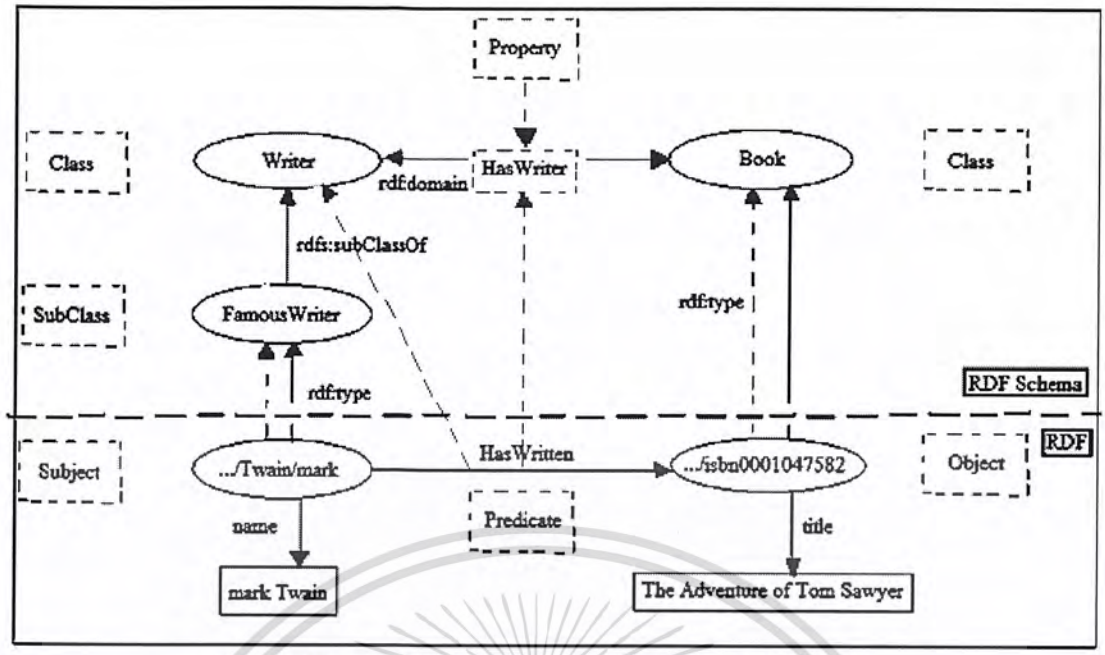
เมื่อทำการอธิบายส่วนของ เมตะเดตา ให้กลายเป็นโหนดคุณสมบัติ และโหนดคลาส รวมถึงมีการสืบทอดเป็น คลาสย่อย และ คุณสมบัติย่อย จากนั้นจะมีการจัดวางความสัมพันธ์ระหว่างโหนดในรูปแบบของ RDF Schema ซึ่งมีความสัมพันธ์กันดังรูป 3.20



รูป 3.20 การอธิบายความสัมพันธ์ของ แหล่งข้อมูล ในรูปแบบของ RDF Schema

จากรูป 3.20 แสดงให้เห็นถึงการจัดวางความสัมพันธ์ของแต่ละโหนดในรูปแบบของ RDF Schema โดยโหนดแหล่งข้อมูล กับโหนดคลาสมีความสัมพันธ์เป็น rdf:type; โหนดแหล่งข้อมูลกับโหนดคุณสมบัติ มีความสัมพันธ์กันเป็น rdfs:subClassOf โหนดคลาสและคลาสย่อย มีความสัมพันธ์กันเป็น rdfs:subClassOf และโหนดคุณสมบัติและคุณสมบัติย่อย มีความสัมพันธ์กันเป็น rdf:type

ในการสร้าง RDF Schema นั้นจะกำหนด คำศัพท์ เพิ่มเข้าไปอธิบายโครงสร้างของเมตะเดตา ที่อธิบายในโครงสร้าง RDF เพื่อให้กลายเป็นข้อมูลที่มีความหมายขึ้นมา



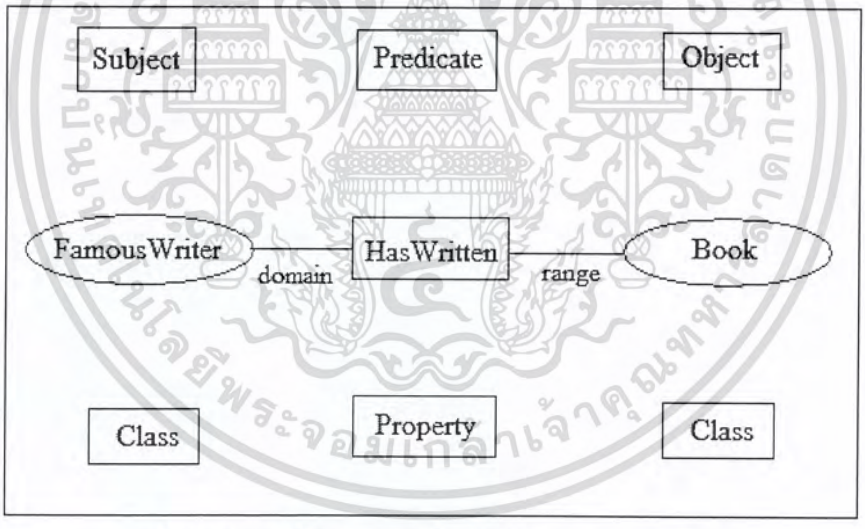
รูป 3.21 ตัวอย่างข้อมูลในการกำหนดส่วนของ RDF Schema

จากรูปที่ 3.21 จะประกอบไปด้วย 2 ส่วน (แบ่งโดยจุดเส้นประ) ส่วนแรกเป็นส่วนการอธิบายข้อมูลโดยแทนข้อมูลในโครงสร้าง RDF ซึ่งอยู่ในส่วนล่าง เพื่ออธิบายว่า แหล่งข้อมูล ที่มี URI เป็น .../twain/mark มีคุณสมบัติเป็น hasWritten และมีค่าของคุณสมบัติเป็น แหล่งข้อมูล .../ISBN00001047582 และมีการอธิบายอีกว่าแหล่งข้อมูลที่มี URI เป็น .../twain/mark นั้นมีคุณสมบัติเป็น name ซึ่งมีค่าของคุณสมบัติเป็นส่วน literal ที่มีค่าเป็น "Mark Twain" และแหล่งข้อมูล .../ISBN00001047582 มี title เป็นคุณสมบัติและมีค่าของคุณสมบัติเป็นส่วน Literal ที่มีค่าเป็น "The Adventure of Tom Sawyer"

จะเห็นได้ว่าจากการอธิบายในส่วนล่างซึ่งเป็นการอธิบายข้อมูลโดยแทนข้อมูลในโครงสร้างการอธิบายข้อมูล RDF เพื่ออธิบายว่าแหล่งข้อมูล ที่มี URI เป็น .../Twain/mark มีคุณสมบัติและค่าของคุณสมบัติที่สัมพันธ์กับสิ่งใด แต่สิ่งเหล่านี้ไม่ได้บ่งบอกถึงความหมายที่ แหล่งข้อมูลนั้นเป็นอยู่โดยแท้จริง ดังนั้นจึงต้องมีการอธิบายในส่วนของ RDF Schema เพิ่มเข้าไปเพื่อทำให้ข้อมูลประกอบไปด้วยความหมายขึ้นมา

ในการอธิบายในส่วนของ RDF Schema นั้น จะทำการกำหนดส่วนที่เรียกว่า คำศัพท์ เพื่ออธิบายขยายส่วนของโครงสร้างเมตะเดตาออกไป โดยเริ่มจากส่วนของ ประธาน นั่นคือ ส่วนของแหล่งข้อมูล .../Twain/mark ซึ่งสามารถอธิบายโดยทำให้แหล่งข้อมูลที่อยู่รูปของ คลาส FamousWriter ตามลูกศรเส้นประ และตามหลักการของ RDF Schema ระหว่างแหล่งข้อมูลและคลาส จะมีความสัมพันธ์กันเป็น rdf:type ต่อมาพิจารณาในส่วนของ ภาคแสดง HasWritten ซึ่งเป็นคุณสมบัติ สามารถอธิบายโดยทำให้คุณสมบัติอยู่ในรูปของคลาส Writer และ คุณสมบัติ

Haswritten ตามลูกศรเส้นประ และส่วนสุดท้ายคือส่วนของ กรรม ซึ่งก็ส่วนของ แหล่งข้อมูล .../ISBN00001047582 ซึ่งสามารถอธิบายโดยให้แหล่งข้อมูลอยู่ในรูปของ คลาส Book ตามลูกศรเส้นประ ซึ่งก็มีความสัมพันธ์กันเป็น rdf:type กับส่วนของกรรม ดังนั้น คลาส และ คุณสมบัติทั้งหมดที่เกิดขึ้นจากการอธิบายโครงสร้าง เมตะเดตา คือ คลาส จะมีทั้งหมด 3 คลาส คือ คลาส FamousWriter, คลาส Writer และ คลาส Book ส่วน คุณสมบัติ จะมี 1 คุณสมบัติ คือ คุณสมบัติ HasWriter โดยในการจัดวางความสัมพันธ์นั้น คลาส FamousWriter จะกลายเป็น คลาสย่อยของ คลาส Writer ซึ่งตามหลักการของ RDF Schema จะมีความสัมพันธ์กันเป็น rdfs:subClassOf ส่วน คุณสมบัติ HasWriter จะถูกจัดวางให้อยู่ระหว่าง คลาส Writerและ คลาส Book โดย คุณสมบัติ HasWriter จะถูกกำหนดให้ใช้ โดเมน และ เรนจ์ ซึ่งจะถูกใช้ในกรณีที่จะอธิบายความสัมพันธ์ของ คลาส 2คลาส ซึ่งถูกคั่นด้วยโหนดคุณสมบัติ ซึ่งมีลักษณะคล้ายกับโครงสร้างRDF ที่ประกอบไปด้วย ประธาน,ภาคแสดง และ กรรม โดยโหนดคลาส ด้านหนึ่งเป็นส่วนของ ประธาน และ คลาส และ คลาส อีกด้านหนึ่งเป็นส่วนของ กรรม และมีโหนดคุณสมบัติ ที่ถูกคั่นอยู่จะเป็นส่วนของ ภาคแสดง ในโครงสร้าง RDF ซึ่งสามารถอธิบายความสัมพันธ์ระหว่างโหนด ได้ดังรูปที่ 3.22



รูป 3.22 การอธิบายความสัมพันธ์ของ คลาส 2 คลาส

จากรูป 3.22 โดเมน จะถูกใช้เพื่อบอกความสัมพันธ์ระหว่างโหนดคุณสมบัติ และคลาส ที่เป็นแหล่งข้อมูล ที่ถูกอธิบายหรือในส่วนของ ประธานจากในตัวอย่างคือ แหล่งข้อมูล (FamousWriter) เป็นคลาสมีความสัมพันธ์กับคุณสมบัติไปยังคลาส ที่เป็นค่าของคุณสมบัติหรือในส่วนของ กรรม จากในตัวอย่างคือ กรรม (Book) ซึ่งเป็นคลาสมีความสัมพันธ์กับ คุณสมบัติ(HasWritten) จากกราฟในรูป 3.22 สามารถนำไปเขียนเป็น RDF Document ได้ดังรูปที่ 3.23

```
<?xml version="1.0"?>
<rdf:RDF
          xmlns:rdf=" http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs=" http://www.w3.org/2000/01/rdf-schema#"
          xmlns:s=" http://www.bookonline.com"
<rdf:Description rdf:about= "http://famouswriters.org/twain/mark">
            <s:hasname>Mark Twain</s:hasname>
            <s:hasWritten rdf:resource=
                        "http://www.books.org/ISBN0000147582">
<rdf:type rdf:resource= http://www.w3.org/2000/01/rdf-
                                          schema#FamousWriters>
</rdf:Description>
<rdf:Description rdf:about=" http:// www.books.org/ISBN0000147582">
          <s:title>The Adventure of Tom Sawyer</s:title>
<rdf:type rdf:resource=" http://www.w3.org/2000/01/rdf-
                                                schema#Book">
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-
                                          schema#FamousWriters">
          <rdf:type rdf:resource=" http://www.w3.org/2000/01/rdf-
                                                schema#Writer">
</rdf:Description>
<rdf:Description
          rdf:about="http://www.movieworld.com/schema#haswritten">
          <rdfs:domain rdf:resource=
                    "http://www.w3.org/2000/01/rdf-schema#Writer">
          <rdfs:range rdf:resource=
                    "http://www.w3.org/2000/01/rdf-schema#Book">
</rdf:Description>
</rdf:RDF >
```

รูป 3.23 RDF Document ของกราฟที่อยู่ในรูป 3.22

จากรูปที่ 3.20 ส่วนของเอกสารที่อยู่ในกรอบสี่เหลี่ยมเส้นประจะเป็นส่วนของการกำหนดคำศัพท์ เพื่ออธิบายโครงสร้างของเมตะเดตา ในส่วนของ RDF Schema ซึ่งจะบ่งบอกถึงความหมายของข้อมูลลักษณะเฉพาะ และความสัมพันธ์ระหว่างกลุ่มของคุณสมบัติและกลายเป็นข้อมูลที่มีความหมาย

3.3 Web Ontology Language (OWL)

ความสามารถในการอธิบาย (Expressivity) ของ RDF และ RDF Schema ที่ได้กล่าวถึงในหัวข้อที่ผ่านมานั้นมีความจำกัดโดยตั้งใจมาก่อนแล้ว โดยที่ RDF นั้นถูกจำกัดให้ใช้กำหนดประโยค (Statement) ที่มีภาคแสดง (Predicate) เชื่อมระหว่างโหนดสองโหนด ส่วน RDF Schema นั้นถูกจำกัดให้กำหนดได้แค่ลำดับชั้นของคลาสย่อย (Subclass hierarchy) และลำดับชั้นของคุณสมบัติ (Property hierarchy) โดยที่สามารถระบุโดเมน (Domain) และเรนจ์ (Range) ของคุณสมบัติที่ต้องการกำหนดได้ด้วย

อย่างไรก็ดี คณะทำงานที่รับผิดชอบเรื่องเว็บออนโทโลยีจาก W3C (Web Ontology Working Group of W3C) ได้ระบุถึงกรณีการใช้งานกรณีพิเศษต่าง ๆ ที่จำเป็นต้องใช้ความสามารถมากกว่าที่มีใน RDF และ RDF Schema

กลุ่มนักวิจัยจำนวนหนึ่งทั้งที่อยู่ในสหรัฐอเมริกา และในทวีปยุโรปได้บ่งชี้ถึงความจำเป็นของภาษาที่นำมาใช้สร้างแบบจำลองออนโทโลยีที่มีความสามารถมากกว่า ทำให้เกิดการร่วมมือระหว่างกลุ่มวิจัยโดยมีจุดมุ่งหมายเพื่อนิยามภาษาใหม่ขึ้นมา ที่มีความสามารถมากกว่าภาษาที่มีอยู่เรียกว่า DAML+OIL ซึ่งเป็นชื่อที่เกิดจากการรวมกันระหว่างโครงงานในสหรัฐอเมริกาที่ชื่อว่า DAML-ONT และภาษาที่กลุ่มวิจัยจากยุโรปคิดค้นขึ้นชื่อ OIL

จุดเริ่มต้นในการนิยามภาษา OWL ขึ้นมาโดยคณะทำงานเว็บออนโทโลยีจาก W3C นั้นเริ่มมาจากการพัฒนาภาษา DAML+OIL ขึ้นมาโดยมีจุดมุ่งหมายเพื่อให้เป็นภาษามาตรฐาน และได้รับการยอมรับในวงกว้างในฐานะภาษาออนโทโลยี (Ontology language) เพื่อใช้ในการพัฒนาเว็บสื่อความหมาย (Semantic Web)

คำว่าออนโทโลยี (Ontology) หมายถึงรายการของแนวคิด และหมวดหมู่ต่าง ๆ ของสาขาวิชาหนึ่ง ๆ ซึ่งจะมีการแสดงความสัมพันธ์ระหว่างกัน

3.3.1 OWL และ RDF/RDFS

3.3.1.1 ความต้องการขั้นพื้นฐานของภาษาออนโทโลยี

ภาษาออนโทโลยี ยินยอมให้ผู้ใช้นำมาใช้เขียนแบบจำลองของเรื่องที่สนใจได้อย่างชัดเจน โดยความต้องการขั้นพื้นฐานหลัก ๆ คือ ไวยากรณ์ที่ชัดเจนเข้าใจง่าย สามารถสนับสนุนการ

ใช้เหตุผล (Reasoning) ได้อย่างมีประสิทธิภาพ สื่อความหมายอย่างเป็นทางการ (Formal semantics) มีความสามารถในการอธิบายมากพอ และความสะดวกสบายในการใช้อธิบาย

การให้ความสำคัญในความต้องการพื้นฐาน ในประเด็นที่ว่าจะต้องมีไวยากรณ์ที่ชัดเจนเข้าใจง่ายนั้นชัดเจน และเป็นที่รู้กันในสาขาวิชาภาษาโปรแกรม ซึ่งเป็นเงื่อนไขที่จำเป็นสำหรับเครื่องจักรที่จะมาประมวลผลข้อมูลความรู้ของภาษาต่าง ๆ ที่เราได้กล่าวถึงมาทั้งหมดจนถึงตอนนี้ต่างมีไวยากรณ์ที่ชัดเจนเข้าใจง่ายทั้งสิ้นทั้ง DAML+OIL และ OWL ต่างก็สร้างขึ้นมาจากรากฐานของภาษา RDF และ RDFS ซึ่งหมายความว่าภาษา OWL นั้นใช้ไวยากรณ์แบบเดียวกัน

เป็นที่ถกเถียงกันว่าไวยากรณ์ของ RDF ที่อิงพื้นฐานจากภาษา XML นั้นเป็นมิตรต่อผู้ใช้งานหรือไม่ โดยที่จริงยังมีทางเลือกอยู่ต่าง ๆ มากมายที่เหมาะกับผู้ใช้มนุษย์ แต่อย่างไรก็ตามข้อบกพร่องที่ว่านั้นนับว่าไม่ได้สำคัญเท่าไหร่นัก เนื่องจากผู้ใช้ส่วนมากนั้นจะพัฒนาออนโทโลยีของตัวเองโดยใช้เครื่องมือต่าง ๆ ช่วยอยู่แล้ว หรือที่เรียกว่าเครื่องมือพัฒนาออนโทโลยี

การสื่อความหมายอย่างเป็นทางการ คือ การอธิบายความหมายขององค์ความรู้อย่างถูกต้องแม่นยำ โดยคำว่า ถูกต้องแม่นยำในที่นี้หมายความหมาย ความหมายที่สื่อออกมานั้นจะไม่เปิดโอกาสให้มีการตีความได้หลายแบบโดยบุคคลที่ต่างกันไป หรือเครื่องจักรที่ต่างกันไป ซึ่งประโยชน์ที่ได้จากการสื่อความหมายอย่างเป็นทางการนี้ทำให้สามารถใช้เหตุผลสรุปเรื่องต่าง ๆ ขึ้นมาจากองค์ความรู้ได้

สำหรับในองค์ความรู้เกี่ยวกับออนโทโลยีเราสามารถสรุปประเด็นต่าง ๆ ได้ดังนี้

1) ความเป็นสมาชิกของคลาส (Class membership) หาก x เป็นอินสแตนซ์ (Instance) ของคลาส C และคลาส C เป็นคลาสย่อยของ คลาส D แล้ว สามารถสรุปได้ว่า x เป็นอินสแตนซ์ของคลาส D ด้วยได้

2) การเท่ากันระหว่างคลาส (Equivalence of classes) หากกล่าวว่าคลาส A นั้นเทียบเท่ากันกับคลาส B และคลาส B เทียบเท่ากันกับคลาส C แล้ว สามารถสรุปได้ว่าคลาส A เทียบเท่ากันกับคลาส C ด้วยได้

3) ความถูกต้อง (Consistency) สมมติให้ x เป็นอินสแตนซ์ของคลาส A โดยที่คลาส A เป็นคลาสย่อยของ $B \cap C$ และยังเป็นคลาสย่อยของคลาส D โดยที่คลาส B และคลาส D นั้นแยกจากกัน (Disjoint) พบว่าเกิดความไม่ถูกต้องขึ้นเนื่องจากคลาส A ควรจะเป็นคลาสว่างเปล่า แต่กลับมีอินสแตนซ์ x เกิดขึ้นมา ซึ่งเป็นตัวบ่งชี้ถึงข้อผิดพลาดในออนโทโลยีในตัวอย่างนี้

4) การจัดแบ่งประเภท(Classification)หากกำหนดให้คู่คุณสมบัติกับค่า (Property-value pair) จำนวนหนึ่งเป็นเงื่อนไขที่พอเพียงสำหรับการเป็นสมาชิกของคลาส A แล้ว หากมีอินดิวิดวล (Individual) x ที่ตอบสนองเงื่อนไขเหล่านั้นได้ เราสามารถสรุปได้ว่า x เป็นอินสแตนซ์ของคลาส A

การสื่อความหมายจัดว่าเป็นเงื่อนไขที่ต้องมีก่อนเพื่อรองรับการใช้เหตุผล (Reasoning support) ประเด็นการสรุปความทั้ง 4 ข้อที่กล่าวข้างต้นนั้นสามารถตรวจสอบได้โดยอาศัยเครื่องจักร แทนที่จะทำด้วยตนเอง ซึ่งการรองรับการใช้เหตุผลนั้นสำคัญเนื่องจาก

1) ช่วยตรวจสอบความถูกต้องของออนโทโลยี และองค์ความรู้ได้

2) ช่วยตรวจหาความสัมพันธ์อันไม่พึงประสงค์ระหว่างคลาสได้

3) ช่วยจำแนกคลาสของแต่ละอินสแตนซ์ได้โดยอัตโนมัติ

การรองรับการใช้เหตุผลโดยอัตโนมัติทำให้เราสามารถตรวจเจอกรณีต่าง ๆ ได้มากกว่าจากการทำด้วยตนเอง การตรวจสอบข้างต้นจัดว่ามีประโยชน์ในการออกแบบออนโทโลยีขนาดใหญ่ซึ่งต้องทำงานร่วมกับผู้ออกแบบหลายคน และยังมีประโยชน์ในกรณีที่ต้องผสาน หรือแบ่งปันออนโทโลยีระหว่างที่มาต่าง ๆ อีกด้วย โดยที่ในปัจจุบันมีเครื่องมือที่ทำหน้าที่นี้ให้ใช้อยู่ อย่างเช่น FaCT และ RACER เป็นต้น

3.3.1.2 ข้อจำกัดในความสามารถในการอธิบายของ RDF Schema

RDF และ RDFS สามารถนำไปใช้แสดงองค์ความรู้เชิงออนโทโลยีได้เพียงบางกรณีเท่านั้น องค์ประกอบหลัก ๆ ที่ใช้ในการนำเสนอของ RDF/RDFS จะพิจารณาเกี่ยวกับการจัดระบบกลุ่มคำศัพท์ (Vocabularies) ให้อยู่ในรูปลำดับชั้นที่มีชนิดกำกับ (Typed hierarchies) ดังเช่น ความสัมพันธ์ที่อธิบายถึงคลาสย่อย และคุณสมบัติย่อย ข้อกำหนดเกี่ยวกับโดเมน และเรนจ์ของคุณสมบัติ และอินสแตนซ์ของคลาสต่างๆ โดยจะเห็นได้ว่ามีลักษณะต่างๆ ที่ยังขาดหายไปเช่น

1) การกำหนดคุณสมบัติของคุณสมบัติที่มีขอบเขตเฉพาะคลาส (Local scope of properties) อย่างในคุณสมบัติ rdfs:range ภาษา RDFS ที่มีไว้สำหรับกำหนดเรนจ์ของคุณสมบัติ ยกตัวอย่างเช่น คุณสมบัติ eats นั้นจะมีผลกับทุก ๆ คลาสเราไม่สามารถเรนจ์ของคุณสมบัติที่มีผลกับคลาสเพียงบางคลาสได้ ตัวอย่างเช่นเราไม่สามารถกำหนดว่าวัวกินแต่พืชในขณะที่สัตว์อื่นๆกินเนื้อด้วย

2) ความแยกจากกันของคลาส (Disjointness of classes) บางทีเราปรารถนาที่จะกำหนดให้คลาสสองคลาสแยกจากกัน ยกตัวอย่างเช่น ต้องการกำหนดให้คลาส male และคลาส female นั้นแยกจากกัน แต่ใน RDF Schema ที่เราทำได้มีเพียงกำหนดให้เป็นคลาสย่อยได้เท่านั้น อย่างเช่น คลาส female เป็นคลาสย่อยของ คลาส person

3) การรวมกันด้วยบูลีนระหว่างคลาส (Boolean combinations of classes) บางทีเราปรารถนาที่จะสร้างคลาสใหม่ขึ้นมาโดยการกระทำการต่าง ๆ ระหว่างคลาสด้วย การยูเนียน (Union) อินเตอร์เซ็คชัน (Intersection) และคอมพลีเมนต์ (Complement) ยกตัวอย่างเช่น การนิยามคลาส person ขึ้นมาโดยกำหนดให้เป็นการยูเนียนกันระหว่างคลาส male และคลาส female โดยที่ทั้งสองคลาสต่างแยก

จากกัน ซึ่งไม่สามารถทำได้ใน RDF Schema

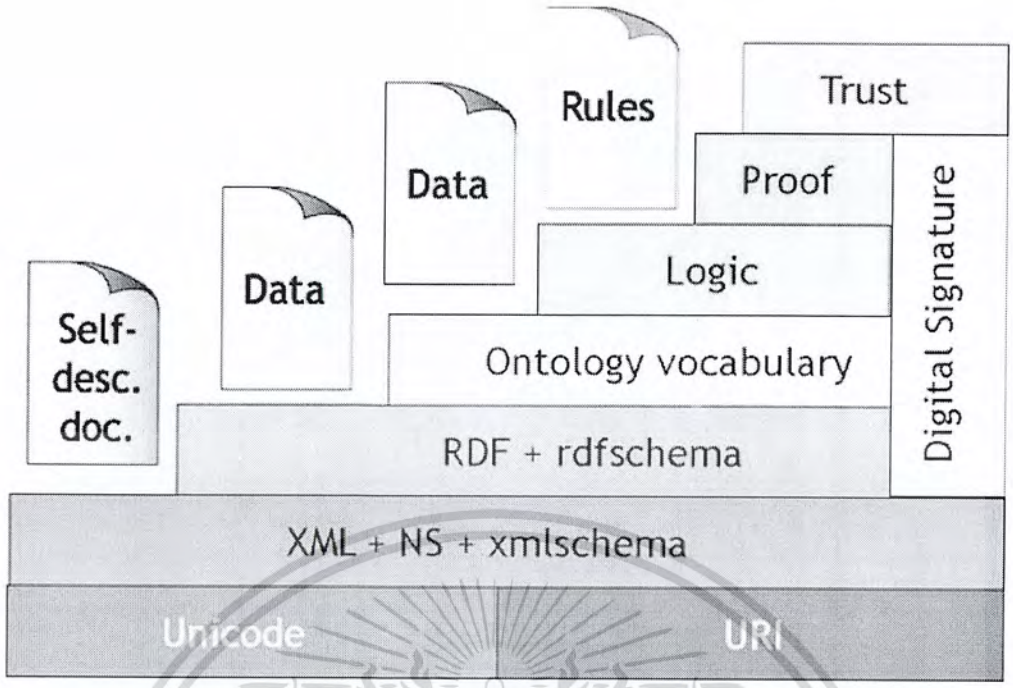
4) ข้อกำหนดจำนวนสมาชิก (Cardinality restrictions) บางทีเราปรารถนาที่จะวางข้อกำหนดว่าคุณสมบัติหนึ่ง ๆ ควรมีค่าเป็นจำนวนในช่วงหนึ่ง ๆ ยกตัวอย่าง เช่น เราต้องการกำหนดให้บุคคลบุคคลหนึ่งมีพ่อแม่ได้ 2 คนเท่านั้น หรือเช่น รายวิชาหนึ่ง ๆ จะต้องมีผู้สอนอย่างน้อยหนึ่ง 1 คน ซึ่ง RDF Schema ไม่สามารถตั้งข้อกำหนดประเภทนี้ได้

5) คุณลักษณะพิเศษของคุณสมบัติต่าง ๆ (Special characteristics of properties) บางทีการกำหนดให้คุณสมบัติหนึ่ง ๆ มีคุณสมบัติถ่ายทอด เช่น คุณสมบัติ "greater than" หรือมีคุณสมบัติเอกลักษณ์ เช่น คุณสมบัติ "is mother of" หรือคุณสมบัติย้อนกลับ เช่น คุณสมบัติ "eats" กับคุณสมบัติ "is eaten by" ทำให้เกิดประโยชน์นานัปการขึ้น โดยเฉพาะในแง่ของการรองรับการใช้เหตุผล

จากคุณลักษณะที่พร่องไปบางส่วนใน RDF Schema ที่ยกตัวอย่างข้างต้นในไปนั้น ทำให้เราต้องการภาษาออนโทโลยีที่มีความสามารถมากกว่าเดิม ซึ่งจะต้องมีคุณสมบัติเหล่านี้ และมากกว่า โดยในการออกแบบภาษาประเภทนี้ ผู้ออกแบบจะต้องคำนึงถึงความสมดุลระหว่างพลังในการอธิบาย และการรองรับการใช้เหตุผลที่มีประสิทธิภาพ กล่าวโดยทั่วไปแล้ว ยิ่งภาษามีคุณสมบัติในการอธิบายมากเท่าไร ก็ยิ่งทำให้การรองรับการใช้เหตุผลด้อยประสิทธิภาพลงเท่านั้น ซึ่งอาจส่งผลให้ไม่สามารถใช้เหตุผลได้เลยก็เป็นได้ ดังนั้นจึงตกลงกันว่า ภาษาที่ต้องการนั้นจะต้องมีความสมดุลในทั้งสองประเด็นเพื่อใช้ในการอธิบายออนโทโลยี และองค์ความรู้ได้มากประเภท

3.3.1.3 ความเข้ากันได้ของ OWL กับ RDF/RDFS

ตามหลักการแล้ว OWL จะเป็นส่วนขยายของ RDF Schema ในแง่ที่ว่า OWL จะใช้ความหมายของคลาส และคุณสมบัติที่มีให้อยู่ก่อนแล้วใน RDF อย่างเช่น rdfs:Class หรือ rdfs:subClassOf เป็นต้น และจะเพิ่มองค์ประกอบของภาษาเข้าไปอีกเพื่อให้รองรับกับความสามารถในการอธิบายที่เพิ่มขึ้น โดยที่ส่วนขยายของ RDF Schema นั้นจะต้องสอดคล้องกับสถาปัตยกรรมแบบลำดับชั้นของเว็บสื่อความหมายได้



รูป 3.24 สถาปัตยกรรมแบบลำดับชั้นของเว็บสื่อความหมาย

3.3.1.4 สามภาษาย่อยในภาษาตระกูล OWL

รูป 3.25 แสดงระดับความสามารถของภาษาย่อย

การจะทำให้ภาษาออนโทโลยี เป็นไปตามความต้องการขั้นพื้นฐานได้ทั้งหมดนั้น ดูเหมือนจะเป็นจริงได้ยาก นั่นคือ รองรับการใช้เหตุผลอย่างมีประสิทธิภาพ และสามารถใช้ในการอธิบายได้โดยง่าย ซึ่งเป็นคุณสมบัติที่เราต้องการ ในภาษาอันทรงพลังที่เกิดจากการผสานเข้ากับ RDF Schema และการใช้ตรรกะอย่างเต็มรูปแบบ ซึ่งจากความต้องการขั้นพื้นฐานดังกล่าวทำให้คณะทำงานเว็บออนโทโลยีจาก W3C ตัดสินใจที่จะนิยามภาษา OWL ให้มีภาษาย่อยต่างๆ 3 ภาษา โดยที่แต่ละภาษาถูกปรับปรุงให้รองรับความต้องการขั้นพื้นฐานในลักษณะต่างๆ ได้แก่

1) ภาษา OWL Fullภาษา OWL ที่สมบูรณ์เรียกว่าภาษา OWL Full และใช้องค์ประกอบของภาษา OWL ทั้งหมด โดยจะยอมให้มีการรวมกันระหว่างองค์ประกอบเหล่านี้เข้าด้วยกันเข้ากับ RDF และ RDF Schema ได้ตามอำเภอใจ

โดยรวมไปถึงความเป็นไปได้ที่จะไปเปลี่ยนความหมายดั้งเดิมขององค์ประกอบต่าง ๆ ของภาษา RDF หรือภาษา OWL ที่มีนิยามไว้ก่อนหน้า โดยการปรับใช้องค์ประกอบที่ว่าใส่กันเอง ยกตัวอย่างเช่น เราสามารถกำหนดข้อกำหนดจำนวนสมาชิกลงบนคลาสที่เป็นคลาสแม่ของทุกคลาสได้ ซึ่งก็คือการจำกัดจำนวนคลาสที่จะถูกนำมาอธิบายได้ ในออนโทโลยีนั้น ๆ นั่นเอง ข้อได้เปรียบของภาษา OWL Full คือมันเข้ากันได้กับ RDF อย่างเต็มรูปแบบในทิศทางชี้ขึ้นทั้งในแง่ของไวยากรณ์ และเชิงความหมาย เอกสาร RDF ที่ถูกต้องจะถือว่าเป็นเอกสาร OWL Full ที่ถูกต้องด้วยเช่นกัน ข้อเสียเปรียบของภาษา OWL Full คือภาษานี้ทรงพลังมากเกินไปที่จนไม่มีการใช้เหตุผลใดที่สมบูรณ์หรือมีประสิทธิภาพพอมารองรับ

2) ภาษา OWL DLOWL DL (OWL Description Logic) จัดเป็นภาษาย่อยของ OWL Full อีกที โดยมีการจำกัดวิธีการนำคอนสตรัคเตอร์ (Constructors) ต่าง ๆ จาก RDF และ OWL มาใช้งาน โดยหลัก ๆ แล้วจะไม่ยอมให้มีการนำคอนสตรัคเตอร์ของ OWL มาประยุกต์ใช้งานในระหว่างกันเองซึ่งเป็นการรับรองว่าตัวภาษาจะเข้ากันกับการอธิบายทางตรรกะ (Description Logic) ซึ่งเป็นงานวิจัยที่ศึกษาเกี่ยวกับการตัดสินใจในส่วนของ First order logic โดยกล่าวได้ว่าภาษา OWL DLนั้นถูกออกแบบมาเพื่อให้รองรับกับการอธิบายตรรกะของธุรกิจนั่นเอง ข้อได้เปรียบของภาษา OWL DL คือทำให้มีการใช้เหตุผลที่มีประสิทธิภาพ ส่วนข้อเสียเปรียบคือเราจะสูญเสียความเข้ากันได้กับภาษา RDF กล่าวคือเอกสาร RDF จะต้องถูกเพิ่มเติม โดยสรุปคือ เอกสาร OWL DL ที่ถูกต้องจัดว่าเป็นเอกสาร RDF ที่ถูกต้องเช่นกัน

3) ภาษา OWL Lite ภาษานี้เป็นภาษาย่อยของ OWL DL อีกชั้นหนึ่ง โดยจะมีข้อจำกัดมากขึ้นไปอีก ยกตัวอย่างเช่น OWL Lite จะไม่สามารถนิยามคลาสที่ถูกแจกแจง (Enumerated class) ได้ ไม่สามารถระบุความแยกจากกันระหว่างคลาสหรือระหว่างอินสแตนซ์ได้ ไม่สามารถกำหนดจำนวนสมาชิกได้ในแบบที่ต้องการ คือกำหนดจำนวนสมาชิกให้เป็นได้แค่ 0 หรือ 1 เท่านั้น ข้อได้เปรียบของภาษานี้คือทั้งเข้าใจง่ายสำหรับผู้ชาย และนำไปอิมพลีเมนต์ลงในเครื่องมือต่าง ๆ ได้ง่าย ส่วนข้อเสียเปรียบคือความสามารถในการอธิบายที่ถูกจำกัด

ผู้พัฒนาออนโทโลยีที่จะนำภาษา OWL มาใช้ควรพิจารณาว่าภาษาย่อยใดที่เข้ากับความต้องการมากที่สุด โดยการเลือกระหว่างภาษา OWL Lite และภาษา OWL DL จะขึ้นอยู่กับว่าผู้ใช้งานต้องการใช้เค้าโครงที่มีความสามารถในการอธิบายที่มีให้ในภาษา OWL DL มากเพียงใด

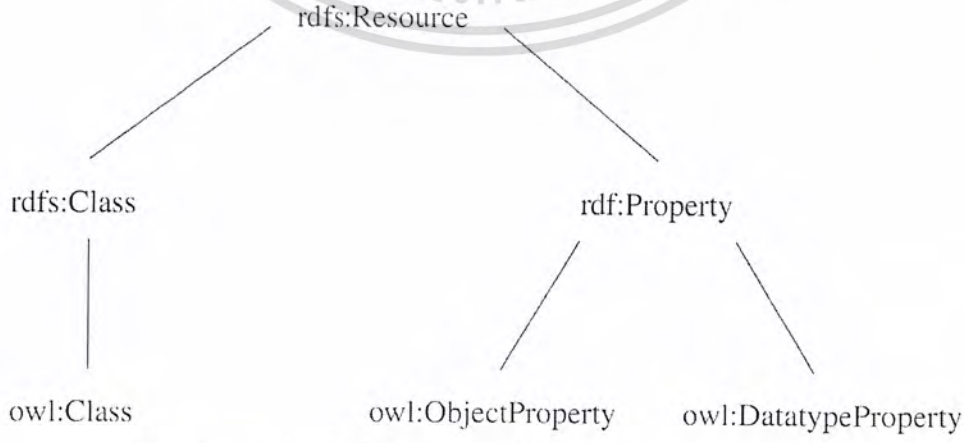
ส่วนการตัดสินใจเลือกระหว่างภาษา OWL DL และ OWL Full จะขึ้นอยู่กับว่าผู้ใช้จำเป็นต้องการใช้เครื่องมือในการทำเมตาโมเดลลิง (Metamodeling) ที่มีใน RDF Schema มากเพียงใด ยกตัวอย่าง เช่น การนิยามคลาสของคลาส หรือการผูกติดคุณสมบัติเข้ากับคลาส เป็นต้น ในการใช้งานภาษา OWL Full เทียบกับการใช้งานภาษา OWL DL นั้นจะพบว่าการรองรับการใช้เหตุผลจะสามารถคาดเดาได้ยากกว่า อันเนื่องมาจากการอิมพลีเมนต์ภาษา OWL Full ได้อย่างสมบูรณ์นั้นในปัจจุบันเป็นเรื่องที่เป็นไปไม่ได้

ความเข้ากันได้ภายในระหว่าง 3 ภาษาย่อยแบบทิศขึ้นโดยเริ่มจากภาษา OWL Lite ไปยังภาษา OWL Full มีดังนี้

1) ทุก ๆ ออนโทโลยีที่ใช้ภาษา OWL Lite นิยามได้ถูกต้องจัดว่าเป็น ออนโทโลยีที่ถูกต้องที่ใช้ภาษา OWL DL นิยาม
2) ทุก ๆ ออนโทโลยีที่ใช้ภาษา OWL DL นิยามได้ถูกต้องจัดว่าเป็น ออนโทโลยีที่ถูกต้องที่ใช้ภาษา OWL Full นิยาม
3) ทุก ๆ กฎของ OWL Lite เป็นกฎของ OWL DL ด้วย
4) ทุก ๆ กฎของ OWL DL เป็นกฎของ OWL Full ด้วย

ภาษา OWL ยังคงใช้ RDF และ RDF Schema อยู่โดยส่วนใหญ่ โดยกล่าวได้ดังนี้

1) ทุกภาษาย่อยของภาษา OWL ใช้ไวยากรณ์ของภาษา RDF
2) อินสแตนซ์ถูกประกาศด้วยภาษา RDF โดยใช้ RDF ในการอธิบายคุณสมบัติต่างๆและในการกำกับข้อมูลของชนิดอินสแตนซ์
3) คอนสตรัคเตอร์ของภาษา OWL เช่น owl:Class, owl:DatatypeProperty และ owl:ObjectPropertyมีความเฉพาะเจาะจงมากกว่าคอนสตรัคเตอร์ที่คล้ายกันในภาษา RDF



รูป 3.26 ความสัมพันธ์เชิงคลาสระหว่าง OWL และ RDF/RDFS

ในรูป 3.26 แสดงถึงองค์ประกอบพื้นฐานต่าง ๆ ที่ใช้ในการสร้างแบบจำลององค์ความรู้ของทั้งใน OWL และ RDF/RDFS

3.3.2 ลักษณะของภาษา OWL

3.3.2.1 ไวยากรณ์

ภาษา OWL ถูกสร้างอยู่บนรากฐานของ RDF และ RDF Schema และใช้ไวยากรณ์ RDF ที่มีพื้นฐานมาจากภาษา XML ด้วยความที่ RDF/XML

3.3.2.2 ส่วนหัว (Header)

เอกสาร OWL มักจะถูกเรียกว่า OWL ontologies และยังเป็นเอกสาร RDF ในขณะเดียวกัน รูตอิลิเมนส์ (Root element) ของ OWL ontology คือ อิลิเมนส์ (Element) rdf:RDF ซึ่งมีการระบุเนมสเปสที่จะนำมาใช้ภายในเอกสารอีกด้วย ดังรูป 3.27

```
<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#">
```

รูป 3.27 ตัวอย่างรูตอิลิเมนส์

OWL ontology อาจเริ่มต้นด้วยการทำการยืนยัน (Assertion) ชุดหนึ่งเพื่อความเป็นระบบระเบียบ โดยที่การยืนยันเหล่านี้จะถูกจับกลุ่มให้อยู่ภายใต้อิลิเมนส์ owl:Ontology ซึ่งประกอบไปด้วยคอมเมนต์ (Comments) การควบคุมเวอร์ชัน (Version control) และการรวมเอาออนโทโลยีอื่นเข้ามา ดังรูป 3.28

```
<owl:Ontology rdf:about="">
  <rdfs:comment>AnexampleOWLontology</rdfs:comment>
  <owl:priorVersion
       rdf:resource="http://www.mydomain.org/uni-ns-
old"/>
  <owl:imports
       rdf:resource="http://www.mydomain.org/persons"/>
  <rdfs:label>UniversityOntology</rdfs:label>
</owl:Ontology>
```

รูป 3.28 ตัวอย่างอิลิเมนส์ owl:Ontology

การยืนยันที่ส่งผลกระทบต่อความหมายเชิงตรรกะของออนโทโลยีคือ owl:imports ซึ่งจะแสดงรายการของออนโทโลยีอื่น ๆ ที่สันนิษฐานว่ามีเนื้อหาเป็นส่วนหนึ่งของออนโทโลยีที่กำลังพิจารณาอยู่ สังเกตว่าเนมสเปสมีไว้สำหรับขจัดความกำกวม ส่วนออนโทโลยีที่ถูกนำเข้ามาใช้

จะให้นิยามต่าง ๆ ที่สามารถนำมาใช้ได้ภายในออนโทโลยี และสังเกตอีกว่า owl:imports นั้นเป็นคุณสมบัติแบบถ่ายทอด นั่นคือหากออนโทโลยี A นำเข้าออนโทโลยี B และออนโทโลยี B นำเข้าออนโทโลยี C แล้ว หมายความว่าออนโทโลยี A ได้นำเข้าออนโทโลยี C เข้ามาด้วย

3.3.2.3 อิลิเมนส์คลาส (Class Elements)

คลาสจะถูกนิยามด้วยการใช้อิลิเมนส์ owl:Class ซึ่งเป็นคลาสย่อยของ rdfs:Class อีกทีหนึ่ง ยกตัวอย่างเช่น เราสามารถนิยามคลาส associateProfessor ได้ดังรูป 3.29

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

รูป 3.29 ตัวอย่างการใช้อิลิเมนส์ owl:Class นิยามคลาส associateProfessor

เรายังสามารถระบุเพิ่มเติมได้อีกว่าคลาสนี้แยกจากกัน ระหว่างคลาส associateProfessor กับคลาส professor โดยการใช้อิลิเมนส์ owl:disjointWith โดยที่เราสามารถเพิ่มอิลิเมนส์นี้เข้าไปในการนิยามคลาสที่ผ่านมาได้ หรือจะระบุเพิ่มเติมเข้าไปในภายหลังได้โดยการอ้างถึงคลาสด้วย ID โดยใช้อิลิเมนส์ rdf:about ซึ่งเป็นกลไกที่ได้รับการสืบทอดมาจาก RDF ดังที่ได้แสดงในรูป 3.30

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

รูป 3.30 ตัวอย่างการใช้อิลิเมนส์ owl:disjointWith

ความเท่าเทียมกันของคลาสสามารถนิยามได้ โดยการใช้อิลิเมนส์ owl:equivalentClass ดังรูป 3.31 ซึ่งเป็นการประกาศคลาสใหม่เพิ่มขึ้นมา แต่มีคุณลักษณะเหมือนกับคลาส faculty ทุกประการ

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

รูป 3.31 ตัวอย่างการใช้อิลิเมนส์ owl:equivalentClass

ท้ายสุดนี้มีคลาสอยู่ 2 คลาสที่ถูกนิยามเอาไว้ก่อนแล้ว ได้แก่ owl:Thing และ owl:Nothing ซึ่งคลาสแรกเป็นคลาสที่สามัญ (General) ที่สุดซึ่งรวมเอาทุกอย่างเข้าไว้ในคลาสนี้ ส่วนคลาสที่สองเป็นคลาสว่างเปล่า หรือกล่าวได้ว่า ทุก ๆ คลาสเป็นคลาสย่อยของคลาส owl:Thing และเป็นคลาสแม่ของ owl:Nothing นั่นเอง

3.3.2.4 อิลิเมนส์คุณสมบัติ (Property Elements)

ใน OWL สามารถแบ่งคุณสมบัติได้ออกเป็น 2 ประเภทหลักดังนี้

1) คุณสมบัติอ็อบเจ็คต์ (Object properties) ซึ่งสร้างความสัมพันธ์ขึ้นระหว่างออบเจ็คต์ 2 ออบเจ็คต์ ตัวอย่างเช่น คุณสมบัติ isTaughtBy และคุณสมบัติ supervises

2) คุณสมบัติชนิดข้อมูล (Data type properties) ซึ่งสร้างความสัมพันธ์ระหว่างออบเจ็คต์ และค่าข้อมูลที่มีชนิดหนึ่ง ๆ (Data type values) อย่างเช่นคุณสมบัติ phone title และ age ใน OWL ไม่มีชนิดข้อมูลใด ๆ ถูกนิยามเอาไว้อยู่ก่อน และไม่มีสิ่งที่อำนวยต่อการนิยามชนิดข้อมูลที่ว่าขึ้นมาด้วย แต่จะใช้ชนิดข้อมูลที่นิยามไว้อยู่ก่อนแล้วใน XML Schema แทน ซึ่งนับว่าเป็นการใช้ประโยชน์จากสถาปัตยกรรมแบบลำดับชั้นของเว็บสื่อความหมาย

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

รูป 3.32 ตัวอย่างนิยามคุณสมบัติชนิดข้อมูล

โดยทั่วไปแล้วชนิดข้อมูลที่ผู้ใช้นิยามขึ้นมาเองมักจะเก็บเอาไว้ใน XML Schema จากนั้นจึงถูกดึงมาใช้ภายใน OWL Ontology ในภายหลัง

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

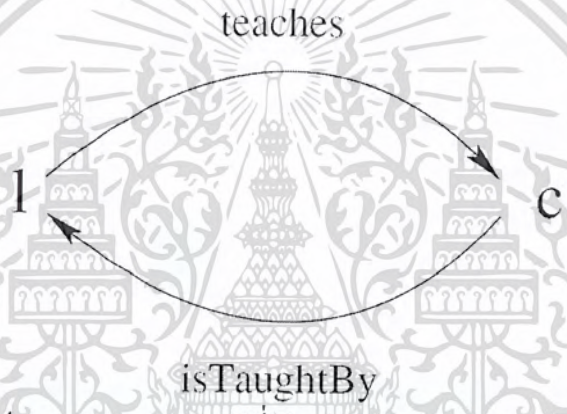
รูป 3.33 ตัวอย่างนิยามคุณสมบัติออบเจ็คต์

สามารถกำหนดโดเมน และเรนจ์ได้มากกว่าอย่างละ 1 อย่าง โดยในกรณีนี้จะเป็นการอินเตอร์เซ็คชันกันระหว่างโดนเมนเอง และระหว่างเรนจ์เองตามลำดับ

นอกจากนี้ OWL ยังยอมให้มีการเชื่อมโยงคุณสมบัติหนึ่งเข้าหาอีกคุณสมบัติที่เป็นส่วนกลับกันของคุณสมบัติแรกเข้าด้วยกันได้ โดยยกตัวอย่างเช่น คุณสมบัติ isTaughtBy และคุณสมบัติ teaches

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

รูป 3.34 ตัวอย่างเชื่อมเข้าหาคุณสมบัติที่เป็นส่วนกลับ



รูป 3.35 แสดงคุณสมบัติที่เป็นส่วนกลับของกันและกัน

รูป 3.35 แสดงถึงความสัมพันธ์ระหว่างคุณสมบัติ และคุณสมบัติส่วนกลับของตัวมันเอง ที่จริงแล้วโดเมน และเรนจ์ของคุณสมบัติหนึ่ง ๆ สามารถสืบทอดมาจากคุณสมบัติที่เป็นส่วนกลับกันได้ นั่นคือสลับโดเมน และเรนจ์ที่ได้รับการสืบทอดมานั่นเอง

ความเท่าเทียมกันของคุณสมบัติสามารถนิยามได้โดยการใช้อิลิเมนส์ owl:equivalentProperty ดังรูป 3.36

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

รูป 3.36 ตัวอย่างเชื่อมเข้าหาคุณสมบัติที่เป็นส่วนกลับ

3.3.2.5 ข้อกำหนดต่าง ๆ บนคุณสมบัติ (Property Restrictions)

ด้วยอิลิเมนส์ rdfs:subClassOf เราสามารถระบุว่าคลาส C เป็นคลาสย่อยของคลาส C' ดังนั้นอินสแตนซ์ของคลาส C จัดว่าเป็นอินสแตนซ์ของคลาส C' ด้วย

สมมติว่าเราแทนที่เราประกาศว่าคลาส C รองรับเงื่อนไขชุดหนึ่ง เราสามารถประกาศว่าอินสแตนซ์ของคลาส C รองรับเงื่อนไขชุดดังกล่าวแทนได้ ซึ่งก็เท่ากับการบอกว่าคลาส C เป็นคลาสย่อยของคลาส C' โดยที่คลาส C' จะสะสมออบเจ็คต์ทั้งหมดที่รองรับกับเงื่อนไขชุดดังกล่าว ซึ่ง OWL จะใช้วิธีการอ้างเช่นนี้ โดยสังเกตว่าโดยทั่วไปแล้วคลาส C' จะเป็นคลาสนิรนาม ตัวอย่างเช่นโค้ดดังรูป3.37ระบุรายวิชาของนักศึกษาปีแรกจะต้องถูกสอนโดยศาสตราจารย์เท่านั้น

```
<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>
```

รูป 3.37 ตัวอย่างการระบุคลาสย่อยนิรนาม

จากรูปที่ 3.37 เราประกาศคลาส firstYearCourse ให้เป็นคลาสย่อยของคลาสนิรนามซึ่งมีกฎเกณฑ์ที่บังคับว่าจะต้องเชื่อมโยงกับคุณสมบัติ isTaughtBy โดยที่ค่าที่คุณสมบัติเชื่อมไปหานั้นจะต้องเป็นอินสแตนซ์ของคลาส Professor โดยการใช้คุณสมบัติของ OWL ที่เรียกว่า owl:allValuesFrom ซึ่งเป็นกฎเกณฑ์ที่ใช้ในการจำแนกคลาสของอินสแตนซ์หนึ่ง ๆ นั่นเอง

owl:allValuesFrom ใช้ในการกำหนดคลาสของค่าที่คุณสมบัติหนึ่ง ๆ จะเชื่อมโยงไปที่ระบุด้วย owl:onProperty ซึ่งก็คือกำหนดคลาส หรือชนิดข้อมูลของเรนจ์ของคุณสมบัติดังกล่าวที่มีผลต่อคลาสหนึ่ง ๆ นั่นเอง

เราอาจประกาศให้รายวิชาเกี่ยวกับสาขาวิชาคณิตถูกสอนโดย David Billington ได้ดังรูป 3.38

```
<owl:Class rdf:about="#mathCourse">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isTaughtBy"/>
      <owl:hasValue rdf:resource="#949318"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

รูป 3.38 ตัวอย่างกำหนดค่าตายตัวให้แก่คุณสมบัติ

owl:hasValue จะใช้ในการกำหนดค่าของคุณสมบัติที่ต้องมีที่ถูก owl:onProperty หยิบยกขึ้นมากล่าวถึง และหากเราประกาศให้อาจารย์ในคณะทุกคนจะต้องสอนวิชาในชั้นปริญญาตรีอย่างน้อย 1 วิชา ทำได้ดังรูป 3.39

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

รูป 3.39 ตัวอย่างการใช้ owl:someValuesFrom

จากการเปรียบเทียบการใช้ owl:allValuesFrom และ owl:someValuesFrom ในรูป 3.37 แล้วรูป 3.39 ตามลำดับนั้น สังเกตได้ว่า การที่เรากำหนดให้ทุก ๆ คนที่มีคุณสมบัติสอนอินสแตนซ์ของคลาสรายวิชาชั้นปีที่ 1 เป็นศาสตราจารย์นั้น ในแง่ของตรรกศาสตร์จัดได้ว่าเป็นการกำหนดตัวบ่งปริมาณสำหรับทุกตัว (Universal quantification) ส่วนในรูป 3.39 นั้น เรากำหนดไว้ว่าอินสแตนซ์ของคลาสอาจารย์ในคณะจะต้องสอนรายวิชาระดับชั้นปริญญาตรีอย่างน้อย 1 วิชา ซึ่งก็เป็นไปได้ว่าอินสแตนซ์นั้น ๆ อาจจะสอนรายวิชาอื่นที่เป็นของระดับชั้นปริญญาโทก็ได้ ในกรณีนี้ในแง่ของตรรกศาสตร์จัดได้ว่าเป็นการกำหนดตัวบ่งปริมาณสำหรับบางตัว (Existensial quantification)

โดยทั่วไปแล้วอิลิเมนส์ owl:Restriction จะมีอิลิเมนส์ owl:Property บรรจุอยู่ภายใน แล้วตามด้วยการประกาศข้อกำหนดต่าง ๆ ตั้งแต่หนึ่งข้อขึ้นไป ซึ่งที่ผ่านมาเราได้แสดงข้อกำหนดชนิดหนึ่งที่กำหนดชนิดของค่าที่คุณสมบัติหนึ่ง ๆ จะเชื่อมโยงไปหาได้ โดยการใช้ owl:allValuesFrom, owl:hasValue, และ owl:someValuesFrom นั่นเอง

นอกจากนี้ยังมีข้อกำหนดอีกชนิดหนึ่งที่เอาไว้ใช้ในการกำหนดจำนวนสมาชิก (Cardinality restrictions) ยกตัวอย่างเช่น กำหนดให้รายวิชาหนึ่ง ๆ จะต้องถูกสอนโดยใครก็ได้อย่างน้อยหนึ่งคนดังรูป 3.40

```
<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>
```

รูป 3.40 ตัวอย่างการกำหนดจำนวนสมาชิกขั้นต่ำด้วย OWL

สังเกตว่าเราจะต้องกำกับไว้ด้วยว่าค่าตามตัวอักษร (Literal) 1 ที่ใช้ในกำกับจำนวนสมาชิกอย่างน้อยของคุณสมบัติ isTaughtBy ในรูป 3.40 ว่าจะต้องตีความแบบชนิดข้อมูล xsd:nonNegativeInteger ซึ่งเป็นชนิดข้อมูลแบบเลขจำนวนเต็มที่ไม่ติดลบที่ได้ประกาศเอาไว้อยู่ก่อนแล้วใน XML Schema โดยเราจะอ้างถึงเนมสเปส xsd ในค่าของแอตทริบิวท์ rdf:dataType ในลักษณะที่มันเป็นเอนทิตี (Entity) ที่ประกาศเอาไว้ในส่วนหัวของเอกสาร OWL

เราอาจกำหนดจำนวนสมาชิกขั้นต่ำ และขั้นสูงได้พร้อม ๆ กัน ดังเช่นตัวอย่างในรูป 3.41 ที่กำหนดให้ภาควิชาหนึ่ง ๆ มีสมาชิกอยู่ในช่วงไม่ต่ำกว่า 10 คน และไม่มากกว่า 30 คน

```
<owl:Class rdf:about="#department">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        10
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        30
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

รูป 3.41 ตัวอย่างการกำหนดจำนวนสมาชิกขั้นต่ำ และขั้นสูง

จากรูป 3.41 สังเกตได้ว่าในการที่จะกำหนดขั้นต่ำ และขั้นสูงเราจะต้องทำการประกาศให้คลาสนั้นเป็นคลาสย่อยของ 2 คลาส โดยที่แต่ละคลาสจะมีการกำหนกฎเกณฑ์ว่าคลาส

ย่อยหนึ่งกำหนดจำนวนสมาชิกขั้นต่ำ และอีกคลาสย่อยใช้กำหนดจำนวนสมาชิกขั้นสูง หรือหากเราต้องการระบุจำนวนสมาชิกตายตัว ยกตัวอย่างเช่น กำหนดให้นักศึกษาปริญญาเอกจะต้องมีอาจารย์ที่ปรึกษา 2 คนเท่านั้น ซึ่งสามารถทำได้โดยกำหนดหมายเลขเดียวกันลงใน owl:minCardinality และ owl:maxCardinality หรือถ้าจะใช้ owl:cardinality เพียงตัวเดียวก็ได้เพื่อความสะดวก

จากตัวอย่างโค้ดในรูป 3.37 ถึงรูป 3.41 สังเกตได้ว่า owl:Restriction จะนิยามคลาสนิรนามขึ้นมาซึ่งไม่มีอัตลักษณ์ (ID) กำกับเอาไว้ และไม่ได้ถูกนิยามขึ้นมาด้วย owl:Class และมีขอบเขตเฉพาะบริเวณ (Local scope) นั่นคือเราจะใช้มันก็ต่อเมื่อจำเป็นต้องกำหนดข้อบังคับขึ้นมาทำให้ต่อจากนี้เมื่อเราเอ่ยคำว่า คลาส จะหมายถึงคลาสได้ 2 ประเภท อย่างแรกคือ คลาสที่นิยามขึ้นมาด้วย owl:Class ที่มีอัตลักษณ์กำกับเอาไว้ และอย่างที่สองคือคลาสนิรนามที่มีผลเฉพาะพื้นที่ (Local anonymous class) ซึ่งมีหน้าที่รวบรวมออบเจ็คต์ที่รองรับกับเงื่อนไขข้อกำหนดของคลาสนิรนามนั้น ๆ หรือเงื่อนไขข้อกำหนดที่เกิดจากการรวมกันระหว่างคลาสชนิดที่สอง

3.3.2.6 คุณสมบัติชนิดพิเศษ (Special Properties)

คุณสมบัติบางประการของอิลิเมนส์ คุณสมบัติสามารถระบุเพิ่มเติมเข้าไปได้โดยตรงในตอนนิยามคุณสมบัติขึ้นมาได้โดยตรงผ่าน rdf:type ได้ โดยคุณสมบัติพิเศษของอิลิเมนส์คุณสมบัติมีดังนี้

1) owl:TransitiveProperty เป็นการกำหนดให้คุณสมบัตินั้นเป็นคุณสมบัติถ่ายทอด ยกตัวอย่างเช่น คุณสมบัติ “has better grade than”, คุณสมบัติ “is taller than”, คุณสมบัติ “is ancestor of” เป็นต้น
2) owl:SymmetricProperty เป็นการกำหนดให้คุณสมบัตินั้น เป็นคุณสมบัติสมมาตร ยกตัวอย่างเช่น คุณสมบัติ “has same grade as”
3) owl:FunctionalProperty เป็นการกำหนดให้คุณสมบัตินั้น เป็นคุณสมบัติเชิงฟังก์ชัน (Functional properties) ซึ่งเป็นการกำหนดให้คุณสมบัติหนึ่ง ๆ มีการเชื่อมโยงไปหาค่าเพียงค่าเดียวสำหรับแต่ละออบเจ็คต์ ยกตัวอย่างเช่น คุณสมบัติ “age”, คุณสมบัติ “height”, คุณสมบัติ “directSupervisor” เป็นต้น แต่หากภายในเอกสาร OWL มีการเชื่อมโยงจากอ็อบเจ็คตผ่านคุณสมบัติเชิงฟังก์ชันไปหาค่ามากกว่าหนึ่งค่า ตัวใช้เหตุผล (Reasoner)สรุปว่าทุกอินสแตนซ์หรือค่าตามอักษรที่คุณสมบัติฟังก์ชันเชื่อมโยงไปหาเป็นสิ่งเดียวกัน
4) owl:InverseFunctionalProperty เป็นการกำหนดให้คุณสมบัติ ที่กำหนดว่าออบเจ็คต์ 2 ออบเจ็คต์ที่ต่างกันจะมีค่าเชื่อมโยงไปหาค่าเดียวกันไม่ได้ ยกตัวอย่างเช่น คุณสมบัติ “isTheSocialSecurityNumberFor” ซึ่งหมายเลขประกันสังคมหมายเลขหนึ่งจะต้องถูกเชื่อมโยงผ่านคุณสมบัตินั้นโดยออบเจ็คต์เพียงแค่ออบเจ็คต์เดียว

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

รูป 3.42 ตัวอย่างการกำหนดคุณสมบัติเพิ่มเติมลงไปในคุณสมบัติ

3.3.2.7 การรวมกันด้วยบูลีน (Boolean Combinations)

เราสามารถนิยามคลาสขึ้นมาใหม่ได้จากการรวมกันด้วยบูลีนระหว่างคลาสที่มีอยู่ก่อนแล้วโดยการยูเนียน, อินเตอร์เซ็คชัน, และคอมพลีเมนต์กันระหว่างคลาส ยกตัวอย่างเช่น นิยามให้คลาสรายวิชาแยกจากกันกับคลาสเจ้าหน้าที่ภาควิชา ดังในรูป 3.43

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

รูป 3.43 ตัวอย่างการนิยามคลาสด้วยการรวมกันด้วยบูลีน

จากการนิยามคลาสในรูป 3.43 เป็นการบอกว่าทุกรายวิชาเป็นอินสแตนซ์ของคลาสที่เป็นคอมพลีเมนต์ของคลาสเจ้าหน้าที่ภาควิชา หรือกล่าวอีกอย่างหนึ่งก็คือไม่มีอินสแตนซ์ของคลาสรายวิชาใดที่เป็นสมาชิกของคลาสเจ้าหน้าที่ภาควิชาด้วยนั่นเอง สังเกตว่าการนิยามรูปแบบนี้สามารถทำได้อีกวิธีโดยการใช้ owl:disjointWith ดังในรูปที่ 7

คลาสที่เกิดจากการรวมกันของคลาสที่ประกาศไว้อยู่ก่อนแล้วทำได้โดยการใช้ owl:unionOf ดังในรูป 3.44

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

รูป 3.44 ตัวอย่างการนิยามคลาสโดยการยูเนียน

การนิยามคลาสในรูป 3.44 นั้นไม่ได้บอกว่าคลาสที่นิยามขึ้นมาใหม่นั้นเป็นคลาสย่อยที่ได้จากการยูเนียน แต่แท้ที่จริงแล้วเป็นคลาสที่เท่ากับการยูเนียน หรือพูดได้ว่าเราได้บอกว่า

คลาสใหม่ที่นิยามขึ้นมานั้นมีความเท่าเทียมกันกับเหล่าคลาสที่นำมายูเนียนเข้าด้วยกัน นอกจากนี้เราไม่ได้ระบุว่าคลาสเจ้าหน้าที่ภาควิชา และคลาสนักศึกษานั้นแยกจากกัน นั่นหมายความว่าเจ้าหน้าที่ภาควิชาอาจจะเป็นนักศึกษาพร้อม ๆ กันได้

```
<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>
```

รูป 3.45 ตัวอย่างการนิยามคลาสโดยการอินเตอร์เซ็คชัน

การนิยามคลาสในรูป 3.45 นั้นเป็นการนิยามคลาสขึ้นมาจากการอินเตอร์เซ็คชันกันระหว่าง 2 คลาส โดยที่มีคลาสหนึ่งเป็นคลาสนิรนามที่ระบุว่าออบเจ็คต์ของคลาสนั้นจะต้องเป็นของภาควิชาวิทยาการคอมพิวเตอร์ และการนำมาอินเตอร์เซ็คชันกันกับคลาสคณาจารย์เพื่อให้อินสแตนซ์ของคลาสนี้เป็นคณาจารย์ของภาควิชาวิทยาการคอมพิวเตอร์

การรวมกันด้วยบูลีนสามารถซ้อนเข้าไปได้เรื่อย ๆ ดังตัวอย่างโค้ดในรูป 3.46 ที่นิยามคลาสเจ้าหน้าที่ฝ่ายบริหารให้เป็นเจ้าหน้าที่ภาควิชา โดยที่เจ้าหน้าที่ฝ่ายบริหารนี้ไม่ได้เป็นทั้งอาจารย์ และเจ้าหน้าที่ฝ่ายเทคนิค

```
<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>
```

รูป 3.46 ตัวอย่างการรวมกันด้วยบูลีนแบบซ้อน

3.3.2.8 การแจกแจงของคลาส (Enumerations)

เราสามารถแจกแจงสมาชิกของคลาสได้ในตอนที่นิยามคลาสขึ้นมาได้ด้วยการใช้อิลิเมนส์ owl:oneOf จากนั้นระบุรายการของอิลิเมนส์ไว้ภายใน ซึ่งหมายความว่าเราไม่สามารถสร้างอินสแตนซ์ของคลาสนั้น ๆ ให้มีอัตลักษณ์ที่ไม่มีอยู่ในรายการได้ ดังรูป 3.47

```
<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>
```

รูป 3.47 ตัวอย่างการแจกแจงของคลาส

3.3.2.9 การสร้างอินสแตนซ์จากคลาสที่ประกาศไว้ (Instances)

การประกาศอินสแตนซ์ทำได้โดยการประกาศด้วยภาษา RDF ดังรูป 3.48 และรูป 3.49 ซึ่งมีความหมายเดียวกันทุกประการ แต่แตกต่างกันที่การใช้ไวยากรณ์ของ XML ในการเขียน โดยในรูป 3.49 จะเป็นการสร้างอินสแตนซ์ด้วยการใช้ไวยากรณ์แบบย่อ โดยการย้ายค่าของ rdf:type จากในรูป 3.48 ไปเป็นแท็กเปิดของการสร้างอินสแตนซ์ในรูป 3.49

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

รูป 3.48 ตัวอย่างการสร้างอินสแตนซ์วิธีที่ 1

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

รูป 3.49 ตัวอย่างการสร้างอินสแตนซ์วิธีที่ 2

เราอาจระบุรายละเอียดเพิ่มเติมเกี่ยวกับอินสแตนซ์นั้น ๆ ได้ด้วยการเชื่อมโยงอินสแตนซ์เข้าหาค่า หรือออบเจ็คต์อื่น ๆ ผ่านคุณสมบัติที่ได้ประกาศเอาไว้ก่อนแล้วในออนโทโลยี ดังรูป 3.50

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

รูป 3.50 การระบุรายละเอียดเพิ่มเติมของอินสแตนซ์ด้วยการใช้คุณสมบัติที่ประกาศไว้ในออนโทโลยี

ถึงแม้จะมีการกำกับอัตลักษณ์ให้แก่ทุก ๆ อินสแตนซ์ก็ตาม ก็ไม่ได้หมายความว่าจะเป็นคนละอินสแตนซ์กันสำหรับใน OWL ดังนั้น หากเรากำหนดให้คุณสมบัติ isTaughtBy เป็นคุณสมบัติเชิงฟังก์ชัน ซึ่งหมายความว่ารายวิชาหนึ่ง ๆ จะถูกสอนได้เพียงอาจารย์ในภาควิชาเพียงคนเดียวเท่านั้น ดังรูป 3.51

```
<owl:ObjectProperty rdf:ID="isTaughtBy">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>
```

รูป 3.51 กำหนดให้คุณสมบัติ isTaughtBy เป็นคุณสมบัติเชิงฟังก์ชัน

จากนั้นทำการสร้างอินสแตนซ์ของรายวิชาหนึ่งขึ้นมา แล้วระบุว่ามีผู้สอน 2 คน ผ่านคุณสมบัติ isTaughtBy ที่ประกาศไว้ในรูป 3.51 ดังในรูป 3.52

```
<course rdf:ID="CIT1111">
  <isTaughtBy rdf:resource="#949318"/>
  <isTaughtBy rdf:resource="#949352"/>
</course>
```

รูป 3.52 สร้างอินสแตนซ์ของรายวิชาหนึ่งขึ้นมา และทำการกำหนดผู้สอนมากกว่า 1 คนให้

การกำหนดเช่นนี้ไม่ได้ทำให้โปรแกรมใช้เหตุผลแจ้งข้อผิดพลาดขึ้นมา นั่นก็เพราะตัวโปรแกรมจะสรุปเอาเองได้ว่าอินสแตนซ์ที่มีอัตลักษณ์ 949318 และอินสแตนซ์ที่มีอัตลักษณ์ 949352 นั้นเป็นอินสแตนซ์เดียวกัน เพื่อไม่ให้เกิดข้อสรุปแบบนี้ขึ้นมาได้ เราต้องหาทางระบุให้ชัดเจนถึงความแตกต่างของแต่ละอินสแตนซ์กัน โดยการใช้ owl:differentFrom ในตอนสร้างอินสแตนซ์ขึ้นมาดังรูป 3.53

```
<lecturer rdf:ID="949318">
  <owl:differentFrom rdf:resource="#949352"/>
</lecturer>
```

รูป 3.53 การระบุความแตกต่างระหว่างอินสแตนซ์

การไล่ระบุความแตกต่างกันระหว่างอินสแตนซ์ด้วย owl:differentFrom ในแต่ละอินสแตนส์ที่เกี่ยวข้องจำเป็นต้องใช้สเตทเมนต์มากเกินความจำเป็น ซึ่ง OWL ก็มีวิธีการระบุความแตกต่างกันระหว่างอินสแตนซ์ที่กระทัดรัดกว่าโดยการระบุรายการของอินสแตนส์ที่แตกต่างกันด้วย owl:AllDifferent ซึ่งมี owl:distinctMembers บรรจุอยู่ภายใน ดังรูป 3.54

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

รูป 3.54 การระบุความแตกต่างระหว่างอินสแตนซ์ด้วยวิธีที่รวบรัดกว่า

สังเกตว่า owl:distinctMembers จะสามารถนำมาใช้ร่วมกับ owl:AllDifferent ได้เท่านั้น

3.3.2.10 ชนิดข้อมูลที่ใช้ได้ใน OWL (Data types)

ถึงแม้ว่าเราจะสามารถสร้างชนิดข้อมูลขึ้นมาเองได้ด้วย XML Schema Definition ยกตัวอย่างเช่น ชนิดข้อมูล adultAge ที่สืบทอดมาจากชนิดข้อมูล integer แต่กำหนดเงื่อนไขว่าจะต้องมีค่ามากกว่า 18 ขึ้นไป หรือชนิดข้อมูลที่สืบทอดมาจากสตริงที่กำหนดเงื่อนไขว่าจะต้อง

ขึ้นต้นด้วยตัวเลขเสมอ OWL ไม่รองรับการใช้ชนิดข้อมูลที่นิยามขึ้นมาเองเหล่านี้ รวมกระทั่งหลาย ๆ ชนิดข้อมูลที่นิยามเอาไว้ก่อนหน้าใน XML Schema ด้วย โดยทั่วไปแล้ว OWL รองรับชนิดข้อมูลทั่ว ๆ ไปที่มักถูกใช้บ่อย ๆ อย่างเช่นข้อความ จำนวนเต็ม จำนวนทศนิยม เวลา วันที่ เป็นต้น

3.3.2.11 ข้อมูลเกี่ยวกับรุ่นเอกสาร (Versioning Information)

จากรูป 3.28 เราใช้ owl:priorVersion เป็นส่วนหนึ่งของส่วนหัวของเอกสารเพื่อใช้บ่งชี้ถึงออนโทโลยีรุ่นก่อนหน้าเทียบกับออนโทโลยีที่กำลังทำงานด้วย ณ ปัจจุบัน ข้อมูลนี้ไม่ได้สื่อความหมายใด ๆ ตัวโปรแกรมใช้เหตุผล แต่มีประโยชน์ต่อมนุษย์ที่มาอ่านออนโทโลยี และโปรแกรมที่มีหน้าที่ในการจัดการบริหารออนโทโลยี

ส่วน owl:versionInfo โดยทั่วไปจะบรรจุข้อความสำหรับให้ข้อมูลเกี่ยวกับออนโทโลยีรุ่นปัจจุบัน

owl:backwardCompatibleWith บรรจุการอ้างอิงไปยังออนโทโลยีอื่น โดยสื่อความหมายว่า ออนโทโลยีที่อ้างอิงถึงเป็นออนโทโลยีรุ่นก่อนหน้านั้น และยังมีความเข้ากันได้กับออนโทโลยีรุ่นก่อนอีกด้วย ซึ่งหมายความว่าอัตลักษณ์ต่าง ๆ ในออนโทโลยีรุ่นก่อนต่างก็มีความหมายเหมือนกันกับออนโทโลยีปัจจุบันทุกประการ ทำให้สามารถตีความด้วยวิธีเดิมได้ ซึ่งเป็นการบอกใบ้ให้ผู้จัดทำเอกสารสามารถปรับเปลี่ยนไปใช้ออนโทโลยีรุ่นใหม่ได้อย่างปลอดภัย โดยการแก้เนมสเปซที่ประกาศเอาไว้ และแก้ URL ของออนโทโลยีที่อ้างถึงใน owl:imports

owl:incompatibleWith เป็นการระบุว่าออนโทโลยีที่กำลังพิจารณานั้นเป็นออนโทโลยีรุ่นถัดไปของออนโทโลยีที่อ้างอิงถึง และไม่มีความเข้ากันได้กับออนโทโลยีรุ่นก่อนแล้ว โดยทั่วไปแล้วผู้ออกแบบออนโทโลยีจะใช้เพื่อบอกให้ชัดเจนว่าเอกสารอื่น ๆ ที่อ้างอิงออนโทโลยีรุ่นก่อนหน้า จะอ้างอิงถึงออนโทโลยีรุ่นปัจจุบันไม่ได้หากไม่ได้ทำการตรวจสอบความถูกต้องเสียก่อน

```
<owl:Ontology rdf:about="">
  <rdfs:comment>Vehicle Ontology, v. 1.1</rdfs:comment>
  <owl:backwardCompatibleWith
     rdf:resource="http://www.example.org/vehicle-1.0"/>
  <owl:priorVersion
     rdf:resource="http://www.example.org/vehicle-1.0"/>
</owl:Ontology>
```

รูป 3.55 ตัวอย่างการระบุข้อมูลเกี่ยวกับรุ่นของเอกสาร

3.3.3 ลำดับชั้นของ OWL (Layering of OWL)

ที่ผ่านมาเราได้อธิบายโครงสร้างต่าง ๆ ของภาษา OWL ไปแล้ว ต่อไปนี้จะเป็นการระบุถึงคุณลักษณะต่าง ๆ ที่แต่ละภาษาย่อยในภาษาตระกูล OWL สามารถใช้งานได้

3.3.3.1 ภาษา OWL Full

ภาษา OWL Full รองรับการใช้งานทุกโครงสร้างในทุกรูปแบบตราบที่เอกสารนั้นยังถูกต้องตามไวยากรณ์ของภาษา RDF

3.3.3.2 ภาษา OWL DL

ข้อจำกัดต่าง ๆ ในภาษา OWL DL มีดังนี้

1) การแบ่งกลุ่มคำศัพท์ (Vocabulary partitioning) ทรัพยากรหนึ่ง ๆ ภายในเอกสารสามารถเป็นได้เพียงหนึ่งในรายการต่อไปนี้ คลาส ชนิดข้อมูล คุณสมบัติชนิดข้อมูล คุณสมบัติออบเจ็คต์ อินสแตนซ์ ค่าข้อมูล หรือเป็นส่วนหนึ่งของกลุ่มคำศัพท์ที่นิยามไว้ให้อยู่แล้ว เท่านั้น นั่นหมายความว่า คลาสคลาสหนึ่งไม่สามารถเป็นอินสแตนซ์ได้ในเวลาเดียวกัน หรือคุณสมบัติหนึ่ง ๆ ไม่สามารถเชื่อมโยงไปยังออบเจ็คต์ และค่าข้อมูลได้ เพราะนั่นหมายความว่า คุณสมบัติที่ว่าจะเป็นทั้งคุณสมบัติชนิดข้อมูล และคุณสมบัติออบเจ็คต์ในเวลาเดียวกัน

2) การต้องระบุชนิดข้อมูล (Explicit typing) ไม่เพียงแต่ทรัพยากรหนึ่ง ๆ จะต้องถูกจัดแบ่งให้เข้าที่เข้าทางแล้ว แต่การจัดแบ่งนี้จะต้องถูกระบุเอาไว้อย่างชัดเจน ยกตัวอย่างเช่น ออนโทโลยีหนึ่งมีการนิยามคลาส C1 ดังในรูป 3.56 นั่นหมายความว่า ตามหลักเหตุผลแล้วอัตลักษณ์ C2 ที่ถูกอ้างถึงจะต้องเป็นอัตลักษณ์ของคลาสด้วย

```
<owl:Class rdf:ID="C1">
  <rdfs:subClassOf rdf:about="#C2" />
</owl:Class>
```

รูป 3.56 การนิยามคลาส C1

ไม่ใช่เพียงเพราะเหตุผลที่ว่าเรนจ์ของ rdfs:subClassOf คือ rdfs:Class แต่จะต้องมีการนิยามคลาส C2 ภายในออนโทโลยีเอาไว้อย่างชัดเจนดังรูป 3.57

```
<owl:Class rdf:ID="C2"/>
```

รูป 3.57 การนิยามคลาส C2 เพื่อความถูกต้อง

3) การแยกกันต่างหากของคุณสมบัติ (Property seperation) เนื่องมาจากข้อจำกัดข้อที่ 1 ซึ่งกล่าวไว้ว่าชุดของคุณสมบัติออบเจ็คต์ และชุดของคุณสมบัติชนิด

ข้อมูลนั้นแยกจากกันให้สรุป ได้ว่าคุณสมบัติพิเศษที่กำหนดลงบนคุณสมบัติ owl:inverseOf, owl:FunctionalProperty,owl:InverseFunctionalProperty, และ owl:SymmetricProperty ไม่สามารถใช้ได้กับคุณสมบัติชนิดข้อมูลได้

4) ไม่สามารถระบุข้อจำกัดจำนวนสมาชิกลงบนคุณสมบัติสมมาตรได้

5) การนิยามคลาสนิรนามทำได้อย่างจำกัดนั่นคือคลาสนิรนาม สามารถเป็นโดเมน และเรนจ์ของ owl:equivalentClass และ owl:disjointWith ได้ และเป็นเรนจ์ของ rdfs:subClassOf ได้เท่านั้น

3.3.3.3 ภาษา OWL Lite

ออนโทโลยีที่ใช้ภาษา OWL Lite อธิบายได้นั้นจะต้องเป็นออนโทโลยีที่ภาษา OWL DL อธิบายได้ด้วย นอกไปจากนี้ยังมีเงื่อนไขเพิ่มเติมที่ต้องทำตามดังนี้

1) ไม่สามารถใช้งานโครงสร้างดังนี้ได้ คือ owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf, และ owl:hasValue

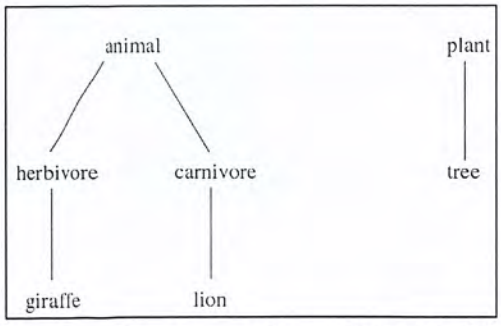
2) การกำหนดจำนวนสมาชิกสามารถกำหนดจำนวนต่ำสุด จำนวนสูงสุด หรือกำหนดจำนวนตายตัวได้เพียง 0 และ 1 เท่านั้น

3) ไม่สามารถใช้ owl:equivalentClass กับคลาสนิรนามได้ ใช้ได้เฉพาะในการอ้างอิงถึงคลาสอื่นที่อัตลักษณ์กำกับ

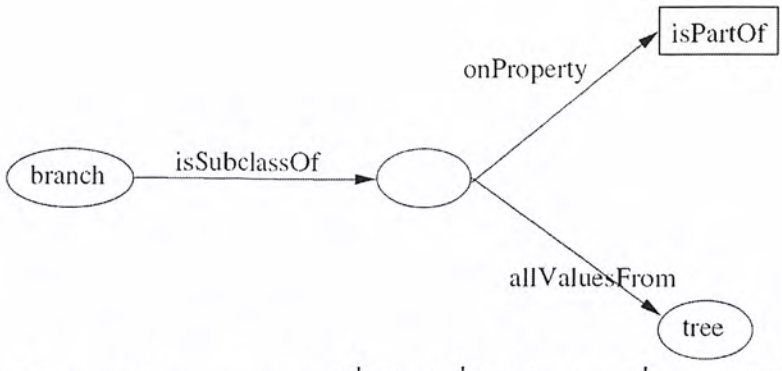
3.3.4 ตัวอย่างออนโทโลยี

3.3.4.1 ออนโทโลยีสัตว์ป่าในทวีปแอฟริกา

ออนโทโลยีนี้จะสร้างกรอบแนวคิดเกี่ยวกับสัตว์ป่าในทวีปแอฟริกาโดยรูป 3.58 แสดงความสัมพันธ์ระหว่างคลาสต่าง ๆ ภายในออนโทโลยี สังเกตว่าคลาสย่อยในรูปจะแสดงคลาสย่อยเพียงบางส่วนเท่านั้น ยังมีมากกว่านั้นภายในออนโทโลยี ส่วนรูป 3.59 แสดงสเตทเมนต์ในรูปแบบกราฟแบบมีทิศทางสำหรับในการนิยามคลาสกิ่งไม้ซึ่งบ่งบอกว่ากิ่งไม้เป็นส่วนหนึ่งของต้นไม้ และข้อคิดเห็นต่าง ๆ ที่อยู่ภายในออนโทโลยีดังในรูป 3.60 จะใส่ลงไปในอิลิเมนส์ rdfs:comment



รูป 3.58 ความสัมพันธ์ระหว่างคลาสภายในออนโทโลยีสัตว์ป่าในทวีปแอฟริกา



รูป 3.59 กราฟแสดงสเตทเมนต์ที่กล่าวว่ากิ่งไม้เป็นส่วนหนึ่งของต้นไม้

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

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

  <owl:Class rdf:ID="plant">
    <rdfs:comment>
      Plants form a class disjoint from animals.
    </rdfs:comment>
    <owl:disjointWith rdf:resource="#animal"/>
  </owl:Class>
  <owl:Class rdf:ID="tree">
    <rdfs:comment>Trees are a type of plant.</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#plant"/>
  </owl:Class>
  <owl:Class rdf:ID="branch">
    <rdfs:comment>Branches are parts of trees.</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#is_part_of"/>
        <owl:allValuesFrom rdf:resource="#tree"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="leaf">
    <rdfs:comment>Leaves are parts of branches.</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#is_part_of"/>
        <owl:allValuesFrom rdf:resource="#branch"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

รูป 3.60 ออนโทโลยีสัตว์ป่าในทวีปแอฟริกา

```
  <owl:Class rdf:ID="herbivore">
    <rdfs:comment>
      Herbivores are exactly those animals that eat only plants or
parts of plants.
    </rdfs:comment>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#animal"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#eats"/>
          <owl:allValuesFrom>
            <owl:Class>
              <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#plant"/>
                <owl:Restriction>
                  <owl:onProperty rdf:resource="#is_part_of"/>
                  <owl:allValuesFrom rdf:resource="#plant"/>
                </owl:Restriction>
              </owl:unionOf>
            </owl:Class>
          </owl:allValuesFrom>
        </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
  <owl:Class rdf:ID="carnivore">
    <rdfs:comment>
      Carnivores are exactly those animals that eat animals.
    </rdfs:comment>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#animal"/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#eats"/>
        <owl:someValuesFrom rdf:resource="#animal"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
  <owl:Class rdf:ID="giraffe">
    <rdfs:comment>
      Giraffes are herbivores, and they eat only leaves.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#herbivore"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#eats"/>
        <owl:allValuesFrom rdf:resource="#leaf"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="lion">
    <rdfs:comment>
      Lions are animals that eat only herbivores.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#carnivore"/>
    <rdfs:subClassOf>
```

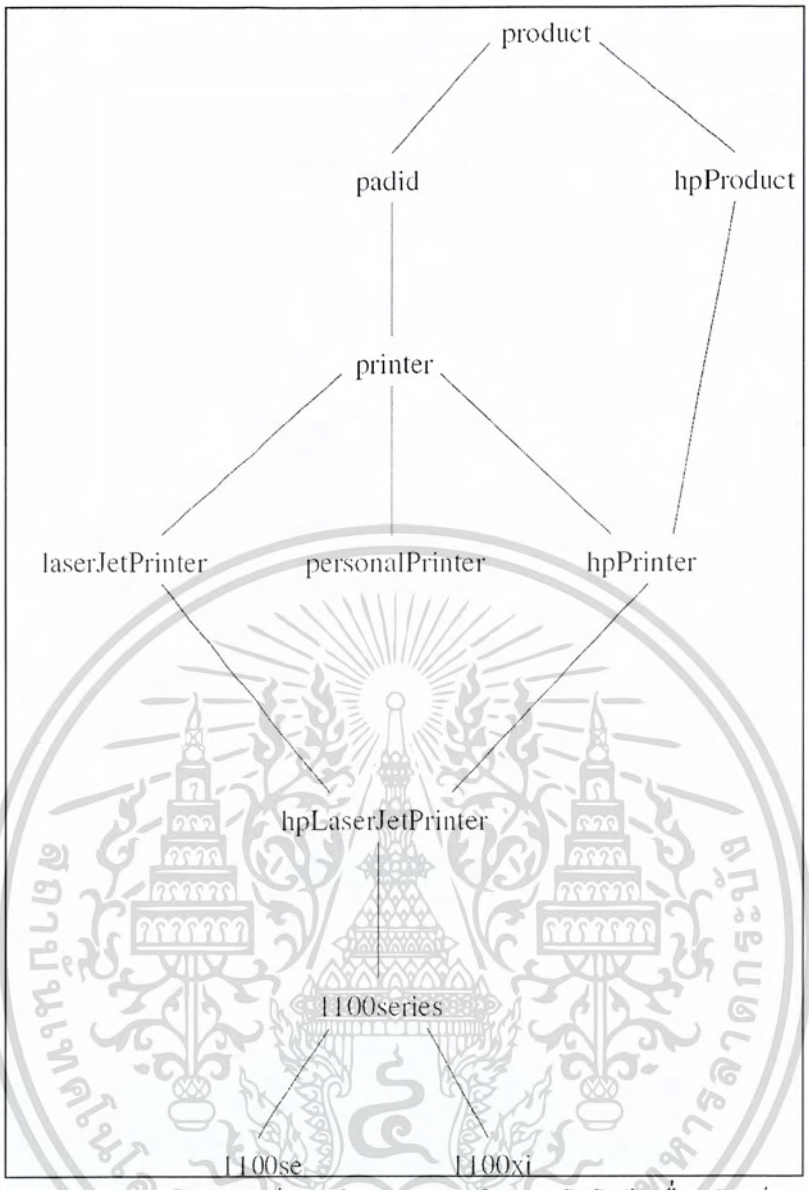
รูป 3.60 ออนโทโลยีสัตว์ป่าในทวีปแอฟริกา (ต่อ)

```
      <owl:Restriction>
        <owl:onProperty rdf:resource="#eats"/>
        <owl:allValuesFrom rdf:resource="#herbivore"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

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

  <owl:TransitiveProperty rdf:ID="is_part_of"/>

  <owl:ObjectProperty rdf:ID="eats">
    <rdfs:domain rdf:resource="#animal"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="eaten_by">
    <owl:inverseOf rdf:resource="#eats"/>
  </owl:ObjectProperty>

</rdf:RDF>
```

รูป 3.60 ออนโทโลยีสัตว์ป่าในทวีปแอฟริกา (ต่อ)

3.3.4.2 ออนโทโลยีเครื่องพิมพ์ (A Printer Ontology)

ความสัมพันธ์ระหว่างคลาสภายในออนโทโลยีนี้แสดงอยู่ในรูป 3.61 และออนโทโลยีแสดงอยู่ในรูป 3.62



รูป 3.61 ความสัมพันธ์ระหว่างคลาสภายในออนโทโลยีเครื่องพิมพ์

```
<!DOCTYPE owl [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns="http://www.cs.vu.nl/~frankh/spool/printer.owl#">
<owl:Ontology rdf:about="">
  <owl:versionInfo>
    My example version 1.2, 17 October 2002
  </owl:versionInfo>
</owl:Ontology>
```

รูป 3.62 ออนโทโลยีเครื่องพิมพ์

```
<owl:Class rdf:ID="product">
  <rdfs:comment>Products form a class.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="padid">
  <rdfs:comment>
    Printing and digital imaging devices form a subclass of
products.
  </rdfs:comment>
  <rdfs:label>Device</rdfs:label>
  <rdfs:subClassOf rdf:resource="#product"/>
</owl:Class>
<owl:Class rdf:ID="hpProduct">
  <rdfs:comment> HP products are exactly those products that
are manufactured by Hewlett Packard.
  </rdfs:comment>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#product"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#manufactured_by"/>
      <owl:hasValue rdf:datatype="&xsd;string">
        Hewlett Packard
      </owl:hasValue>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
<owl:Class rdf:ID="printer">
  <rdfs:comment>
    Printers are printing and digital imaging devices.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#padid"/>
</owl:Class>
<owl:Class rdf:ID="personalPrinter">
  <rdfs:comment>
    Printers for personal use form a subclass of printers.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#printer"/>
</owl:Class>
<owl:Class rdf:ID="hpPrinter">
  <rdfs:comment>
    HP printers are HP products and printers.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#printer"/>
  <rdfs:subClassOf rdf:resource="#hpProduct"/>
</owl:Class>
<owl:Class rdf:ID="laserJetPrinter">
  <rdfs:comment>
    Laser jet printers are exactly those printers that use
laser jet printing technology.
  </rdfs:comment>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#printer"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#printingTechnology"/>
      <owl:hasValue rdf:datatype="&xsd;string"> laser jet
      </owl:hasValue>
```

รูป 3.62 ออนโทโลยีเครื่องพิมพ์ (ต่อ)

```
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
<owl:Class rdf:ID="hpLaserJetPrinter">
  <rdfs:comment> HP laser jet printers are HP products and
laser
    jet printers.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#laserJetPrinter"/>
  <rdfs:subClassOf rdf:resource="#hpPrinter"/>
</owl:Class>
<owl:Class rdf:ID="1100series">
  <rdfs:comment> 1100series printers are HP laser jet
printers with 8ppm printing speed and 600dpi printing
resolution.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#hpLaserJetPrinter"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#printingSpeed"/>
      <owl:hasValue
rdf:datatype="&xsd;string">8ppm</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#printingResolution"/>
      <owl:hasValue rdf:datatype="&xsd;string"> 600dpi
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="1100se">
  <rdfs:comment> 1100se printers belong to the 1100 series
and cost $450.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#1100series"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasValue rdf:datatype="&xsd;integer"> 450
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="1100xi">
  <rdfs:comment> 1100xi printers belong to the 1100 series
and cost $350.
  </rdfs:comment>
  <rdfs:subClassOf rdf:resource="#1100series"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasValue rdf:datatype="&xsd;integer">350
      </owl:hasValue>
```

รูป 3.62 ออนโทโลยีเครื่องพิมพ์ (ต่อ)

```
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="manufactured_by">
  <rdfs:domain rdf:resource="#product"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="price">
  <rdfs:domain rdf:resource="#product"/>
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="printingTechnology">
  <rdfs:domain rdf:resource="#printer"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="printingResolution">
  <rdfs:domain rdf:resource="#printer"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="printingSpeed">
  <rdfs:domain rdf:resource="#printer"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

</rdf:RDF>
```

รูป 3.62 ออนโทโลยีเครื่องพิมพ์ (ต่อ)

จุดประสงค์ที่ออนโทโลยีนี้ต้องการแสดง คือคลาสย่อยที่สืบทอดมาจากคลาสแม่เดียวกันไม่จำเป็นต้องแยกจากกัน ยกตัวอย่างเช่น เครื่องพิมพ์ส่วนตัวอาจจะเป็นเครื่องพิมพ์ยี่ห้อเอชพี หรือเครื่องพิมพ์เลเซอร์เจ็ตก็ได้ ถึงแม้ว่าทั้ง 3 คลาสย่อยจะสืบทอดมาจากคลาสเครื่องพิมพ์ที่เป็นคลาสของเครื่องพิมพ์ทั้งหมดภายในออนโทโลยี

3.3.5 นิยามภาษา OWL ด้วยตัวภาษา OWL เอง (OWL in OWL)

ในหัวข้อนี้จะแสดงนิยามของภาษา OWL ด้วยตัวภาษา OWL เองซึ่งนี่ไม่ใช่นิยามเต็ม แต่นำมาแสดงเพื่ออธิบายแง่คิดต่าง ๆ เกี่ยวกับ OWL ที่ยังไม่ได้อธิบายถึง

3.3.5.1 เนมสเปส (Namespaces)

URI ของเอกสารปัจจุบันซึ่งก็คือเอกสารที่นิยามภาษา OWL นี้จะถูกนิยามให้เป็นเนมสเปสดั้งเดิม (Default namespace) ดังนั้นจะไม่มีการใช้คำขึ้นต้น (Prefix) owl: ภายในเอกสารและสังเกตถึงการนิยามเอนทิตีเอาไว้ในเอกสาร ซึ่งมีไว้สำหรับย่อค่าของแอทริบิวต์ภายในเอกสารให้สั้นลงนั่นเอง

```
<?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/">
```

รูป 3.63 การประกาศเนมสเปสต่าง ๆ ภายในเอกสาร

3.3.5.2 คลาสของคลาส หรือเมตาคลาส (Classes of Classes)

คลาสที่เป็นคลาสของทุกคลาสในภาษา OWL นั้นเป็นคลาสย่อยของคลาสที่เป็นคลาสของทุกคลาสในภาษา RDF Schema ดังรูป 3.64

```
<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>
```

รูป 3.64 การนิยามคลาสของทุกคลาสในภาษา OWL

ส่วนคลาส Thing ซึ่งจัดว่าเป็นคลาสของออบเจ็คต์ที่ทั่วไปที่สุดในภาษา OWL และ Nothing ที่เป็นคลาสที่เฉพาะเจาะจงที่สุด คือเป็นคลาสย่อยที่ว่างเปล่าของทุกคลาส ซึ่งมีความสัมพันธ์ดังรูป 3.65 ซึ่งนิยามด้วยภาษา OWL ได้ดังรูป 3.66

$$\text{Thing} = \text{Nothing} \cup \text{Nothing}^{c}$$
$$\text{Nothing} = \text{Thing}^{c} = \text{Nothing}^{c} \cap \text{Nothing}^{cc} = \emptyset$$

รูป 3.65 ความสัมพันธ์ระหว่างคลาส Thing และคลาส Nothing

```
<Class rdf:ID="Thing">
  <rdfs:label>Thing</rdfs:label>
  <unionOf rdf:parseType="Collection">
    <Class rdf:about="#Nothing"/>
    <Class>
      <complementOf rdf:resource="#Nothing"/>
    </Class>
  </unionOf>
</Class>

<Class rdf:ID="Nothing">
  <rdfs:label>Nothing</rdfs:label>
  <complementOf rdf:resource="#Thing"/>
</Class>
```

รูป 3.66 นิยามคลาส Thing และคลาส Nothing

3.3.5.3 ความเท่าเทียมกันของคลาส (Class Equivalence)

เราระบุความเท่าเทียมกันของคลาสโดยใช้ owl:EquivalenceClass ซึ่งทำให้มีความสัมพันธ์แบบคลาสย่อยโดยนัย ซึ่งจะใช้ในการระบุระหว่างคลาส 2 คลาส ซึ่งคล้ายกับ owl:EquivalenceProperty ส่วนความแยกจากกันระหว่างคลาสใช้ระบุได้ในระหว่างกลุ่มคลาส โดยการนิยามทั้ง 3 อย่างเป็นดังรูป 3.67

```
<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>
```

รูป 3.67 นิยามของ owl:EquivalenceClass, owl:EquivalenceProperty และ owl:disjointWith

ความเท่ากัน และความไม่เท่ากันสามารถนำมาใช้ระบุกับสิ่งใด ๆ ก็ได้โดยในภาษา OWL Full สเตทเมนต์ที่ใช้ในการระบุประเด็นเหล่านี้สามารถนำมาใช้กับคลาสได้ ส่วน

owl:sameAs มีคุณสมบัติเหมือนกับ owl:saveIndividualAs ทุกประการโดยนิยามของทั้ง 3 อย่างเป็นดังรูป 3.68

```
<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>
```

รูป 3.68 นิยามของ owl:sameIndividualAs, owl:differentFrom, และ owl:sameAs

ส่วนคุณสมบัติ owl:distinctMembers จะมีไว้สำหรับใช้กับคลาส owl:AllDifferent เท่านั้น ดังนิยามในรูป 3.69

```
<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>
```

รูป 3.69 นิยามของ owl:distinctMembers และ owl:AllDifferent

3.3.5.4 การสร้างคลาสจากคลาสอื่น ๆ (Building Classes from Other Classes)

owl:unionOf จะทำการสร้างคลาสจากรายการคลาสดังนิยามในรูป 3.70

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

รูป 3.70 นิยามของ owl:unionOf

ส่วน owl:intersectionOf และ owl:oneOf จะใช้สำหรับนิยามคลาสใหม่ขึ้นมาจากรายการเหมือนกัน เพียงแต่เป็นรายการของอินสแตนซ์ ส่วน owl:complementOf จะใช้ในการนิยามคลาสโดยอิงจากอีกคลาสหนึ่งโดยมีนิยามตามในรูป 3.71

```
<rdf:Property rdf:ID="complementOf">
  <rdfs:label>complementOf</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>
```

รูป 3.71 นิยามของ owl:complementOf

3.3.5.5 การตั้งข้อกำหนดลงบนคลาส (Restricting Properties of Classes)

ข้อกำหนดต่าง ๆ ใน OWL ใช้ในการนิยามคลาสที่มีอ็อบเจ็คต์ที่รองรับตามเงื่อนไขที่แนบมากับคลาสนั้น ๆ โดยมีนิยามดังในรูป 3.72

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

รูป 3.72 นิยามของ owl:Restriction

คุณสมบัติที่จะแสดงนิยามต่อไปนี้สามารถระบุได้เฉพาะในการนิยามข้อกำหนดของคลาสเท่านั้น นั่นก็คือโดเมนของคุณสมบัติเหล่านี้จะต้องเป็น owl:Restriction เท่านั้น แต่จะมีเรนจ์ที่แตกต่างกันไปในแต่ละคุณสมบัติโดยนิยามอยู่ในรูป 3.73

```
<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="someValuesFrom">
  <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>

<rdf:Property rdf:ID="maxCardinality">
  <rdfs:label>minCardinality</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</rdf:Property>

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

รูป 3.73 นิยามของ owl:onProperty, owl:allValuesFrom, owl:someValuesFrom, owl:hasValue, owl:minCardinality, owl:maxCardinality, และ owl :cardinality

สังเกตว่า owl:Property จะใช้สำหรับกำหนดข้อกำหนดของคุณสมบัติอ็อบเจ็คต์และคุณสมบัติชนิดข้อมูลจึงมีเรนจ์ไปยัง rdf:Property ซึ่งไม่ได้เฉพาะเจาะจงไปที่ชนิดของคุณสมบัติ และก็เหมือนกับที่เรนจ์ของ owl:allValuesFrom และ owl:someValuesFrom ที่ไม่ได้เป็น owl:Class แต่เป็นคลาสที่เฉพาะเจาะจงน้อยกว่าคือ rdfs:Class

3.3.5.6 คุณสมบัติ (Properties)

owl:ObjectProperty เป็นกรณีพิเศษของ rdf:Property ดังนั้นในนิยามในรูป 3.74 จึงทำการนิยามให้เป็นคลาสย่อยของ rdf:Property

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

รูป 3.74 นิยามของ owl:ObjectProperty

ส่วนนิยามของ owl:DatatypeProperty จะมีนิยามลักษณะเหมือนกับนิยามของ owl:ObjectProperty ในรูป 3.74

ส่วน owl:TransitiveProperty, owl:SymmetricProperty, owl:FunctionalProperty, และ owl:InverseFunctionalProperty ที่เป็นคลาสของคุณสมบัติซึ่งมีคุณสมบัติพิเศษเพิ่มเข้ามาจะเป็นคลาสย่อยของคลาสคุณสมบัติอ็อบเจ็คต์ได้เท่านั้น ดังนิยามในรูป 3.75

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

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

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

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

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

รูป 3.75 นิยามของ owl:TransitiveProperty, owl:SymmetricProperty, owl:FunctionalProperty, และ owl:InverseFunctionalProperty

สุดท้ายนี้ owl:inverseOf จะเป็นคุณสมบัติที่ทำหน้าที่เชื่อมคุณสมบัติอ็อบเจ็คต์ 2 คุณสมบัติเข้าด้วยกันดังนิยามในรูป 3.76

```
<rdf:Property rdf:ID="inverseOf">
  <rdfs:label>inverseOf</rdfs:label>
  <rdfs:domain rdf:resource="#ObjectProperty"/>
  <rdfs:range rdf:resource="#ObjectProperty"/>
</rdf:Property>
```

รูป 3.76 นิยามของ owl:inverseOf

และข้อสังเกตที่จะแสดงในรูป 3.77 นี้เป็นจริง แต่ไม่ได้อธิบายเอาไว้ในเอกสารอ้างอิงออนไลน์

```
<TransitiveProperty rdf:ID="&rdfs;subClassOf"/>
<TransitiveProperty rdf:ID="&rdfs;subProperty/>

<TransitiveProperty rdf:ID="EquivalentClass"/>
<SymmetricProperty rdf:ID="EquivalentClass"/>

<SymmetricProperty rdf:ID="disjointWith"/>

<TransitiveProperty rdf:ID="EquivalentProperty"/>
<SymmetricProperty rdf:ID="EquivalentProperty"/>

<TransitiveProperty rdf:ID="sameIndividualAs"/>
<SymmetricProperty rdf:ID="sameIndividualAs"/>

<SymmetricProperty rdf:ID="differentFrom"/>

<SymmetricProperty rdf:ID="complementOf"/>

<rdf:Property rdf:about="complementOf">
  <rdfs:subPropertyOf rdf:resource="disjointWith"/>
</rdf:Property>

<rdf:Property rdf:about="cardinality">
  <rdfs:subPropertyOf rdf:resource="mincardinality"/>
  <rdfs:subPropertyOf rdf:resource="maxcardinality"/>
</rdf:Property>

<SymmetricProperty rdf:ID="inverseOf"/>

<rdf:Property rdf:about="inverseOf">
  <inverseOf rdf:resource="inverseOf"/>
</rdf:Property>
```

รูป 3.77 ข้อสังเกตเพิ่มเติมต่าง ๆ ของโครงสร้างต่าง ๆ ภายใน OWL

ถึงแม้ว่าข้อสังเกตต่าง ๆ ในรูป 3.77 จะนำความหมายที่สื่อในโครงสร้างต่าง ๆ ของ OWL ออกมาแสดง แต่ก็ไม่ได้แสดงความหมายทั้งหมด ดังนั้นจึงควรอ้างอิงเอกสารที่อธิบายมาตรฐาน OWL ด้วย

3.4 RDFlib

RDFLib เป็นไลบรารีสำหรับทำงานกับกราฟข้อมูล RDF ที่พัฒนาด้วยภาษาไพธอนที่มีขนาดไม่ใหญ่นัก สามารถดาวน์โหลดได้ที่เว็บไซต์ http://rdflib.net โดยความสามารถหลัก ๆ คือเราสามารถการสืบค้นข้อมูลจากกราฟ RDF ที่อ่านเข้ามาด้วยภาษา SPARQL และใช้ในการบันทึกกราฟข้อมูล RDF ออกเป็นไฟล์ในรูปแบบต่าง ๆ ได้แก่ RDF/XML, N-Triple, N3 เป็นต้น หรือสามารถเก็บบันทึกลงในระบบฐานข้อมูลต่าง ๆ ได้เช่น SQLite, MySQL, และ SleepyCat เป็นต้น ในหัวข้อนี้จะสาธิตวิธีการใช้งานไลบรารีพอสังเขป และจะพัฒนาโปรแกรมประยุกต์ตัวอย่างในการทดลองในภายหลัง

ตัวอย่างโค้ดในรูป 3.78 แสดงการอ่านข้อมูลกราฟ RDF ที่บันทึกอยู่ในอินเตอร์เน็ตแล้วทำการพิมพ์ทริปเปิลทั้งหมดที่อยู่ภายในกราฟออกมา โดยในบรรทัดที่ 3 ทำการประกาศอ็อบเจ็คต์ชนิด rdflib.Graph.ConjunctiveGraph ขึ้นมา แล้วสั่งให้อ่านข้อมูลกราฟ RDF ที่เก็บบันทึกอยู่บนอินเตอร์เน็ตพร้อมกำหนดรูปแบบให้เป็นแบบ N-Triple ในบรรทัดที่ 4 จากนั้นทำการพิมพ์ทริปเปิลทั้งหมดที่อยู่ในกราฟ g ทำให้รู้ว่าแต่ละทริปเปิลจะถูกบันทึกด้วยชนิดข้อมูลทัพเพิล (Tuple) ที่มีสมาชิกทั้งหมด 3 ตัว ดังรูป 3.79

```
>> import rdflib
>> from rdflib.Graph import ConjunctiveGraph
>> g = ConjunctiveGraph()
>> g.parse("http://semprog.com/people/colin", format="nt")
>> for triple in g:
>> ...   print triple
...
```

รูป 3.78 โค้ดการใช้งาน RDFlib

```
(rdflib.term.BNode('ub1bL9C10'),
rdflib.term.URIRef('http://xmlns.com/foaf/0.1/name'),
rdflib.term.Literal(u'Jamie Taylor'))
...
```

รูป 3.79 รูปแบบการเก็บทริปเปิลด้วย RDFlib

เราสามารถสืบค้นทริปเปิลภายในกราฟขึ้นมาได้โดยการกำหนดรูปแบบทริปเปิลโดยใช้เมธอด triples() ของคลาส Graph จากนั้นส่งผ่านทัพเพิลที่เก็บรูปแบบทริปเปิลที่เราต้องการสืบค้นเข้าไป ดังตัวอย่างในรูป 3.80 ซึ่งทำการสืบค้นหาทริปเปิลที่มีภาคแสดงเป็น foaf:knows แล้วทำการเก็บทริปเปิลที่สืบค้นมาได้ไว้ในลิสต์ (List)

```
>> list( g.triples((None,rdflib.URIRef('foaf:knows'),None)) )
```

รูป 3.80 โค้ดการสืบค้นด้วยเมธอด triples()

ส่วนตัวอย่างโค้ดในรูป 3.81 เป็นการสาธิตการบันทึกข้อมูลกราฟ ที่เราอ่านเก็บไว้ในหน่วยความจำด้วยอ็อบเจ็คต์ชนิด ConjunctiveGraph ชื่อ g ลงในไฟล์ชื่อ colin.xml ในรูปแบบ RDF/XML แทน โดยการสั่งเปิดไฟล์ในบรรทัดแรกเพื่อเขียนข้อมูล จากนั้นส่งผ่านสตริงที่เมธอด serialize ของอ็อบเจ็คต์ ConjunctiveGraph คืนกลับมาให้เข้าไปในเมธอด write() ของอ็อบเจ็คต์ชนิดไฟล์ (file)

```
>> outfile = open("colin.xml", "w")
>> outfile.write(g.serialize(format="pretty-xml"))
```

รูป 3.81 โค้ดการบันทึกข้อมูลกราฟ RDF ลงบนสื่อบันทึกข้อมูล

จากนั้นโค้ดในรูป 3.82 จะทำการอ่านข้อมูลกราฟ RDF เข้ามาจากไฟล์ที่บันทึกอยู่ในเครื่องเข้ามา สังเกตว่าไลบรารีจะทำการตรวจจับรูปแบบการบันทึกข้อมูลกราฟ RDF ให้เอง แล้วจากนั้นทำการแสดงข้อมูลของกราฟที่อ่านเข้ามาในแปรชนิด ConjunctiveGraph ชื่อ newg ในรูปแบบ N3 ในบรรทัดที่ 3

```
>> newg = ConjunctiveGraph()
>> newg.parse("colin.xml")
>> newg.serialize(format="n3")
```

รูป 3.82 โค้ดการอ่านข้อมูลกราฟ RDF โดยการอ่านข้อมูลที่บันทึกอยู่ในเครื่อง

สังเกตว่าข้อมูลกราฟที่เก็บอยู่ในอ็อบเจ็คต์ g และ newg นั้นคือกราฟเดียวกัน สังเกตได้จากการนำกราฟกราฟหนึ่งมาหักลบกับอีกกราฟหนึ่ง แล้วดูจำนวนทริปเปิลที่อ็อบเจ็คต์กราฟนั้น ๆ เก็บอยู่ โดยการเรียกฟังก์ชัน len() ดังแสดงในรูป 3.83 ซึ่งผลลัพธ์ที่ได้คือเลข 0 นอกจากนี้ยังสามารถนำกราฟมาผสานเข้ากันได้ด้วยการนำอ็อบเจ็คต์กราฟมาบวกกัน

```
>> newg -= g
>> len(newg)
0
```

รูป 3.83 โค้ดการกระทำกันระหว่างอ็อบเจ็คต์ชนิด Graph

จากนั้นทำสาธิตการเพิ่มทริปเปิลที่สื่อความหมายว่าเราเป็นเพื่อนกับ Colin ที่เป็นข้อมูลกราฟในอ็อบเจ็คต์ g โดยขั้นแรกให้ทำการกำหนด URI เพื่อใช้ในการระบุทรัพยากรที่บ่งบอกถึงตัวเราก่อน ซึ่งไลบรารี RDFLib ได้จัดเตรียมคลาส URIRef เอาไว้ให้ใช้ โดยการส่งสตริงเข้าไปในคอนสตรัคเตอร์ แล้วเก็บลงในตัวแปร me ดังรูป 3.84

```
>> me = URIRef("http://my.uriref.com/goes/here")
```

รูป 3.84 โค้ดการสร้างอ็อบเจ็คต์ชนิด URIRef

ต่อมาทำการสร้าง URIRef ที่สื่อถึงภาคแสดง rdf:type สำหรับใช้ในการบอกชนิดของอินสแตนซ์ภายในข้อมูลกราฟ RDF โดยการสร้างอ็อบเจ็คต์ชนิด Namespace ขึ้นมาพร้อมส่ง URI เข้าไปยังคอนสตรัคเตอร์ ซึ่งการใช้งานจะเหมือนกับตัวแปรชนิดข้อมูล dict ในภาษาไพธอนทุกประการ เพราะฉะนั้นในการสร้าง URIRef ขึ้นมาจากเนมสเปสที่ประกาศเอาไว้ในอ็อบเจ็คต์ทำได้โดยการส่งคีย์ไปยังอ็อบเจ็คต์ในลักษณะเดียวกันกับวิธีการใช้ตัวแปร dict ค่าที่ได้จะเป็นอ็อบเจ็คต์ชนิด URIRef ดังแสดงในรูป 3.85

```
>> RDF = rdflib.Namespace("http://www.w3.org/TR/rdf-schema/#")
>> rdf-type-predicate = RDF["type"]
```

รูป 3.85 โค้ดสร้างอ็อบเจ็คต์ชนิด Namespace และ URIRef จากเนมสเปสที่ประกาศไว้

เนื่องจากเนมสเปส FOAF ถูกนิยามเอาไว้ในข้อมูลกราฟ RDF ที่เก็บอยู่ในหน่วยความจำอยู่แล้ว เราสามารถดึงออกมาใช้ได้เลยโดยไม่ต้องประกาศขึ้นมาใหม่โดยการเรียกเมธอด namespaces โดยจะคืนทัพเพิลที่ประกอบไปด้วยสมาชิก 2 ตัว ตัวแรกคือชื่อเนมสเปส ตัวที่สองคือ URIRef ของเนมสเปสนั้น ๆ ทำให้ได้ URIRef ของเนมสเปส FOAF มาใช้ในการสร้างอ็อบเจ็คต์ชนิด Namespace ชื่อ foaf ดังโค้ดในรูป 3.86

```
>>> foafURIRef = [x for x in g.namespaces() if x[0] ==
u'foaf'][0][1]
>>> foaf = rdflib.Namespace(foafURIRef)
```

รูป 3.86 โค้ดสร้างอ็อบเจ็คต์ชนิด Namespace จาก URIRefที่นิยามไว้ก่อนแล้วในกราฟข้อมูล RDF

เมื่อได้เนมสเปสที่จำเป็นสำหรับการอ้างถึงภาคแสดงครบแล้ว ให้ทำการเพิ่มทริปเปิลที่มีการประกาศอินสแตนส์ตัวเราเข้าไป ซึ่งมีภาคแสดงเป็น rdf:type เชื่อมโยงไปหา foaf:person ตามด้วยทริปเปิลที่สื่อความหมายว่า Colin รู้จักอินสแตนซ์ที่แทนตัวเรา ดังโค้ดในรูป 3.87 โดยการเรียกเมธอด add ของคลาส ConjunctiveGraph เป็นอันเสร็จสิ้น

```
>> g.add((me, rdf-type-predicate, foaf["person"]))
>> g.add(URIRef("http://semprog.com/people/colin"),
foaf["knows"], me)
```

รูป 3.87 โค้ดสร้างอ็อบเจ็คต์ชนิด Namespace จากURIRef ของคลาส ConjunctiveGraph

สังเกตว่าเราสามารถใช้สตริงแทนการประกาศอ็อบเจ็คต์ชนิด URIRef แทนได้ แต่การกระทำบางอย่างที่มีไว้ให้ในไลบรารี RDFLib อาจไม่สามารถทำงานเข้ากันได้กับสตริงได้ จึงแนะนำให้สร้างอ็อบเจ็คต์ URIRef เสมอ

3.4.1 การเก็บข้อมูลกราฟ RDF ลงในระบบฐานข้อมูล

ข้อได้เปรียบอย่างหนึ่งในการทำงานกับข้อมูล RDF ด้วยไลบรารี RDFLib คือเราสามารถโหลดชุดทริปเปิลลงไปในตัวเก็บกราฟ (Graph store) ที่จะยังคงอยู่เมื่อโปรแกรมทำงานเสร็จสิ้นแล้ว ทำให้เราสามารถเขียนโค้ดที่ทำงานร่วมกับตัวเก็บกราฟ และใช้งานกราฟที่ได้เก็บบันทึกลงไปก่อนหน้านั้นแล้ว โดย RDFLib ออกแบบไลบรารีในลักษณะที่ยอมให้มีการเขียนปลั๊กอิน (Plug-in) เพิ่มเข้ามาได้ ทำให้เราสามารถเพิ่มตัวพาร์เซอร์ (Parser), ตัวแปลงรูปแบบกราฟ (Serializer), ตัวประสานงานกับระบบจัดเก็บข้อมูล (Storage) ซึ่งไลบรารี RDFLib นี้รองรับการทำงานร่วมกับระบบฐานข้อมูลเช่น MySQL และ SleepyCat เป็นต้น แต่ตัวอย่างการใช้งานที่จะแสดงนั้นจะทำการเก็บข้อมูลกราฟ RDF ผ่านระบบฐานข้อมูล SQLite เนื่องจากเป็นระบบฐานข้อมูลที่ใช้งานง่าย ทำงานได้บนหลายแพลตฟอร์ม (Platform) ทำงานอยู่ภายในโปรเซสของโปรแกรมประยุกต์นั้น ๆ และไม่จำเป็นต้องทำการตั้งค่าใด ๆ ก่อนการใช้งาน โดยอันดับแรกให้ติดตั้งแพ็คเกจ (Package) ชื่อ pysqlite ซึ่งสามารถดาวน์โหลดได้จาก http://pysqlite.org/ จากนั้นทำสร้างไฟล์ฐานข้อมูล SQLite ขึ้นมาแล้วทำการเพิ่มทริปเปิลลงไปเพื่อใช้ในการบันทึกลงในฐานข้อมูลดังที่แสดงในรูป 3.88 สังเกตพารามิเตอร์ (Parameter) create ในบรรทัดที่ 3 การระบุให้มีค่าความจริงเป็นจริงหมายถึงสั่งให้สร้างฐานข้อมูลขึ้นมาใหม่หากยังไม่มีอยู่

```
>> import rdflib
>> from rdflib import Literal
>> store = rdflib.plugin.get('SQLite', rdflib.store.Store)('rdf-test.ts')
>> store.open('.', create=True)
>> g = rdflib.ConjunctiveGraph(store)
>> semprog = rdflib.Namespace("http://semprog.com/people/")
>> foaf = rdflib.Namespace("http://xmlns.com/foaf/0.1/")
>> g.add((semprog["jamie"], foaf["name"], Literal("Jamie Taylor")))
>> g.add((semprog["jamie"], foaf["mbox"], Literal("jamie@semprog.com")))
>> g.serialize(format="nt") #just to check our work
>> g.commit()
```

รูป 3.88 โค้ดบันทึกข้อมูลกราฟ RDF ลงฐานข้อมูล

หลังจากทำตามในรูป 3.88 แล้วให้ปิดโปรแกรม Python ไปแล้วทดลองทำการโหลดกราฟที่บันทึกลงไว้ในฐานข้อมูลที่เพิ่งสร้างด้วยโค้ดที่แสดงในรูป 3.89 สังเกตว่าเรากำหนดให้ค่าความจริงเป็นเท็จในพารามิเตอร์ create ของเมธอด open เพราะว่าไลบรารี pysqlite กำหนดไว้ว่าหากต้องการเปิดฐานข้อมูลที่มีอยู่แล้วขึ้นมาอ่านแล้วกำหนดค่าพารามิเตอร์ create เป็นค่าความจริงเป็น

จริงแล้ว จะส่งเอ็กเซ็ปชัน (Exception) ชนิด OperationalError เพื่อป้องกันไม่ให้สร้างฐานข้อมูลขึ้นมาใหม่จากฐานข้อมูลที่มีอยู่แล้ว

```
>> import rdflib
>> store = rdflib.plugin.get('SQLite', rdflib.store.Store)('rdf-test.ts')
>> store.open('.', create=False)
>> g = rdflib.ConjunctiveGraph(store)
>> g.serialize(format="nt")
```

รูป 3.89 โค้ดอ่านข้อมูลกราฟ RDF จากฐานข้อมูล

3.4.2 การสืบค้นข้อมูลด้วยไลบรารี RDFLib โดยใช้ภาษา SPARQL (SPARQL Queries in RDFLib)

คลาส Graph ซึ่งเป็นคลาสแม่ของคลาส ConjunctiveGraph นั้นมีเมธอดสำหรับใช้สืบค้นข้อมูลกราฟที่เก็บไว้ในตัวเก็บทริปเปิลด้วยภาษา SPARQL ไว้ให้ใช้อยู่ โดยเมธอดนั้นจะรับสตริงที่เก็บชุดคำสั่งสืบค้นภาษา SPARQL และอาจส่งผ่านดิคชันนารีที่ใช้สำหรับระบุเนมสเปสไปยัง URIRef เข้าไปในพารามิเตอร์ initNS หรือสามารถกำหนดเนมสเปสต่าง ๆ ผ่านคำสั่ง PREFIX ที่มีในภาษา SPARQL ลงไปในสตริงที่ส่งผ่านเข้าไปในเมธอดก็ได้ ตัวอย่างการใช้เมธอดแสดงอยู่ในรูป 3.90

```
from rdflib.Graph import ConjunctiveGraph, Namespace
FBNAMESPACE = Namespace("http://rdf.freebase.com/ns/")
g = ConjunctiveGraph()
g.parse("sample-movie-data.n3", format="n3")
results = g.query("""SELECT ?film ?year
          WHERE { ?film fb:film.film.initial_release_date ?year. }""", \
                initNs={'fb':FBNAMESPACE})

for triple in results:
    print triple
```

รูป 3.90 โค้ดสืบค้นข้อมูลจากกราฟภายในตัวเก็บทริปเปิล

นอกจากนั้นเมธอดสืบค้นข้อมูลในไลบรารี RDFLib ยังรองรับการสืบค้นข้อมูลรูปแบบ CONSTRUCT อีกด้วยดังตัวอย่างในรูป 3.91

```
from rdflib.Graph import ConjunctiveGraph, Namespace

FBNAMESPACE = Namespace("http://rdf.freebase.com/ns/")
g = ConjunctiveGraph()
g.parse("sample-movie-data.n3", format="n3")
results = g.query("""CONSTRUCT {
  ?who <http://employment.history/was_employed_in> ?year
}
WHERE {
     ?film fb:film.film.starring ?who .
     ?film fb:film.film.initial_release_date ?year .
   }""", initNs={'fb':FBNAMESPACE}).serialize(format="xml")
for result in results:
     print results
```

รูป 3.91 โค้ดสืบค้นข้อมูลรูปแบบ CONSTRUCT ด้วยไลบรารี RDFLib

3.5 SPARQL

SPARQL ย่อมาจาก Simple Protocol and RDF Query Language เหมือนกับที่ SQL เป็นภาษามาตรฐานที่ใช้ในการสืบค้นข้อมูลบนระบบฐานข้อมูลเชิงสัมพันธ์ต่าง ๆ ภาษา SPARQL ก็เป็นภาษาที่ใช้ในการสืบค้นข้อมูลที่ได้รับการกำหนดมาตรฐานโดย W3C เพื่อใช้สืบค้นข้อมูลในกราฟ RDF ซึ่งนอกจากจะใช้สืบค้นข้อมูลได้แล้ว ยังมีความสามารถอื่น ๆ อีกเช่น สร้างกลุ่มทริปเปิลใหม่ขึ้นมาโดยอิงจากผลการสืบค้น เป็นต้น โดยหลักการที่ใช้ในการสืบค้นข้อมูลของภาษา SPARQL คือให้ผู้สืบค้นทำการกำหนดรูปแบบทริปเปิล (Triple patterns) ที่มีตัวแปรกำกับอยู่เพื่อให้ตัวประมวลผล SPARQL นำรูปแบบกราฟดังกล่าวไปจับคู่เข้ากับกราฟ RDF ที่ทำต้องการทำการสืบค้น

เครื่องมือสำหรับใช้สืบค้นข้อมูลที่ใช้ในหัวข้อนี้ชื่อว่า Twinkle พัฒนาด้วยภาษา Java และตัวประมวลผลภาษา SPARQL ชื่อว่า ARQ ซึ่งสามารถสืบค้นข้อมูลกราฟ RDF ที่เก็บอยู่บนเครื่องหรือไฟล์เอกสาร RDF ที่อยู่บนเครื่องระยะไกลก็ได้ ลักษณะเครื่องมือแสดงในรูป 3.94 โดยผลลัพธ์ที่ได้จากการสืบค้นจะแสดงอยู่ใน 2 รูปแบบคือ รูปแบบตาราง และรูปแบบ RDF/XML

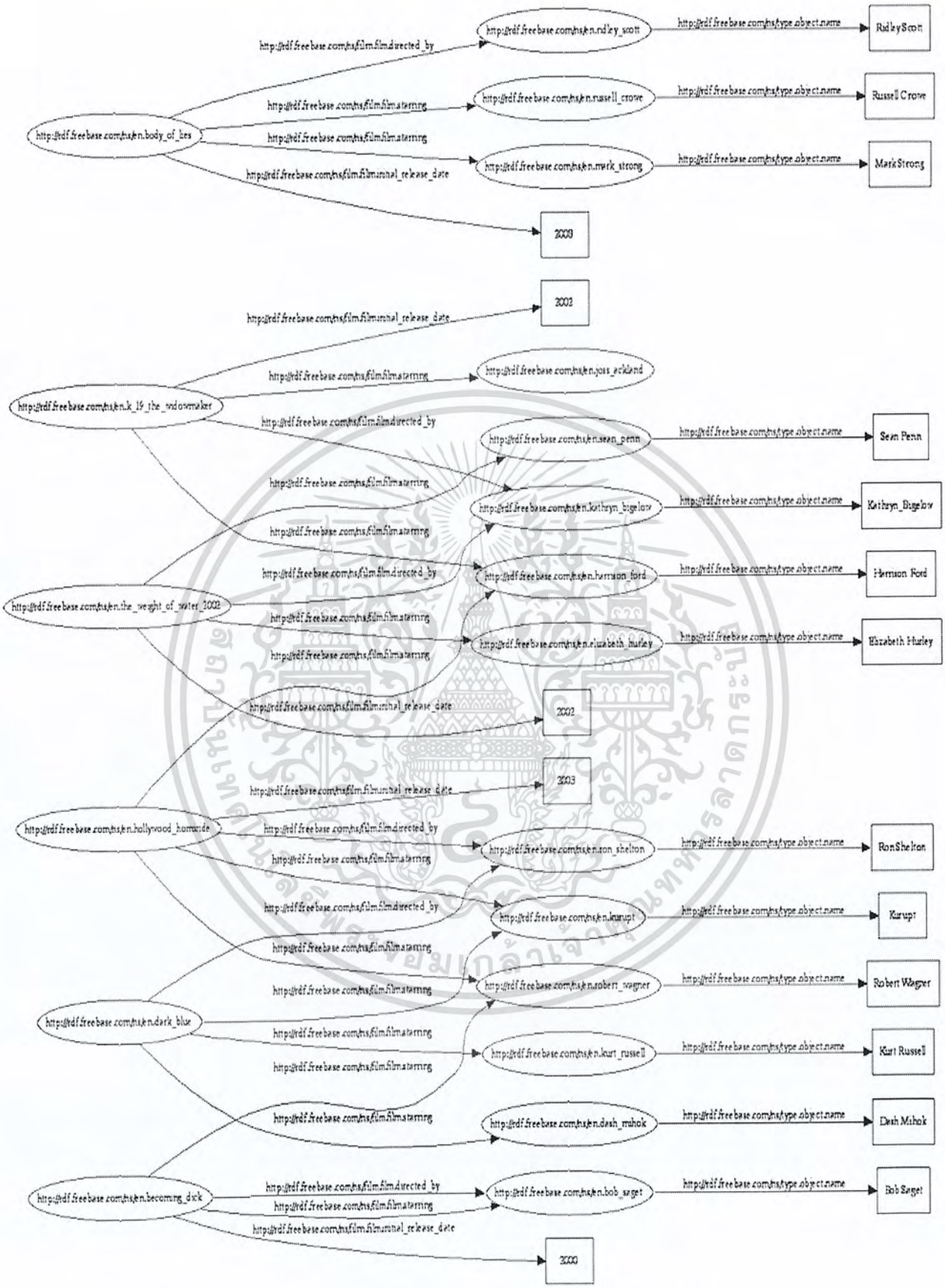
กราฟข้อมูล RDF ที่เราจะทำการสืบค้นในหัวข้อนี้แสดงอยู่ในรูป 3.92 โดยเขียนอยู่ในรูปของ N3 และเมื่อนำมาแสดงในรูปแบบของกราฟจะเป็นดังรูป 3.93 ข้อมูลตัวอย่างนี้ความสัมพันธ์ระหว่างภาพยนตร์ กับผู้กำกับ นักแสดง และปีที่เริ่มฉาย

```
@prefix fb: <http://rdf.freebase.com/ns/> .

<http://rdf.freebase.com/ns/en.hollywood_homicide>
  <http://rdf.freebase.com/ns/film.film.directed_by>
  <http://rdf.freebase.com/ns/en.ron_shelton> ;
<http://rdf.freebase.com/ns/film.film.starring>
  <http://rdf.freebase.com/ns/en.harrison_ford> ,
  <http://rdf.freebase.com/ns/en.kurupt> ,
  <http://rdf.freebase.com/ns/en.robert_wagner> ;
<http://rdf.freebase.com/ns/film.film.initial_release_date>  "2003"
.

<http://rdf.freebase.com/ns/en.k_19_the_widowmaker>
  <http://rdf.freebase.com/ns/film.film.directed_by>
  <http://rdf.freebase.com/ns/en.kathryn_bigelow> ;
<http://rdf.freebase.com/ns/film.film.starring>
  <http://rdf.freebase.com/ns/en.harrison_ford> ,
  <http://rdf.freebase.com/ns/en.joss_ackland> ;
<http://rdf.freebase.com/ns/film.film.initial_release_date>  "2002"
.

<http://rdf.freebase.com/ns/en.dark_blue>
  <http://rdf.freebase.com/ns/film.film.directed_by>
  <http://rdf.freebase.com/ns/en.ron_shelton> ;
<http://rdf.freebase.com/ns/film.film.starring>
  <http://rdf.freebase.com/ns/en.kurupt> ,
  <http://rdf.freebase.com/ns/en.kurt_russell> ,
  <http://rdf.freebase.com/ns/en.dash_mihok> .

<http://rdf.freebase.com/ns/en.the_weight_of_water_2002>
  <http://rdf.freebase.com/ns/film.film.directed_by>
  <http://rdf.freebase.com/ns/en.kathryn_bigelow> ;
<http://rdf.freebase.com/ns/film.film.starring>
  <http://rdf.freebase.com/ns/en.sean_penn> ,
  <http://rdf.freebase.com/ns/en.elizabeth_hurley> ;
<http://rdf.freebase.com/ns/film.film.initial_release_date>  "2002"
.

<http://rdf.freebase.com/ns/en.becoming_dick>
  <http://rdf.freebase.com/ns/film.film.directed_by>
  <http://rdf.freebase.com/ns/en.bob_saget> ;
<http://rdf.freebase.com/ns/film.film.starring>
  <http://rdf.freebase.com/ns/en.robert_wagner> ,
  <http://rdf.freebase.com/ns/en.bob_saget> ;
<http://rdf.freebase.com/ns/film.film.initial_release_date>  "2000"
.

<http://rdf.freebase.com/ns/en.body_of_lies>
  <http://rdf.freebase.com/ns/film.film.directed_by>
  <http://rdf.freebase.com/ns/en.ridley_scott> ;
<http://rdf.freebase.com/ns/film.film.starring>
  <http://rdf.freebase.com/ns/en.russell_crowe> ,
  <http://rdf.freebase.com/ns/en.mark_strong> ;
<http://rdf.freebase.com/ns/film.film.initial_release_date>  "2008"
.
```

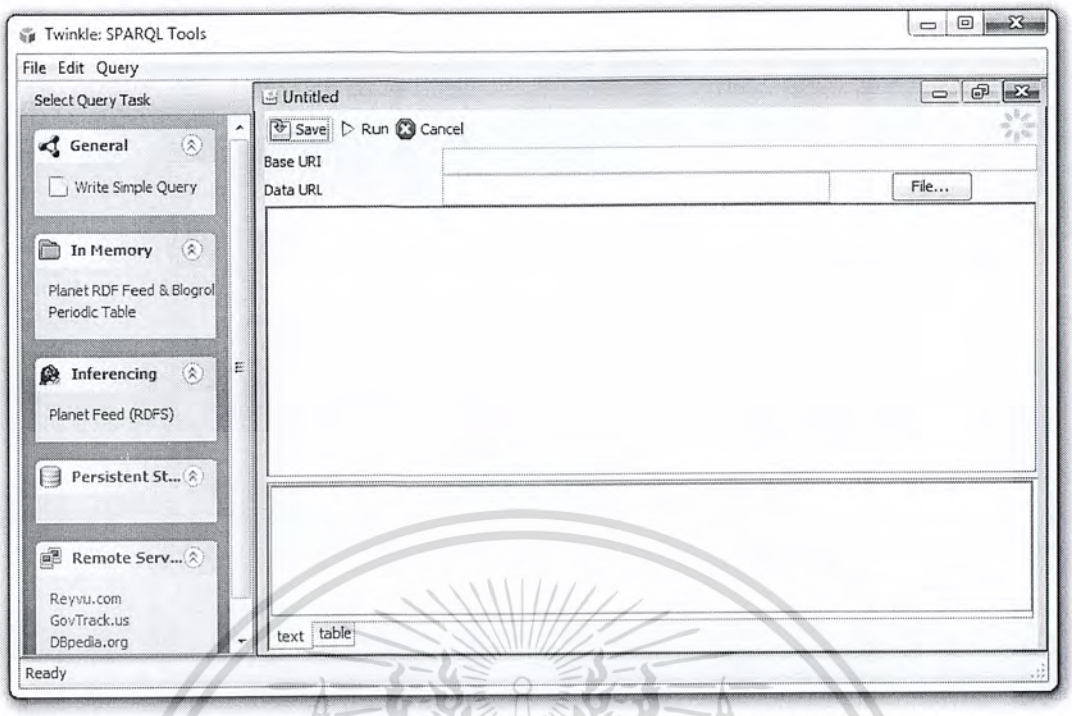
รูป 3.92 ข้อมูลกราฟ RDF เกี่ยวกับข้อมูลภาพยนตร์

```
<http://rdf.freebase.com/ns/en.kurt_russell>
  <http://rdf.freebase.com/ns/type.object.name> "Kurt Russell" .
<http://rdf.freebase.com/ns/en.dash_mihok>
  <http://rdf.freebase.com/ns/type.object.name> "Dash Mihok" .
<http://rdf.freebase.com/ns/en.sean_penn>
  <http://rdf.freebase.com/ns/type.object.name> "Sean Penn" .
<http://rdf.freebase.com/ns/en.elizabeth_hurley>
  <http://rdf.freebase.com/ns/type.object.name> "Elizabeth Hurley"
.
<http://rdf.freebase.com/ns/en.kathryn_bigelow>
  <http://rdf.freebase.com/ns/type.object.name>   "Kathryn_Bigelow"
.
<http://rdf.freebase.com/ns/en.bob_saget>
  <http://rdf.freebase.com/ns/type.object.name> "Bob Saget" .
<http://rdf.freebase.com/ns/en.ridley_scott>
  <http://rdf.freebase.com/ns/type.object.name> "Ridley Scott" .
<http://rdf.freebase.com/ns/en.russell_crowe>
  <http://rdf.freebase.com/ns/type.object.name> "Russell Crowe" .
<http://rdf.freebase.com/ns/en.mark_strong>
  <http://rdf.freebase.com/ns/type.object.name> "Mark Strong" .
<http://rdf.freebase.com/ns/en.ron_shelton>
  <http://rdf.freebase.com/ns/type.object.name> "Ron Shelton" .
<http://rdf.freebase.com/ns/en.harrison_ford>
  <http://rdf.freebase.com/ns/type.object.name> "Harrison Ford" .
<http://rdf.freebase.com/ns/en.robert_wagner>
  <http://rdf.freebase.com/ns/type.object.name> "Robert Wagner" .
<http://rdf.freebase.com/ns/en.kurupt>
  <http://rdf.freebase.com/ns/type.object.name> "Kurupt" .
```

รูป 3.92 ข้อมูลกราฟ RDF เกี่ยวกับข้อมูลภาพยนตร์ (ต่อ)



รูป 3.93 ข้อมูลกราฟ RDF ในรูปแบบของกราฟ



รูป 3.94 ลักษณะเครื่องมือที่ใช้ในการสืบค้นข้อมูลกราฟ RDF ชื่อว่า Twinkle

การสืบค้นในภาษา SPARQL มีทั้งหมดอยู่ 4 รูปแบบดังนี้ SELECT, CONSTRUCT, ASK และ DESCRIBE ซึ่งต่างก็ให้ผลลัพธ์โดยอยู่บนพื้นฐานของรูปแบบกราฟที่ผู้ใช้ทำการสืบค้น ซึ่งจะค่อย ๆ กล่าวไปตามลำดับ

3.5.1 การสืบค้นรูปแบบ SELECT

การสืบค้นรูปแบบ SELECT จะเริ่มต้นด้วยการประกาศคำนำหน้า หรือ PREFIX เพื่อใช้แทนการพิมพ์ URI แบบเต็มเหมือนกับที่เราประกาศเนมสเปสในเอกสาร RDF ซึ่งจะถูกนำมาใช้ในตลอดการสืบค้น เนื่องจากเราเอาข้อมูลมาจาก Freebase จึงทำการเพิ่ม PREFIX ดังในรูป 3.95 สังเกตว่าจะต้องมีเครื่องหมายโคลอน (:) ต่อท้ายคำนำหน้าที่ประกาศเสมอ

```
PREFIX fb: http://rdf.freebase.com/ns/
```

รูป 3.95 ประกาศ PREFIX สำหรับใช้ในการสืบค้น

นอกจากนี้ในส่วนเริ่มต้นของการสืบค้นเราสามารถประกาศฐาน URI ได้ด้วย โดยใช้คำสั่ง BASE แล้วตามด้วย URI ฐานที่ต้องการกำหนดให้ดังในรูป 3.96 ซึ่งจะถูกนำไปเชื่อมเข้ากับ URI สัมพัทธ์ภายในการสืบค้นในภายหลัง ยกตัวอย่างเช่น <b006ww0v.rdf> ที่ปรากฏอยู่ในการสืบค้นเมื่อตัวประมวลผล SPARQL พิจารณา ก็จะทำการเชื่อม URI ฐานเข้ากับ URI สัมพัทธ์ที่

ประกาศเอาไว้ในสัญลักษณ์ได้ ทำให้ได้ผลลัพธ์เป็นURIสัมบูรณ์คือ http://www.//bbc.co.uk programmes/b006ww0v.rdf แต่มีข้อแม้คือสามารถประกาศ URI ฐานด้วยคำสั่ง BASE ได้เพียงครั้งเดียวต่อการสืบค้นหนึ่ง ๆ

```
BASE http://www.bbc.co.uk/programmes/
```

รูป 3.96 ประกาศ BASE สำหรับใช้ในการสืบค้น

การสืบค้นรูปแบบ SELECT นั้นจะต้องระบุเซ็ตย่อยของตัวแปรที่ต้องการทราบข้อมูลต่อท้าย โดยเป็นเซ็ตเดียวกับตัวแปรที่ใช้ในการระบุกลุ่มรูปแบบทริปเปิลในการสืบค้น ตามด้วยประโยคย่อย (Clause) WHERE ซึ่งใช้ในการระบุรูปแบบทริปเปิลที่ต้องการนำไปเทียบ โดยแต่ละรูปแบบทริปเปิลจะประกอบไปด้วย ประธาน, ภาคแสดง, และกรรม ตามลำดับ และตัวแปรจะต้องมีเครื่องหมายคำถาม (?) หรือเครื่องหมายดอลลาร์ ($) นำหน้าเสมอ ซึ่งสามารถใช้แทนกันได้ทุกประการ

รูป 3.97 แสดงการสืบค้นโดยการหาว่าผู้กำกับคนไหนแสดงบทในภาพยนตร์เรื่องที่ตนเองกำกับ และรูป 3.98 แสดงผลการสืบค้น

```
PREFIX fb:<http://rdf.freebase.com/ns/>
SELECT ?who ?film
WHERE
{
  ?film fb:film.film.directed_by ?who .
  ?film fb:film.film.starring ?who .
}
```

รูป 3.97 โค้ดภาษา SPARQL

Who	film
http://rdf.freebase.com/ns/en.bob_saget	http://rdf.freebase.com/ns/en.becoming_dick

รูป 3.98 ผลลัพธ์จากการสืบค้น

3.5.2 การกำหนดข้อจำกัดด้วยประโยคย่อย OPTIONAL และ FILTER

บางทีเราอาจต้องการผลลัพธ์จากการสืบค้นโดยผลลัพธ์ที่ได้ไม่จำเป็นต้องจับคู่เข้ากันกับรูปแบบทริปเปิลบางรูปแบบก็ได้ ซึ่งมักจะมีประโยชน์ในกรณีที่เราทำการสืบค้นบนข้อมูลที่ไม่สมบูรณ์ ยกตัวอย่างจากในข้อมูลตัวอย่าง เช่น ภาพยนตร์บางเรื่องไม่มีปีที่เริ่มฉายกำกับเอาไว้ โดย

ภาษา SPARQL ได้มีประโยคย่อย OPTIONAL เอาไว้สำหรับระบุรูปแบบทริปเปิลที่หากไม่สามารถนำไปจับคู่เข้ากับกราฟได้ก็จะไม่ตัดผลลัพธ์นั้นทิ้งไปดังตัวอย่างการสืบค้นในรูป 3.99 ซึ่งทำการสืบค้นภาพยนตร์เรื่องที่ Ron Shelton กำกับพร้อมทั้งหาปีที่เริ่มฉาย แต่หากภาพยนตร์นั้นไม่มีปีที่เริ่มฉายกำกับก็ให้นำมาแสดงในผลลัพธ์ด้วยดังจะเห็นในรูป 3.100 ที่ภาพยนตร์เรื่อง Dark Blue ที่ไม่มีปีที่เริ่มฉายกำกับอยู่ ก็ติดเข้ามาในผลลัพธ์ด้วย

```
PREFIX fb: <http://rdf.freebase.com/ns/>
SELECT ?film ?reldate
WHERE
{
  ?film fb:film.film.directed_by fb:en.ron_shelton .
  OPTIONAL { ?film fb:film.film.initial_release_date ?reldate .}
}
```

รูป 3.99 โค้ดภาษา SPARQL

film	reldate
http://rdf.freebase.com/ns/en.dark_blue	
http://rdf.freebase.com/ns/en.hollywood_homicide	2003

รูป 3.100 ผลลัพธ์จากการสืบค้น

การระบุรูปแบบทริปเปิลนั้นมีประโยชน์ ในการพิจารณาดูว่าเซ็ตของความสัมพันธ์ที่อธิบายด้วยรูปแบบทริปเปิลนั้นมีอยู่ในกราฟข้อมูลที่เราต้องการสืบค้นหรือไม่ แต่หากเราต้องการจำกัดผลลัพธ์ของการสืบค้น โดยการกรองข้อมูลที่ได้จากส่วนประกอบของรูปแบบทริปเปิลไม่ว่าจะเป็น ประธาน ภาคแสดง หรือกรรม ก็สามารถทำได้โดยการใช้ประโยคย่อย FILTER ซึ่งรองรับโอเปอร์เรเตอร์ (Operators) กับข้อมูลชุดหนึ่งซึ่งมีลักษณะเหมือนกับที่ใช้ในภาษา XPath 2.0 แสดงในตาราง 3.1 ตาราง 3.2 และตาราง 3.3 ทำให้เราสามารถทดสอบค่าของตัวแปรกับเงื่อนไขที่กำหนดเอาไว้ในประโยคย่อย FILTER ได้ และหากประโยคย่อย FILTER คืนค่าความจริงเป็นเท็จกลับมาผลลัพธ์จากการสืบค้นนั้น ๆ จะถูกคัดออกไป

ตาราง 3.1 SPARQL Unary Operators

Operator	Type(A)	Function	Result type
XQuery Unary Operators			
! A	xsd:boolean (EBV)	fn:not(A)	xsd:boolean
+ A	Numeric	op:numeric-unary-plus(A)	numeric
- A	Numeric	op:numeric-unary-minus(A)	numeric

ตาราง 3.1 SPARQL Unary Operators (ต่อ)

SPARQL Tests			
BOUND(A)	Variable	bound(A)	xsd:boolean
isIRI(A) **isURI(A)**	RDF term	isIRI(A)	xsd:boolean
isBLANK(A)	RDF term	isBlank(A)	xsd:boolean
isLITERAL(A)	RDF term	isLiteral(A)	xsd:boolean
SPARQL Accessors			
STR(A)	literal	str(A)	simple literal
STR(A)	IRI	str(A)	simple literal
LANG(A)	literal	lang(A)	simple literal
DATATYPE(A)	typed literal	datatype(A)	IRI
DATATYPE(A)	simple literal	datatype(A)	IRI

ตาราง 3.2 SPARQL Binary Operators

Operator	Type(A)	Type(B)	Function	Result type
Logical Connectives				
A || B	xsd:boolean (EBV)	xsd:boolean (EBV)	logical-or(A, B)	xsd:boolean
A && B	xsd:boolean (EBV)	xsd:boolean (EBV)	logical-and(A, B)	xsd:boolean
XPath Tests				
A = B	numeric	numeric	op:numeric-equal(A, B)	xsd:boolean
A = B	simple literal	simple literal	op:numeric-equal(fn:compare(A, B), 0)	xsd:boolean
A = B	xsd:string	xsd:string	op:numeric-equal(fn:compare(STR(A), STR(B)), 0)	xsd:boolean
A = B	xsd:boolean	xsd:boolean	op:boolean-equal(A, B)	xsd:boolean
A = B	xsd:dateTime	xsd:dateTime	op:dateTime-equal(A, B)	xsd:boolean
A != B	numeric	numeric	fn:not(op:numeric-equal(A, B))	xsd:boolean
A != B	simple literal	simple literal	fn:not(op:numeric-equal(fn:compare(A, B), 0))	xsd:boolean
A != B	xsd:string	xsd:string	fn:not(op:numeric-equal(fn:compare(STR(A), STR(B)), 0))	xsd:boolean
A != B	xsd:boolean	xsd:boolean	fn:not(op:boolean-equal(A, B))	xsd:boolean

ตาราง 3.2 SPARQL Binary Operators (ต่อ)

A != B	xsd:dateTime	xsd:dateTime	fn:not(op:dateTime-equal(A, B))	xsd:boolean
A < B	numeric	numeric	op:numeric-less-than(A, B)	xsd:boolean
A < B	simple literal	simple literal	op:numeric-equal(fn:compare(A, B), -1)	xsd:boolean
A < B	xsd:string	xsd:string	op:numeric-equal(fn:compare(STR(A), STR(B)), -1)	xsd:boolean
A < B	xsd:boolean	xsd:boolean	op:boolean-less-than(A, B)	xsd:boolean
A < B	xsd:dateTime	xsd:dateTime	op:dateTime-less-than(A, B)	xsd:boolean
A > B	numeric	numeric	op:numeric-greater-than(A, B)	xsd:boolean
A > B	simple literal	simple literal	op:numeric-equal(fn:compare(A, B), 1)	xsd:boolean
A > B	xsd:string	xsd:string	op:numeric-equal(fn:compare(STR(A), STR(B)), 1)	xsd:boolean
A > B	xsd:boolean	xsd:boolean	op:boolean-greater-than(A, B)	xsd:boolean
A > B	xsd:dateTime	xsd:dateTime	op:dateTime-greater-than(A, B)	xsd:boolean
A <= B	numeric	numeric	logical-or(op:numeric-less-than(A, B), op:numeric-equal(A, B))	xsd:boolean
A <= B	simple literal	simple literal	fn:not(op:numeric-equal(fn:compare(A, B), 1))	xsd:boolean
A <= B	xsd:boolean	xsd:boolean	fn:not(op:boolean-greater-than(A, B))	xsd:boolean
A <= B	xsd:dateTime	xsd:dateTime	fn:not(op:dateTime-greater-than(A, B))	xsd:boolean
A >= B	numeric	numeric	logical-or(op:numeric-greater-than(A, B), op:numeric-equal(A, B))	xsd:boolean
A >= B	simple literal	simple literal	fn:not(op:numeric-equal(fn:compare(A, B), -1))	xsd:boolean

ตาราง 3.2 SPARQL Binary Operators (ต่อ)

A >= B	xsd:string	xsd:string	fn:not(op:numeric-equal(fn:compare(STR(A), STR(B)), -1))	xsd:boolean
A >= B	xsd:boolean	xsd:boolean	fn:not(op:boolean-less-than(A, B))	xsd:boolean
A >= B	xsd:dateTime	xsd:dateTime	fn:not(op:dateTime-less-than(A, B))	xsd:boolean
XPath Arithmetic				
A * B	numeric	numeric	op:numeric-multiply(A, B)	numeric
A / B	numeric	numeric	op:numeric-divide(A, B)	numeric; but xsd:decimal if both operands are xsd:integer
A + B	numeric	numeric	op:numeric-add(A, B)	numeric
A - B	numeric	numeric	op:numeric-subtract(A, B)	numeric
SPARQL Tests				
A = B	RDF term	RDF term	RDFterm-equal(A, B)	xsd:boolean
A != B	RDF term	RDF term	fn:not(RDFterm-equal(A, B))	xsd:boolean
sameTERM(A)	RDF term	RDF term	sameTerm(A, B)	xsd:boolean
langMATCHES(A, B)	simple literal	simple literal	langMatches(A, B)	xsd:boolean
REGEX(STRING, PATTERN)	simple literal	simple literal	fn:matches(STRING, PATTERN)	xsd:boolean

ตาราง 3.3 SPARQL Trinary Operator

Operator	Type(A)	Type(B)	Type(C)	Function	Result type
SPARQL Tests					
REGEX(STRING, PATTERN, FLAGS)	simple literal	simple literal	simple literal	fn:matches(STRING, PATTERN, FLAGS)	xsd:boolean

ในขณะที่ประโยคย่อย OPTIONAL มีเอาไว้เพื่อรับมือกับข้อมูลที่ตกหล่นไป เราสามารถใช้ประโยคย่อย OPTIONAL ร่วมกับการกระทำ bound ของประโยคย่อย FILTER เพื่อหาว่าโหนดใดไม่มีความสัมพันธ์ที่เราต้องการสืบค้นอยู่ดังในรูป 3.101 เป็นการสืบค้นหาภาพยนตร์ที่กำกับโดย Ron Shelton และไม่มีปีที่เริ่มฉายกำกับเอาไว้

```
PREFIX fb: <http://rdf.freebase.com/ns/>

SELECT ?film
WHERE
{
  ?film fb:film.film.directed_by fb:en.ron_shelton .
  OPTIONAL { ?film fb:film.film.initial_release_date ?reldate .}
    FILTER (!bound(?reldate))
}
```

รูป 3.101 โค้ดภาษา SPARQL

ซึ่งแน่นอนว่าผลลัพธ์ที่ได้จากการสืบค้นคือภาพยนตร์เรื่อง Dark Blue ดังรูป 3.102 โดยฟังก์ชัน bound จะทำหน้าที่ดูว่าตัวแปรของคำตอบจากการสืบค้นที่ถูกส่งเข้ามานั้นมีค่ากำกับอยู่หรือไม่ หากไม่มีจะคืนค่าความจริงเป็นเท็จออกมา แล้วถูกนิเสธ ทำให้ถูกนำไปเก็บอยู่ในชุดของคำตอบของการสืบค้น

film
http://rdf.freebase.com/ns/en.dark_blue

รูป 3.102 ผลลัพธ์จากการสืบค้น

เราสามารถใช้โอเปอร์เรเตอร์สำหรับใช้กับประโยคย่อย FILTER เพื่อตรวจดูว่าค่าของตัวแปรนั้นตรงตามเงื่อนไขที่เราระบุหรือไม่ ยกตัวอย่างเช่น เราสามารถดูว่าค่าของตัวแปรชนิดสตริง (String) นั้นตรงตามรูปแบบที่ระบุไว้หรือไม่ ซึ่งสำหรับการคัดกรองประเภทนี้ภาษา SPARQL ได้จัดเตรียมโอเปอร์เรเตอร์ regex เอาไว้ให้ดังจะเห็นได้จากในตารางที่ 3.2 และตารางที่ 3.3 ซึ่งจะรับตัวแปรที่ใช้สืบค้น จากนั้นรูปแบบของสตริงโดยเป็นไปตามภาษา Regular Expression และสุดท้ายค่าแฟล็กต่าง ๆ สำหรับกำหนดให้ไม่สนใจตัวอักษรเล็ก หรือตัวอักษรใหญ่ในค่าของตัวแปร และรูปแบบสตริงที่ส่งผ่านเข้าไปในโอเปอร์เรเตอร์ ดังการสืบค้นในรูป 3.103 ที่สืบค้นหาภาพยนตร์ที่มีดาราร่วมแสดงที่ชื่อมีคำว่า Russell ปนอยู่ด้วย ผลลัพธ์จากการสืบค้นเป็นดังรูป 3.104

```
PREFIX fb:<http://rdf.freebase.com/ns/>

SELECT distinct ?who ?film
WHERE
{
  ?film fb:film.film.starring ?star .
  ?star fb:type.object.name ?who .
    FILTER regex(?who, "russell", "i")
}
```

รูป 3.103 โค้ดภาษา SPARQL

who	film
Russell Crowe	http://rdf.freebase.com/ns/en.body_of_lies
Kurt Russell	http://rdf.freebase.com/ns/en.dark_blue

รูป 3.104 ผลลัพธ์จากการสืบค้น

โอเปอร์เรเตอร์ regex ในภาษา SPARQL จะมีลักษณะเหมือนกับโอเปอร์เรเตอร์ fn:matches ในภาษา XPath 2.0 ทุกประการซึ่งหมายความว่าเราสามารถส่งผ่านรูปแบบสตริงที่ซับซ้อนเข้าไปในโอเปอร์เรเตอร์นี้ได้ ยกตัวอย่างเช่น เราสามารถจำกัดให้คำตอบในผลลัพธ์จากการสืบค้นมีเพียงแค่ดาราชื่อ Russell Crowe ได้เพียงการเปลี่ยนโอเปอร์เรเตอร์ที่ใช้ในประโยคย่อย FILTER ให้เป็น regex(?who, "^russell", "i") แทน

นอกจากนี้ยังมีโอเปอร์เรเตอร์อื่น ๆ ให้ใช้อีกตามที่ระบุเอาไว้ในตารางทั้ง 3 ตารางข้างต้น ยกตัวอย่างเช่นการใช้โอเปอร์เรเตอร์สำหรับทดสอบความไม่เท่ากันในการสืบค้นหาภาพยนตร์เรื่องที่ฉายหลังปี 2002 ดังในรูป 3.105 และผลลัพธ์ในรูป 3.106

```
PREFIX fb:<http://rdf.freebase.com/ns/>

SELECT ?film ?when
WHERE
{
  ?film fb:film.film.initial_release_date ?when .
    FILTER (?when > "2002")
}
```

รูป 3.105 โค้ดภาษา SPARQL

Film	when
http://rdf.freebase.com/ns/en.body_of_lies	2008
http://rdf.freebase.com/ns/en.hollywood_homicide	2003

รูป 3.106 ผลลัพธ์จากการสืบค้น

3.5.3 การสืบค้นด้วยรูปแบบกราฟหลาย ๆ รูปแบบ (Querying with Multiple Graph Patterns)

ที่ผ่านมาเราได้แสดงแต่การสืบค้นที่ใช้รูปแบบกราฟเพียงรูปแบบเดียวต่อการสืบค้นครั้งหนึ่ง ๆ ในขณะที่ภาษา SPARQL ยอมให้เราระบุรูปแบบกราฟมากกว่า 1 รูปแบบได้ต่อการสืบค้นหนึ่งครั้ง โดยการครอบกลุ่มของทริปเปิลด้วยเครื่องหมาย { และ } เพื่อแยกกลุ่มทริปเปิลออกเป็นรูปแบบกราฟหนึ่ง ๆ หากไม่มีการกำกับส่วนขยาย หรือมอดิไฟเออร์ (Modifier) รูปแบบกราฟทุกรูปแบบจะถูกนำมาพิจารณาเข้าด้วยกันหมดเพื่อใช้ในการหาคำตอบราวกับว่ามีรูปแบบกราฟอยู่เพียงรูปแบบเดียว และที่สำคัญ เมื่อมีการจัดกลุ่มทริปเปิลออกเป็นหลาย ๆ รูปแบบกราฟ เราจะต้องใส่ประโยคย่อย FILTER กำกับเข้าไปในแต่ละรูปแบบกราฟด้วย ดังในรูป 3.107 เป็นการสืบค้นหาผู้กำกับที่มีชื่อจริงความยาว 3 ตัวอักษร และดาราที่มีชื่อขึ้นต้นด้วยตัวอักษร B ซึ่งสองเงื่อนไขนี้ดูจะไม่เกี่ยวกันเลย แต่ในการสืบค้นนี้ จะต้องพิจารณาตัวแปร ?name ที่รองรับกับทั้ง 2 เงื่อนไข โดยผลลัพธ์ที่ได้แสดงในรูป 3.108

```
PREFIX fb:<http://rdf.freebase.com/ns/>

SELECT ?name
WHERE
{
  {
    ?film fb:film.film.directed_by ?person .
    ?person fb:type.object.name ?name
      filter regex(?name, "^... ", "i")
  }
  {
    ?film fb:film.film.starring ?actor .
    ?actor fb:type.object.name ?name
      filter regex(?name, "^b", "i")
  }
}
```

รูป 3.107 โค้ดภาษา SPARQL

name
Bob Saget

รูป 3.108 ผลลัพธ์จากการสืบค้น

จะเห็นได้ว่า Bob Saget เป็นทั้งผู้กำกับ และดาราในคนเดียว และสอดคล้องกับเงื่อนไขทั้งสอง นอกจากนี้เราอาจเพิ่มคำหลัก (Keyword) คำว่า UNION เข้าไปในการสืบค้นเพื่อเป็นการสั่งให้พิจารณารูปแบบกราฟแต่ละรูปแบบโดยอิสระจากกัน แล้วนำคำตอบที่ได้จากแต่ละรูปแบบกราฟมารวมเข้าด้วยกัน ซึ่งมีประโยชน์ในกรณีที่คำตอบที่จะได้จากทั้ง 2 รูปแบบต่างมีประโยชน์พอ ๆ กัน ดังตัวอย่างการสืบค้นในรูป 3.109และผลลัพธ์แสดงในรูป 3.110

```
PREFIX fb:<http://rdf.freebase.com/ns/>

SELECT DISTINCT ?name
WHERE
{
  {
    ?film fb:film.film.directed_by ?person .
    ?person fb:type.object.name ?name
      filter regex(?name, "^... ", "i")
  }
  UNION
  {
    ?film fb:film.film.starring ?actor .
    ?actor fb:type.object.name ?name
      filter regex(?name, "^b", "i")
  }
}
```

รูป 3.109 โค้ดภาษา SPARQL

name
Bob Saget
Ron Shelton

รูป 3.110 ผลลัพธ์จากการสืบค้น

พบว่าในผลลัพธ์มีบุคคลชื่อ Ron Shelton รวมเข้ามาอยู่ในชุดคำตอบด้วย เนื่องมาจากตัวแปร ?name ไม่จำเป็นต้องสอดคล้องกับเงื่อนไขทั้ง 2 เงื่อนไขแล้วนั่นเอง

และการตรวจสอบดูว่าทรัพยากร 2 ทรัพยากรนั้นเป็นทรัพยากรเดียวกันหรือไม่ในชุดคำตอบก็มีประโยชน์เหมือนกัน เช่นในตัวอย่างการสืบค้นในรูป 3.111 ที่ต้องการดูว่าผู้กำกับคนไหนที่เคยทำงานกับดาราคนหนึ่ง ซึ่งในการสืบค้นนี้คือ Harrison Ford จากนั้นทำการสืบค้นหาว่ามีดาราคนไหนที่ร่วมแสดงภาพยนตร์เรื่องนั้นอีก แล้วจึงนำไปสืบค้นหาภาพยนตร์เรื่องที่ผู้กำกับ และดาราที่แสดงร่วมทำงานร่วมกันอีก ซึ่งผลลัพธ์ของการสืบค้นครั้งนี้เราสนใจแต่ภาพยนตร์เรื่องอื่นที่ผู้กำกับ และดาราแสดงร่วมทำงานร่วมกัน จึงทำการกำหนดข้อจำกัดในประโยคย่อย FILTER ว่าค่า

ของตัวแปร ?othermovie จะต้องไม่เท่ากับค่าตัวแปร ?movie ทำให้ได้ผลลัพธ์ดังที่แสดงในรูป 3.112

```
PREFIX fb: <http://rdf.freebase.com/ns/>

SELECT ?othermovie ?director ?costar
WHERE
{
  ?movie fb:film.film.starring fb:en.harrison_ford .
  ?movie fb:film.film.directed_by ?director .
  ?movie fb:film.film.starring ?costar .
  ?othermovie fb:film.film.directed_by ?director .
  ?othermovie fb:film.film.starring ?costar .
    FILTER (?othermovie != ?movie)
}
```

รูป 3.111 โค้ดภาษา SPARQL

othermovie	director	costar
http://rdf.freebase.com/ns/en.dark_blue	http://rdf.freebase.com/ns/en.ron_shelton	http://rdf.freebase.com/ns/en.kurupt

รูป 3.112 ผลลัพธ์จากการสืบค้น

3.5.4 การสืบค้นรูปแบบ CONSTRUCT

เป็นรูปแบบการสืบค้นที่นำตัวแปรที่ได้จากชุดคำตอบจากการสืบค้นมาสร้างเป็นกราฟใหม่ขึ้นมา ซึ่งการสืบค้นรูปแบบนี้เราสามารถเขียนโค้ดที่โปรแกรมประยุกต์ให้ทำหน้าที่นี้แทนได้ แต่ภาษา SPARQL ก็จัดหาการสืบค้นรูปแบบนี้มาให้ใช้เพื่อให้ได้ทริปเปิลออกมาเป็นผลลัพธ์ และนำไปเพิ่มลงในกราฟข้อมูลในภายหลังได้ โดยเราจะต้องกำหนดแม่แบบ (Template) ของกราฟที่ต้องการสร้างขึ้นมาใหม่ลงไปในประโยคย่อย CONSTRUCT ซึ่งประโยคย่อยนี้จะมาแทนที่ประโยคย่อย SELECT ในขณะที่ส่วนอื่น ๆ ของการสืบค้นยังคงทำหน้าที่เดิม

ตัวอย่างการสืบค้นในรูป 3.114 เป็นการสร้างทริปเปิลเพื่ออธิบายว่าใครถูกว่าจ้างในปีไหนบ้าง โดยอิงจากข้อมูลตัวอย่าง

```
PREFIX fb:<http://rdf.freebase.com/ns/>

CONSTRUCT
{
  ?who <http://employment.history/was_employed_in> ?year
}
WHERE
{
  {
    ?film fb:film.film.starring ?who .
    ?film fb:film.film.initial_release_date ?year .
  }
  UNION
  {
    ?film fb:film.film.directed_by ?who .
    ?film fb:film.film.initial_release_date ?year .
  }
}
```

รูป 3.113 โค้ดภาษา SPARQL

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:fb="http://rdf.freebase.com/ns/"
    xmlns:j.0="http://employment.history/" >
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.joss_ackland">
    <j.0:was_employed_in>2002</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.kathryn_bigelow">
    <j.0:was_employed_in>2002</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.bob_saget">
    <j.0:was_employed_in>2000</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.kurupt">
    <j.0:was_employed_in>2003</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.mark_strong">
    <j.0:was_employed_in>2008</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.sean_penn">
    <j.0:was_employed_in>2002</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.ridley_scott">
    <j.0:was_employed_in>2008</j.0:was_employed_in>
  </rdf:Description>
```

รูป 3.114 ผลลัพธ์จากการค้นหา

```
  <rdf:Description rdf:about="http://rdf.freebase.com/ns/en.harrison_ford">
    <j.0:was_employed_in>2002</j.0:was_employed_in>
    <j.0:was_employed_in>2003</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description rdf:about="http://rdf.freebase.com/ns/en.russell_crowe">
    <j.0:was_employed_in>2008</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description rdf:about="http://rdf.freebase.com/ns/en.ron_shelton">
    <j.0:was_employed_in>2003</j.0:was_employed_in>
  </rdf:Description>
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.elizabeth_hurley">
    <j.0:was_employed_in>2002</j.0:was_employed_in>
  </rdf:Description>
</rdf:RDF>
```

รูป 3.114 ผลลัพธ์จากการค้นหา (ต่อ)

ผลลัพธ์ที่ได้จากการค้นหาในรูป 3.114 จะเป็นเอกสาร RDF ที่เครื่องมือ Twinkle สร้างขึ้นมาให้ในรูปแบบ RDF/XML และทำการกำหนดเนมสเปสใหม่ขึ้นมาให้คือ j.0

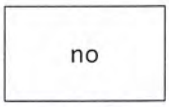
รูปแบบของทริปเปิลที่ได้เพิ่มขึ้นมานั้นจากการสืบค้นรูปแบบ CONSTRUCT นั้นจะแตกต่างกันไปตามแต่ละเครื่องมือที่ใช้ในการสืบค้นด้วยภาษา SPARQL

3.5.5 การสืบค้นรูปแบบ ASK และ DESCRIBE

ในภาษา SPARQL ได้นิยามการสืบค้นรูปแบบ ASK มาให้สำหรับทดสอบดูว่ารูปแบบกราฟที่ต้องการสืบค้นนั้นมีอยู่ในกราฟข้อมูล RDF หรือไม่ การใช้งานทำได้โดยการใช้คำว่า ASK แทนคำว่า WHERE ในการสืบค้น โดยผลลัพธ์ที่ได้คือคำว่า yes และ no ดังเช่นในตัวอย่างในรูป 3.115 และผลลัพธ์ในรูป 3.116 ที่ทำการสืบค้นว่ามีภาพยนตร์เรื่องใดที่ Bob Saget และ Harrison Ford แสดงร่วมกันหรือไม่

```
PREFIX fb:<http://rdf.freebase.com/ns/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

ASK
{
  ?film fb:film.film.starring fb:en.bob_saget .
  ?film fb:film.film.starring fb:en.harrison_ford .
}
```

รูป 3.115 โค้ดภาษา SPARQL



รูป 3.116 ผลลัพธ์จากการสืบค้น

ส่วนการสืบค้นรูปแบบ DESCRIBE จำคืนข้อมูลที่เป็นประโยชน์เกี่ยวกับทรัพยากรที่สืบค้นที่ตัวประมวลผล SPARQL พบซึ่งผลลัพธ์จะแตกต่างกันไปในแต่ละตัวประมวลผล ในทางทฤษฎีแล้วการสืบค้นรูปแบบ DESCRIBE ควรจะช่วยให้ผู้สืบค้นมีความเข้าใจเกี่ยวกับบริบทของทรัพยากรที่ค้นหาอยู่

การสืบค้นรูปแบบ DESCRIBE อย่างง่ายที่สุด คือการสืบค้นที่ถามตัวประมวลผล SPARQL ว่ารู้อะไรเกี่ยวกับทรัพยากรนั้น ๆ บ้างดังตัวอย่างในรูป 3.117 ซึ่งทำให้ได้ผลลัพธ์ดังที่แสดงในรูป 3.118 พบว่าได้ออกมาเพียงทริปเปิลเดียว ที่อธิบายชื่อของทรัพยากร http://rdf.freebase.com/ns/en.harrison_ford แทนที่จะได้ทริปเปิลที่บอกว่าทรัพยากรนั้นแสดงภาพยนตร์เรื่องใดบ้าง

```
DESCRIBE http://rdf.freebase.com/ns/en.harrison_ford
```

รูป 3.117 โค้ดภาษา SPARQL

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:fb="http://rdf.freebase.com/ns/" >
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.harrison_ford">
    <fb:type.object.name>Harrison Ford</fb:type.object.name>
  </rdf:Description>
</rdf:RDF>
```

รูป 3.118 โค้ดภาษา SPARQL

และอีกตัวอย่างหนึ่งของการสืบค้นรูปแบบ DESCRIBE โดยทำการแทนที่ประโยคย่อย SELECT ในการสืบค้นได้เลย ซึ่งผลลัพธ์ที่ได้จะเป็นข้อมูลที่ตัวประมวลผล SPARQL คิดว่ามีประโยชน์ในการอธิบายทรัพยากรที่สืบค้น ดังในตัวอย่างในรูป 3.119 ที่ต้องการ DESCRIBE ผู้กำกับที่กำกับภาพยนตร์ในปี 2003 และผลลัพธ์แสดงในรูป 3.120

```
PREFIX fb:<http://rdf.freebase.com/ns/>

DESCRIBE ?director
WHERE
{
  ?film fb:film.film.initial_release_date "2003" .
  ?film fb:film.film.directed_by ?director .
}
```

รูป 3.119 โค้ดภาษา SPARQL

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:fb="http://rdf.freebase.com/ns/" >
  <rdf:Description
rdf:about="http://rdf.freebase.com/ns/en.ron_shelton">
    <fb:type.object.name>Ron Shelton</fb:type.object.name>
  </rdf:Description>
</rdf:RDF>
```

รูป 3.120 ผลลัพธ์จากการสืบค้น

3.5.6 การสืบค้นระหว่างกราฟข้อมูล (Querying between Named and Background Graphs)

เราสามารถทำการสืบค้นข้อมูลจากกราฟหลายกราฟได้ในการสืบค้นครั้งหนึ่ง โดยจะมีกราฟหลัก (Background graph) ที่เราทำงานด้วยในปัจจุบันซึ่งไม่มีชื่อเรียก และไม่มีกราฟมีชื่ออยู่ภายในที่อ้างอิงผ่าน URI และกราฟมีชื่อ (Named graph) ซึ่งเป็นกราฟอื่นที่จะเข้าไปสืบค้นข้อมูลมา โดยในภาษา SPARQL จะเรียกกราฟกราฟหนึ่งว่า RDF Dataset

รูป 3.121 ถึงรูป 3.123 แสดง RDF Dataset ตัวอย่างที่จะทำการสืบค้นโดยเขียนอธิบายอยู่ในรูป N3

```
# Background graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<http://example.org/bob> dc:publisher "Bob" .
<http://example.org/alice> dc:publisher "Alice" .
```

รูป 3.121 RDF Dataset หรือ Background graph

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX data: <http://example.org/foaf/>

# Graph: http://example.org/foaf/aliceFoaf
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .
_:a foaf:knows _:b .
_:b rdfs:seeAlso <http://example.org/foaf/bobFoaf> .
<http://example.org/foaf/bobFoaf>                                    rdf:type
foaf:PersonalProfileDocument .
_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
_:b foaf:age 32 .
```

รูป 3.122 RDF Dataset หรือ http://example.org/foaf/aliceFoaf graph

```
# Graph: http://example.org/foaf/bobFoaf
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
 _:1 foaf:mbox <mailto:bob@work.example> .
 _:1 rdfs:seeAlso <http://example.org/foaf/bobFoaf> .
_:1 foaf:age 35 .
<http://example.org/foaf/bobFoaf>                                    rdf:type
foaf:PersonalProfileDocument .
```

รูป 3.123 RDF Dataset หรือ http://example.org/foaf/bobFoaf

ในตัวอย่างการสืบค้นในรูป 3.124 จะแสดงการเจาะจงกราฟที่ต้องการเข้าไปสืบค้นด้วยการเพิ่มประโยคย่อย GRAPH เข้าไปแล้วต่อท้ายด้วย ตัวแปร หรือ IRI ที่อ้างไปยังกราฟ โดยในตัวอย่างจะใช้การอ้างถึง IRI อย่างย่อโดยการใช้คำนำหน้าว่า data:bobFoaf และได้ผลลัพธ์แสดงในรูป 3.123

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX data: <http://example.org/foaf/>

SELECT ?age
WHERE
{
  GRAPH data:bobFoaf
  {
    ?x foaf:mbox <mailto:bob@work.example> .
    ?x foaf:age ?age
  }
}
```

รูป 3.124 โค้ดภาษา SPARQL

เราสามารถทำการสืบค้นข้อมูลบนกราฟข้อมูลมากกว่าหนึ่งกราฟได้โดยการใช้ประโยคย่อย FROM NAMED แล้วต่อท้ายด้วย IRI ที่อ้างอิงไปยังกราฟมีชื่อ ดังเช่นตัวอย่างในรูป 3.125 และผลลัพธ์จากการสืบค้นในรูป 3.126

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

SELECT ?src ?bobNick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
{
  GRAPH ?src
  {
    ?x foaf:mbox <mailto:bob@work.example> .
    ?x foaf:nick ?bobNick
  }
}
```

รูป 3.125 โค้ดภาษา SPARQL

Src	bobnick
http://example.org/foaf/aliceFoaf	Bobby
http://example.org/foaf/bobFoaf	Robert

รูป 3.126 ผลลัพธ์จากการสืบค้น

นอกจากนี้ยังมีคำสั่งอื่น ๆ อีกที่เกี่ยวข้องกับการแสดงผลลัพธ์จากการสืบค้น เช่น คำสั่ง ORDER BY สำหรับใช้ในการเรียงผลลัพธ์ วิธีใช้งานคือระบุตัวแปรที่ใช้ในรูปแบบทริปเปิลในการค้นหาตามหลัง และอาจระบุด้วยว่าให้เรียงจากมากไปน้อยที่ตัวแปรหนึ่ง ๆ ด้วยโอเปอเรเตอร์ DESC โดยส่งผ่านตัวแปรที่ต้องการเข้าไป ยกตัวอย่างเช่น การเพิ่มประโยคย่อย ORDER BY ?who DESC(?film) เข้าไปในการสืบค้นในรูป 3.103 จะทำให้ได้ผลลัพธ์ที่มีการเรียงชื่อจากตัวอักษรน้อยไปยังตัวอักษรมากในคอลัมน์ who เป็นอันดับแรก และหากมีการเท่ากันในคอลัมน์แรก จึงจะไปจัดเรียงในคอลัมน์ film จากมากไปน้อยทีหลัง และยังมีคำสั่ง LIMIT ตามด้วยตัวเลขจำนวนเต็มบวกเพื่อระบุจำนวนผลลัพธ์มากที่สุดที่ต้องการให้ตัวประมวลผลภาษา SPARQL ส่งคืน ซึ่งมักจะใช้คู่กับคำสั่ง OFFSET ซึ่งมีตัวเลขจำนวนเต็มบวกกำกับเหมือนกัน สำหรับระบุว่าผลลัพธ์แรกสุดที่นำมาแสดงนั้นเป็นผลลัพธ์ที่เท่าไหร่ในบรรดาผลลัพธ์ทั้งหมด

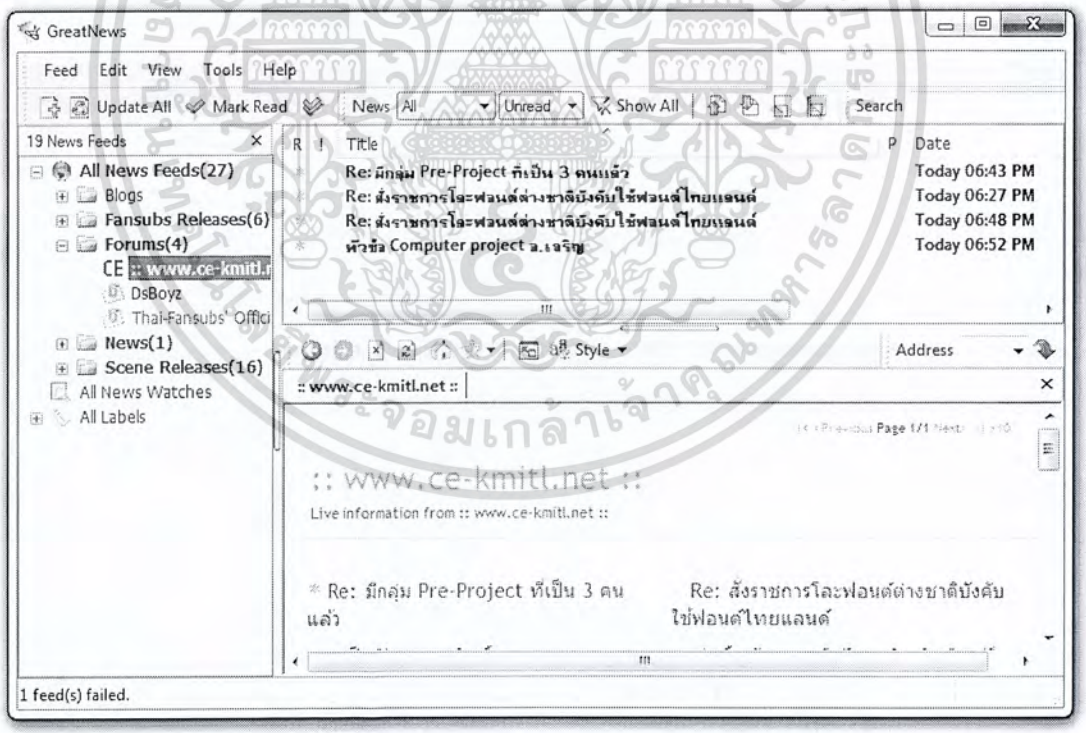
สังเกตว่าเราสามารถทำได้เพียงสืบค้นข้อมูลจากข้อมูลกราฟ RDF ขึ้นมาอย่างเดียว ไม่เหมือนกับภาษา SQL ที่มีคำสั่งสำหรับแก้ไขข้อมูลได้แก่คำสั่ง INSERT และคำสั่ง DELETE ซึ่ง

ข้อจำกัดนี้ทำให้เรายอมให้ระบบภายนอกเข้ามาใช้ข้อมูลของเราได้โดยไม่ต้องกังวลว่าข้อมูลกราฟจะถูกแก้ไข ซึ่งเป็นการออกแบบเชิงสถาปัตยกรรมที่ทรงพลัง

3.6 ตัวอย่างการนำ RDF ไปประยุกต์ใช้งาน

ตัวอย่างการนำ RDF ไปประยุกต์ใช้งานในปัจจุบันตัวอย่างหนึ่ง คือ การใช้ RSS (RDF Site Summary) โดยนำมาใช้เป็นตัวบรรยายข้อมูลของข้อมูล (Metadata) ที่อเนกประสงค์ และกะทัดรัด RSS เป็นการประยุกต์ใช้ภาษา XML ซึ่งตรงตามมาตรฐาน RDF ที่กำหนดโดย W3C และรองรับการขยายด้วยการใช้เนมสเปสของภาษา XML และการจัดแบ่งประเภทของภาษา RDF

เอกสาร RSS ในรูปแบบกะทัดรัดที่สุดที่มีไว้เพื่ออธิบายช่องทางข่าวสาร โดยข้อมูลข่าวสารจะมี URL ที่สามารถเข้าถึงได้กำกับไว้อยู่ โดยแต่ละข้อมูลข่าวสารนั้นจะประกอบไปด้วย หัวเรื่อง ลิงค์ และรายละเอียดโดยสังเขป ข้อมูลข่าวสารที่เก็บบันทึกอยู่ใน RSS โดยทั่วไปมักจะมาจาก เว็บข่าวสาร กระดานสนทนา เป็นส่วนใหญ่ โดยโปรแกรมประยุกต์ที่ทำงานกับเอกสาร RSS เรียกว่าตัวรับฟีด (Feed subscriber) ยกตัวอย่างเช่น Google Reader, GreatNews เป็นต้น



รูป 3.127 ตัวรับฟีดชื่อ GreatNews

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
>
  <channel rdf:about="http://example.com/news.rss">
    <title>Example Channel</title>
    <link>http://example.com/</link>
    <description>My example channel</description>
    <items>
      <rdf:Seq>
        <rdf:li resource="http://example.com/2002/09/01/"/>
        <rdf:li resource="http://example.com/2002/09/02/"/>
      </rdf:Seq>
    </items>
  </channel>
  <item rdf:about="http://example.com/2002/09/01/">
     <title>News for September the First</title>
     <link>http://example.com/2002/09/01/</link>
     <description>other things happened today</description>
     <dc:date>2002-09-01</dc:date>
  </item>
  <item rdf:about="http://example.com/2002/09/02/">
     <title>News for September the Second</title>
     <link>http://example.com/2002/09/02/</link>
     <dc:date>2002-09-02</dc:date>
  </item>
</rdf:RDF>
```

รูป 3.128 ตัวอย่างเอกสาร RSS

จากรูปที่ 3.128 เริ่มมาจะเป็นการอธิบายโหนดชนิดช่องทาง (Channel) โดยมีคุณสมบัติดังต่อไปนี้ ชื่อช่องทาง ลิงค์ไปยังช่องทาง รายละเอียดช่องทางโดยสังเขป และข่าวสารที่เอกสาร RSS มีอยู่ หลังจากนั้นจึงค่อยอธิบายรายละเอียดของข่าวสารแต่ละข่าวที่ได้แจกแจงไว้ในภาคแสดง items ที่ใช้ในการอธิบายช่องทาง โดยแต่ละข่าวสารโดยทั่วไปจะประกอบไปด้วย หัวเรื่อง ลิงค์ไปยังข่าวสาร รายละเอียดโดยสังเขป และอาจมีการใส่ข้อมูลอื่น ๆ อธิบายเพิ่มเติมไปได้ด้วย ดังเช่น การกำกับวันที่ของแต่ละข่าวสารเอาไว้ด้วยภาคแสดง dc:date เป็นต้น

นอกจากนี้ยังมีบริการอื่น ๆ ที่นำเอาภาษา RDF ไปใช้งาน อย่างเช่น เว็บไซต์ DBpedia.org เป็นต้น ที่นำเอาข้อมูลจากเว็บไซต์ Wikipedia.com มาแปลงเป็นข้อมูลรูปแบบภาษา RDF เพื่อให้ระบบภายนอกเข้ามาสืบค้น ดังที่ได้แสดงไว้ในการทดลอง

3.7 การเปลี่ยนรูปแบบจำลองเชิงสัมพันธ์ให้เป็นแบบจำลอง RDF

เนื่องจากข้อมูลที่ถูกจัดระเบียบแล้ว (Formatted data) ส่วนมากถูกเก็บบันทึกเอาไว้ด้วยฐานข้อมูลเชิงสัมพันธ์ (Relational databases) เพื่อให้สามารถนำข้อมูลดังกล่าวนำมาใช้งานได้ในแง่ของซีแมนติคจึงต้องแปลงให้อยู่ในรูปแบบ RDF

ในการใช้งานฐานข้อมูลเชิงสัมพันธ์ จะต้องมีการสร้างข้อตกลงกันในเรื่องของเค้าร่างฐานข้อมูล (Schema) ก่อนที่จะนำข้อมูลมาเก็บบันทึกลงได้ ซึ่งความยากลำบากของการสร้างข้อตกลงกันในในเรื่องของเค้าร่างฐานสำหรับข้อมูลภายในพื้นที่ร่วม อย่างเช่นเว็บไซต์ในปัจจุบันนั้นกำลังเพิ่มขึ้น ส่วน RDF จะยินยอมให้ข้อมูลถูกบันทึก และสืบค้นได้โดยไม่จำเป็นต้องกำหนดเค้าร่างขึ้นมาก่อน นอกจากนี้ RDF ยังยินยอมให้เค้าร่างมีการเปลี่ยนแปลงเกิดขึ้นได้โดยไม่ขึ้นอยู่กับเนื้อข้อมูลทำให้ไม่จำเป็นต้องมีข้อมูลที่ถูกตัดทิ้งออกไป หรือถูกเติมด้วยค่า NULL เหมือนอย่างที่ทำกันในฐานข้อมูลเชิงสัมพันธ์ และค่า NULL ที่ว่าอาจมีความหมายต่างกันสำหรับแต่ละแหล่งข้อมูล และจะให้ผลลัพธ์จากการสืบค้นที่แตกต่างกันตามระบบฐานข้อมูลที่ใช้ในแต่ละแหล่งข้อมูล ดังนั้นการที่ไม่ต้องมีการใช้งานค่า NULL นั้นส่งผลกระทบในแง่บวก และกลายเป็นประเด็นสำคัญโดยเฉพาะอย่างยิ่งเมื่องานหลักของเว็บสื่อความหมาย (Semantic web) คือการรวบรวมข้อมูลจากหลาย ๆ แหล่งที่แตกต่างกัน

แบบจำลอง RDF จะมีความเป็นโครงสร้างน้อยกว่าแบบจำลองเชิงสัมพันธ์โดยทั่ว ๆ ไป แต่ก็สเตทเมนต์ RDF ก็ยังมีโครงสร้างที่ตายตัวอันประกอบไปด้วย Subject, Predicate และ Object ทำให้เป็นไปได้ที่จะนำเสนอข้อมูลในรูปแบบเชิงสัมพันธ์ นอกจากนี้ยังยินยอมให้ข้อมูลที่มีโครงสร้างต่างกันเล็กน้อยถูกเก็บบันทึกเข้าด้วยกันได้ในรูปแบบสเตทเมนต์ RDF จึงทำให้ไม่จำเป็นต้องเลือกระหว่างการออกแบบเค้าร่างที่ดี กับประสิทธิภาพในการสืบค้นข้อมูลที่ดีเพื่อที่จะเก็บบันทึกข้อมูลที่มีโครงสร้างที่แตกต่างออกไป

วิธีการเปลี่ยนรูปที่จะนำเสนอต่อไปนี้ประกอบไปด้วยองค์ประกอบหลัก 2 องค์ประกอบคือการระบุซีแมนติคของแบบจำลองเชิงสัมพันธ์ และการเปลี่ยนรูปซีแมนติคที่ระบุได้ในขั้นตอนแรกให้อยู่ในรูปแบบจำลอง RDF

3.7.1 Relational Model and RDF Model

ในหัวข้อนี้จะเอ่ยถึงแนวคิดพื้นฐานของแบบจำลองเชิงสัมพันธ์ และความสัมพันธ์ระหว่างแบบจำลองเชิงสัมพันธ์ และแบบจำลอง RDF

3.7.1.1 แบบจำลองเชิงสัมพันธ์ (Relational Model)

แบบจำลองที่เป็นรากฐานฐานข้อมูลเชิงสัมพันธ์คือแบบจำลองเชิงสัมพันธ์ ประกอบไปด้วย

1) โดเมน (Domain) คือเซ็ตไม่ว่างเปล่า (Non empty set) ซึ่งตามปกติแล้วจะเป็นแหล่งรวม ของค่าข้อมูล และจะสมมติทุก ๆ โดเมนมีชื่อกำกับเอาไว้

2) เค้าร่าง (Schema) คือรายการของชื่อโดเมน เขียนอธิบายว่า $R = \{A_1, \ldots, A_n\}$

3) รีเลชัน (Relation) คือเซ็ตย่อยของผลคูณคาร์ทีเซียน ระหว่างโดเมนทุกตัวภายในรายการ สำหรับเค้าร่างหนึ่ง ๆ

4) ทิวเพิล (Tuple) คือสมาชิกของความสัมพันธ์

5) คีย์ (Key) อธิบายด้วยตัวอย่างเช่น คีย์ K สำหรับความสัมพันธ์ r ในเค้าร่าง R เป็นเซ็ตย่อยของ R ซึ่งใช้ระบุว่าทิวเพิล 2 ทิวเพิลใด ๆ จะเท่ากันได้นั้น จะต้องมีค่าข้อมูล K เท่ากัน

เค้าร่างฐานข้อมูลเชิงสัมพันธ์ประกอบไปด้วยเค้าร่างเชิงสัมพันธุชุดหนึ่ง รวมเข้าด้วยกันกับข้อบังคับต่าง ๆ ชุดหนึ่งซึ่งควบคุมให้ข้อมูลภายในฐานข้อมูลมีความถูกต้อง (Integrity constraints) ข้อบังคับดังกล่าวคือเพรดิเคต (Predicate) ที่มีผลกับรีเลชันซึ่งจะอธิบายข้อบังคับต่าง ๆ ซึ่งข้อบังคับชนิดที่นิยมใช้กันมากที่สุดก็คือ ข้อบังคับในการอ้างอิง (Referential integrity constraints) ข้อบังคับในการอ้างอิงหนึ่ง ๆ จะเกี่ยวข้องกับเซ็ต 2 เซ็ตของแอตทริบิวต์เช่น S1 และ S2 ภายในรีเลชัน T1 และ T2 ตามลำดับโดยที่เซ็ตหนึ่ง (เช่น S1) จะเป็นคีย์ของรีเลชั่นหนึ่ง ๆ (เช่น T1) ส่วนอีกเซ็ตหนึ่งจะเป็นคีย์นอก (Foreign key) หาก T2[S2] เป็นเซ็ตย่อยของ T1[S1] อาจพูดได้ว่าข้อบังคับในการอ้างอิงเป็นยึดเหนี่ยวกันระหว่างกันของรีเลชันต่าง ๆ ภายในฐานข้อมูล ดังนั้นในรีเลชันจึงจำเป็นจะต้องระบุว่าอะไรเป็นคีย์หลัก และควรจะมีข้อบังคับอะไรบ้างเพื่อความถูกต้องของข้อมูล

3.7.1.2 ความสัมพันธ์ระหว่างแบบจำลองเชิงสัมพันธ์ และแบบจำลอง RDF

RDF triple สามารถอธิบายข้อเท็จจริงง่าย ๆ ได้อย่างเช่นความสัมพันธ์ระหว่างของ 2 สิ่งโดยใช้เพรดิเคตในการตั้งชื่อความสัมพันธ์ที่ว่า และประธาน กับกรรมสำหรับของ 2 สิ่งข้างต้น วิธีการนำเสนอที่คุ้นเคยกันดีคือการนำข้อเท็จจริงที่ว่ามาเป็นแถวในตารางในฐานข้อมูลเชิงสัมพันธ์

โดยตารางจะมี 2 คอลัมน์ซึ่งตรงกันกับประธาน และกรรมของ RDF triple ส่วนชื่อของตารางจะเป็นตามชื่อของเพรดิเคตใน RDF triple นั้น ๆ

ฐานข้อมูลเชิงสัมพันธ์ยอมให้ตารางหนึ่งตารางมีคอลัมน์จำนวนเท่าไหร่ก็ได้ ซึ่งข้อมูลใน 1 แถวจะแสดงถึงข้อมูลที่เกี่ยวข้องกับสิ่งที่ตารางนั้นต้องการอธิบาย โดยข้อมูลแต่ละแถวนั้นจะต้องถูกแยกย่อยองค์ประกอบเพื่อเอามานำเสนอให้อยู่ในรูปแบบ RDF ยิ่งไปกว่านั้นในแบบจำลองเชิงสัมพันธ์บริสุทธิ์นั้น ทุกตารางจะต้องมีคีย์หลัก นั่นคือคอลัมน์ที่เก็บค่าที่ไม่ซ้ำกับแถวอื่นใด เพื่อใช้ในการระบุแถวข้อมูลนั้น ๆ ซึ่งใช้ไม่ได้กับเว็บ วิธีแก้ปัญหาวิธีหนึ่งคือให้ใช้คีย์หลักเป็นหลักกลางใน RDF triple หรือใช้อัตลักษณ์ประจำแถวข้อมูล แล้วทำการแปลงจากแถวของตารางไปเป็น RDF triples ตามขั้นตอนดังนี้

1) คีย์หลักจะตรงกับโหนดที่เป็นประธานร่วมซึ่งมีคุณสมบัติ rdf:type ที่มีค่าเป็นชื่อของตารางนั้น ๆ
2) ชื่อคอลัมน์ของตารางจะกลายเป็นภาคแสดง หรือเพรดิเคต
3) ค่าข้อมูลภายในเซลล์จะเป็นกรรม

ดังนั้นข้อเท็จจริงที่ซับซ้อนกว่านั้นจะถูกอธิบายด้วย RDF ด้วยการรวมกันผ่านความสัมพันธ์ไบนารี RDF สามารถใช้อธิบายข้อเท็จจริงเกี่ยวกับเรื่องใด ๆ ก็ได้ผ่านการใช้รายการคำศัพท์ (Vocabulary) แบบ URI ซึ่งขยายได้ โดย URI จะสร้างมาเพื่อสำหรับอะไรก็ได้ที่สามารถถูกตั้งชื่อได้จึงทำให้ข้อเท็จจริงที่อธิบายด้วย RDF เกี่ยวกับเรื่องใดก็ได้

3.7.2 การเปลี่ยนรูปแบบจำลองเชิงสัมพันธ์ให้เป็นแบบจำลอง RDF

ขั้นตอนการเปลี่ยนรูปประกอบด้วย 2 ขั้นตอนหลักคือ ระบุซีแมนติคภายในแบบจำลองเชิงสัมพันธ์ที่กำลังพิจารณา แล้วจึงค่อยทำการเปลี่ยนรูปให้กลายเป็นแบบจำลอง RDF

3.7.2.1 ระบุซีแมนติคภายในแบบจำลองเชิงสัมพันธ์

การระบุซีแมนติคภายในแบบจำลองเชิงสัมพันธ์นั้นอาจต้องอ้างถึงวิธีการบางวิธีในกระบวนการวิศวกรรมย้อนกลับ (Reverse engineering) บางวิธี ซึ่งในกระบวนการดังกล่าวสำหรับหัวข้อนี้จะประกอบไปด้วย 3 ขั้นตอนคือ ระบุเอนทิตี, ระบุความสัมพันธ์ และระบุจำนวนสมาชิก ซึ่งขั้นตอน 3 ขั้นตอนดังกล่าวจะใช้ได้ต่อเมื่อเค้าร่างเชิงสัมพันธ์ที่จะนำมาเปลี่ยนรูปนั้นจะต้องอยู่ในรูป 3NF เป็นอย่างต่ำ นั่นหมายความว่าจะไม่มีการขึ้นต่อกันแบบถ่ายทอด (Transitive dependencies) และแอตทริบิวต์แบบนอนคีย์ (Non-key attributes) ทุกตัวมีความพึ่งพิงแบบฟังก์ชัน (Functionally dependent) อยู่บนคีย์ของรีเลชัน

ขั้นตอน 3 ขั้นตอนที่จะนำเสนอต่อไปนี้จะยกตัวอย่างประกอบอันเป็นแบบจำลองเชิงสัมพันธ์เกี่ยวกับธุรกิจ ดังแสดงในตาราง 3.4

ตาราง 3.4 ตัวอย่างเค้าร่างเชิงสัมพันธ์

รายการตาราง	รายการแอตทริบิวต์	คีย์นอก และรีเลชันที่อ้างอิง
Account	accountID, customerID, isbalance	customerID(Customer)
Customer	customerID, name, city	-
Subbank	bankID, bankname	-
Loan	loanID, customerID, loanAmount	customerID(Customer)
SavingAccount	accountID, interrate	accountID(Account)
CheckingAccount	accountID, overdraftNumber	accountID(Account)
Establish	bankID, customerID, edate	bankID(Bank), customerID(Customer)
Payment	loanID, paymentID, paymentDate, paymentAmount	loadID(Loan)

แอตทริบิวต์ภายในตารางที่มีเส้นใต้ขีดอยู่หมายถึงเป็นคีย์หลักของรีเลชัน ส่วนชื่อภายในวงเล็บที่ตามท้ายชื่อของคีย์นอกนั้นอ้างถึงชื่อตารางที่ตัวมันเองเป็นคีย์หลักอยู่ ส่วนข้อมูลตัวอย่างของบางตารางจะแสดงอยู่ในตาราง 3.5 ถึง 3.7

ตาราง 3.5 ข้อมูลตัวอยย่างของตาราง Saving Account

accountID	interrate
102	20

ตาราง 3.6 ข้อมูลตัวอย่างของตาราง Account

accountID	customerID	isbalance
102	1004	F

ตาราง 3.7 ข้อมูลตัวอย่างของตาราง Customer

customerID	Name	City
1004	John Doe	Washington

3.7.3 กระบวนการเปลี่ยนรูป

3.7.3.1 การระบุเอนทิตี

ระบุเอนทิตี (Identifying entity) ตารางเชิงสัมพันธ์จะถูกเปลี่ยนรูปไปเป็นเอนทิตีได้ 2 กรณีดังนี้ กรณีแรกคือทุกตารางที่คีย์หลักประกอบขึ้นจากแอตทริบิวต์เพียงแอตทริบิวต์เดียว และกรณีที่สองสำหรับตารางที่คีย์หลักมีมากกว่า 1 แอตทริบิวต์และมีแอตทริบิวต์อย่างน้อย 1 ตัวที่ไม่ใช่คีย์นอก ตารางที่เข้าข่ายสองกรณีนี้จะถูกเรียกว่าตารางเอนทิตี (Entity-table) จากในตัวอย่างเค้าร่างเชิงสัมพันธ์พบว่ามีเพียงตาราง Establish เท่านั้นที่ไม่ได้เป็นตารางเอนทิตี

3.7.3.2 การระบุความสัมพันธ์

ระบุความสัมพันธ์ (Identifying relationship) โดยทั่วไปแล้วมีความสัมพันธ์อยู่ 3 ประเภทที่สามารถนำเสนอในแบบจำลองเชิงสัมพันธ์ได้ดังนี้ ความสัมพันธ์แบบเชื่อมติด (Association), ความสัมพันธ์แบบถ่ายทอด (Inheritance) และความสัมพันธ์แบบรวมกัน (Aggregation) โดยระบุขั้นตอนตามนี้

1) ระบุความสัมพันธ์แบบเชื่อมติด (Identifying association) ตาราง 2 ตารางจะมีความเชื่อมติดกันได้ก็ต่อเมื่อรองรับกรณีใดกรณีหนึ่งใน 2 กรณีดังนี้ กรณีแรกคือ ทุกตารางที่คีย์หลักประกอบด้วยแอตทริบิวต์มากกว่า 1 ตัว และเป็นคีย์นอกทั้งหมด ซึ่งนำเสนอความเชื่อมติดระหว่างเอนทิตีต่าง ๆ ที่อยู่ในตารางที่กำลังพิจารณา โดยจะมีความสัมพันธ์แบบ Many-to-many และเรียกตารางที่รองรับกับกรณีนี้ว่า ตารางเชื่อมติด (Association-table) แต่หากตารางเชื่อมติดดังกล่าวมีแอตทริบิวต์ของตัวเองด้วยจะทำให้ตารางที่กำลังพิจารณานี้เป็นตารางเอนทิตีด้วย กรณีที่สองเกิดจากความสัมพันธ์ที่เกิดขึ้นระหว่างเอนทิตีกับอีกเอนทิตีซึ่งสอดคล้องกับคีย์นอกภายในรีเลชัน เว้นแต่ว่ามันจะเป็นส่วนหนึ่งของคีย์หลัก จากในตัวอย่างเค้าร่างเชิงสัมพันธ์พบว่าคีย์หลักของตาราง Establish ประกอบขึ้นจาก 2 แอตทริบิวต์ และเป็นคีย์นอกทั้งหมด ทำให้ถูกระบุว่าเป็นตารางเชื่อมติด นอกจากนี้จากการที่คีย์นอกปรากฏภายในรีเลชันต่าง ๆ ทำให้เกิดความสัมพันธ์แบบเชื่อมติดเกิดขึ้นระหว่าง Account กับ Customer และ Customer กับ Loan สังเกตว่าจะไม่มีความสัมพันธ์แบบเชื่อมติดเกิดขึ้นระหว่างตารางเอนทิตี Account กับ SavingAccount, Account กับ CheckingAccount และ Loan กับ Payment

2) ระบุความสัมพันธ์แบบถ่ายทอด (Identifying inheritance) ทุก ๆ คู่ของตารางเอนทิตี (ET1, ET2) ที่มีคีย์หลักเหมือนกันอาจมีความสัมพันธ์แบบถ่ายทอดกันได้ ET1 จะมีความสัมพันธ์แบบถ่ายทอดในรูปแบบ “is-a” กับ ET2 ก็ต่อเมื่อคีย์หลักของ ET1 เป็นคีย์นอกซึ่งเป็นคีย์หลักของ ET2 หรือในภาษา RDFS กล่าวได้ว่า ET1 เป็นคลาสย่อยของ ET2 นั่นเอง

จากในตัวอย่างเค้าร่างเชิงสัมพันธ์พบว่าตารางเอนทิตี SavingAccount มีความสัมพันธ์รูปแบบ “is-a” กับตารางเอนทิตี Account เพราะทั้งคู่ต่างมีคีย์หลักเหมือนกัน ส่วนคีย์หลักในตาราง SavingAccount ยังเป็นคีย์นอกซึ่งเป็นคีย์หลักในตาราง Account อีกด้วย นอกจากนี้ตารางเอนทิตี CheckingAccount ก็มีความสัมพันธ์รูปแบบ “is-a” กับตารางเอนทิตี Account ด้วย

3) ระบุความสัมพันธ์แบบรวมกัน (Identifying aggregation) พิจารณาตารางเอนทิตี ET1 ที่มีคีย์หลักประกอบขึ้นมาจากแอตทริบิวต์มากกว่า 1 ตัว และมีอย่างน้อย 1 แอตทริบิวต์ที่ไม่ใช่คีย์นอก หากคีย์นอกใน ET1 เป็นคีย์หลักในตาราง ET2 แล้วจะเกิดการรวมกันแบบ “is-part-of” ระหว่าง ET1 และ ET2 ตามลำดับ

จากตัวอย่างเค้าร่างเชิงสัมพันธ์พบว่าคีย์หลักของตารางเอนทิตี Payment มี 2 แอตทริบิวต์ และหนึ่งในนั้น (paymentID) ไม่ใช่คีย์นอก จากนั้นคีย์นอก (loanID) ยังอ้างถึงคีย์หลักของตารางเอนทิตี Loan ดังนั้นตารางเอนทิตี Payment มีความสัมพันธ์รูปแบบ “is-part-of” กับตาราง Loan

3.7.3.4 ระบุจำนวนสมาชิก

ระบุจำนวนสมาชิก (Identifying cardinalities) มีจำนวนสมาชิกที่เกิดจากการเชื่อมติดกันอยู่ 3 ประเภทนั่นคือ แบบ 1-1, 1-many และ many-many ซึ่งประเภทสุดท้ายนั้นได้ถูกระบุไปแล้วผ่านการจำแนกให้ไปเป็นตารางเชื่อมติดในขั้นตอนก่อนหน้านั้น

1) ระบุความเชื่อมติดแบบ 1-1 (Identifying 1-1 association) ให้พิจารณาความเชื่อมติด A(ET1, ET2) ระหว่าง 2 ตารางเอนทิตี ET1 และ ET2 หาก ET1 มีคีย์ให้เลือก (Candidate key) มากกว่า 1 ชุด และหนึ่งในนั้นเป็นคีย์นอกซึ่งอ้างถึง ET2 แล้วอาจเกิดความเชื่อมติดระหว่าง ET1 และ ET2 แบบ 1-1 ได้

จากตัวอย่างเค้าร่างเชิงสัมพันธ์ หากตารางเอนทิตี Loan มีคีย์ให้เลือก 2 ชุด นั่นคือ loanID และ customerID เป็นต้น แอตทริบิวต์ใดแอตทริบิวต์หนึ่งในนั้นสามารถระบุแถวข้อมูลในตาราง Loan ที่แตกต่างกันได้ จึงทำให้สามารถเกิดความสัมพันธ์แบบเชื่อมติดแบบ 1-1 ได้ในระหว่าง 2 ตารางเอนทิตีนี้ผ่านคีย์นอก customerID ซึ่งหมายความว่าลูกค้าคนหนึ่งสามารถกู้เงินได้เพียง 1 ครั้ง และจะกู้ได้อีกก็ต่อเมื่อใช้เงินกู้เก่าไปแล้วเท่านั้น

2) ระบุความเชื่อมติดแบบ 1-many (Identifying 1-many association) หากในความสัมพันธ์แบบเชื่อมติด A(ET1, ET2) โดยที่ตารางเอนทิตี ET2 มีคีย์นอกซึ่งอ้างอิงถึงตารางเอนทิตี ET1 แต่ตารางเอนทิตี ET1 ไม่ได้มีคีย์นอกที่อ้างอิงถึงตารางเอนทิตี ET2 แล้วเป็นไปได้ที่อินสแตนซ์มากกว่า 1 ตัวในตาราง ET2 มีความเชื่อมติดกับแต่ละอินสแตนซ์ใน ET1

3.7.4 การเปลี่ยนรูปให้เป็นแบบจำลอง RDF

เพื่อให้ได้แบบจำลอง RDF ที่เทียบเท่ากับแบบจำลองเชิงสัมพันธ์ให้ทำตามขั้นตอนดังนี้

1) สร้าง RDFS class สำหรับแต่ละตารางเอนทิตี

2) สำหรับทุก ๆ แถวของแต่ละตาราง ให้สร้าง URI สำหรับระบุอินสแตนซ์ขึ้นจากแอตทริบิวต์ต่าง ๆ ที่เป็นคีย์หลัก

3) กำหนด URI สำหรับภาคแสดงให้กับแต่ละแอตทริบิวต์ที่ไม่เป็นคีย์

4) กำหนดภาคแสดง rdf:type ให้แก่อินสแตนซ์ในขั้นตอนที่ 2 โดยเชื่อมกับ URI ของ RDFS class ที่สอดคล้อง

5) สำหรับแต่ละคอลัมน์ที่ไม่ได้เป็นคีย์หลักหรือคีย์นอกให้สร้างทริปเปิล ซึ่งมีประธานเป็น URI จากขั้นตอนที่ 2 มีภาคแสดงเป็น URI จากขั้นตอนที่ 3 และกรรมเป็นค่าภายในคอลัมน์นั้น

6) พิจารณาตารางเชื่อมติดดูว่ามีแอตทริบิวต์ของตัวเองหรือไม่ หากมีก็จะได้รับการสร้าง RDFS class ตามขั้นตอนที่ 1 ตามปกติ แต่หากไม่มี ก็ไม่จำเป็นต้องสร้าง RDFS class ใหม่ให้ แต่ให้ทำการเชื่อมโยงหากันโดยตรง ระหว่างอินสแตนซ์ของ 2 RDFS class ที่เกี่ยวข้องตรง ๆ

7) สำหรับคอลัมน์ที่อ้างถึงคีย์นอกในตารางเอนทิตีให้ทำสร้าง URI สำหรับภาคแสดงขึ้นมาแล้วจากนั้นสร้างทริปเปิลที่มีประธานเป็นอินสแตนซ์ของคลาสนั้น ๆและมีกรรมเป็น

อินสแตนซ์ของอีกคลาสที่มาจากอีกตารางเอนทิตีหนึ่งที่ถูกอ้างถึงผ่านคีย์นอกข้างต้น

นอกจากขั้นตอนข้างต้นแล้ว บางทีเราสามารถเพิ่มภาคแสดงที่ย้อนทิศกลับได้เพื่อความสะดวกในการสืบค้นบนกราฟข้อมูล RDF ที่ได้แปลงมา ทำให้สามารถเริ่มต้นทำการสืบค้นจากอินสแตนซ์อื่น ๆ ได้

3.7.5 การนำไปใช้ประโยชน์

ขั้นตอนการเปลี่ยนรูปแบบจำลองเชิงสัมพันธ์ไปเป็นแบบจำลอง RDF ที่เสนอในหัวข้อนี้สามารถนำไปใช้ประโยชน์ในการนิยามการแปลงกราฟ RDF จำลองด้วยเครื่องมือสำหรับจุดประสงค์นี้ได้เป็นอย่างดี

3.8 Open Sesame

OpenRDF Sesame เป็นไลบรารีภาษา java สำหรับใช้ทำงานกับข้อมูลกราฟ RDF พัฒนาโดยบริษัท Aduana จุดเด่นของไลบรารีนี้คือ นอกจากความสามารถในการเก็บข้อมูลกราฟ RDF ในหน่วยความจำ หรือทำการบันทึกลงเป็นไฟล์ด้วยไวยากรณ์ต่าง ๆ สำหรับอธิบายข้อมูลกราฟ RDF ที่กำหนดได้เหมือนไลบรารีอื่น ๆ ได้แล้ว OpenRDF Sesame ยังมีองค์ประกอบที่ทำหน้าที่เป็นทริปเปิลสโตร์ (Triple Store) ได้อีกด้วย

ทริปเปิลสโตร์ทำหน้าที่เปรียบเสมือนระบบฐานข้อมูลที่ทำงานกับข้อมูล RDF ซึ่งเป็นตัวกลางเปิดให้ผู้ใช้ภายนอกเข้ามาสืบค้นข้อมูลได้ด้วยภาษา SPARQL หรือให้โปรแกรมประยุกต์ที่เก็บข้อมูลในรูปแบบ RDF สามารถเชื่อมต่อเข้ามาในทริปเปิลสโตร์ผ่าน API ของ Sesame เพื่อเพิ่ม ลบ หรือสืบค้นทริปเปิลที่เก็บอยู่ภายในที่เก็บ (Repository) ในทริปเปิลสโตร์ได้ ซึ่งทริปเปิลสโตร์ที่มีมาให้ใน Sesame เรียกว่า OpenRDF Sesame Server และมีเว็บแอปพลิเคชันชื่อ OpenRDF Workbench สำหรับบริหารจัดการข้อมูลกราฟ RDF ในที่เก็บต่าง ๆ ที่ได้ประกาศสร้างเอาไว้ แทนจัดการผ่านคอนโซลที่มีมาให้ในชุดไลบรารีตรง ๆ ซึ่งช่วยอำนวยความสะดวกได้อย่างมาก การนิยามที่เก็บใหม่ขึ้นมาใน OpenRDF Sesame Server จำเป็นจะต้องระบุว่า ที่เก็บมีอัตลักษณ์เป็นอะไรเพื่อใช้ในการเชื่อมต่อมายังที่เก็บข้อมูล และที่เก็บนั้นมีคำอธิบายว่าอย่างไร เพื่ออธิบายภาพรวมของข้อมูลที่เก็บอยู่ในที่เก็บดังกล่าว และสุดท้ายคือที่เก็บดังกล่าวจะใช้กลไกใดในการเก็บบันทึกข้อมูลกราฟ RDF ลงบนเครื่องเซิร์ฟเวอร์ โดยสามารถแบ่งกลไกออกเป็น 3 ประเภทหลัก ๆ ได้ดังนี้

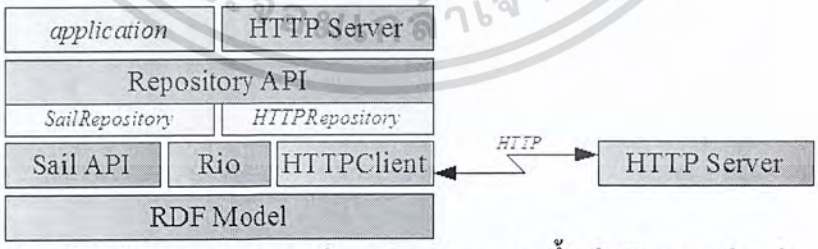
1) เก็บกราฟข้อมูล RDF ทั้งกราฟเอาไว้ในหน่วยความจำเข้าถึงโดยสุ่ม เป็นกลไกประเภทที่ทำงานได้รวดเร็วที่สุด และจะบันทึกข้อมูลกราฟลงในดิสก์อยู่สม่ำเสมอ ข้อจำกัดที่สำคัญ

สำหรับกลไกนี้คือ ต้องมั่นใจว่ากราฟข้อมูล RDF จะไม่ขยายโตเกินไปกว่าขนาดหน่วยความจำที่ OpenRDF Sesame Server สามารถใช้งานได้

2) เก็บข้อมูลกราฟ RDF ลงบนดิสก์ในรูปแบบเฉพาะของ Sesame เอง กลไกนี้จะให้ประสิทธิภาพที่ต่ำกว่าในข้อที่ 1 แต่ได้ขจัดข้อจำกัดที่ห้ามขนาดของกราฟข้อมูล RDF มีขนาดใหญ่กว่าขนาดหน่วยความจำเข้าถึงโดยสุ่ม

3) เก็บข้อมูลกราฟ RDF ลงบนระบบฐานข้อมูลเชิงสัมพันธ์อย่างเช่น MySQL หรือ PostgreSQL ซึ่งให้ประสิทธิภาพช้ากว่าในกลไกประเภทที่ 2 แต่มีข้อดีก็ต่อเมื่อผู้พัฒนาใช้งานระบบฐานข้อมูลเชิงสัมพันธ์ในองค์กรที่มีผู้ควบคุมดูแล คอยสำรองข้อมูลให้อยู่เสมอ ทำให้มีความเสี่ยงที่ข้อมูลจะเสียหายต่ำที่สุดในบรรดากลไกทั้ง 3 ประเภท

นอกจากนี้ Sesame ยังมีส่วนประกอบที่เรียกว่า Sail API (The Storage And Inference Layer API) ที่เข้าใจความหมายของ RDF Schema ทำให้เมื่อมีการเพิ่มทริปเปิลที่มีความสอดคล้องกับ RDF Schema ลงไปในที่เก็บที่ได้มีการนิยาม RDF Schema ดังกล่าวเอาไว้แล้วสามารถทำการสรุปชนิดของอินสแตนซ์ภายในข้อมูลกราฟ RDF ในที่เก็บดังกล่าวได้ ยกตัวอย่างเช่น ทำการนิยามคลาสมนุษย์ แล้วทำการนิยามคลาสย่อยจากคลาสมนุษย์มาเป็นคลาสผู้ชาย และคลาสผู้หญิง เมื่อทำการเพิ่มทริปเปิลที่ทำการประกาศอินสแตนซ์เพิ่มเติม (เช่น นาย ก. เป็นผู้ชาย) ลงในที่เก็บที่สร้างเอาไว้ใน Sesame Server แล้ว ส่วนประกอบที่เรียกว่า Sail API จะทำการเพิ่มอีกทริปเปิลหนึ่งว่า นาย ก. เป็นมนุษย์ให้โดยอัตโนมัติเป็นผลมาจากการสรุปที่ได้จาก RDF Schema ที่นิยามเอาไว้ในที่เก็บนั้น หรือสามารถทำการสรุปชนิดของอินสแตนซ์ผ่านโดเมนหรือเรนจ์ของภาคแสดงในทริปเปิลหนึ่ง ๆ ได้อีกด้วย



รูป 3.129 ส่วนประกอบต่าง ๆ ภายใน Sesame และการขึ้นต่อกันของแต่ละส่วนประกอบ

จากรูปที่ 3.129 พบว่ามีที่เก็บอยู่ 2 ประเภทใน Sesame นั่นคือประเภท Sail และประเภท HTTP โดยประเภทแรกคือที่เก็บข้อมูลกราฟ RDF บนหน่วยความจำและจะหายไปเมื่อโปรแกรมประยุกต์ทำงานเสร็จสิ้น แต่มีความสามารถในการเข้าใจความหมายของ RDF Schema ภายในที่เก็บที่สร้าง

ขึ้นอย่างที่ได้กล่าวไปแล้วในข้างต้น ส่วนประเภทหลังคือที่เก็บประเภทที่ไม่ได้อยู่ในหน่วยความจำ แต่เป็นที่เก็บที่อยู่ระยะไกลโดยติดต่อกันผ่านโปรโตคอล HTTP ไปยัง Sesame Server เพื่อทำการสืบค้น เพิ่ม หรือลบข้อมูลได้ ที่เก็บประเภทนี้จะคงอยู่แม้โปรแกรมประยุกต์ได้ทำงานเสร็จสิ้นไปแล้ว

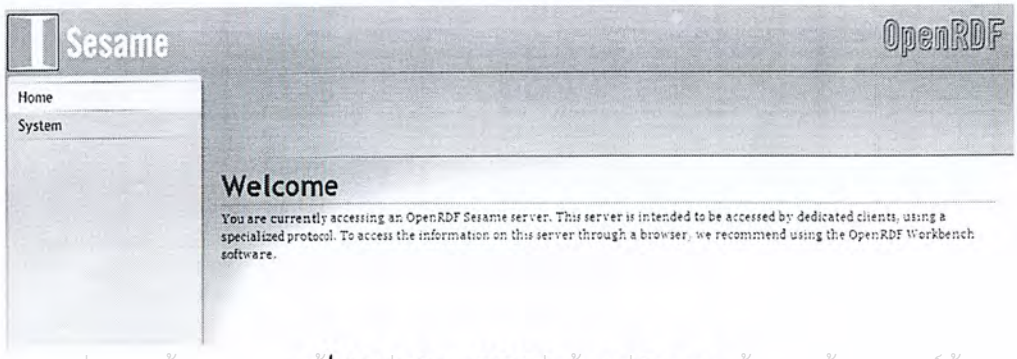
3.8.1 การติดตั้งและใช้งาน OpenRDF Sesame โดยสังเขป

ลักษณะการใช้งาน OpenRDF Sesame ในโครงงานนี้คือจะใช้งานในลักษณะ ของการติดต่อไปยังทริปเปิลสโตร์ซึ่งอยู่ระยะไกล ดังนั้นจึงจำเป็นต้องติดตั้งเซิร์ฟเวอร์ลงในตัวเก็บ เซิร์ฟเล็ท (Servlet Container) ก่อน โดยโครงงานนี้เลือกใช้ Apache Tomcat 6 เป็น โปรแกรมสำหรับรันเซิร์ฟเล็ท หลังจากนั้นให้ทำการดีพลอยไฟล์นามสกุล WAR ทั้งสองไฟล์ที่พบในโฟลเดอร์ /openrdf-sesame-2.3.2/war/ ลงบน Apache Tomcat 6 ได้ผลลัพธ์ดังรูปที่ 3.130

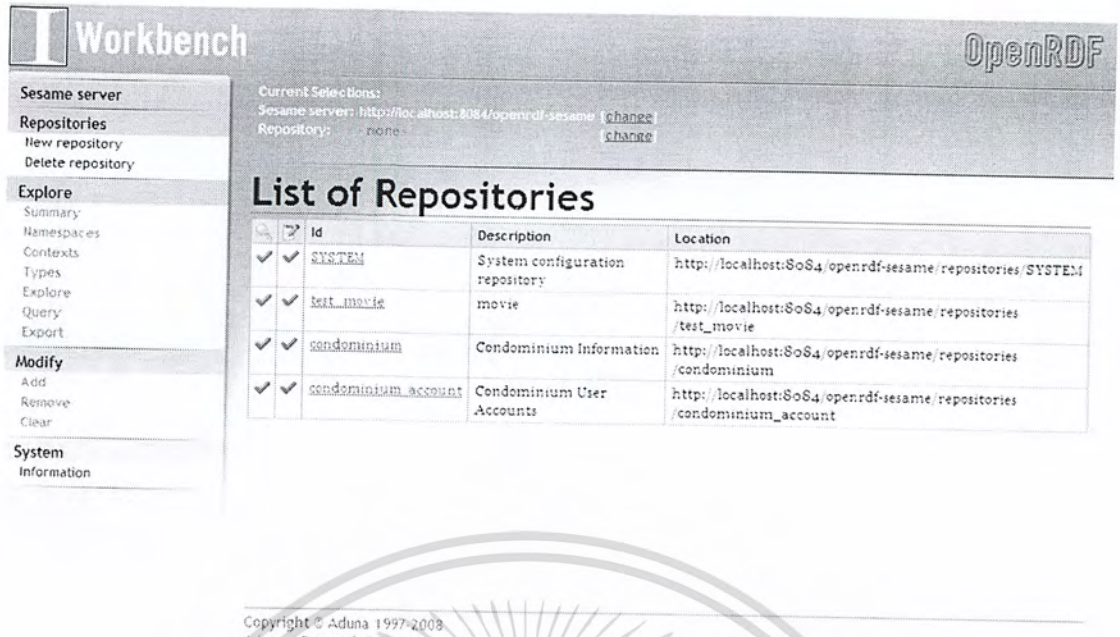
/openrdf-sesame	OpenRDF Sesame	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/openrdf-workbench		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

รูป 3.130 ดีพลอยเว็บแอปพลิเคชันที่แนบมากับ Sesame

เว็บแอปพลิเคชันในโฟลเดอร์ /openrdf-sesame คือ Sesame Server ที่คอยรับการสืบค้น การเพิ่มหรือลบข้อมูลผ่านทางโปรโตคอล HTTP ส่วนอีกโฟลเดอร์หนึ่ง คือ /openrdf-workbench เป็นเว็บแอพพลิเคชันสำหรับใช้บริหารจัดการข้อมูลกราฟ RDF ภายในที่เก็บที่ได้สร้างเอาไว้ใน Sesame Server ดังแสดงไว้ในรูป3.131 และรูป3.132



รูป 3.131 OpenRDF Sesame Server



รูป 3.132 OpenRDF Workbench

จากที่แสดงในรูป3.132 พบว่า Workbench จะบังคับให้ผู้ใช้เลือกที่เก็บข้อมูลที่ต้องการทำงานด้วย เมื่อเลือกที่เก็บจากในตารางแล้วจะสามารถแก้ไข สำรวจ สืบค้นข้อมูลกราฟ RDF ภายในที่เก็บที่เลือกเอาไว้ในตอนแรกได้

3.82 การเชื่อมต่อเข้ากับที่เก็บข้อมูลกราฟ RDF จากโปรแกรมประยุกต์

หลังจากสร้างที่เก็บข้อมูลกราฟ RDF ผ่านเว็บแอปพลิเคชัน OpenRDF Workbench เรียบร้อยแล้ว โปรแกรมประยุกต์สามารถเข้าถึงที่เก็บข้อมูลดังกล่าวเพื่อสืบค้น สร้าง และลบข้อมูลกราฟ RDF ได้ผ่าน API ของ OpenRDF Sesame ดังในโค้ดภาษา java ที่นิยามคลาสชื่อ RemoteGraph สำหรับใช้ติดต่อที่เก็บข้อมูลกราฟ RDF ที่เก็บอยู่ใน OpenRDF Sesame Server เพื่อความสะดวกรวดเร็วในการใช้พัฒนาโปรแกรมประยุกต์ ดังแสดงในรูป 3.133

```
import java.net.URISyntaxException;
import org.openrdf.query.*;
import org.openrdf.model.vocabulary.*;
import org.openrdf.repository.*;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.inferencer.fc.ForwardChainingRDFSInferencer;
import org.openrdf.sail.memory.MemoryStore;
import org.openrdf.rio.*;
import org.openrdf.model.*;
import java.net.URL;
import java.net.URLConnection;
import java.util.*;
import java.io.*;
import org.openrdf.repository.http.HTTPRepository;

public class RemoteGraph {
  private Repository therepository = null;
  // useful -local- constants
  static RDFFormat NTRIPLES = RDFFormat.NTRIPLES;
  static RDFFormat N3 = RDFFormat.N3;
  static RDFFormat RDFXML = RDFFormat.RDFXML;
  static String RDFTYPE =  RDF.TYPE.toString();
  /**
   * Remote Sesame repository with optional inferencing
* @param inferencing
   */
  public RemoteGraph(String serverURL, String repoID){
    try {
      therepository = new HTTPRepository(serverURL, repoID);
      therepository.initialize();
    } catch (RepositoryException e) {
      e.printStackTrace();
    }
  }
  /**
   *  Literal factory
   *
   * @param s the literal value
   * @param typeuri uri representing the type (generally xsd)
   * @return
   */
  public org.openrdf.model.Literal Literal(String s, URI typeuri) {
    try {
      RepositoryConnection con = therepository.getConnection();
      try {
        ValueFactory vf = con.getValueFactory();
        if (typeuri == null) {
          return vf.createLiteral(s);
        } else {
          return vf.createLiteral(s, typeuri);
        }
      } finally {
        con.close();
      }
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }
  /**
   * Untyped Literal factory
   *
   * @param s the literal
   * @return
   */
```

รูป 3.133 นิยามคลาส RemoteGraph สำหรับเชื่อมต่อไปยังที่เก็บข้อมูลกราฟ RDF ใน Open RDF Sesame Server

```
public org.openrdf.model.Literal Literal(String s) {
  return Literal(s, null);
}

/**
 *  URIref factory
 *
 * @param uri
 * @return
 */
public URI URIref(String uri) {
  try {
    RepositoryConnection con = therepository.getConnection();
    try {
      ValueFactory vf = con.getValueFactory();
      return vf.createURI(uri);
    } finally {
      con.close();
    }
  } catch (Exception e) {
    e.printStackTrace();
    return null;
  }
}

/**
 *  BNode factory
 *
 * @return
 */
public BNode bnode() {
  try{
    RepositoryConnection con = therepository.getConnection();
    try {
      ValueFactory vf = con.getValueFactory();
      return vf.createBNode();
    } finally {
      con.close();
    }
  }catch(Exception e){
    e.printStackTrace();
    return null;
  }
}
 /**
 *  Insert Triple/Statement into graph
 *
 * @param s subject uriref
 * @param p predicate uriref
 * @param o value object (URIref or Literal)
 */
public void add(URI s, URI p, Value o) {
  try {
       RepositoryConnection con = therepository.getConnection();
       try {
        ValueFactory myFactory = con.getValueFactory();
        Statement st = myFactory.createStatement((Resource) s, p, (Value) o);
        con.add(st);
       } finally {
        con.close();
       }
    }
    catch (Exception e) {
       // handle exception
    }
}
```

รูป 3.133 นิยามคลาส RemoteGraph สำหรับเชื่อมต่อไปยังที่เก็บข้อมูลกราฟ RDF ใน Open RDF Sesame Server (ต่อ)

```
   /**
    *  Remove Triple/Statement into graph
    *
    * @param s subject uriref
    * @param p predicate uriref
    * @param o value object (URIref or Literal)
    */
  public void remove(URI s, URI p, Value o) {
    try {
         RepositoryConnection con = therepository.getConnection();
         try {
          List tupleToDelete = this.tuplePattern(s, p, o);
          if(tupleToDelete.size() > 0) {
          for(Object _tuple: tupleToDelete) {
             org.openrdf.model.impl.ContextStatementImpl tuple =
(org.openrdf.model.impl.ContextStatementImpl) _tuple;
             ValueFactory myFactory = con.getValueFactory();
             Statement st = myFactory.createStatement(tuple.getSubject(),
tuple.getPredicate(), tuple.getObject());
             con.remove(st);
          }
           }
         } finally {
          con.close();
         }
      }
      catch (Exception e) {
         // handle exception
      }
  }
  /**
   *  Import RDF data from a string
   *
   * @param rdfstring string with RDF data
   * @param format RDF format of the string (used to select parser)
   */
  public void addString(String rdfstring,  RDFFormat format) {
    try {
      RepositoryConnection con = therepository.getConnection();
      try {
        StringReader sr = new StringReader(rdfstring);
        con.add(sr, "", format);
      } finally {
        con.close();
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   *  Import RDF data from a file
   *
   * @param location of file (/path/file) with RDF data
   * @param format RDF format of the string (used to select parser)
   */
  public void addFile(String filepath,  RDFFormat format) {
    try {
      RepositoryConnection con = therepository.getConnection();
      try {
        con.add(new File(filepath), "", format);
      } finally {
        con.close();
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

รูป 3.133 นิยามคลาส RemoteGraph สำหรับเชื่อมต่อไปยังที่เก็บข้อมูลกราฟ RDF ใน Open RDF Sesame Server (ต่อ)

```
/**
   *  Import data from URI source
   *  Request is made with proper HTTP ACCEPT header
   *  and will follow redirects for proper LOD source negotiation
   *
   * @param urlstring absolute URI of the data source
   * @param format RDF format to request/parse from data source
   */
  public void addURI(String urlstring, RDFFormat format) {
    try {
      RepositoryConnection con = therepository.getConnection();
      try {
        URL url = new URL(urlstring);
        URLConnection uricon = (URLConnection) url.openConnection();
        uricon.addRequestProperty("accept", format.getDefaultMIMEType());
        InputStream instream = uricon.getInputStream();
        con.add(instream, urlstring, format);
      } finally {
        con.close();
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
    /**
   *  dump RDF graph
   *
   * @param out output stream for the serialization
   * @param outform the RDF serialization format for the dump
   * @return
   */
  public void dumpRDF(OutputStream out, RDFFormat outform) {
    try {
      RepositoryConnection con = therepository.getConnection();
      try {
        RDFWriter w = Rio.createWriter(outform, out);
        con.export(w);
      } finally {
        con.close();
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   *  Convenience URI import for RDF/XML sources
   *
   * @param urlstring absolute URI of the data source
   */
  public void addURI(String urlstring) {
    addURI(urlstring, RDFFormat.RDFXML);
  }

  /**
   *  Tuple pattern query - find all statements with the pattern, where null
   *  is a wildcard
   *
   * @param s subject (null for wildcard)
   * @param p predicate (null for wildcard)
   * @param o object (null for wildcard)
   * @return serialized graph of results
   */
   public List tuplePattern(URI s, URI p, Value o) {
    try{
RepositoryConnection con = therepository.getConnection();
      try
```

รูป 3.133 นิยามคลาส RemoteGraph สำหรับเชื่อมต่อไปยังที่เก็บข้อมูลกราฟ RDF ใน Open RDF Sesame Server (ต่อ)

```
      RepositoryConnection con = therepository.getConnection();
      try {
        RepositoryResult repres = con.getStatements(s, p, o, true);
        ArrayList reslist = new ArrayList();
        while (repres.hasNext()) {
          reslist.add(repres.next());
        }
        return reslist;
      } finally {
        con.close();
      }
    }catch(Exception e){
      e.printStackTrace();
    }
    return null;
  }
  public String runSPARQL(String qs, RDFFormat format) {
    try{
      RepositoryConnection con = therepository.getConnection();
      try {
        GraphQuery query =
          con.prepareGraphQuery(
          org.openrdf.query.QueryLanguage.SPARQL, qs);
        StringWriter stringout = new StringWriter();
        RDFWriter w = Rio.createWriter(format, stringout);
        query.evaluate(w);
        return stringout.toString();
      } finally {
        con.close();
      }
    }catch(Exception e){
      e.printStackTrace();
    }
    return null;
  }
  /**
   *  Execute a SELECT SPARQL query against the graph
   * @param qs SELECT SPARQL query
   * @return list of solutions, each containing a hashmap of bindings
   */
  public List runSPARQL(String qs) {
    try{
      RepositoryConnection con = therepository.getConnection();
      try {
        TupleQuery query =
          con.prepareTupleQuery(
          org.openrdf.query.QueryLanguage.SPARQL, qs);
        TupleQueryResult qres = query.evaluate();
        ArrayList reslist = new ArrayList();
        while (qres.hasNext()) {
          BindingSet b = qres.next();
          Set names = b.getBindingNames();
          HashMap hm = new HashMap();
          for (Object n : names) {
            hm.put((String) n, b.getValue((String) n));
          }
          reslist.add(hm);
        }
        return reslist;
      } finally {
        con.close();
      }
    }catch(Exception e){
      e.printStackTrace();
    }
    return null;
  }
}
```

รูป 3.133 นิยามคลาส RemoteGraph สำหรับเชื่อมต่อไปยังที่เก็บข้อมูลกราฟ RDF ใน Open RDF Sesame Server (ต่อ)

จากนิยามคลาส RemoteGraph ในรูป3.133 คอนสตรัคเตอร์ของคลาสนี้ จะรับสตริง 2 สตริงคือ URL ของ OpenRDF Sesame Server และอัตลักษณ์ของที่เก็บข้อมูลกราฟ RDF ส่วนเมธอดอื่น ๆ ภายในคลาสนี้ได้นิยามเอาไว้เพื่ออำนวยความสะดวกสำหรับการเข้าใช้งานที่เก็บข้อมูล เช่น เมธอดสร้างโหนดทุกประเภทที่มีการใช้งานใน RDF ได้แก่ โหนดว่าง โหนดที่มี URI กำกับ และโหนดที่มีค่าข้อมูลตามตัวอักษร เมธอดสำหรับการเพิ่มและลบทริปเปิลในที่เก็บที่ได้เชื่อมต่อเอาไว้ เมธอดสำหรับรันคำสั่งสืบค้นด้วยภาษา SPARQL ไม่ว่าจะเป็นการสืบค้นข้อมูล หรือคำสั่งสืบค้นรูปแบบ CONSTRUCT สำหรับสร้างกราฟข้อมูล RDF ขึ้นมาใหม่จากกราฟข้อมูล RDF ในที่เก็บข้อมูล เมธอดสำหรับเพิ่มข้อมูลกราฟ RDF ที่เก็บอยู่ในไฟล์ ในสตริง หรือในอินเตอร์เน็ตลงในที่เก็บข้อมูล RDF ที่สร้างเอาไว้ใน OpenRDF Sesame Server เป็นต้น ข้อจำกัดของคลาสนี้คือ ไม่มีการใช้โครงสร้างข้อมูลที่ประกาศแบบ Generic ทำให้การดึงเอาผลลัพธ์ที่ได้จากการสืบค้นข้อมูลด้วยภาษา SPARQL จำเป็นจะต้องแปลงชนิดข้อมูลหลายขั้นตอนเพื่อเข้าถึงตัวข้อมูลจริงได้

นอกจากความสามารถในการเข้าถึงข้อมูลทริปเปิลภายในที่เก็บที่สร้างเอาไว้ใน OpenRDF Sesame Server แล้วไลบรารี OpenRDF Sesame ยังสามารถเข้าใช้งานแหล่งข้อมูลกราฟ RDF จำลองที่ต้นฉบับข้อมูลจริงถูกเก็บบันทึกเอาไว้ในระบบฐานข้อมูลเชิงสัมพันธ์อย่าง D2R Server ได้อีกด้วย ดังตัวอย่างโค้ดภาษา java ในรูป 3.134

```
import de.fuberlin.wiwiss.d2rq.sesame.D2RQRepository;
import de.fuberlin.wiwiss.d2rq.sesame.D2RQSource;

import org.openrdf.model.Value;
import org.openrdf.sesame.Sesame;
import org.openrdf.sesame.constants.QueryLanguage;
import org.openrdf.sesame.query.QueryResultsTable;
import org.openrdf.sesame.repository.SesameRepository;

...

try{
    // Initialize repository
    D2RQSource source = new D2RQSource("file:///where/you/stored/the/d2rq-mapping.n3", "N3");
    SesameRepository repos = new D2RQRepository("urn:youRepository", source,
Sesame.getService());

    // Query the repository
    String query = "SELECT ?x, ?y WHERE (<http://www.conference.org/conf02004/paper#Paper1>,
?x, ?y)";
    QueryResultsTable result = repos.performTableQuery(QueryLanguage.RDQL, query);

    // print the result
    int rows = result.getRowCount();
    int cols = result.getColumnCount();
```

รูป 3.134 โค้ดตัวอย่างเชื่อมต่อไปยังกราฟข้อมูล RDF จำลองที่สร้างไว้บน D2R Server

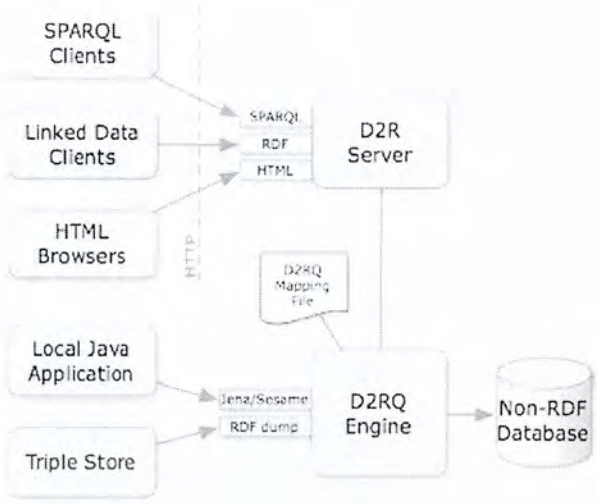
```
for(int i = 0; i < rows; i++){
        for(int j = 0; j < cols; j++){
            Value v = result.getValue(i,j);
            System.out.print(v.toString() + "    ");
        }
        System.out.println();
    }
} catch(Exception e){
    // catches D2RQException from D2RQSource construcor
    // catches java.io.IOException,
    //         org.openrdf.sesame.query.MalformedQueryException,
    //         org.openrdf.sesame.query.QueryEvaluationException,
    //         org.openrdf.sesame.config.AccessDeniedException
    //         from performTableQuery
    e.printStackTrace();
}
```

รูป 3.134 โค้ดตัวอย่างเชื่อมต่อไปยังกราฟข้อมูล RDF จำลองที่สร้างไว้บน D2R Server (ต่อ)

จากรูป 3.134 เป็นการแสดงโค้ดตัวอย่างสำหรับเข้าถึงกราฟข้อมูล RDF จำลองที่สร้างไว้บน D2R Server และทำการสืบค้นข้อมูลในที่เก็บข้อมูลดังกล่าวด้วยภาษาสืบค้น RDQL โดยยังจำเป็นต้องใช้ไฟล์ที่ระบุการแปลง ข้อมูลที่มีโครงสร้างแบบฐานข้อมูลเชิงสัมพันธ์ไปเป็นข้อมูลที่มีโครงสร้างแบบ RDF ซึ่งไม่ปลอดภัยนัก เพราะในไฟล์ที่ระบุการแปลงดังกล่าวนั้นเก็บบันทึกรหัสผ่านสำหรับเข้าถึงระบบฐานข้อมูลเชิงสัมพันธ์เอาไว้ด้วย นอกจากวิธีการเชื่อมต่อที่แสดงในรูป 3.134 แล้ว อีกวิธีหนึ่งคือส่งคำสั่งสืบค้นรูปแบบต่าง ๆ ผ่าน URL ไปยังเซิร์ฟเวอร์แล้วรับเอาผลลัพธ์นำมาประมวลผล ซึ่งไม่จำเป็นต้องมีไฟล์ที่ระบุการแปลงฐานข้อมูลดังกล่าวเลย

3.9 D2R Server

D2R Server เป็นเครื่องมือสำหรับตีแผ่เนื้อหาข้อมูลภายในฐานข้อมูลเชิงสัมพันธ์ลงบนซีแมนติคเว็บ ซึ่งรองรับระบบฐานข้อมูลเชิงสัมพันธ์จากหลายผู้ให้บริการ ได้แก่ MySQL, Oracle, PostgreSQL, และ Microsoft SQL Server ข้อมูลภายในฐานข้อมูลจะถูกแปลงให้อยู่ในรูปแบบ RDF ผ่านภาษาเชิงประกาศสำหรับการแปลงโดยเฉพาะที่เรียกว่า ภาษา D2RQ ซึ่งหน้าที่หลักคือ ระบุวิธีการระบุทรัพยากร และระบุวิธีการสร้างค่าคุณสมบัติ (Property values) จากข้อมูลภายในฐานข้อมูล และจากข้อมูลที่ระบุการแปลงฐานข้อมูลนี้เอง D2R Server สามารถให้บริการข้อมูล RDF แก่เว็บเอเจนท์, นำเสนอข้อมูลในรูปแบบ XHTML สำหรับผู้ใช้งานทั่วไปให้ดูง่าย, และรับคำสั่งสืบค้นข้อมูลภาษา SPARQL เข้ามาได้โดย D2R Engine จะทำหน้าที่แปลงคำสั่งสืบค้นข้อมูลภาษา SPARQL ให้เป็นคำสั่งสืบค้นภาษา SQL แล้วนำไปประมวลผลบนระบบฐานข้อมูลเชิงสัมพันธ์ที่ฐานข้อมูลดังกล่าวเชื่อมต่ออยู่



รูป 3.135 การให้บริการของ D2R Server แก่โปรแกรมประยุกต์ชนิดต่าง ๆ

3.9.1 การแปลงข้อมูลเชิงสัมพันธ์ให้อยู่ในรูปแบบ RDF

D2R Server มีภาษาสำหรับการแปลงที่เรียกว่า D2RQ ซึ่งเขียนอธิบายในรูปของ RDF ได้ ใช้สำหรับเก็บบันทึกการแปลงระหว่างเค้าร่างฐานข้อมูลที่มีลักษณะแตกต่างกันออกไป ตามระบบฐานข้อมูลเชิงสัมพันธ์ที่ใช้ และ RDF Schema ข้อมูลการแปลงที่อธิบายด้วยภาษา D2RQ จะระบุวิธีการระบุทรัพยากร และระบุวิธีการสร้างค่าคุณสมบัติ (Property values) จากข้อมูลภายในฐานข้อมูล อ็อบเจ็คต์ศูนย์กลางในที่ใช้ระบุการแปลงด้วยภาษา D2RQ คือ อ็อบเจ็คต์ที่เรียกว่า ClassMap ซึ่งนำเสนอการแปลงจากกลุ่มของเอนทิตีที่ถูกอธิบายเอาไว้ภายในฐานข้อมูลไปเป็นคลาสคลาสหนึ่ง หรือกลุ่มของคลาสที่คล้ายกันในหมู่ทรัพยากร แต่ละ ClassMap จะมีชุด PropertyBridges ซึ่งระบุถึงวิธีการสร้างการอธิบายทรัพยากรที่ถูกระบุโดย ClassMap หรือเป็นการระบุภาคแสดงจากแอตทริบิวต์ในเอนทิตีที่ถูก ClassMap แปลงนั่นเอง ส่วนค่าของคุณสมบัติสามารถสร้างขึ้นมาได้โดยตรงจากค่าข้อมูลภายในฐานข้อมูล หรืออาจจะผ่านการใช้แม่แบบ โดยทั้งหมดที่กล่าวมานี้สามารถใช้เครื่องมือที่ติดมากับ D2R Server ช่วยในการสร้างไฟล์สำหรับแปลงฐานข้อมูลได้อัตโนมัติ ซึ่งผลลัพธ์ที่ได้จะเป็นกลุ่มคำศัพท์ RDF สำหรับแต่ละฐานข้อมูลที่ป้อนเข้าเครื่องมือดังกล่าว ไฟล์สำหรับการแปลงที่ได้จากเครื่องมือข้างต้นสามารถนำมาปรับแต่งให้ดีขึ้นได้ในภายหลังโดยนักพัฒนาเอง โดยขั้นตอนการแปลงที่เครื่องมือดังกล่าวใช้มีความใกล้เคียงกับอัลกอริธึมที่เสนอวิธีการแปลงรูปแบบจำลองเชิงสัมพันธ์ไปเป็นแบบจำลอง RDF ดังที่ได้เสนอเอาไว้ในบททฤษฎี โดยมีข้อจำกัดที่ไม่สามารถนิยามลำดับชั้นของคลาส หรือคุณสมบัติได้เนื่องจากภาษา D2RQ ไม่รองรับ

3.9.2 การจอง URI

ในแต่ละ ClassMap แต่ละแถวข้อมูลของเอนทิตีในฐานข้อมูลจะถูกกำหนด URI ให้โดยมีแม่แบบ URI ที่ตายตัว ยกตัวอย่างเช่น แม่แบบ "products/product@@Products.ID@@" จะให้ URI ผลลัพธ์อย่างเช่น products/product1134 โดยมาจากการใส่ค่าจาก Products.ID ที่เป็นแอตทริบิวต์ภายในฐานข้อมูลลงไปในแม่แบบ URI ข้างต้น

D2R Server จะเปลี่ยน URI สัมพัทธ์ให้เป็น URI สัมบูรณ์ด้วยการขยาย URI สัมพัทธ์โดยการเติม URI ฐานของเซิร์ฟเวอร์เข้าไปไว้ข้างหน้า ซึ่งเป็นกลไกการจอง URI ที่แนะนำเพราะเป็นการรับรองว่าอัตลักษณ์ที่ได้อยู่ในขอบเขต URI ของผู้ดูแลเซิร์ฟเวอร์ และยังทำให้ D2R Server สามารถตอบคำร้องขอ HTTP ที่เข้ามาสอบถามเกี่ยวกับ URI ต่าง ๆ เหล่านี้ได้ เพื่ออธิบายข้อมูลเพิ่มเติมของทรัพยากรที่มีการร้องขอเข้ามา แต่ถ้าหากฐานข้อมูลมีการเก็บ URI สำหรับใช้ในการระบุข้อมูลอื่น ๆ เอาไว้เองอยู่แล้ว อย่างเช่นในตารางที่อธิบายเอกสารเว็บเป็นต้น ก็สามารถใช้ URI ภายนอกแทน URI ที่สร้างขึ้นมาจากแม่แบบได้เช่นกัน

3.9.3 ประสิทธิภาพ

จากการทดสอบโดยเทียบเวลาที่ใช้ในการสืบค้นข้อมูลตามแม่แบบกราฟแต่ละแบบโดยเทียบกับ Jena2 DB ซึ่งเป็นไลบรารีภาษา java สำหรับทำงานกับข้อมูลกราฟ RDF เช่นกัน แต่ไม่สามารถทำหน้าที่เป็นทริปเปิลสโตร์ได้ ผลลัพธ์จากการทดสอบแสดงไว้ในตาราง 3.8

ตาราง 3.8 เปรียบเทียบประสิทธิภาพของ D2RQ Engine เทียบกับ Jena2DB

	Jena2 DB	D2RQ Engine
สืบค้น (s ? ?)	1.83ms	0.01ms
สืบค้น (? p o)	1.94ms	0.97ms
สืบค้น (? p ?)	42.431ms	72ms
สืบค้น (? ? o)	1.72ms	3.23ms

3.9.4 ขั้นตอนการนำข้อมูลภายในฐานข้อมูลเชิงสัมพันธ์เผยแพร่ในรูปแบบ RDF โดยสังเขป

ภายใน D2R Server จะมีเครื่องมือสำหรับสร้างไฟล์ที่เก็บคำสั่งภาษา D2RQ ขึ้นมาให้โดยอัตโนมัติเรียกว่า generate-mapping โดยสามารถนำไฟล์คำสั่งภาษา D2RQ ดังกล่าวไปปรับแต่งภายหลังได้ พารามิเตอร์ที่เครื่องมือ generate-mapping รับแสดงไว้ในรูป 3.136

```
generate-mapping [-u username] [-p password] [-d driverclass]
[-o outfile.n3] [-b base uri] jdbcURL
```

รูป 3.136 พารามิเตอร์ที่ generate-mapping รับ

รูป 3.136 พบว่าผู้ใช้จะต้องระบุชื่อผู้ใช้ รหัสผ่าน คลาสสำหรับต่อระบบฐานข้อมูล ชื่อไฟล์ที่เก็บการแปลงที่จะให้เซฟลงดิสก์ URI ฐาน และตำแหน่งเค้าร่างฐานข้อมูลในรูปแบบ JDBC ซึ่งรายละเอียดการเชื่อมต่อกับระบบฐานข้อมูลเชิงสัมพันธ์นี้จะไปปรากฏอยู่ในส่วนหัวของไฟล์ที่เก็บคำสั่งการแปลงด้วยภาษา D2RQ

คลาสสำหรับต่อระบบฐานข้อมูลจากผู้ให้บริการต่าง ๆ มีดังนี้

1) MySQL ใช้ com.mysql.jdbc.Driver

2) PostgreSQL ใช้ org.postgresql.Driver

3) Oracle ใช้ oracle.jdbc.OracleDriver

4) Microsoft SQL Server ใช้ com.microsoft.sqlserver.jdbc.SQLServerDriver

พารามิเตอร์ URI ฐานใช้สำหรับการสร้างเนมสเปซให้แก่ชุดคำสั่งสำหรับฐานข้อมูลที่ทำการแปลง หากไม่ระบุเอาไว้ เครื่องมือจะกำหนดให้เป็น URL ไปยัง D2R Server

หลังจากที่ได้ไฟล์ที่แสดงการแปลงฐานข้อมูลแล้ว สามารถนำไฟล์ดังกล่าวมาใช้กับเครื่องมือที่เรียกว่า dump-rdf ได้ ซึ่งมีไว้สำหรับแปลงข้อมูลภายในฐานข้อมูล ณ ขณะเวลานั้นเก็บลงในไฟล์ข้อมูลกราฟ RDF ไฟล์เดียว โดยเครื่องมือดังกล่าวรับพารามิเตอร์ดังรูป 3.137

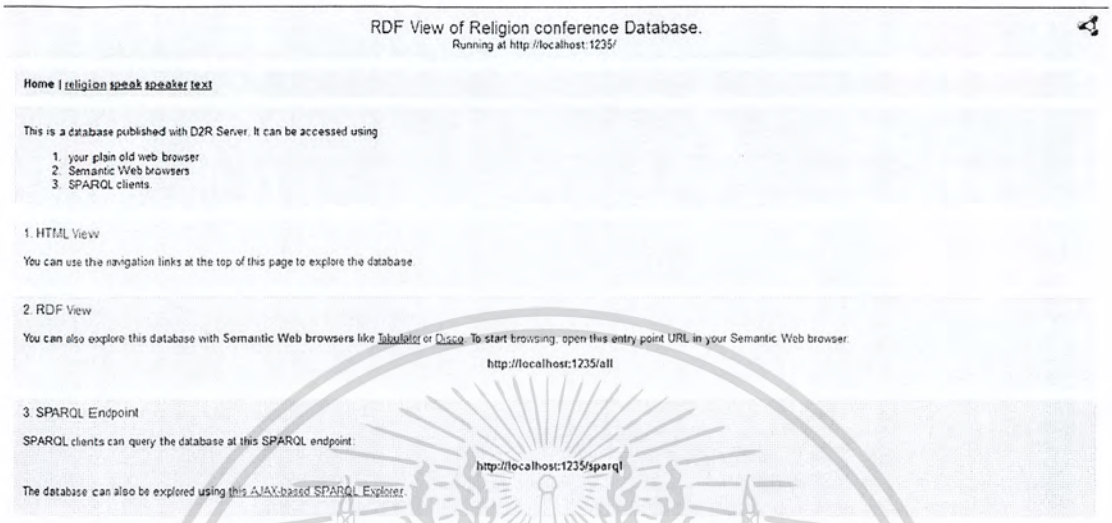
```
dump-rdf -u username [-p password] -d driverclass -j jdbcURL
[-f fetchSize] [output parameters]
```

รูป 3.137 พารามิเตอร์ที่ dump-rdf รับ

พารามิเตอร์ fetchSize คือจำนวนแถวของข้อมูลในฐานข้อมูลที่จะให้เครื่องมือรับเข้ามาในแต่ละการร้องขอข้อมูล ซึ่งมีความสำคัญต่อการควบคุมการใช้หน่วยความจำสำหรับทั้ง D2RQ และเซิร์ฟเวอร์ระบบฐานข้อมูลเอง ค่าตั้งต้นของพารามิเตอร์นี้คือ 500 หรือ Integer.MIN_VALUE หากทำงานกับระบบฐานข้อมูล MySQL เพื่อที่จะใช้งานในโหมดสตรีมมิง

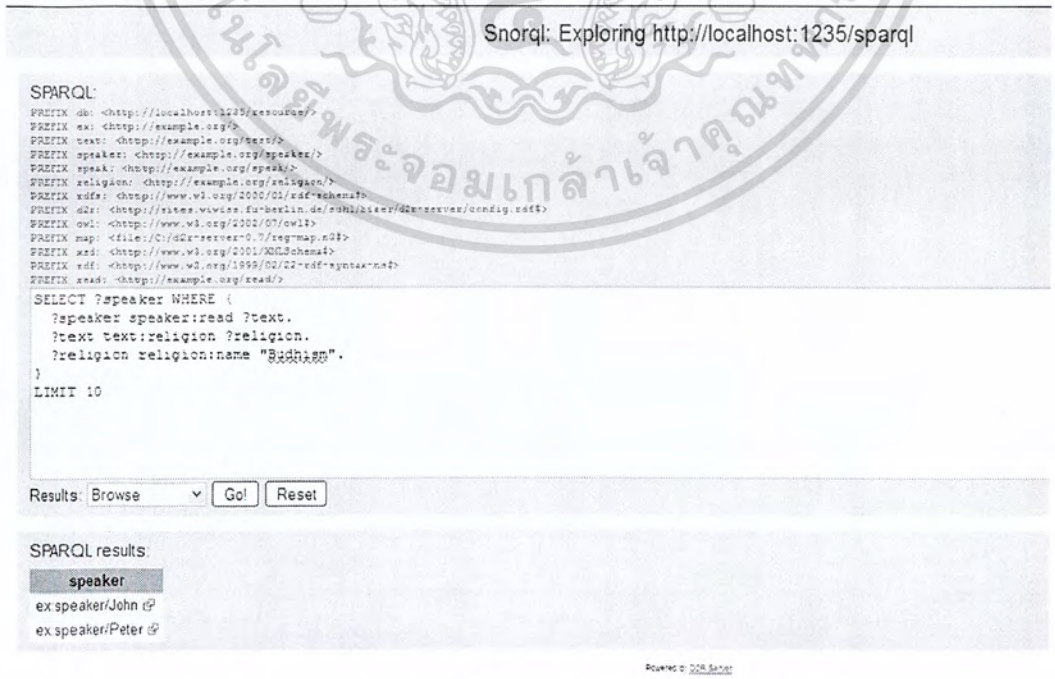
หรืออีกวิธีหนึ่งคือส่งชื่อไฟล์นี้เข้าเครื่องมือชื่อ d2r-server เพื่อทำการตั้งเว็บเซิร์ฟเวอร์สำหรับรับใช้เผยแพร่ข้อมูลที่ต้องการเผยแพร่ในรูปแบบ RDF เมื่อเครื่องมือทำการตั้งเซิร์ฟเวอร์

เรียบร้อยแล้วจะแจ้งพอร์ตที่ใช้สำหรับการเข้าถึง ซึ่งโดยปกติจะถูกระบุเอาไว้ในส่วนหัวของไฟล์สำหรับแปลงอยู่แล้ว และเมื่อเข้าไปตาม URL ที่เครื่องมือแจ้งมาจะได้ดังรูป 3.138



รูป 3.138 เว็บไซต์ที่ d2r-server สร้างขึ้นมาให้โดยอัตโนมัติ

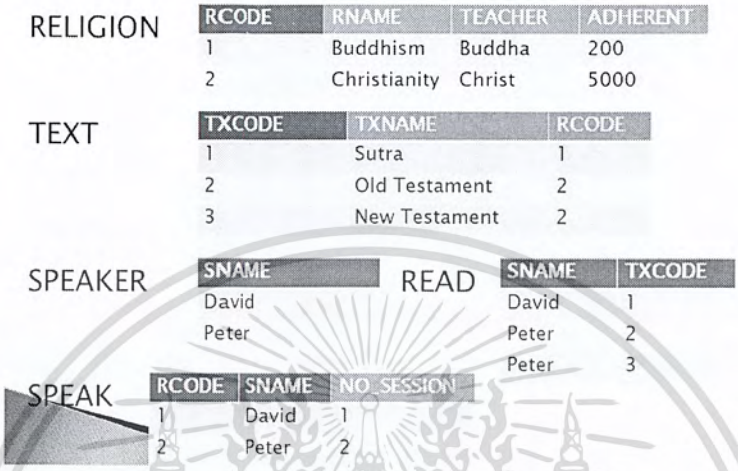
ภายในเว็บไซต์ที่ d2r-server สร้างขึ้นมาให้ จะมีหน้าเว็บไซต์ที่รับคำสั่งสืบค้นภาษา SPARQL และแสดงผลลัพธ์ออกที่ด้านล่างของหน้าเว็บดังแสดงในรูป 3.139



รูป 3.139 หน้าเว็บไซต์ที่เปิดรับคำสั่งสืบค้นภาษา SPARQL

3.9.4 ตัวอย่างไฟล์เก็บคำสั่งแปลงฐานข้อมูลเชิงสัมพันธ์ไปเป็นข้อมูลกราฟ RDF

ในหัวข้อนี้จะใช้ตัวอย่างฐานข้อมูลการประชุมวิชาการทางศาสนาโดยมีเค้าร่างฐานข้อมูลเชิงสัมพันธ์ และตัวอย่างข้อมูลดังแสดงในรูป 3.140



รูป 3.140 เค้าร่างฐานข้อมูลเชิงสัมพันธ์ของการประชุมวิชาการศาสนา

แอตทริบิวท์ในความสัมพันธ์ที่มีสีเข้มคือคีย์หลักประจำแต่ละความสัมพันธ์ เมื่อป้อนเค้าร่างฐานข้อมูลเชิงสัมพันธ์ข้างต้นในเครื่องมือชื่อ generate-mapping แล้วจะได้ไฟล์ที่เก็บการแปลงในรูปแบบภาษา D2RQ ดังแสดงในรูป 3.141 และรูป 3.142 ซึ่งเป็นเพียงบางส่วนของคำสั่งการแปลงที่นำมาแสดง และถูกปรับแต่งแล้ว

```
map:database a d2rq:Database;
        d2rq:jdbcDriver "com.mysql.jdbc.Driver";
        d2rq:jdbcDSN "jdbc:mysql://localhost/rdftest2";
        d2rq:username "root";
        d2rq:password "root";
        jdbc:autoReconnect "true";
        jdbc:zeroDateTimeBehavior "convertToNull";
        .

# Server configuration
<> a d2r:Server;
    rdfs:label "RDF View of Religion conference Database.";
    d2r:port 1235;
    d2r:documentMetadata [
        rdfs:comment "This is comment...";
    ]
```

รูป 3.141 อธิบายการเชื่อมต่อเข้าสู่ระบบฐานข้อมูลเชิงสัมพันธ์ และค่าตั้งต้นของ D2R Server

โค้ดในรูป 3.141 ซึ่งเป็นคำสั่งภาษา D2RQ ซึ่งใช้ไวยากรณ์ของภาษา RDF มีหน้าที่กำหนดการเชื่อมต่อเข้าสู่ระบบฐานข้อมูลเชิงสัมพันธ์ และระบุเค้าร่างข้อมูลที่จะนำมาสร้างกราฟข้อมูล RDF จำลอง โดยส่วนแรกจะระบุคลาสภาษา java สำหรับติดต่อกับระบบฐานข้อมูลเชิงสัมพันธ์ รหัสผ่านต่าง ๆ และตำแหน่งที่อยู่ฐานข้อมูลรูปแบบ JDBC ส่วนที่สองเป็นการกำหนดข้อความที่จะให้แสดงในหน้าเว็บไซต์ที่ได้มาจากการเริ่มต้นการทำงาน D2R Server

และในรูป3.142 เป็นส่วนหนึ่งของคำสั่งภาษา D2RQ ซึ่งทำงานกับความสัมพันธ์ religion ในเค้าร่างฐานข้อมูลเชิงสัมพันธ์ที่แสดงไว้ในรูป 3.141 โดยเริ่มแรกให้ประกาศอ็อบเจ็คต์ชนิด ClassMap พร้อมระบุแม่แบบ URI สำหรับใช้กำหนดให้แต่ละอินสแตนซ์ภายในเอนทิตีที่กำลังพิจารณา และระบุว่าอินสแตนซ์ที่ถูกสร้างผ่านอ็อบเจ็คต์ ClassMap นี้จะเป็นอินสแตนซ์ของคลาสใดในข้อมูลกราฟ RDF ที่กำลังทำการแปลง สุดท้ายระบุข้อความกำกับสำหรับคลาส RDFS ใหม่ที่ได้จากอ็อบเจ็คต์ ClassMap สำหรับให้นักพัฒนาอ่านเข้าใจได้ด้วยภาษาธรรมชาติ

```
# Table religion
map:religion a d2rq:ClassMap;
        d2rq:dataStorage map:database;
        d2rq:uriPattern "http://example.org/religion/@@religion.RCODE@@";
        d2rq:class religion:Religion;
        d2rq:classDefinitionLabel "Details about religions";
        .
map:religion_RNAME a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:religion;
        d2rq:property religion:name;
        d2rq:propertyDefinitionLabel "religion RNAME";
        d2rq:column "religion.RNAME";
        .
map:religion_TEACHER a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:religion;
        d2rq:property religion:teacher_name;
        d2rq:propertyDefinitionLabel "religion TEACHER";
        d2rq:column "religion.TEACHER";
        .
map:religion_ADHERENT a d2rq:PropertyBridge;
        d2rq:belongsToClassMap map:religion;
        d2rq:property religion:adherent;
        d2rq:propertyDefinitionLabel "religion ADHERENT";
        d2rq:column "religion.ADHERENT";
        d2rq:datatype xsd:int;
        .
map:religion_SPOKEN_BY a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:religion;
    d2rq:property religion:spoken_by;
    d2rq:refersToClassMap map:speak;
    d2rq:join "religion.RCODE <= speak.RCODE"
```

รูป 3.142 ตัวอย่างคำสั่งภาษา D2RQ ที่ทำงานกับเอนทิตี religion ในเค้าร่างฐานข้อมูลเชิงสัมพันธ์

ส่วนอ็อบเจ็คต์อื่น ๆ ในรูป 3.142 ที่มีชนิดเป็น PropertyBridge นั้นจะทำงานกับแอตทริบิวต์ต่าง ๆ ในเอนทิตีที่ถูก ClassMap อธิบายเอาไว้ การใช้โดยสังเขปคือ กำหนดเพรดิเคตให้ หากแอตทริบิวต์ดังกล่าวเป็นคีย์นอกซึ่งอ้างอิงไปยังคีย์หลักของความสัมพันธ์อื่น ๆ ภายในเค้าร่างฐานข้อมูล ให้ระบุด้วยว่ามีการอ้างอิงไปยังเอนทิตีที่ถูกอธิบายด้วยอ็อบเจ็คต์ชนิด ClassMap ใด และระบุเงื่อนไขในการระบุอินสแตนซ์ที่ตรงกันผ่านภาคแสดง d2rq:join

นอกจากนี้ยังมีคำสั่งอื่น ๆ ที่ทำงานกับอ็อบเจ็คต์ชนิด PropertyBridge ที่ไม่ได้อธิบายเอาไว้ในหัวข้อนี้อีกพอสมควรผู้ที่สนใจสามารถศึกษาเพิ่มเติมได้ในเว็บไซต์

http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/

3.10 การแปลงข้อมูลกราฟ RDF จาก Schema

โดยทั่วไปใช้คำสั่งสืบค้นรูปแบบ Construct ดังตัวอย่างรูป 3.143

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a     foaf:givenname      "Alice" .
_:a     foaf:family_name    "Hacker" .

_:b     foaf:firstname      "Bob" .
_:b     foaf:surname        "Hacker" .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT {  ?x  vcard:N _:v .
             _:v  vcard:givenName ?gname .
             _:v  vcard:familyName ?fname }

WHERE
  {
      { ?x  foaf:firstname  ?gname } UNION { ?x  foaf:givenname     ?gname } .
      { ?x  foaf:surname    ?fname } UNION { ?x  foaf: family_name  ?fname } .

  }
```

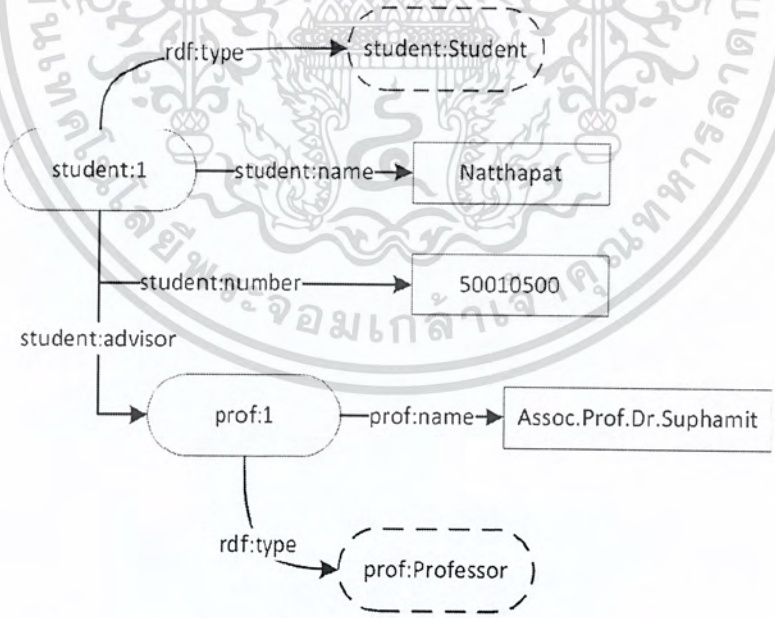
รูป 3.143 ตัวอย่างคำสั่งสืบค้นด้วย Construct

ข้อสังเกตเกี่ยวกับคำสั่ง Construct คือ ไม่สามารถกำหนด URI สำหรับ Resource URI ใหม่ให้แก่กราฟผลลัพธ์ได้ และสามารถสร้าง Blank node ได้ ตัวอย่างรูปที่ 3.144

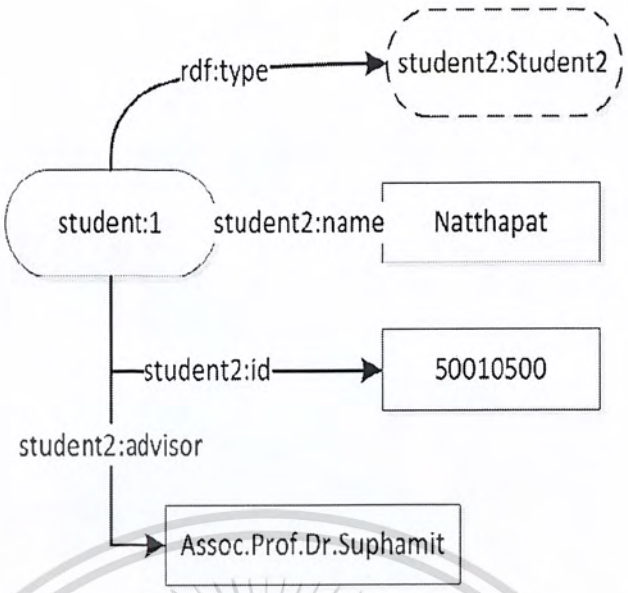
```
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .

_:v1    vcard:N             _:x .
_:x     vcard:givenName     "Alice" .
_:x     vcard:familyName    "Hacker" .

_:v2    vcard:N             _:z .
_:x     vcard:givenName     "Bob" .
_:x     vcard:familyName    "Hacker" .
```

รูป 3.144 ตัวอย่างการสร้าง Blank node

ด้วยเหตุนี้จึงไม่แปลง Resource URI ที่ได้จากกราฟต้นทาง และไม่สามรถแปลงกราฟที่มี RDF Container ได้อย่างสมบูรณ์แบบ



รูป 3.145 กราฟข้อมูลนักศึกษาและอาจารย์ ต้นแบบที่จะถูกแปลง



รูป 3.146 กราฟข้อมูลนักศึกษาและอาจารย์ ผลลัพธ์ที่ถูกแปลงแล้ว

ขั้นตอนการสร้างกราฟที่ใช้ในการแปลง

1) ระบุ Attibute ภายใน Class ปลายทางที่ต้องการแปลงไปให้หมดและระบุ Predicate URI ที่ต้องการเพื่อนำค่าจากกราฟต้นทางมาใส่ ใช้ Container Sequence ในการเก็บ URI ใน คลาสปลายทางจะระบุได้แค่ระดับเดียวกันเพราะยอมให้เก็บได้แค่ Literal value หรือไม่ก็ Resource node
2) ระบุ Attribute ภายใน Class ต้นทางที่สอดคล้องกับ Attribute ที่ระบุเอาไว้ในฝั่งกราฟที่อธิบาย Class ปลายทาง และระบุลำดับ Predicate URI ที่ใช้ในการเข้าถึง Object ที่ต้องการ
3) เชื่อมโหนดของ Class ต้นทางไปยัง Class ปลายทางด้วย willBeConvertedTo เชื่อมโหนดของ Attribute ใน Class ต้นทางไปยังโหนดของ Attribute ใน Class ปลายทางที่สอดคล้องผ่าน mapsTo

ข้อจำกัดของการแปลง

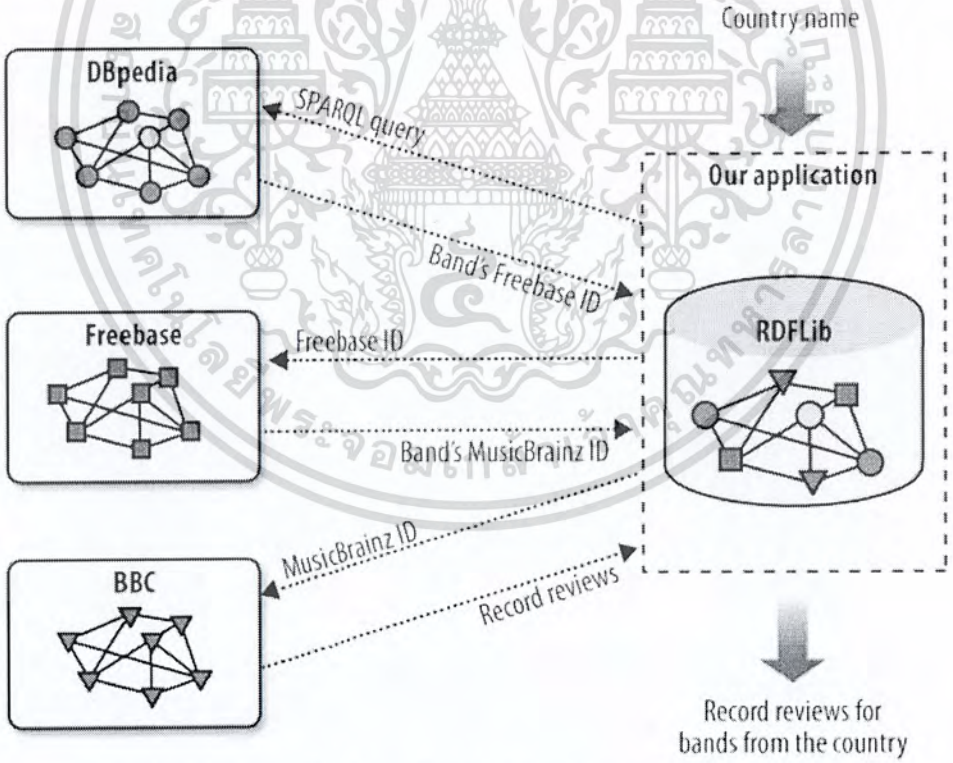
1) ทำงานกับกราฟที่มี Blank node ไม่ได้
2) RDF Graph ที่ได้จากการ Map ด้วย D2RQ จะไม่มี Blank node อยู่แล้ว
3) ไม่สามารถ Map ข้อมูลที่เก็บด้วย Container ได้ เนื่องจากข้อจำกัดของคำสั่ง CONSTRUCT และข้อจำกัดของกราฟสำหรับแปลง Class ด้วยทั้งคู่

บทที่ 4

การทดลอง

4.1 การทดลองที่ 1 สร้างโปรแกรมประยุกต์ที่ใช้งานข้อมูล RDF จากแหล่งข้อมูลต่าง ๆ

การทดลองสร้างโปรแกรมประยุกต์ด้วยภาษาไพธอนที่สาธิตให้เห็นถึงการเชื่อมโยงกันของข้อมูล โดยโปรแกรมประยุกต์นี้จะรับชื่อประเทศไปทำงาน เพื่อใช้สืบค้นรายชื่อศิลปินของประเทศนั้น รวมถึงรายการอัลบั้ม และบทวิจารณ์ต่าง ๆ ซึ่งจะต้องไปดึงเอาข้อมูลจาก 3 แหล่งข้อมูลนำมาผนวกเข้าด้วยกัน โดยบทวิจารณ์จะเก็บอยู่ในเว็บไซต์ของ British Broadcasting Company (BBC) แต่ว่า URIref ที่ใช้อธิบายศิลปินแต่ละกลุ่มนั้นเป็น URIref เดียวกับที่เว็บไซต์ MusicBrainz ใช้ ซึ่ง URIref เหล่านี้ไม่สามารถนำไปใช้หาคำอธิบายของทรัพยากรต่อได้ จึงนำ URIref เหล่านี้ไปสืบค้นหา URIref ที่อธิบายทรัพยากรเดียวกันจากในเว็บไซต์ Freebase อีกต่อหนึ่ง ดังเช่นแผนภาพแสดงลำดับการสืบค้นข้อมูลในโปรแกรมประยุกต์ที่สร้างในการทดลองนี้ในรูปที่ 4.1



รูป 4.1 แผนภาพลำดับการสืบค้นข้อมูลจากแหล่งข้อมูล 3 แหล่ง

4.1.1 จุดประสงค์การทดลอง

1) เพื่อแสดงตัวอย่างการผสานข้อมูลเข้าด้วยกันจากเว็บไซต์หลายแห่ง รวมถึงแสดงให้เห็นถึงบทบาทสำคัญของ URIref ในกราฟข้อมูล RDF ที่ใช้อธิบายทรัพยากรหนึ่ง ๆ

2) เพื่อแสดงการนำไลบรารี RDFlib มาใช้ในการสร้างโปรแกรมประยุกต์ ซึ่งในโปรแกรมอาศัยผนวกข้อมูลกราฟที่ได้มาใหม่เข้ากับกราฟเดิมที่อยู่ในหน่วยความจำ

4.1.2 ขั้นตอนการทดลอง

1) ทำการประกาศเนมสเปสที่จะใช้ ในตลอดทั้งโปรแกรม เพื่อสืบค้นข้อมูลจากแหล่งข้อมูลต่างๆ จากนั้นนำมาเก็บรวบรวมลงตัวแปรชนิดข้อมูล dict สำหรับใช้ส่งเข้าในเมธ็อด query และนิยามฟังก์ชันที่ใช้ติดต่อกับแหล่งข้อมูลภายนอกผ่านโปรโตคอลผ่านโปรโตคอล (Protocol) HTTP ดังรูปที่ 4.2

```
import urllib2
from urllib import quote
from StringIO import StringIO
from rdflib import Namespace, Graph, URIRef, ConjunctiveGraph
countryname = "England"
dbpedia_sparql_endpoint_URL = "http://dbpedia.org/sparql?default-
graph-uri=http%3A//dbpedia.org&query="

#namespaces we will use
owl = Namespace("http://www.w3.org/2002/07/owl#")
fb = Namespace("http://rdf.freebase.com/ns/")
foaf = Namespace("http://xmlns.com/foaf/0.1/")
rev = Namespace("http://purl.org/stuff/rev#")
dc = Namespace("http://purl.org/dc/elements/1.1/")
rdfs = Namespace("http://www.w3.org/2000/01/rdf-schema#")

nsdict = {'owl':owl, 'fb':fb, 'foaf':foaf, 'rev':rev, 'dc':dc,
'rdfs':rdfs}
#utitlies to fetch URLs
def _geturl (url):
    try:
        reqObj = urllib2.Request(url)
        urlObj = urllib2.urlopen(reqObj)
        response = urlObj.read()
        urlObj.close()
    except:
        print("NO DATA")
        response = ""
    return response
def sparql(url, query):
    return _geturl(url + quote(query))
def fetchRDF(url, g):
    try:
        g = g.parse(url)
    except:
        print("Error fetching graph from {0}".format(url)
```

รูป 4.2 โค้ดประกาศเนมสเปส และฟังก์ชันที่ทำงานกับโปรโตคอล HTTP

2) นิยามฟังก์ชันสำหรับทำงานกับแต่ละแหล่งข้อมูล โดยจะรับตัวแปรชนิดข้อมูลกราฟที่ใช้ภายในโปรแกรมประยุกต์แล้วพิจารณาดูว่ามีข้อมูลใหม่นอกเหนือจากที่มีอยู่แล้วในโปรแกรมหรือไม่ แล้วนำมาผสานรวมเข้าไปในกราฟ โดยอันดับแรกนิยามฟังก์ชันสำหรับสืบค้นข้อมูลจาก DBPedia เพื่อสร้างกราฟที่เชื่อมโยงระหว่าง URIref ของศิลปินที่ประกาศไว้ใน MusicBrainz และ URIref ที่ประกาศไว้ใน Freebase ก่อนดังในรูปที่ 4.3

```
def getBands4Location(g):
    """query DBpedia to get a list of rock bands from a
location"""

    dbpedia_query = """
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX owl: <http://www.w3.org/2002/07/owl#>
        PREFIX dbpp: <http://dbpedia.org/property/>
        PREFIX dbpo:<http://dbpedia.org/ontology/>
        PREFIX dbpr:<http://dbpedia.org/resource/>

        CONSTRUCT {
          ?band owl:sameAs ?link .
        } WHERE {
          ?loc dbpp:commonName '""" + countryname + """'@en .
          ?band dbpo:hometown ?loc .
          ?band rdf:type dbpo:MusicalArtist .
          ?band dbpo:genre dbpr:Rock_music .
          ?band owl:sameAs ?link .
            FILTER regex(?link, "freebase")
        }"""
    print "Fetching DBpedia SPARQL results (this may take a few
seconds)"
    dbpedia_data = sparql(dbpedia_sparql_endpoint_URL,
dbpedia_query)
    g.parse(StringIO(dbpedia_data),format='n3') #put results in
local triplestore
    print "done with dbpedia query
```

รูป 4.3 โค้ดฟังก์ชันที่ทำหน้าที่สืบค้นข้อมูลจาก DBpedia

3) นิยามฟังก์ชันที่ทำหน้าที่สืบค้นข้อมูลวงดนตรีที่มีอัตลักษณ์ของ Freebase แต่ไม่มีข้อมูลเกี่ยวกับชื่อวงดนตรีกำกับเก็บไว้ในตัวแปรกราฟภายในโปรแกรมประยุกต์ ซึ่งฟังก์ชันจะติดต่อไปยัง URIref เหล่านั้นเพื่อผสานข้อมูลต่าง ๆ เกี่ยวกับวงดนตรีที่เก็บบันทึกอยู่บน Freebase เข้ากันกับข้อมูลกราฟภายในโปรแกรมประยุกต์ดังในรูปที่ 4.4

```
def getMBZIDs(g):
    """Query the local triplestore to find the Freebase links for
each band
    and load the Freebase data for the band into the local
triplestore"""

    #Freebase provides the canonical name for each band,
    #so if the name is missing, we know that we haven't asked
Freebase about it
    fbquery = """
        SELECT ?fblink WHERE{
            ?band owl:sameAs ?fblink .
            OPTIONAL { ?fblink fb:type.object.name ?name . }
            FILTER regex(?fblink, "freebase", "i")
            FILTER (!bound(?name))
        }"""

    freebaserefs = [fref[0] for fref in g.query(fbquery,
initNs=nsdict)]

    print "Fetching " + str(len(freebaserefs)) + " items from
Freebase"
    for fbref in freebaserefs:
        fetchRDF(str(fbref), g
```

รูป 4.4 โค้ดฟังก์ชันที่ทำหน้าที่สืบค้นข้อมูลวงดนตรีจาก Freebase

4) นิยามกลุ่มฟังก์ชันที่ทำการสืบค้นหาวงดนตรีที่มี URIref ของ BBC กำกับอยู่ แต่ไม่มีบทวิจารณ์เก็บบันทึกไว้ โดยฟังก์ชันแรกจะไปสืบค้นข้อมูลเกี่ยวกับอัลบั้มที่วงดนตรีแต่ละวงได้ผลิตออกมา และฟังก์ชันที่สองจะไปสืบค้นหาบทวิจารณ์ของแต่ละอัลบั้มที่วงดนตรีของประเทศนั้น ๆ ได้ผลิตออกมาจาก BBC นำมาผนวกเข้ากับตัวแปรกราฟภายในโปรแกรมประยุกต์ ดังรูปที่ 4.5

```
def getBBCArtistData(g):
    """For each MusicBrainz ID in the local graph try to retrieve
a review from
    the BBC"""

    #BBC will provide the album review data,
    #so if its missing we haven't retrieved BBC Artist data
    bbcartist_query = """
    SELECT ?bbcuri

    WHERE {
        ?band owl:sameAs ?bbcuri .
        OPTIONAL{
             ?band foaf:made ?a .
             ?a dc:title ?album .
             ?a rev:hasReview ?reviewuri .
        }
        FILTER regex(?bbcuri, "bbc", "i")
        FILTER (!bound(?album))
    }"""

    result = g.query(bbcartist_query, initNs=nsdict)
    print "Fetching " + str(len(result)) + " artists from the
BBC"

    for bbcartist in result:
        fetchRDF(str(bbcartist[0]), g)

def getBBCReviewData(g):
    #BBC review provides the review text
    #if its missing we haven't retrieved the BBC review
    album_query = """
    SELECT ?artist ?title ?rev
    WHERE {
      ?artist foaf:made ?a .
      ?a dc:title ?title .
      ?a rev:hasReview ?rev .
    }"""

    bbc_album_results = g.query(album_query, initNs=nsdict)
    print "Fetching " + str(len(bbc_album_results)) + " reviews
from the BBC"

    #get the BBC review of the album
    for result in bbc_album_results:
        fetchRDF(result[2], g)
```

รูป 4.5 โค้ดฟังก์ชันที่ทำหน้าที่สืบค้นรายชื่ออัลบั้ม และบทวิจารณ์ของอัลบั้มต่าง ๆ

5) จากนั้นนำฟังก์ชันต่าง ๆ ที่ได้นิยามไว้มาเรียกใช้งานตามลำดับที่แสดงไว้ในรูป 4.5 พร้อมทั้งแสดงบทวิจารณ์ของอัลบั้มต่าง ๆ ที่สืบค้นมาได้ ดังโค้ดที่แสดงในรูป 4.6

```
if __name__ == "__main__":
    g = ConjunctiveGraph()
    print "Number of Statements in Graph: " + str(len(g))

    getBands4Location(g)
    print "Number of Statements in Graph: " + str(len(g))

    getMBZIDs(g)
    print "Number of Statements in Graph: " + str(len(g))

    getBBCArtistData(g)
    print "Number of Statements in Graph: " + str(len(g))

    getBBCReviewData(g)
    print "Number of Statements in Graph: " + str(len(g))

    final_query = """
    SELECT ?name ?album ?reviewtext
    WHERE {
        ?fbband fb:type.object.name ?name .
        ?fbband owl:sameAs ?bband .
        ?bband foaf:made ?bn0 .
        ?bn0 dc:title ?album .
        ?bn0 rev:hasReview ?rev .
        ?rev rev:text ?reviewtext .
        FILTER ( lang(?name) = "en" )
    }"""

    finalresult = g.query(final_query, initNs=nsdict)
    for res in finalresult:
        print "ARTIST: " + res[0] + " ALBUM: " + res[1]
        print "------------------------------"
        print res[2]
        print "=============================
```

รูป 4.6 โค้ดเรียกใช้ฟังก์ชันต่าง ๆ ที่นิยามเอาไว้ เพื่อสืบค้นข้อมูลจากแหล่งต่าง ๆ

4.1.3 ผลลัพธ์จากการทดลอง

ผลลัพธ์จากการเรียกโปรแกรมประยุกต์แสดงไว้ในรูป 4.7 โดยสังเกตเห็นว่ามีเพียงอัลบั้มเดียวเท่านั้นที่มีบทวิจารณ์

```
Number of Statements in Graph: 0
Fetching DBpedia SPARQL results (this may take a few seconds)
done with dbpedia query
Number of Statements in Graph: 8
Fetching 8 items from Freebase
Number of Statements in Graph: 609
Fetching 6 artists from the BBC
Number of Statements in Graph: 704
Fetching 1 reviews from the BBC
```

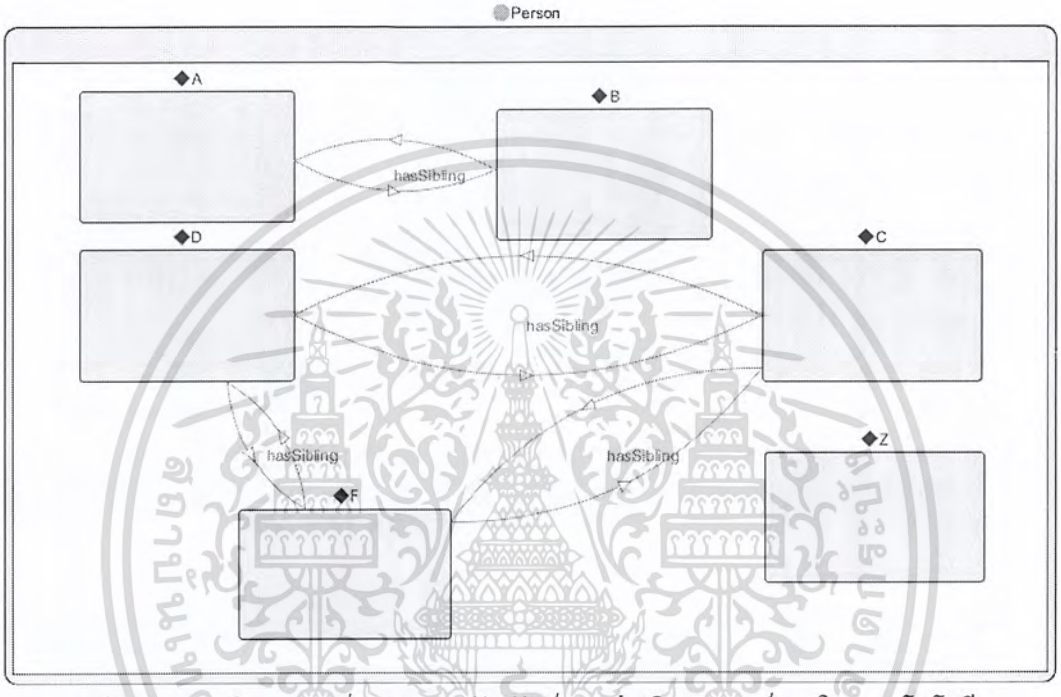
รูป 4.7 ผลลัพธ์จากการเรียกโปรแกรมประยุกต์

```
Number of Statements in Graph: 720
ARTIST: Aqualung ALBUM: Still Life
-----------------------------
<p>Aqualung follow their successful debut eponymous album with a sublime
new rec
ord titled Still Life. The music here is as melodically mellow as ever and
the l
yrical content a similar mixture of odes and epitaphs to love.</p>

<p>"Brighter Than Sunshine", the current single, starts the ten track feast
for
your ears. Your winter blues will be banished as you bask in this
triumphantly u
plifting anthem.</p>

<p>Although Still Life never quite reaches such dizzy heights again, the
remaini
ng songs are still better than your average album fodder. Another high is
reache
d with "7 Keys" as its trombone bass-line swoop in and out of the
track.</p>

<p>A wide range of instrumentation is employed ranging from the regular
strings
to the more exotic harpsichord and the downright bizarre 'frog' and
'bowls'. How
ever this wide assortment is blended together so seamlessly you'll hardly
notice
 they're all there.</p>

<p>It would be no surprise to find Coldplay, Radiohead and U2 amongst
singer Mat
t Hale's music collection. His voice is often reminiscent of Thom Yorke's
and th
e general Aqualung mood is very similar to that of the Oxford quartet. On
the ot
her hand the intro to "Listen Again" is a dead ringer to the start of
Coldplay's
 "Politik", whereas tracks such as "Take Me Home" put me more in mind of
U2's mo
re introspective moments.</p>

<p>By the end of the album the down-beat tempo can become rather dirgeful
but if
 you're rejoicing in the fact that cannabis has been downgraded you'll
probably
appreciate Still Life all the more.</p>

<p>Aqualung became successful after "Strange And Beautiful (I'll Put A
Spell On
You)" was used in a car advert. It wouldn't be a surprise or a shame if we
found
 any more of their tracks appearing in such commercial breaks. Whatever it
takes
 to spread the word and spread some aural happiness!</p>
==============================
```

รูป 4.7 ผลลัพธ์จากการเรียกโปรแกรมประยุกต์ (ต่อ)

4.2 การทดลองที่ 2 ทดสอบความสามารถในการเข้าใจออนโทโลยีของภาษาสืบค้น SPARQL

ในการทดลองนี้ จะทำการสร้างออนโทโลยีอย่างง่ายซึ่งแสดงถึงความสัมพันธ์ระหว่างอินสแตนซ์บุคคลที่มีภาคแสดง hasSibling เชื่อมโยงระหว่างกันเพื่อแสดงคุณสมบัติความเป็นพี่น้องกัน โดยภาคแสดง hasSibling นี้จะมีคุณสมบัติเป็นคุณสมบัติถ่ายทอด และคุณสมบัติสมมาตร



รูป 4.8 คลาส อินสแตนส์ และความสัมพันธ์ระหว่างอินสแตนส์ภายในออนโทโลยี

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/Ontology1284724820.owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1284724820.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Person"/>
  <owl:SymmetricProperty rdf:ID="hasSibling">
rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  </owl:SymmetricProperty>
```

รูป 4.9 โค้ดอธิบายคลาส ภาคแสดง และอินสแตนส์ของออนโทโลยีสำหรับใช้ในการทดลอง

```
  <Person rdf:ID="Z"/>
  <Person rdf:ID="D">
    <hasSibling>
      <Person rdf:ID="C">
        <hasSibling rdf:resource="#D"/>
        <hasSibling>
          <Person rdf:ID="F">
            <hasSibling rdf:resource="#C"/>
            <hasSibling rdf:resource="#D"/>
          </Person>
        </hasSibling>
      </Person>
    </hasSibling>
    <hasSibling rdf:resource="#F"/>
  </Person>
  <Person rdf:ID="A">
    <hasSibling>
      <Person rdf:ID="B">
        <hasSibling rdf:resource="#A"/> <!--WE ARE GOING TO DELETE THIS
LINE>
      </Person>
    </hasSibling>
  </Person>
</rdf:RDF>

        <!-- Created with Protege (with OWL Plugin 3.4.4, Build 579)
                        http://protege.stanford.edu -->
```

รูป 4.9 โค้ดอธิบายคลาส ภาคแสดง และอินสแตนส์ของออนโทโลยีสำหรับใช้ในการทดลอง (ต่อ)

เนื่องจากออนโทโลยี และอินสแตนส์ในการทดลองนี้ใช้โปรแกรม Protégé สร้างขึ้นซึ่งโปรแกรมมีความเข้าใจออนโทโลยีจึงทำการเขียนอธิบายข้อมูลที่ได้จากการสรุปลงไปในเอกสาร RDF ไว้ให้แล้ว ยกตัวอย่างเช่น หากเราระบุว่าบุคคล A เป็นพี่น้องกับบุคคล B แล้ว ย่อมสรุปได้โดยนัยว่า บุคคล B ก็เป็นพี่น้อง A ด้วย ดังเช่นโค้ดที่ทำตัวหนาเอาไว้ในรูป 4.9 ซึ่งเราจะทำการลบออกเพื่อทดสอบว่าภาษา SPARQL นั้นสามารถทำความเข้าใจออนโทโลยี แล้วสืบค้นข้อมูลที่ได้จากการสรุปออกมาได้หรือไม่ โดยจะใช้เครื่องมือ Twinkle ซึ่งใช้ตัวประมวลผลภาษา SPARQL ชื่อ ARQ และไลบรารี RDFlib ในการสืบค้นข้อมูลจากเอกสาร RDF ในรูป 4.9

4.2.1 จุดประสงค์การทดลอง

เพื่อทดสอบความเข้าใจออนโทโลยีของภาษา SPARQL

4.2.2 ขั้นตอนการทดลอง

1) ทำการสืบค้นด้วยเครื่องมือ Twinkle ด้วยคำสั่งสืบค้นภาษา SPARQL รูปที่ 4.10

2) ทำการสืบค้นด้วยตัวประมวลผลที่มีให้ในไลบรารี RDFlib โดยการเขียนโค้ดดังแสดงในรูป 4.11

```
PREFIX ab: <http://www.owl-ontologies.com/Ontology1284724820.owl#>

SELECT ?subject ?object
WHERE
{
  ?subject ab:hasSibling ?object.
}
ORDER BY ?subject
```

รูป 4.10 คำสั่งสืบค้น

```
from rdflib import ConjunctiveGraph, Namespace, URIRef

ab = Namespace("http://www.owl-ontologies.com/Ontology1284724820.owl#")
nsDict = {'ab':ab}

g = ConjunctiveGraph()
g.parse("AB.owl", format="xml")

query_string = \
        """SELECT ?subject ?object
           WHERE
           {
              ?subject ab:hasSibling ?object.
           }
           ORDER BY ?subject"""

results = g.query(query_string, initNs=nsDict)
for result in results:
    print("{0} has sibling {1}.".format(result[0][-1], result[1][-1]))
```

รูป 4.11 โค้ดโปรแกรมที่สืบค้นข้อมูลแบบเดียวกัน

4.2.3 ผลลัพธ์จากการทดลอง

จากผลการสืบค้นข้อมูลในรูป 4.11 และรูป 4.12 พบว่าไม่มีคำตอบที่บ่งชี้ว่าบุคคล B เป็นพี่น้องกับบุคคล A ซึ่งสรุปได้ว่าภาษาสืบค้น SPARQL ไม่มีความเข้าใจออนโทโลยีที่กำกับไว้ในเอกสาร RDF จึงหมายความว่าภาษา SPARQL จะทำงานอยู่ในระดับชั้น RDF เท่านั้น

subject	object
http://www.owl-ontologies.com/Ontology1284724820.owl#A	http://www.owl-ontologies.com/Ontology1284724820.owl#B
http://www.owl-ontologies.com/Ontology1284724820.owl#C	http://www.owl-ontologies.com/Ontology1284724820.owl#D
http://www.owl-ontologies.com/Ontology1284724820.owl#C	http://www.owl-ontologies.com/Ontology1284724820.owl#F
http://www.owl-ontologies.com/Ontology1284724820.owl#D	http://www.owl-ontologies.com/Ontology1284724820.owl#C
http://www.owl-ontologies.com/Ontology1284724820.owl#D	http://www.owl-ontologies.com/Ontology1284724820.owl#F
http://www.owl-ontologies.com/Ontology1284724820.owl#F	http://www.owl-ontologies.com/Ontology1284724820.owl#C
http://www.owl-ontologies.com/Ontology1284724820.owl#F	http://www.owl-ontologies.com/Ontology1284724820.owl#D

รูป 4.12 ผลลัพธ์การสืบค้นด้วยเครื่องมือ Twinkle

```
A has sibling B.
C has sibling F.
C has sibling D.
D has sibling F.
D has sibling C.
F has sibling D.
F has sibling C.
```

รูป 4.13 ผลลัพธ์จากการสืบค้นด้วยโปรแกรมที่ใช้ไลบรารี RDFlib

4.3 การทดลองที่ 3 การทดลอง OWL

การทดลองสร้าง Ontology(.owl) และไฟล์ RDF โดยใช้โปรแกรม Protege โดยเป็น Ontology ของข้อมูลรายละเอียดและข้อมูลการแต่งงานของประธานาธิบดี

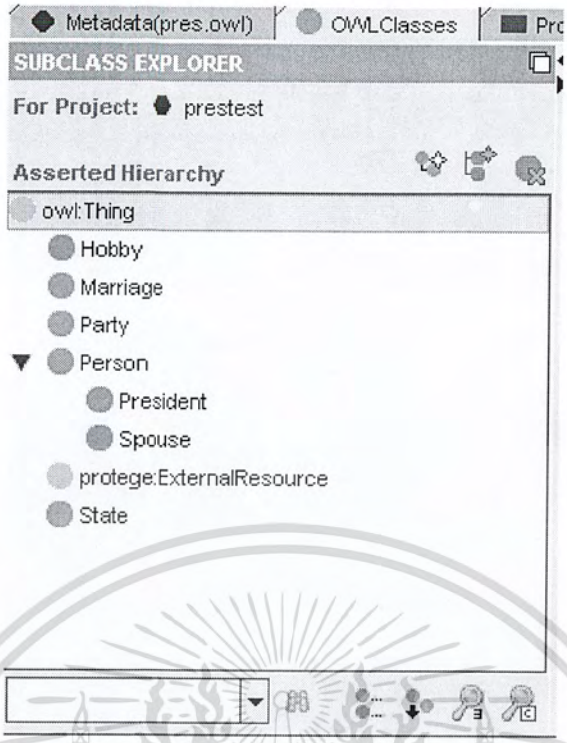
4.3.1 จุดประสงค์ของการทดลอง

1) เพื่อศึกษาโครงสร้างและการทดลองเรื่อง Ontology

2) เพื่อทดสอบการ Query ด้วยภาษา SPARQL

4.3.2 ขั้นตอนการทดลอง

ทำการตั้งชื่อ Ontology ว่า prestest.owl โดยมี URI เป็น http://www.owl-ontologies.com/pres.owl และโดยจะมี คลาส owl:Thing เป็นหลัก(ใน protégé owl คลาส ทุก คลาส ต้องสืบทอดมาจาก คลาส owl:Thing) และมีคลาส ลูกประกอบด้วย คลาส Person ,Hobby ,Marriage ,Party และ State และ คลาส ย่อยของ คลาส Person ก็ประกอบไปด้วย President และ Spouse อีก ซึ่งจะเก็บ โดเมน อันประกอบด้วย บุคคลซึ่งเป็นประธานาธิบดีและภรรยาของประธานาธิบดี งานอดิเรก พรรคที่สังกัด และสถานที่เกิด นอกจากนั้นต้องมีการกำหนด คุณสมบัติ และคุณสมบัติ value ของ รายละเอียดต่างๆ รวมถึง instance ทั้งหมด

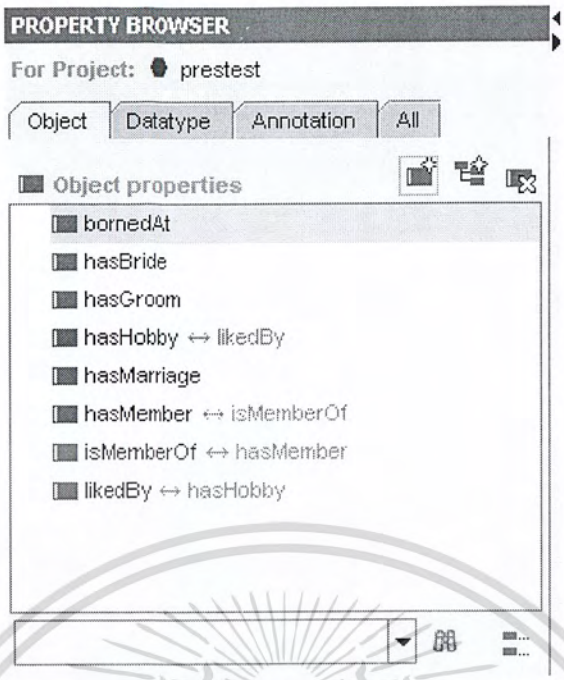


รูป 4.14 การกำหนด คลาส ของ Ontology

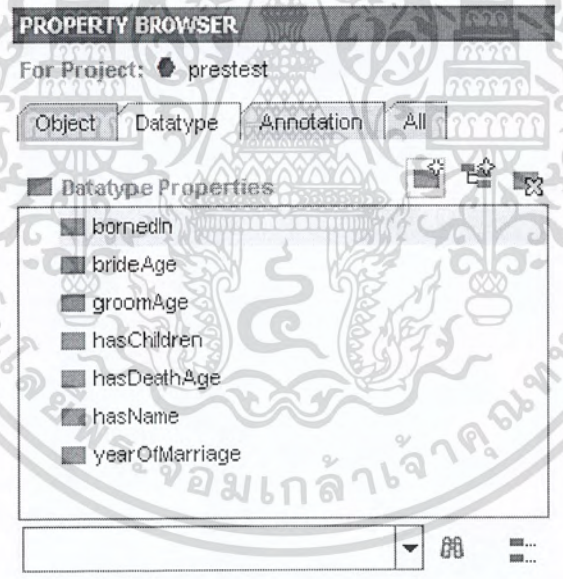
ในคลาสของ owl นี้ จะสามารถใส่ขอบเขตหรือข้อกำหนด เพิ่มเติมที่เรียกว่า Restriction ในการควบคุม ให้ instance ที่อยู่ใน โดเมน เช่น ประธานาธิบดีสามารถสังกัดพรรคได้เพียง หนึ่งพรรค ก็จะถูกกำหนดโดยส่วนนี้

รูป 4.15 ตัวอย่างการกำหนด Restriction ของคลาส President

การเพิ่ม คุณสมบัติ ทั้ง คุณสมบัติ ชนิด กรรม และ ชนิดข้อมูล อีกทั้งต้องกำหนดขอบเขต โดเมน และ เรนจ์ เช่น โดเมน ประธานาธิบดี คุณสมบัติ กรรม เป็น bornAt และ เรนจ์ เป็น State เพื่อ บ่งบอกว่าประธานาธิบดีท่านหนึ่ง เกิดที่ รัฐใด

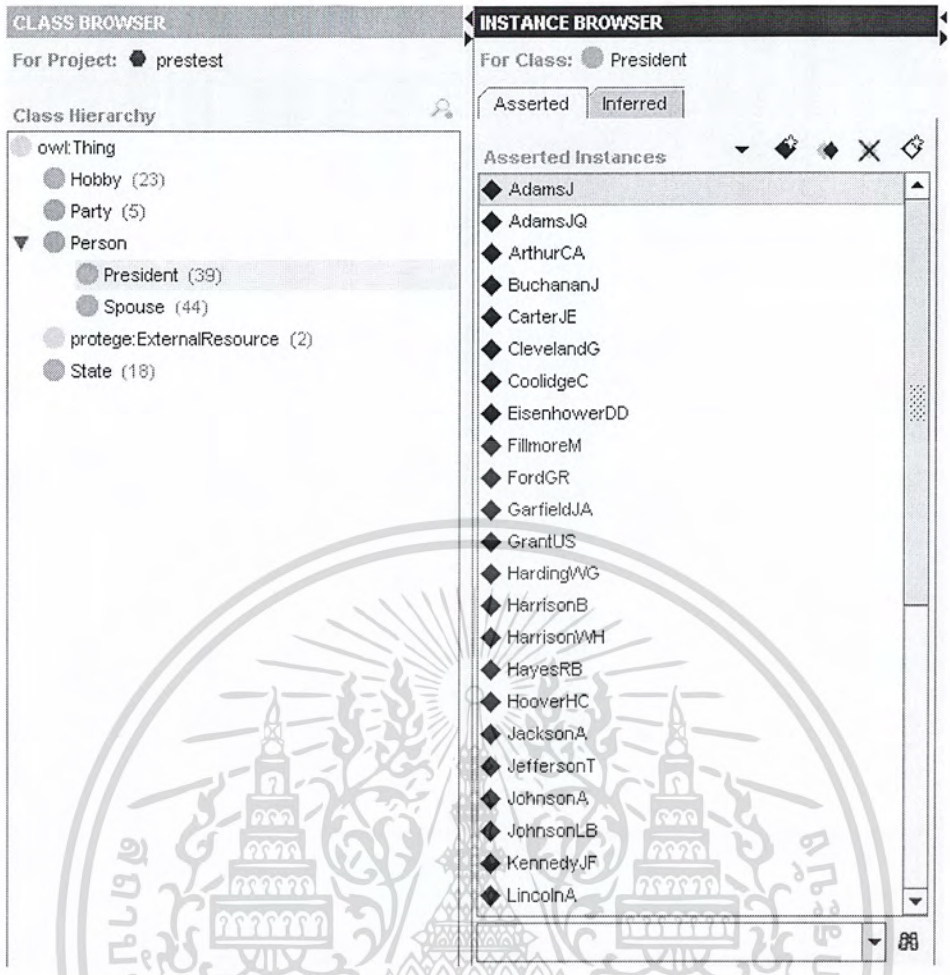


รูป 4.16 การใส่ Object Property

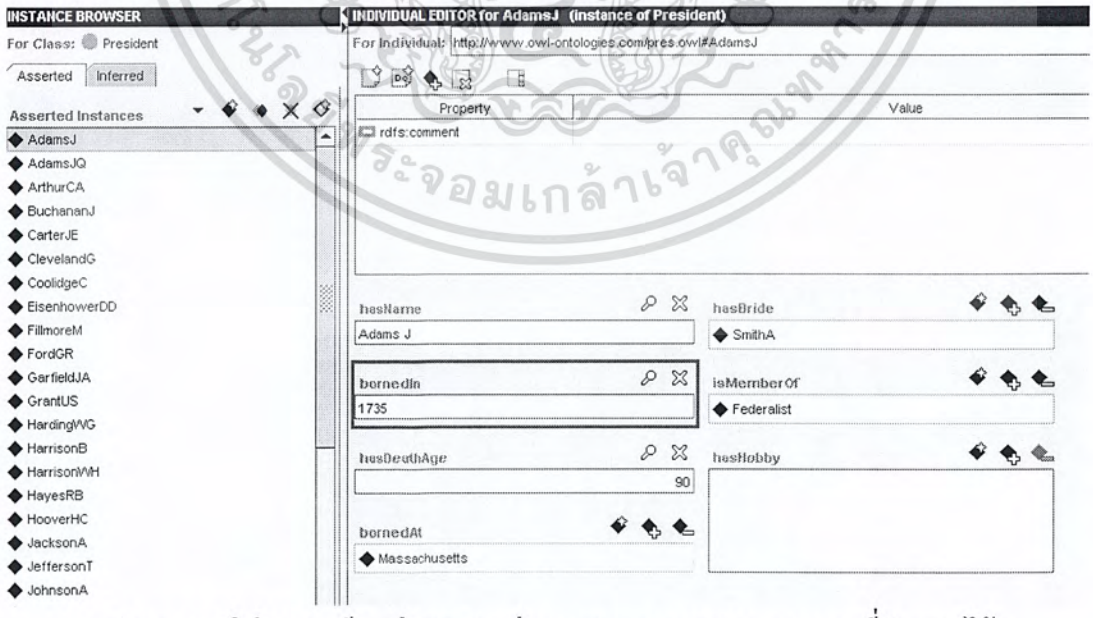


รูป 4.17 การใส่ ชนิดข้อมูล Property

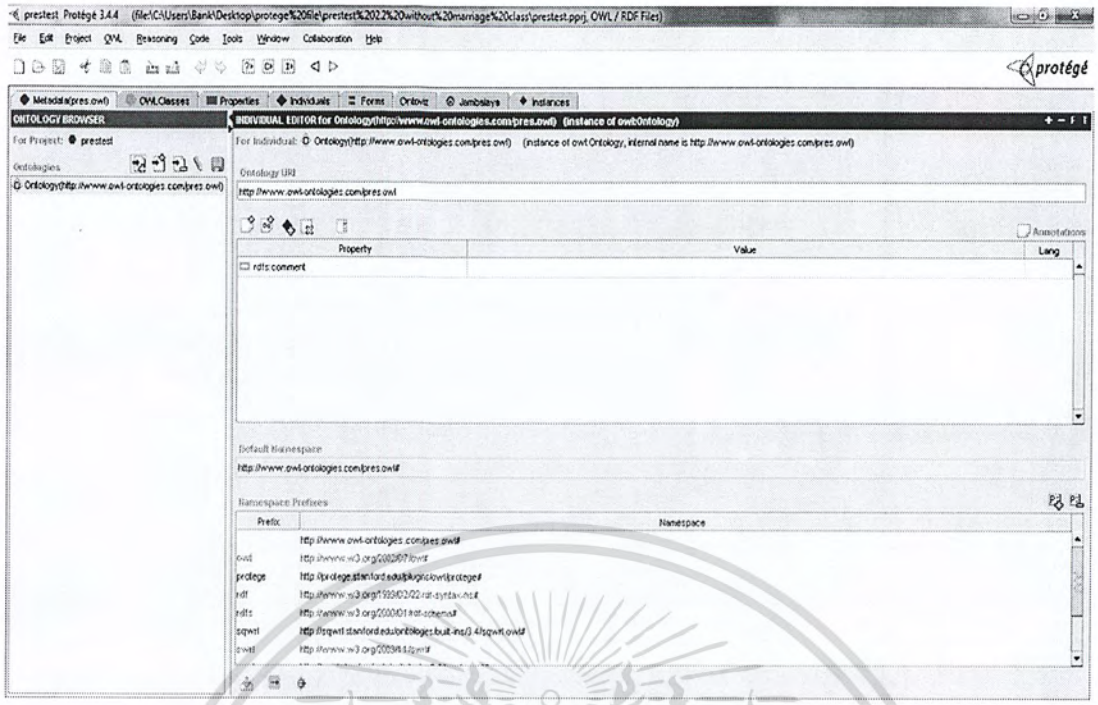
อีกทั้งต้องระบุ instance ที่เกิดขึ้นต่างๆในแต่ละ คลาส ที่กำหนดไว้ข้างต้น อันประกอบด้วย instance ของ คลาส Hobby ,Person ,Party และ State ซึ่งเป็นการระบุว่า งานอดิเรกมีอะไร พรรคประกอบด้วยพรรคอะไร และมีรัฐอะไรบ้าง มีประธานาธิบดีเป็นใคร ซึ่งจะมีรายละเอียดไปอีกว่าประธานาธิบดี



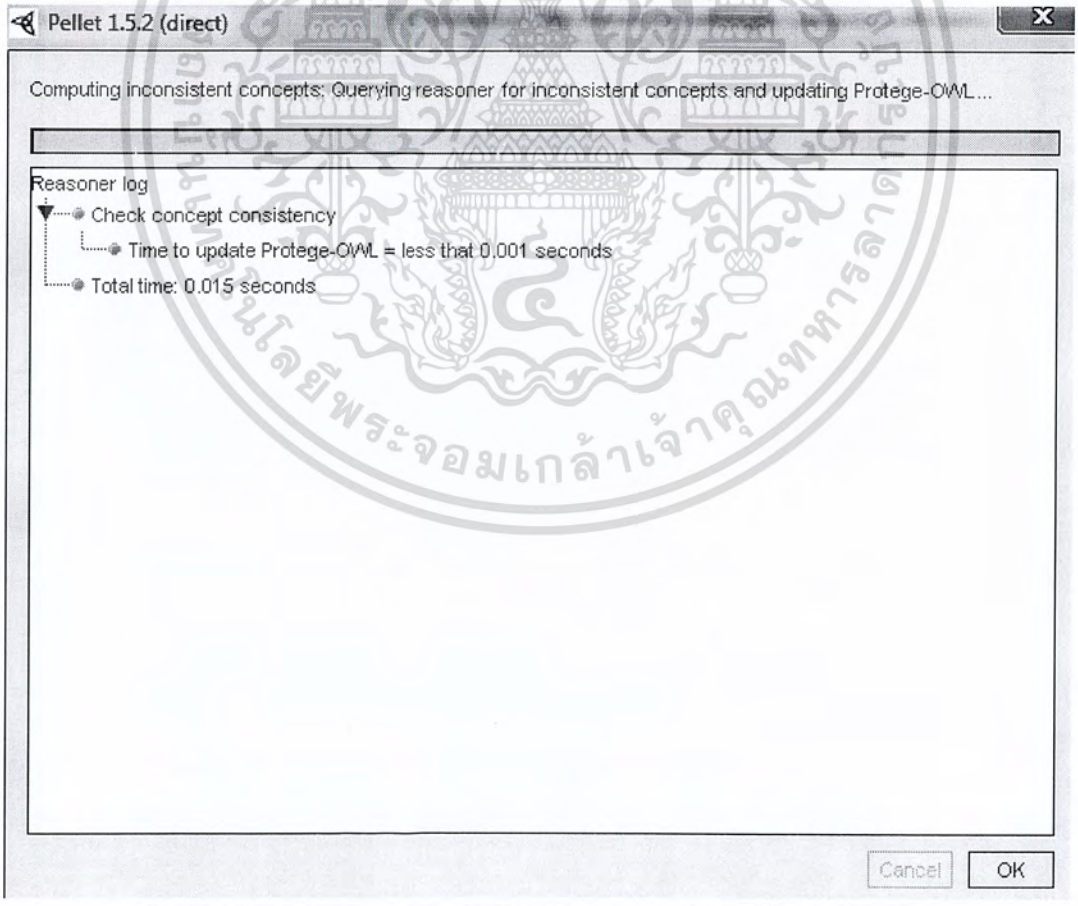
รูป 4.18 ตัวอย่าง instance ของ คลาส President



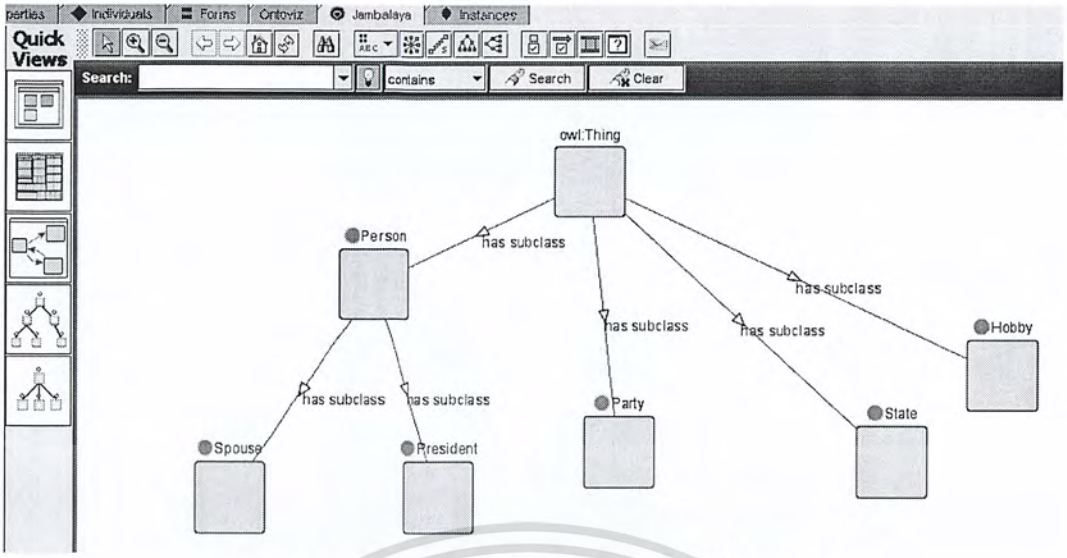
รูป 4.19 การใส่รายละเอียดย่อยของ แต่ละ instance ตาม Restriction ที่กำหนดไว้



รูป 4.20 ภาพโปรแกรมโดยรวม



รูป 4.21 มี Reasoning check ความถูกต้องของ ontology โดย Pellet 1.5.2



รูป 4.22 ความสัมพันธ์ระหว่างกราฟออนโทโลยี President

รูป 4.23 source code ที่ Protégé สามารถ generate ได้

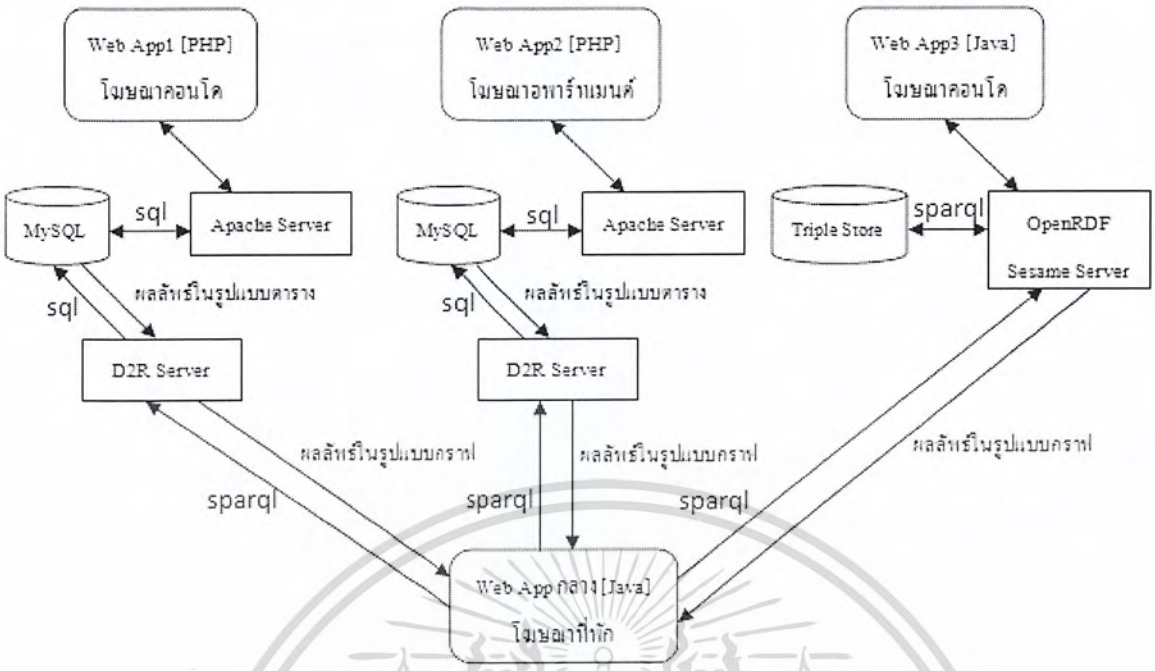
บทที่ 5
พัฒนาแอพพลิเคชัน RDF และ RDFS

5.1 การทดลองพัฒนาแอพพลิเคชัน

ทดลองสร้างเว็บแอปพลิเคชันกลางที่เผยแพร่ข้อมูลในรูปแบบข้อมูลกราฟ RDF ที่ทำหน้าที่สืบค้นข้อมูลจากเว็บไซต์ 3 เว็บไซต์ย่อย ได้แก่

1) เว็บไซต์คอนโดมีเนียมที่พัฒนาด้วยภาษา PHP ใช้ระบบฐานข้อมูลเชิงสัมพันธ์ MySQL และใช้เครื่องมือ D2R Server ในการเผยแพร่เนื้อหาข้อมูลที่เก็บในรูปเชิงสัมพันธ์ให้อยู่ในรูปกราฟ RDF
2) เว็บไซต์อพาร์ตเมนต์ที่พัฒนาด้วยภาษา PHP ใช้ระบบฐานข้อมูลเชิงสัมพันธ์ MySQL และใช้เครื่องมือ D2R Server ในการเผยแพร่เนื้อหาข้อมูลที่เก็บในรูปเชิงสัมพันธ์ให้อยู่ในรูปกราฟ RDF
3) เว็บไซต์คอนโดมีเนียมที่พัฒนาด้วยภาษา Java โดยใช้ JSP (Java Servlet Page) และใช้ทริปเปิลสโตร์ OpenRDF Sesame Server สำหรับเก็บข้อมูลโฆษณาในรูปแบบ RDF และยังเป็นตัวกลางที่เปิดให้ภายนอกเข้ามาสืบค้นข้อมูลภายในด้วยภาษา SPARQL

โดยจะรับเงื่อนไขในการสืบค้นจากผู้ใช้ แล้วกระจายคำสั่งสืบค้นไปยังเว็บไซต์ทั้ง 3 แห่ง จากนั้นทำการแปลงข้อมูลกราฟ RDF ในแต่ละแหล่งที่มีโครงสร้างต่างกัน ให้มีโครงสร้าง RDF Schema เหมือนกับที่นิยามเอาไว้สำหรับเว็บแอปพลิเคชันกลาง จากนั้นจึงแสดงผลลัพธ์จากการสืบค้น



รูป 5.1 ทิศทางการสืบค้นข้อมูลของแอพพลิเคชันกลาง

5.2 จุดประสงค์การทดลอง

จุดประสงค์การทดลองมีดังนี้

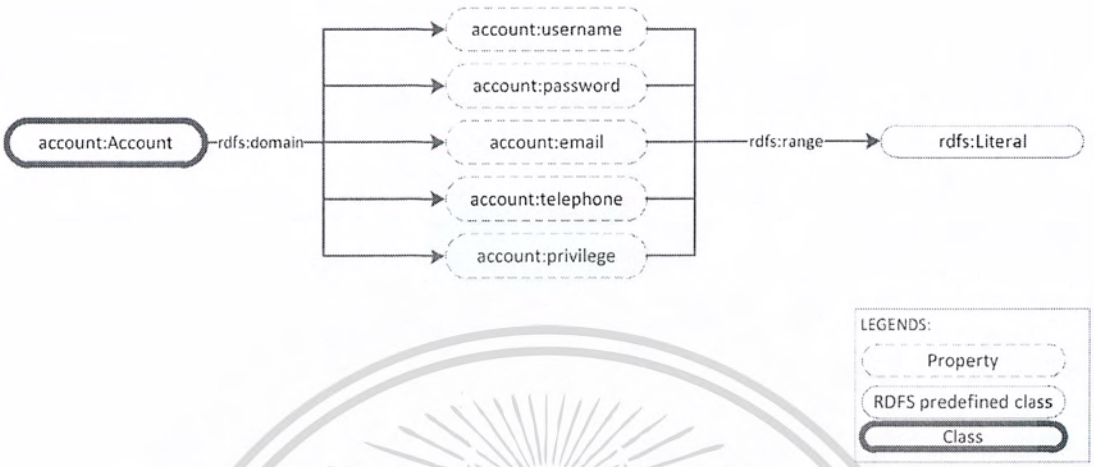
1) ทดลองสร้างโปรแกรมประยุกต์ที่ใช้งานฐานข้อมูลที่ขับเคลื่อนด้วย RDF
2) ศึกษาการใช้งานไลบรารี OpenRDF Sesame
3) ศึกษาการใช้งาน OpenRDF Sesame Server
4) ศึกษาการบริหารจัดการที่เก็บข้อมูลกราฟ RDF ผ่านเว็บแอปพลิเคชัน OpenRDF Workbench
5) ศึกษาการใช้งาน D2R Server เพื่อสร้างเซิร์ฟเวอร์สำหรับเผยแพร่เนื้อหาข้อมูลในรูปแบบเชิงสัมพันธ์ให้อยู่ในรูปแบบกราฟข้อมูล RDF

5.3 ขั้นตอนการทดลอง

5.3.1 สร้างเว็บไซต์คอนโดมีเนียม

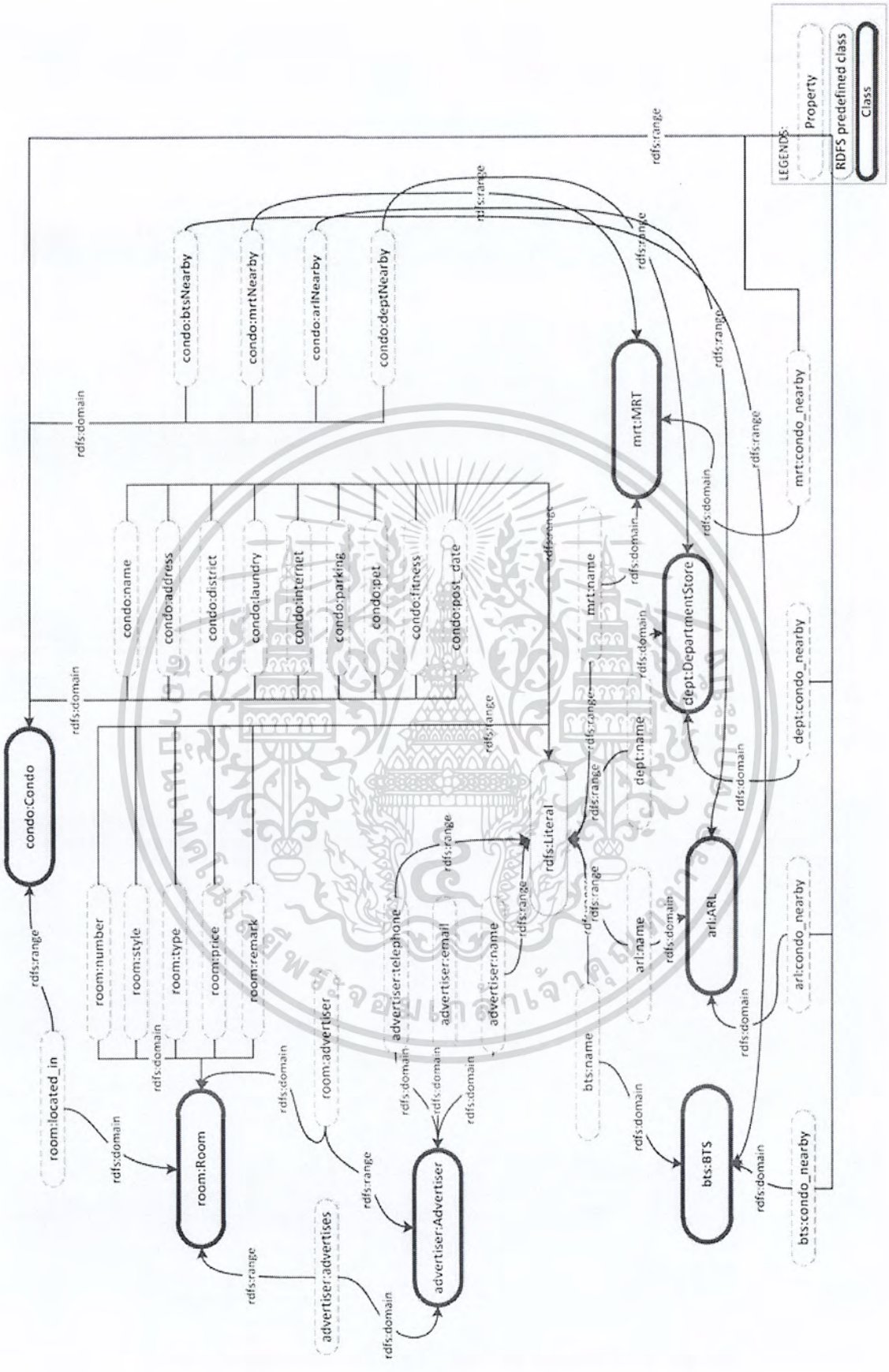
สร้างเว็บไซต์คอนโดมีเนียมที่พัฒนาด้วยภาษา Java โดยใช้ JSP และใช้ OpenRDF Sesame Server เป็นทริปเปิลสโตร์ ขั้นตอนดังนี้

1) นิยาม RDF Schema สำหรับเก็บข้อมูลสมาชิกในเว็บ เช่น รหัสผ่าน เป็นต้น โดยจะสมาชิกเว็บไซต์จะมีสองระดับคือ ระดับผู้ใช้งาน และระดับผู้ดูแลเว็บไซต์



รูป 5.2 RDF Schema สำหรับเก็บข้อมูลสมาชิกเว็บไซต์คอนโดมีเนียม

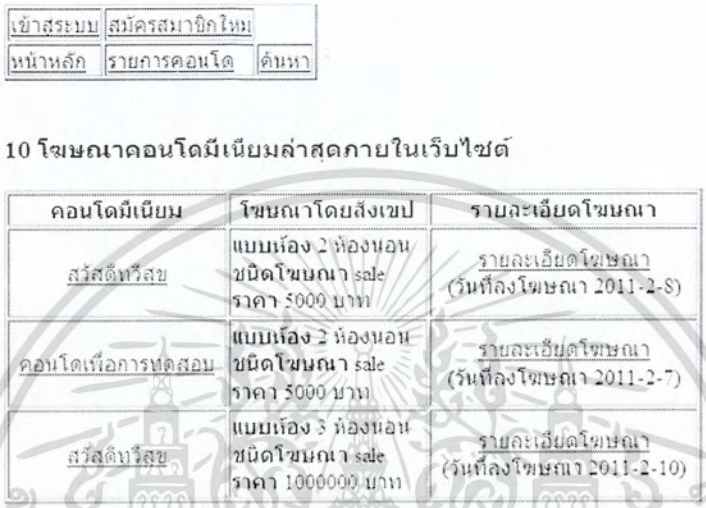
2) นิยาม RDF Schema สำหรับใช้เก็บข้อมูลเกี่ยวกับโฆษณาคอนโดมีเนียม



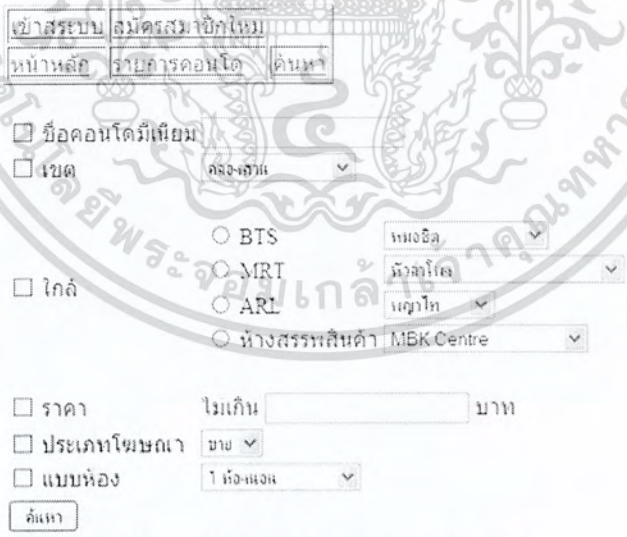
รูป 5.3 RDF Schema สำหรับเก็บข้อมูลที่จำเป็นสำหรับเก็บโฆษณา

3) สร้างที่เก็บข้อมูลสำหรับ RDF Schema ทั้งสองลงใน OpenRDF Sesame Server โดยใช้งานเว็บแอปพลิเคชัน OpenRDF Workbench

4) พัฒนาเว็บไซต์ที่เป็นประเด็นหลักของการทดลองนี้โดยใช้ JSP และคลาส RemoteGraph ที่ครอบการทำงานของ OpenRDF Sesame API



รูป 5.4 หน้าหลักของเว็บไซต์



รูป 5.5 หน้าค้นหาโฆษณาของเว็บไซต์

```
String[] criterias = request.getParameterValues("criteria");
if(criterias == null)
{
  %>
  <font color="RED">คุณไม่ได้เลือกเกณฑ์ในการค้นหา</font><br><br>
  <a href="javascript:history.go(-1);">กลับ</a>
  <%
}
else
{
  // Find condos that match the search criterias

  // Fetches all the criterias that are meant for picking the right condominiums
  List<String> condo_additional_patterns = new ArrayList<String>();
  List<String> condo_filters = new ArrayList<String>();
  for(String criteria : criterias)
  {
    if(criteria.equals("name"))
    {
      String condo_name = new String(request.getParameter("name").getBytes("ISO8859_1"), "UTF-
8");
      //condo_additional_patterns.add("?CONDO condo:name ?NAME. ");
      condo_filters.add("FILTER(regex(?NAME, \"" + condo_name + "\")) ");
    }
    else if(criteria.equals("district"))
    {
      String condo_district = new
String(request.getParameter("district").getBytes("ISO8859_1"), "UTF-8");
      condo_additional_patterns.add("?CONDO condo:district ?DISTRICT. ");
      condo_filters.add("FILTER(regex(?DISTRICT, \"" + condo_district + "\")) ");
    }
    else if(criteria.equals("nearbyPlace"))
    {
      String nearby_place = request.getParameter("nearbyPlace");
      String placeURI = request.getParameter(nearby_place);

      if(nearby_place.equals("BTS"))
        condo_additional_patterns.add("?CONDO condo:btsNearby <" + placeURI + ">. ");
      else if(nearby_place.equals("MRT"))
        condo_additional_patterns.add("?CONDO condo:mrtNearby <" + placeURI + ">. ");
      else if(nearby_place.equals("ARL"))
        condo_additional_patterns.add("?CONDO condo:arlNearby <" + placeURI + ">. ");
      else if(nearby_place.equals("DEPT"))
        condo_additional_patterns.add("?CONDO condo:deptNearby <" + placeURI + ">. ");
    }
  }
  StringBuilder _queryCondo = new StringBuilder();
  _queryCondo.append("PREFIX condo:<http://localhost:8084/condo_rdf/condo#> ");
  _queryCondo.append("SELECT ?CONDO ?NAME ");
  _queryCondo.append("WHERE { ?CONDO a condo:Condo. ?CONDO condo:name ?NAME. ");
  for(String addt_pattern : condo_additional_patterns)
    _queryCondo.append(addt_pattern);
  for(String filter : condo_filters)
    _queryCondo.append(filter);
  _queryCondo.append(" }");

  List condo_list = rg.runSPARQL(_queryCondo.toString());
  if(condo_list.size() > 0)
  {
    List<String> condoURI = new ArrayList<String>();
    List<String> condo_names = new ArrayList<String>();
    for(Object _condo : condo_list)
    {

condoURI.add(((org.openrdf.model.impl.URIImpl)((HashMap)_condo).get("CONDO")).stringValue());

condo_names.add(((org.openrdf.model.impl.LiteralImpl)((HashMap)_condo).get("NAME")).stringValu
e());
    }
```

รูป 5.6 โค้ดสำหรับค้นหาโฆษณาภายในฐานข้อมูล RDF โดยสร้างคำสั่งสืบค้น SPARQL

```
StringBuilder _queryCondo = new StringBuilder();
  _queryCondo.append("PREFIX condo:<http://localhost:8084/condo_rdf/condo#> ");
  _queryCondo.append("SELECT ?CONDO ?NAME ");
  _queryCondo.append("WHERE { ?CONDO a condo:Condo. ?CONDO condo:name ?NAME. ");
  for(String addt_pattern : condo_additional_patterns)
    _queryCondo.append(addt_pattern);
  for(String filter : condo_filters)
    _queryCondo.append(filter);
  _queryCondo.append(" }");

  List condo_list = rg.runSPARQL(_queryCondo.toString());
  if(condo_list.size() > 0)
  {
    List<String> condoURI = new ArrayList<String>();
    List<String> condo_names = new ArrayList<String>();
    for(Object _condo : condo_list)
    {

condoURI.add(((org.openrdf.model.impl.URIImpl)((HashMap)_condo).get("CONDO")).stringValue());

condo_names.add(((org.openrdf.model.impl.LiteralImpl)((HashMap)_condo).get("NAME")).stringValu
e());
    }

    List<String> room_filters = new ArrayList<String>();
    for(String criteria : criterias)
    {
      if(criteria.equals("price"))
      {
        room_filters.add("FILTER ( ?PRICE <= " + request.getParameter("price") + " ) ");
      }
      else if(criteria.equals("type"))
      {
        room_filters.add("FILTER ( regex(?TYPE, \"" + new
String(request.getParameter("type").getBytes("ISO8859_1"), "UTF-8") + "\")) ");
      }
      else if(criteria.equals("style"))
      {
        room_filters.add("FILTER ( regex(?STYLE, \"" + new
String(request.getParameter("style").getBytes("ISO8859_1"), "UTF-8") + "\")) ");
      }
    }

    for(int i = 0; i < condo_list.size(); ++i)
    {
      StringBuilder _queryRooms = new StringBuilder();
      _queryRooms.append("PREFIX room:<http://localhost:8084/condo_rdf/room#> ");
      _queryRooms.append("SELECT ?ROOM ?NUMBER ?STYLE ?TYPE ?PRICE ");
      _queryRooms.append("WHERE { ?ROOM a room:Room; room:locatedIn <" + condoURI.get(i) + ">;
room:number ?NUMBER; room:style ?STYLE; room:type ?TYPE; room:price ?PRICE. ");
      for(String filter : room_filters)
      {
        _queryRooms.append(filter);
      }
      _queryRooms.append(" }");

      List room_list = rg.runSPARQL(_queryRooms.toString());
      if(room_list.size() > 0)
      {
        %>
        <h3><%=condo_names.get(i)%></h3>
        <table border="1" width="60%">
          <tr>
            <td><b><center>ราคา</center></b></td>
            <td><b><center>ข้อมูลทั่วไป</center></b></td>
            <td><b><center>รายละเอียด</center></b></td>
          </tr>

        <%
```

รูป 5.6 โค้ดสำหรับค้นหาโฆษณาภายในฐานข้อมูล RDF โดยสร้างคำสั่งสืบค้น SPARQL (ต่อ)

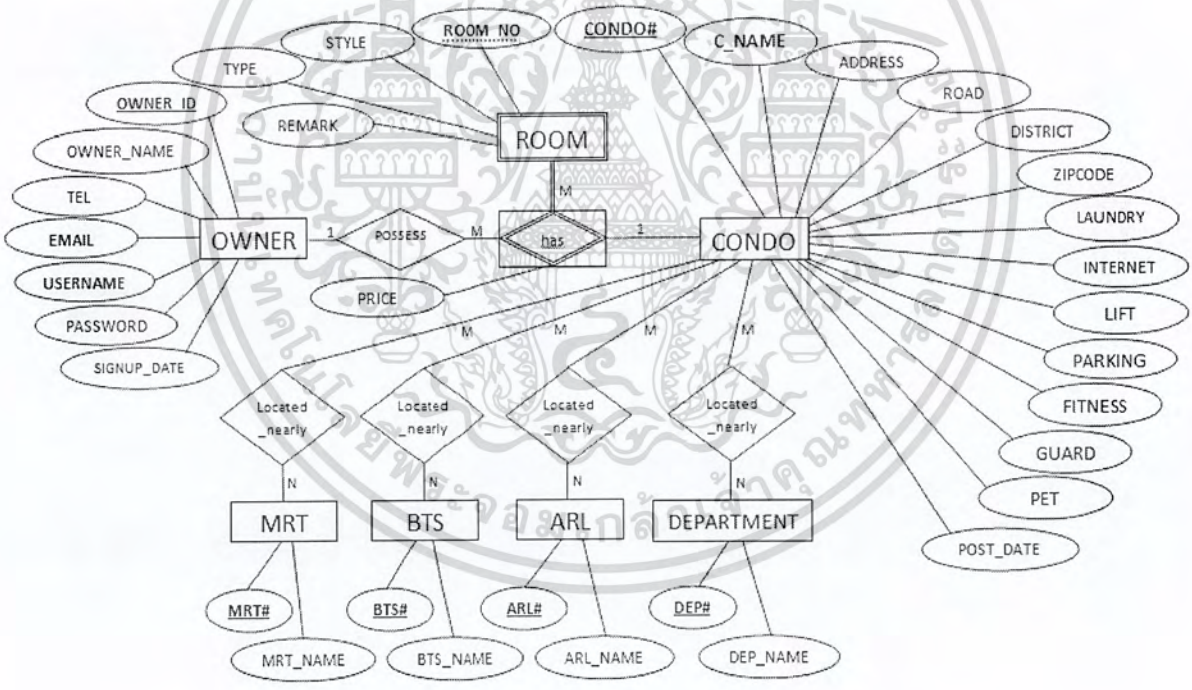
```
        %>
        </table>
        <%
      }
    }

  }
  else
  {
    %>
    <font color="RED">ไม่พบโฆษณาที่เข้าข่าย</font><br><br>
    <a href="javascript:history.go(-1);">กลับ</a>
    <%
  }
}
%>
```

รูป 5.6 โค้ดสำหรับค้นหาโฆษณาภายในฐานข้อมูล RDF โดยสร้างคำสั่งสืบค้น SPARQL (ต่อ)

5.3.2 สร้างเว็บไซต์ให้บุคคลภายนอกสมัครสมาชิกลงโฆษณาคอนโดมีเนียมด้วยภาษา PHP

มีขั้นตอนการพัฒนาดังนี้

1) กำหนดเค้าร่างฐานข้อมูลเชิงสัมพันธ์สำหรับใช้เก็บฐานข้อมูล



รูป 5.7 แผนภาพ ER อธิบายข้อมูลโฆษณาคอนโดมีเนียม

CONDO

CONDO#	CNAME	ADDRESS	ROAD	DISTRICT	ZIPCODE	LAUNDRY	INTERNET	LIFT	PARKING	FITNESS	GUARD	PET	POST_DATE

ROOM_IN_CONDO

CONDO#	ROOM#	STYLE	TYPE	PRICE	REMARK	OWNER#

OWNER

OWNER#	OWNER_NAME	TEL	EMAIL	USERNAME	PASSWORD	SIGNUP_DATE

MRT_NEARBY

CONDO#	MRT#

BTS_NEARBY

CONDO#	BTS#

ARL_NEARBY

CONDO#	ARL#

DEPARTMENT_NEARBY

CONDO#	DEP #

MRT

MRT #	MRT_NAME

BTS

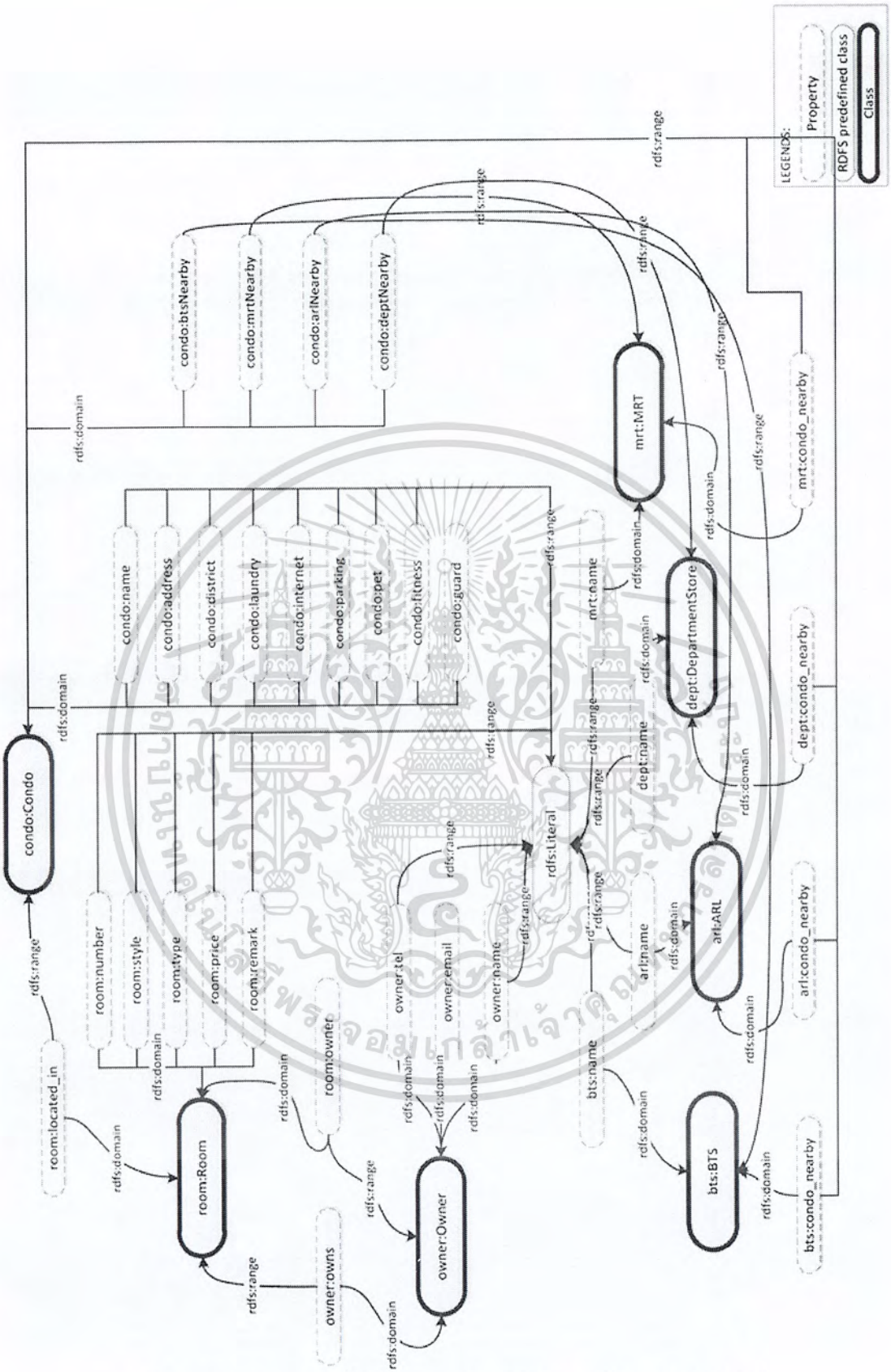
BTS#	BTS_NAME

รูป 5.8 เค้าร่างฐานข้อมูลเชิงสัมพันธ์สำหรับเก็บข้อมูลโฆษณาคอนโดมีเนียม

2) พัฒนาเว็บแอปพลิเคชันด้วยภาษา PHP

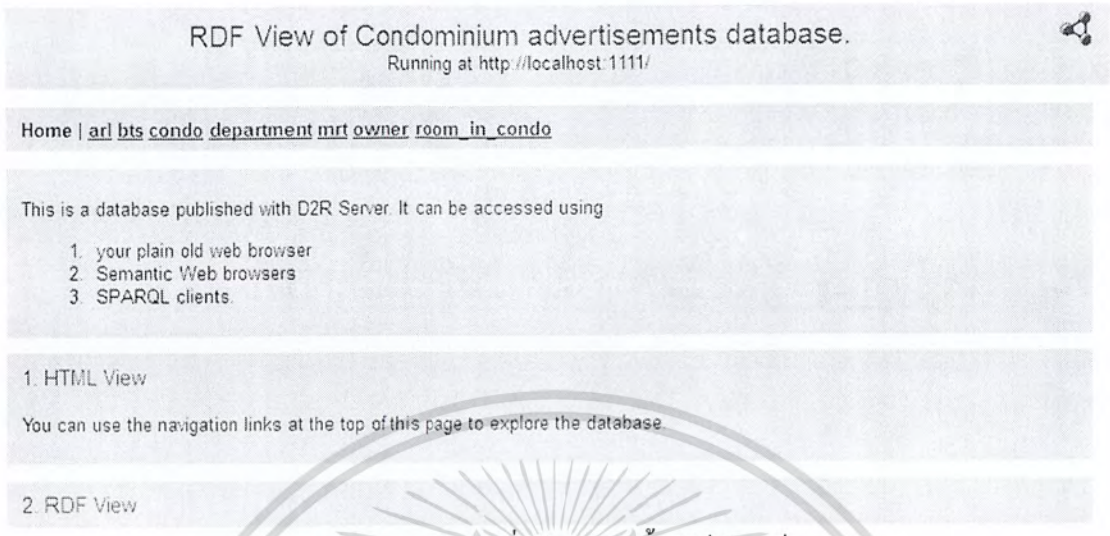
รูป 5.9 หน้าเว็บไซต์หลักของเว็บไซต์คอนโดมีเนียมที่พัฒนาด้วยภาษา PHP

3) แปลงเค้าร่างฐานข้อมูลเชิงสัมพันธ์ที่แสดงในรูป 5.8 ให้เป็น RDF Schema เก็บลงในไฟล์ที่แสดงการแปลง ได้ผลลัพธ์ดังรูป 5.10



รูป 5.10 RDF Schema ที่ได้จากการแปลงเค้าร่างฐานข้อมูลเชิงสัมพันธ์

4) นำเอาไฟล์ที่เก็บการแปลงไปใช้ตั้งเซิร์ฟเวอร์ด้วยเครื่องมือชื่อ d2r-server

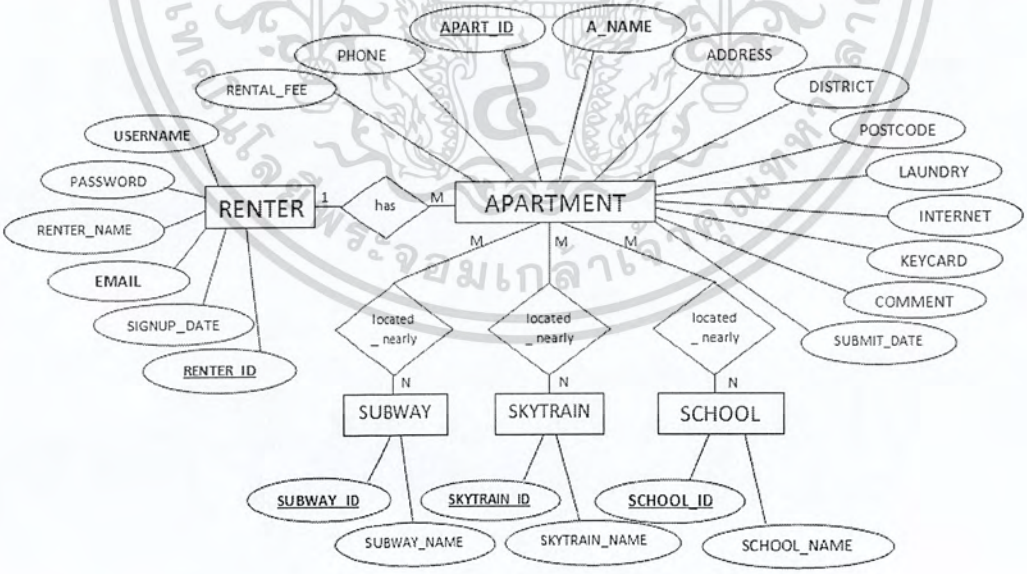


รูป 5.11 หน้าเว็บที่ได้จากการตั้งเซิร์ฟเวอร์

5.3.3 สร้างเว็บไซต์ให้บุคคลภายนอกสมัครสมาชิกลงโฆษณาคอนโดมีเนียมอพาร์ตเมนต์

มีขั้นตอนดังนี้

1) กำหนดเค้าร่างฐานข้อมูลเชิงสัมพันธ์สำหรับใช้เก็บข้อมูลโฆษณาอพาร์ตเมนต์



รูป 5.12 แผนภาพ ER อธิบายข้อมูลโฆษณาอพาร์ตเมนต์

APARTMENT

APAR_ID	A_NAME	ADDRESS	DISTRICT	POSTCODE	LAUNDRY	INTERNET	KEYCARD	PHONE	RENTAL_FEE	COMMENT	SUBMIT_DATE	RENTER_ID

RENTER

RENTER_ID	RENTER_NAME	EMAIL	USERNAME	PASSWORD	SIGNUP_DATE

SUBWAY_NEARBY

APART_ID	SUBWAY_ID

SKYTRAIN_NEARBY

APART_ID	SKYTRAIN_ID

SCHOOL_NEARBY

APART_ID	SCHOOL_ID

SUBWAY

SUBWAY_ID	SUBWAY_NAME

SKYTRAIN

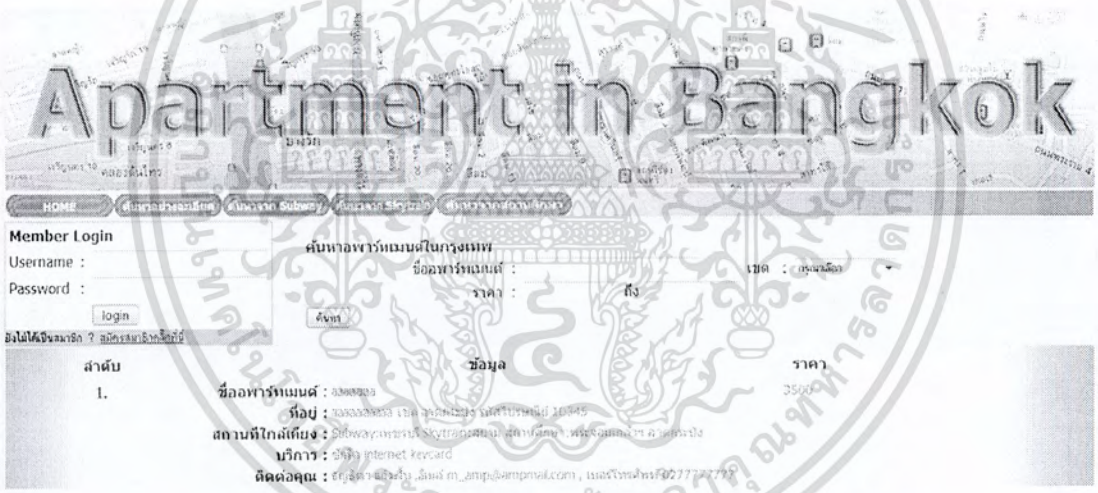
SKYTRAIN_ID	SKYTRAIN_NAME

SCHOOL

SCHOOL_ID	SCHOOL_NAME

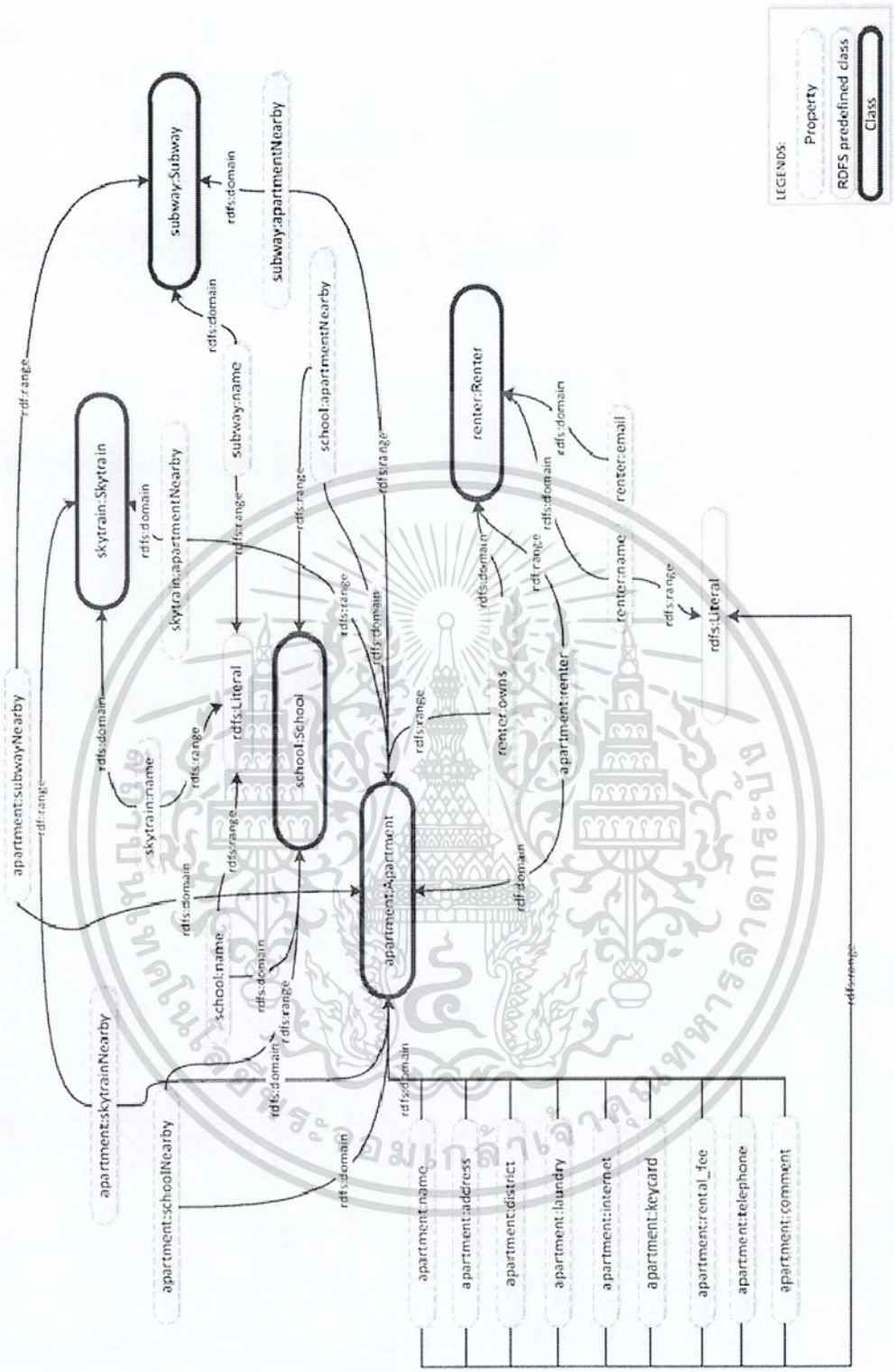
รูป 5.13 เค้าร่างฐานข้อมูลเชิงสัมพันธ์สำหรับเก็บข้อมูลโฆษณาอพาร์ตเมนต์

2) พัฒนาเว็บแอปพลิเคชันด้วยภาษา PHP



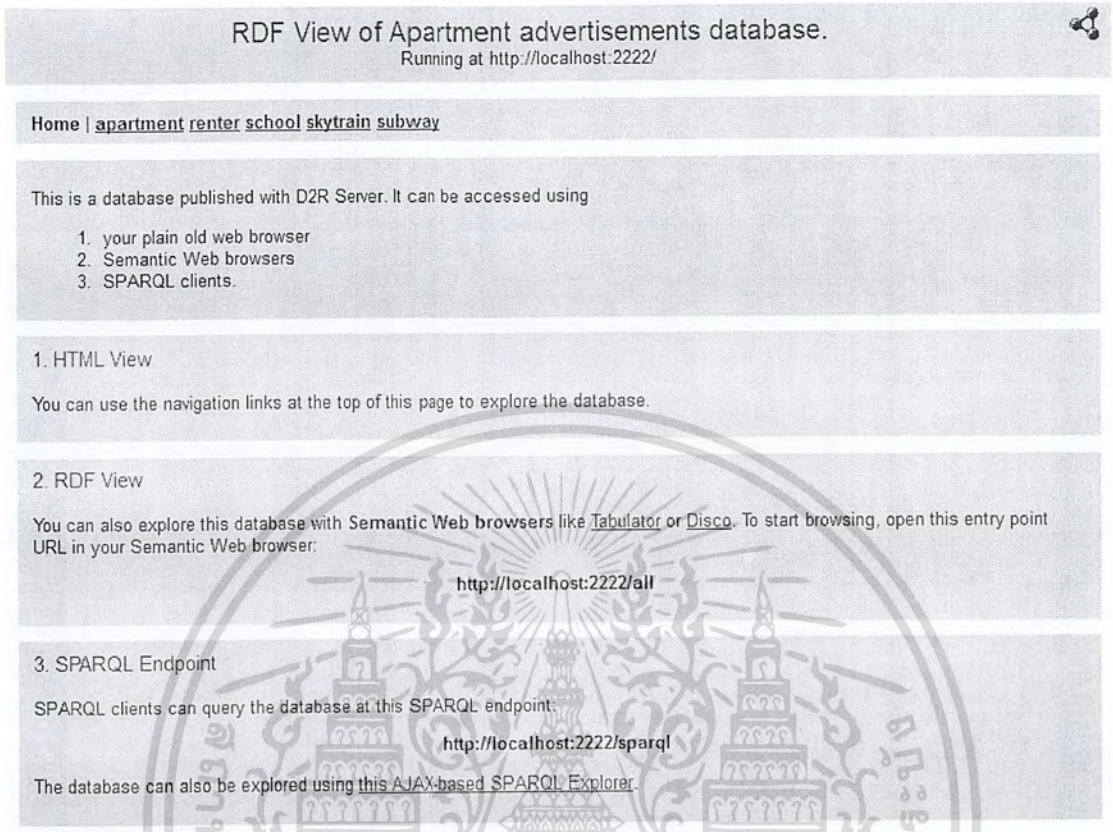
รูป 5.14 หน้าหลักของเว็บไซต์ที่พัฒนาขึ้นมา

3) แปลงเค้าร่างฐานข้อมูลเชิงสัมพันธ์ที่แสดงในรูป 5.13 ให้เป็น RDF Schema เก็บลงในไฟล์ที่แสดงการแปลง ได้ผลลัพธ์ดังรูป 5.15



รูป 5.15 RDF Schema ที่ได้มาจากการแปลงเค้าร่างฐานข้อมูลเชิงสัมพันธ์

4) นำเอาไฟล์ที่เก็บการแปลงไปใช้ตั้งเซิร์ฟเวอร์ด้วยเครื่องมือชื่อ d2r-server

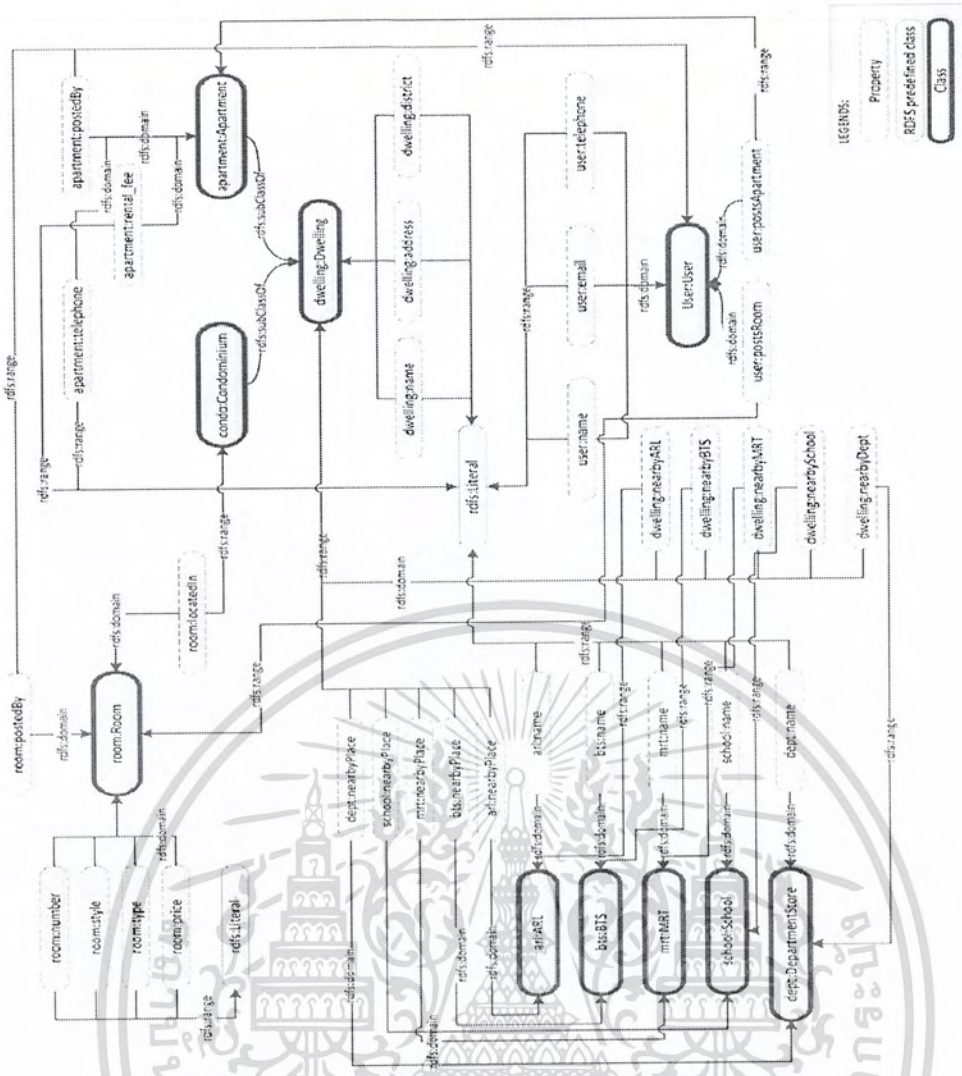


รูป 5.16 หน้าเว็บที่ได้จากการตั้งเซิร์ฟเวอร์

5.3.4 สร้างเว็บแอปพลิเคชันกลางที่สืบค้นข้อมูลจาก 3 เว็บแอปพลิเคชันข้างต้น

มีขั้นตอนดังนี้

1) สร้าง RDF Schema สำหรับเก็บข้อมูลโฆษณาโดยสังเขปจาก 3 เว็บแอปพลิเคชันข้างต้น



รูป 5.17 RDF Schema สำหรับเว็บเอปพลิเคชันกลาง

2) พัฒนาเว็บเอปพลิเคชันกลาง

ที่อยู่ ในละครเรื่องเนื้อคู่อยากรู้ว่าใคร เขตดินแดง

ข้อมูลห้องโดยสังเขป	ราคา	ที่มาโฆษณา
หมายเลขห้อง 1480 **แบบห้อง** 2 ห้องนอน **ประเภทโฆษณา** sale **ติดต่อ** S-Diox	5000 บาท	รายละเอียดโฆษณา
หมายเลขห้อง 1313 **แบบห้อง** 3 ห้องนอน **ประเภทโฆษณา** sale **ติดต่อ** S-Diox	1000000 บาท	รายละเอียดโฆษณา

รูป 5.18 หน้าหลักของเว็บแอปพลิเคชันกลาง

บทที่ 6
บทวิจารณ์และสรุป

6.1 บทสรุป

6.1.1 บทสรุปโดยภาพรวม

ในมุมมองของฐานข้อมูลแล้ว จะกล่าวได้ว่าเอกสาร XML นั่นก็คือฐานข้อมูล XML นั่นเอง ซึ่งสามารถ query ข้อมูลในเอกสาร XML ได้โดยใช้เทคโนโลยี XPath ซึ่งเป็นภาษาที่ใช้ค้นหาจุดเริ่มต้นของข้อมูลภายในเอกสาร XML ต่อจากนั้นจึงใช้ XQuery ที่มีไวยากรณ์ FLWOR เข้ามช่วยในการค้นหาข้อมูลที่ต้องการภายในเอกสาร XML อีกที่หนึ่ง

โดยปกติแล้วเอกสาร XML ไม่จำเป็นต้องใช้การกำหนดโครงสร้างเอกสารแบบ DTD หรือ XML schema ก็สามารถใช้งานได้ ถ้าเอกสาร XML นั้นมีคุณสมบัติ well-formed อยู่แล้ว แต่ถ้ามีการกำหนดโครงสร้างเอกสารแบบ DTD หรือ XML schema เพิ่มเติม เพื่อเพิ่มคุณสมบัติ valid ให้กับเอกสาร XML ทำให้รูปแบบโครงสร้างของเอกสาร XML ของเอกสาร XML หลายๆเอกสารที่มีข้อมูลที่เกี่ยวข้องกัน อยู่ในรูปแบบเดียวกัน นั่นคือเป็นการกำหนดว่าเอกสาร XML มีแท็กอะไรบ้าง มีแอตทริบิวต์อะไรบ้าง ค่าของแอตทริบิวต์มีค่าอะไรบ้าง เป็นต้น และยังเป็นสัญญาระหว่างองค์กรที่ใช้เอกสาร XML ในการแลกเปลี่ยนข้อมูลเพื่อให้รูปแบบเอกสาร XML ที่ใช้มีรูปแบบตรงกัน และเพื่อความถูกต้องในการนำเอกสาร XML ไปใช้งานด้วยเนื่องจากเอกสาร XML เป็นเอกสารที่ถูกสร้างโดยอนุณาตให้ผู้เขียนเอกสารกำหนดรูปแบบ และเนื้อหาเองได้ ต่อมาจึงได้มีเทคโนโลยี RDF (Resource Description Framework) เพื่อกำหนดรูปแบบในการอธิบายเนื้อหาข้อมูลเอกสารให้อยู่ในรูปแบบเป็นมาตรฐานเดียวกันโดยเทคโนโลยี RDF นี้ มีความสามารถในเรื่องต่างๆ 3 เรื่องที่เหนือกว่า XML คือ

1) เทคโนโลยี RDF มีซีแมนติก ในการอธิบายข้อมูลที่อยู่ภายในเอกสารซึ่ง XML นั้นไม่มี
2) สามารถใช้ RDFS (RDF Schema) มาเสริมเทคโนโลยี RDF ได้ ทำให้เอกสารสามารถนำเสนอความสัมพันธ์ระหว่างข้อมูลต่างๆ ที่มีความสัมพันธ์เกี่ยวข้องกันภายในเอกสาร RDF ได้
3) เทคโนโลยี RDF มี ออนโทโลยี ซึ่งอยู่ในรูปแบบของ OWL ในการบัญญัติศัพท์ต่างๆ ให้ผู้ใช้เอกสาร RDF ที่ใช้ ออนโทโลยี เดียวกันเข้าใจตรงกันได้ มีประโยชน์ในกรณีที่ถ้ามีเอกสาร RDF ที่เก็บข้อมูลประเภทเดียวกันอยู่หลายๆ เอกสาร ซึ่งเก็บอยู่ในที่ต่างๆ กัน จะสามารถจัดการเอกสาร RDF เหล่านั้นให้อยู่ในรูปแบบ ออนโทโลยี เดียวกัน ต่อจากนั้นก็จะสามารถใช้ข้อมูลในเอกสาร RDF ต่างๆ ได้เข้าใจตรงกัน

เราสามารถ query ข้อมูลในเอกสาร RDF ได้โดยใช้ภาษา SPARQL ซึ่งทำได้ง่ายกว่าการ query เอกสาร XML ด้วยภาษา XPath ร่วมกับ XQuery เนื่องจากความสามารถที่เหนือกว่าของ RDF ดังกล่าว 3 ข้อนั่นเอง

6.1.2 เทคโนโลยี XML ในมุมมองของฐานข้อมูล

6.1.2.1 โครงสร้างข้อมูล (data structure)

XML มีโครงสร้างข้อมูลเป็นลำดับชั้นต้นไม้ (Hierarchy) และมีความสัมพันธ์เป็นแบบพ่อลูก คือ พ่อ(parent) 1 คนมีลูก(child) ได้หลายคน แต่ลูกมีพ่อได้คนเดียว (ความสัมพันธ์แบบ 1 to Many) หรือแบบพ่อคนเดียวมีลูก 1 คน(ความสัมพันธ์แบบ 1 to 1) จากความสัมพันธ์แบบ 1 to Many ของ XML ทำให้ XML สามารถรองรับการออกแบบข้อมูลที่มีลักษณะเป็น Repeating Group (ข้อมูลชนิดเดียวกันแต่มีหลายค่า)

ถึงแม้ XML จะมีโครงสร้างเป็นลำดับชั้นต้นไม้ (Hierarchy) ซึ่งไม่มีความสัมพันธ์ของข้อมูลเป็นแบบลูกมีพ่อได้หลายคน แต่ XML ก็สามารถรองรับความสัมพันธ์แบบ Many to Many ได้เพราะมี DTD หรือ XML Schema มาช่วย เช่น ชนิดของ attribute ที่ชื่อ ID จะคล้าย Primary Key ใน relational database ส่วน IDREF และ IDREFS จะคล้ายกับ Foreign Key ใน relational database

6.1.2.2 ความถูกต้องของข้อมูล (data integrity)

Data integrity ของเอกสาร XML คือ คุณสมบัติ well-formed ของเอกสาร XML นั่นเอง และ element ลูกสามารถมี element พ่อได้เพียง element เดียว และ element หนึ่งๆ จะมี Attributes หรือไม่มีก็ได้ ถ้ามีสามารถมีได้มากกว่าหนึ่ง Attributes

6.1.2.3 Data manipulation language

ภาษาที่ใช้ในการสืบค้นเอกสาร XML คือภาษา XQuery ซึ่งมีความสามารถดังต่อไปนี้

1) เลือกแสดงผล Elements และ Attributes ทั้งหมดหรือบางส่วนของ XML ได้
2) สามารถเลือกแสดงผล Elements และ Attributes ตามเงื่อนไขที่กำหนดได้
3) มี Function ต่างๆ ให้เลือกใช้มากมาย เช่น AVG, SUM, MIN, MAX, COUNT
4) สามารถคำนวณตัวเลขได้ เช่น +, -, *, div
5) สามารถกรองข้อมูลเป็นกลุ่มๆ แม้จะไม่มีคำสั่ง Group By
6) สามารถนำเอกสารสองเอกสารมา Join กันได้
7) สามารถทำ Subquery ได้
8) สามารถใช้เอกสารหนึ่งๆ ได้มากกว่า 1 copy
9) สามารถทำ correlated Subqueries (การอ้างตารางนอก Subqueries) ได้
10) สามารถตรวจสอบความมีอยู่จริงของ Subqueries โดยใช้ฟังก์ชัน exists

6.1.3 เทคโนโลยี RDF ในมุมมองของฐานข้อมูล

6.1.3.1 โครงสร้างข้อมูล (data structure)

RDF จะมีโครงสร้างข้อมูลเป็น Graph ส่วนประกอบของ RDF Graph มี 3 ส่วน คือ

1) Subject คือหัวข้อหรือ Resource ที่ต้องการจะอธิบาย โดย Resource ที่ต้องการอธิบายต้องเป็น URI หรือ Blank node เท่านั้น

2) Predicate หรือ Property คือ คุณลักษณะของ Resource

3) Object คือค่าของ Property โดยจะเป็น Literal, URI หรือ Blank node ก็ได้

6.1.3.2 ความถูกต้องของข้อมูล (data integrity)

1) Resource ต้องเป็น URI เท่านั้น

2) Subject สามารถเป็น Resource หรือ Blank node ก็ได้

3) Object เป็นได้สามาอย่างคือ Literal, Resource, และ Blank node

4) Literal จะไม่มี Predicate หรือ property

5) Predicate หรือ property จะมีทิศทางจาก Subject ไป Object เสมอ

6) จะต้องมี <rdf:RDF> เป็น Root element เสมอ

7) Graphหนึ่งหนึ่ง ซึ่งประกอบไปด้วย Subject, Property, Object ต้องอยู่ใน <rdf:Description>

8) Subject ต้องเป็นค่าของ Attribute ชื่อ rdf:about

9) Property ต้องเป็น element ลูก ของ element ที่ชื่อ <rdf:Description>

10) Object ที่เป็น Literal เทียบได้กับ PCDATA ที่อยู่ใน element ของ Property นั้นๆ ในเอกสาร XML

11) Object ที่เป็น Resource คือค่าของ Attribute ชื่อ rdf:resource ที่อยู่ใน element ของ property นั้นๆ

12) Blank node จะใช้เมื่อใน element ของ property มี element ลูกอีก เช่น <rdf:Alt>หรือ element property มี Attribute เป็นrdf:parseType=”Collection”

6.1.3.3 Data manipulation language

ภาษาที่ใช้ในการสืบค้นเอกสาร RDF คือภาษา SPARQL ซึ่งมีความสามารถดังต่อไปนี้

1) สามารถเลือกแสดงผลโดยใช้หลักของRDFกราฟ คือ Subject, Predicate, Object

2) สามารถเลือกแสดงผลตามเงื่อนไขที่ต้องการได้ เพราะมี FILTER() ในการกรอง

3) สามารถคำนวณตัวเลขได้ (+, -, *, /) หรือเปรียบเทียบ (>, <, =, !=) หรือ Logical (||, &&) ได้ตามชนิดข้อมูลที่ประกาศใน XML Schema

4) สามารถกรองข้อมูลเป็นกลุ่มๆ โดยใช้หลักการของ Empty Group Pattern และ

Projection ช่วยได้

5) สามารถนำเอาเอกสารสองเอกสารมา join กันได้ โดยการใช้ Named, Empty Group Pattern, Projection

6) สามารถใช้เอกสารหนึ่งๆ ได้มากกว่า 1 copy โดยใช้ Empty Group Pattern

7) ไม่สามารถทำ subqueries ได้

8) ไม่สามารถทำ Correlated Subqueries (การอ้างตารางนอก Subqueries) ได้ เพราะไม่มี subqueries

9) สามารถ bound ตรวจสอบได้ว่ามีตามเงื่อนไขหรือไม่ จะคล้ายกับ exists() ใน XQuery

10) มี ORDER BY ที่ใช้ในการเรียงผลลัพธ์

11) มี Constructor Functions ต่างๆให้เรียกใช้

12) offset และ limit ในการกำหนดจำนวนการแสดงผล

13) มีรูปแบบการแสดงหลายอย่าง SELECT, CONSTRUCT, ASK, DESCRIBE

14) มีฟังก์ชันในการตรวจสอบว่าเป็นค่าอะไรบ้าง เช่น isIRI, isBlank, iaLiteral,

6.1.4 ความสามารถของเอกสาร RDF ที่เหนือกว่าเอกสาร XML

เอกสาร RDF มีความสามารถที่เหนือกว่าเอกสาร XML อยู่สามประเด็นดังนี้

1) เทคโนโลยี RDF มี semantic ในการอธิบายข้อมูลที่อยู่ภายในเอกสาร ซึ่ง XML ไม่มี

2) สามารถใช้ RDFS (RDF Schema) มาเสริมเทคโนโลยี RDF ได้ ทำให้เอกสารสามารถนำเสนอความสัมพันธ์ระหว่างข้อมูลต่างๆ ที่มีความสัมพันธ์เกี่ยวข้องกันภายในเอกสาร RDF ได้

3) เทคโนโลยี RDF มี ontology ซึ่งอยู่ในรูปแบบของ OWL ในการบัญญัติศัพท์ต่างๆ ให้ผู้ใช้เอกสาร RDF ที่ใช้ ontology เดียวกันเข้าใจตรงกันได้ มีประโยชน์ในกรณีที่ ถ้ามีเอกสาร RDF ที่เก็บข้อมูลประเภทเดียวกันอยู่หลายๆ เอกสาร ซึ่งเก็บอยู่ในที่ต่างๆ กัน จะสามารถจัดการเอกสาร RDF เหล่านั้นให้อยู่ในรูปแบบ ontology เดียวกัน ต่อจากนั้นก็จะสามารถใช้ข้อมูลในเอกสาร RDF ต่างๆ ได้เข้าใจตรงกัน

6.1.5 ความสามารถของภาษา SPARQL และ XQuery

1) SPARQL มี productivity มากกว่า XQuery ในการ Query เอกสาร RDF เพราะ SPARQL มีความง่ายในการใช้งานมากกว่า XQuery เนื่องจากสามารถ query เอกสาร RDF ได้โดยใช้ syntax สั้นๆ เพียงไม่กี่บรรทัดก็จะได้ผลลัพธ์เหมือนกับการใช้ Xquery ในการ query เอกสาร XML ซึ่งมีความยาว และยุ่งมากกว่า แต่ SPARQL มีข้อจำกัดที่ไม่สามารถทำ subquery ได้ จึงต้องใช้ Xquery มาช่วยโดยการนำ Xquery มา query subquery เอกสาร RDF แทนการใช้ SPARQL เนื่องจากเอกสาร RDF ก็เขียนโดยใช้

syntax XML เหมือนกัน

2) การ query โดยใช้ SPARQL นั้น คนที่ทำการ query แค่ต้องรู้ว่าต้องการ query อะไร เท่านั้นก็สามารถ query ข้อมูลได้เลย ซึ่งต่างจาก XQuery ที่จำเป็นต้องรู้โครงสร้างของข้อมูลก่อน

3) SPARQL สามารถ query ข้อมูลที่มี ontology ได้ ซึ่งการ query โดยใช้ SPARQL เป็นภาษาธรรมชาติ นั่นคือใกล้เคียงกับภาษามนุษย์มากกว่าการใช้ XQuery

4) SPARQL สามารถถามถึงคุณสมบัติที่เกี่ยวกับ RDF ได้ เช่น isIRI, isBlank, isLiteral, lang เป็นต้น ในขณะที่ XQuery ไม่มีคุณสมบัติเหล่านี้

5) SPARQL คำนวณเปรียบเทียบชนิดข้อมูลที่ไม่ใช่ primitive type ได้ เช่น ข้อมูลชนิดวันที่ ในขณะที่ XQuery ทำไม่ได้

6) การ query โดยใช้ SPARQL นั้นมีความง่ายกว่าการใช้ XQuery เนื่องจากผู้ใช้ไม่จำเป็นต้องมีประสบการณ์ในการ query มาก่อน และไม่เป็นต้องรู้ไวยากรณ์ระดับยากก็สามารถ query ข้อมูลได้

7) SPARQL เหมาะกับการออกรายงาน เพราะมีรูปแบบการแสดงผลหลายอย่าง เช่น SELECT, CONSTRUCT, ASK, DESCRIBE ส่วน XQuery มีรูปแบบการแสดงผลเพียงอย่างเดียวคือ return

6.1.6 เปรียบเทียบความสามารถของภาษา SQL และภาษา XQuery

1) ภาษา XQuery มีความสามารถอย่างน้อยเท่ากับภาษา SQL เพราะ XQuery สามารถ query ข้อมูลได้อย่างที่ SQL ทำได้

2) ภาษา XQuery มีคุณสมบัติเป็น Relational Complete เพราะการที่ภาษาใดจะมีคุณสมบัติเป็น Relational Complete ต้องมีคุณสมบัติอย่างน้อยเทียบเท่า Relational Algebra หรือ Relational Calculus เนื่องจากภาษา SQL มีคุณสมบัติเป็น Relational Complete ดังนั้นเมื่อภาษา XQuery มีความสามารถเทียบเท่า SQL ดังกล่าว จึงทำให้ภาษา XQuery มีความสามารถอย่างน้อยเทียบเท่า Relational Algebra หรือ Relational Calculus ด้วย ทำให้ภาษา XQuery มีคุณสมบัติเป็น Relational Complete ตามไปด้วย

3) ภาษา XQuery สามารถ query ข้อมูลที่มีโครงสร้างเป็นลำดับชั้น (hierarchy) ได้ ซึ่งภาษา SQL ไม่สามารถทำได้

6.2 วิจารณ์สิ่งที่ได้จากโครงงาน

1) รู้จักภาษา XML และเทคโนโลยีที่เกี่ยวข้องกับภาษา XML เช่น XSL, DTD, XML Schema, DOM, XPath และXquery เป็นต้น

2) รู้จักเทคโนโลยี RDF, RDFS, OWL และภาษาสืบค้น SPARQL

3) สามารถสืบค้นข้อมูลในเอกสาร XML ด้วยภาษา XQuery และ Xpath ได้

4) สามารถสืบค้นข้อมูลในเอกสาร RDF ด้วยภาษา SPARQL ได้โดยใช้ไลบรารี RDFlib

5) รู้จักคุณประโยชน์ของเทคโนโลยีของภาษา XML ในปัจจุบัน

6) รู้จักการนำข้อมูลที่เก็บในรูปเชิงสัมพันธ์มาเผยแพร่ ในรูปแบบกราฟ RDF ด้วยเครื่องมือ D2RQ Server

7) รู้จักการใช้งาน RDF ในฐานะฐานข้อมูลโดยอาศัยทริปเปิลสโตร์ และ API สำหรับทำงานกับข้อมุลกราฟ RDF

6.3 ปัญหาอุปสรรค และแนวทางการแก้ไข

1) XML มีเทคโนโลยีที่เกี่ยวข้องเป็นจำนวนมาก จึงจำเป็นต้องใช้เวลาในการศึกษา สามารถแก้ไขได้โดยให้เวลากับการศึกษาพอสมควร

2) โปรแกรมที่ใช้ในการประมวลผลคำสั่งสืบค้นภาษา XQuery ไม่มีดีบักเกอร์แนบติดมาให้ใช้ ภาษา Xquery ยังอยู่ในขั้นตอนการพัฒนาอาจมีการเปลี่ยนแปลงได้ในอนาคต

3) เอกสารที่อธิบายเทคโนโลยีต่าง ๆ เป็นภาษาอังกฤษ และยากต่อการทำความเข้าใจ สามารถแก้ไขโดยการให้เวลากับการศึกษาเอกสารเหล่านี้เป็นอย่างมาก และแลกเปลี่ยนความรู้กันในกลุ่มเพื่อป้องกันการเข้าใจผิด

4) วรรณกรรมวิชาการบางฉบับไม่สมบูรณ์ทำให้ใช้เวลาศึกษามากกว่าปกติ

5) อุปสรรคปัญหาเกี่ยวกับการใช้ภาษา การทำงานแต่ละ library ไม่เหมือนกัน แต่สามารถแก้ได้โดยการ เข้ารหัสข้อความแบบ UTF-8

6.4 แนวทางการพัฒนาต่อ

1) ศึกษา และทดลองการเขียนกฎเกณฑ์กำกับออนโทโลยีด้วยภาษา SWRL (Semantic Web Rule Language)

2) ศึกษา และทดลองพัฒนาเว็บสื่อความหมาย

3) ศึกษากลวิธีการรวมออนโทโลยี

บรรณานุกรม

กนกวรรณ ตันติพาณิชย์กูล และ กิตติศักดิ์ จิรภิวงศ์. 2550. “ฐานข้อมูลเอ็กซ์เอ็มแอล.” วิทยานิพนธ์วิศวกรรมศาสตร์บัณฑิต ภาควิชาวิศวกรรมคอมพิวเตอร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.

Gregoris, A. and Frank van H. 2008. **A Semantic Web Primer.** London : The MIT Press.

Toby, S, Colin, E. and Jamie, T. 2009. **Programming the Semantic Web.** Cambridge : O'Reilly.

ชวลิต จิรทีปติสุนทร, ผู้แปล. 2543. XML Step by Step. กรุงเทพฯ สำนักพิมพ์สามย่าน.COM.

สราวุธ อ้อยศรีสกุล. 2551. เริ่มคิด-เริ่มสร้าง-เริ่มใช้ XML. ปรับปรุงครั้งที่ 2. กรุงเทพฯ Witty Group.

ศุภชัย สมพานิช. 2544. เข้าใจและใช้งานภาษา XML ฉบับโปรแกรมเมอร์. กรุงเทพฯ Inforpress.

Cristina F. 2004. **SPARQL Query Language for RDF.** [Online]. Available : http://www.wsmo.org/wsml/papers/presentations/sparql.ppt

Joshua T. 2005. **RDF in Depth.** [Online]. Available : http://www.rdfabout.com/intro/?section=contents

Daniel K. 2005. **RDFlib 2.4.0 Documentation.** [Online]. Available : http://rdflib.net/rdflib-2.4.0/html/index.html

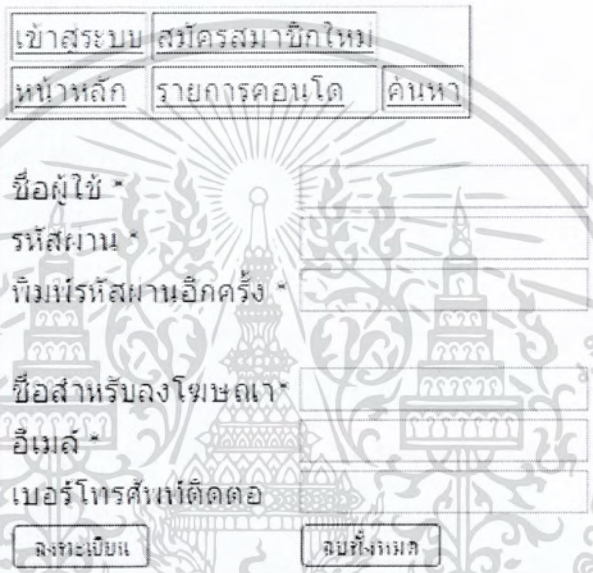
ภาคผนวก

คู่มือการใช้งาน

ก.1 วิธีการใช้งานเว็บไซต์คอนโดมีเนียมที่ขับเคลื่อนด้วยฐานข้อมูล RDF

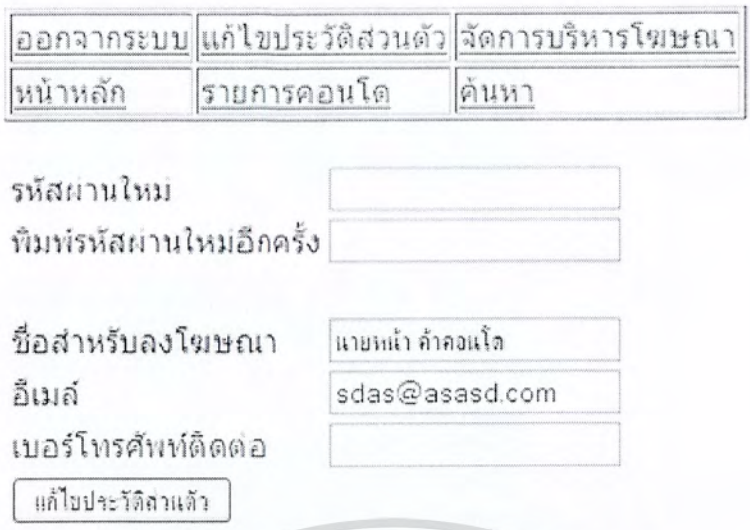
มีลำดับการการใช้งานดังนี้

1) สมัครเป็นสมาชิกเว็บไซต์โดยการคลิกที่ลิงค์ “สมัครสมาชิกใหม่”



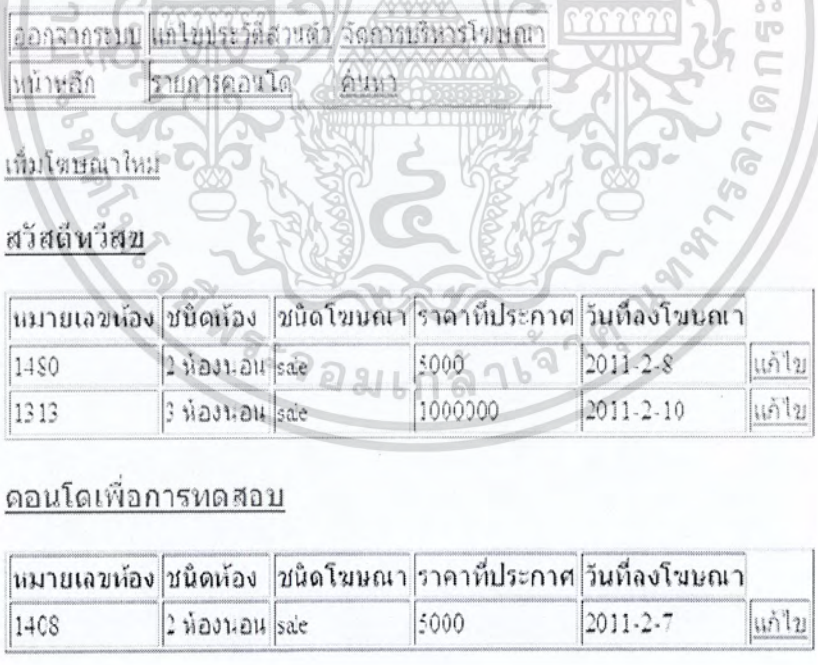
รูป ก.1 หน้าเว็บสมัครสมาชิกใหม่

2) กรอกข้อมูลที่จำเป็นแล้วกดลงทะเบียนแล้วทำการเข้าสู่ระบบ โดยการกรอกชื่อผู้ใช้ และรหัสผ่าน เว็บไซต์จะพาไปหน้าหลัก สามารถแก้ไขประวัติส่วนตัวได้โดยการคลิก “แก้ไขประวัติส่วนตัว” ที่ส่วนเมนู



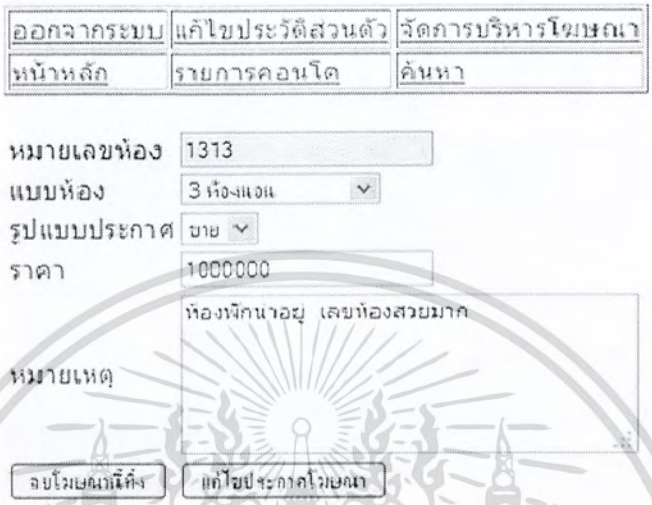
รูป ก.2 หน้าเว็บสำหรับแก้ไขประวัติส่วนตัว

3) หากต้องการเพิ่ม แก้ไข หรือลบโฆษณาประกาศที่สมาชิกเว็บได้เคยประกาศเอาไว้ ให้คลิก "จัดการบริหารโฆษณา" จะพบหน้าเว็บในรูปก.3 ที่แสดงรายการโฆษณาของสมาชิกนั้น ๆ



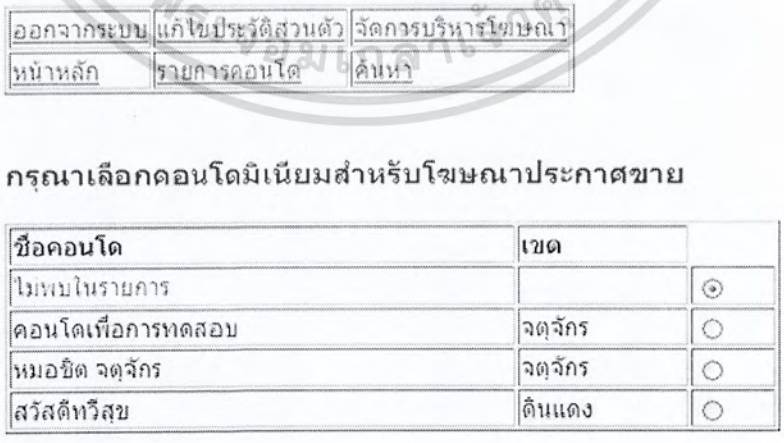
รูป ก.3 รายการโฆษณาคอนโดมีเนียมของสมาชิก

4) หากต้องการแก้ไขรายละเอียดห้องในคอนโดหรือลบโฆษณาทิ้ง ให้กดลิงค์ “แก้ไข” ที่ทางขวาของแถวในตารางที่แสดงรายการโฆษณาในคอนโดแต่ละแห่งของสมาชิกที่เข้ามาในระบบ แสดงดังในรูป ก.4



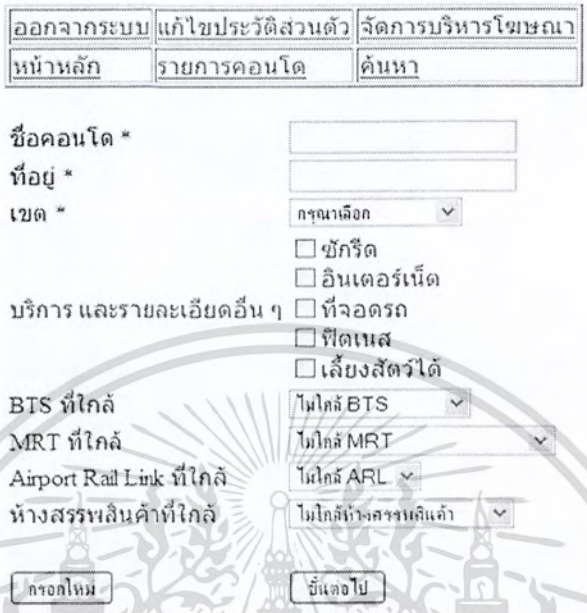
รูป ก.4 หน้าเว็บสำหรับแก้ไขโฆษณา

5) หากต้องการประกาศโฆษณาใหม่ลงในเว็บไซต์ให้คลิกลิงค์ “เพิ่มโฆษณาใหม่” ในรูป ก.3 จะปรากฏหน้าเว็บไซต์ดังรูป ก.5 ให้ผู้ใช้เลือกคอนโดมีเนียมที่ต้องการประกาศขายห้องพัก หากไม่มีคอนโดมีเนียมที่ต้องการ ให้เลือก “ไม่พบในรายการ” แล้วกดปุ่ม “ขั้นตอนต่อไป”



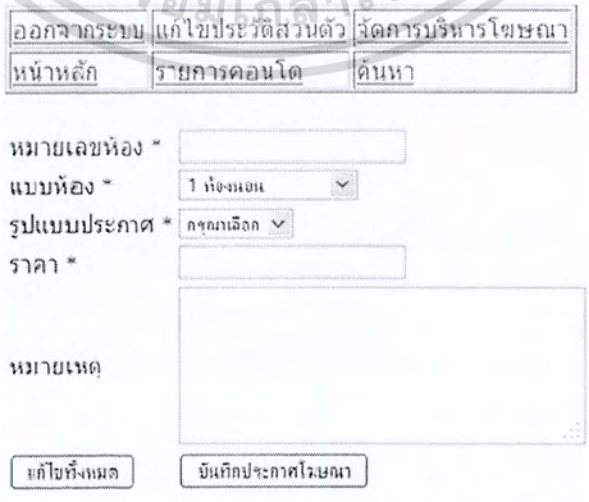
รูป ก.5 หน้าเว็บให้ผู้ใช้เลือกคอนโดมีเนียมที่ต้องการลงโฆษณา

6) สมมติว่าคอนโดมีเนียมที่ต้องการประกาศโฆษณาห้องพักไม่มีอยู่ในฐานข้อมูล จะปรากฏหน้าเว็บดังรูป ก.6 เพื่อกรอกรายละเอียดต่าง ๆ เกี่ยวกับคอนโดมีเนียม



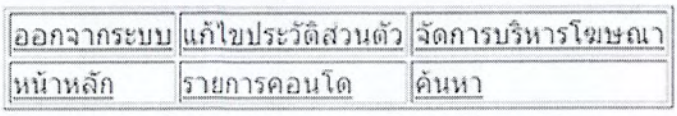
รูป ก.6 หน้าเว็บกรอกรายละเอียดคอนโดมีเนียมแห่งใหม่ที่สมาชิกต้องการลงประกาศโฆษณา

7) หลังจากกรอกรายละเอียดคอนโดมีเนียมเสร็จกดปุ่ม “ขั้นต่อไป” แล้วจะพาไปยังหน้าเว็บไซต์สำหรับกรอกข้อมูลรายละเอียดห้องภายในคอนโดมีเนียมที่เพิ่งเลือก หรือเพิ่งสร้างขึ้นมาใหม่ในขั้นตอนก่อนหน้านี้



รูป ก.7 หน้าเว็บไซต์สำหรับกรอกรายละเอียดโฆษณาห้องพัก

8) เมื่อกรอกเสร็จแล้วให้กดปุ่ม “บันทึกประกาศโฆษณา” แล้วโฆษณาใหม่ที่สมาชิกเว็บไซต์เพิ่งประกาศไป จะไปอยู่ที่หน้าหลักของเว็บไซต์ เนื่องจากหน้าหลักจะแสดง 10 โฆษณาล่าสุดที่ได้ประกาศเอาไว้ ดังในรูป ก.8



ออกจากระบบ	แก้ไขประวัติส่วนตัว	จัดการบริหารโฆษณา
หน้าหลัก	รายการคอนโด	ค้นหา

10 โฆษณาคอนโดมีเนียมล่าสุดภายในเว็บไซต์

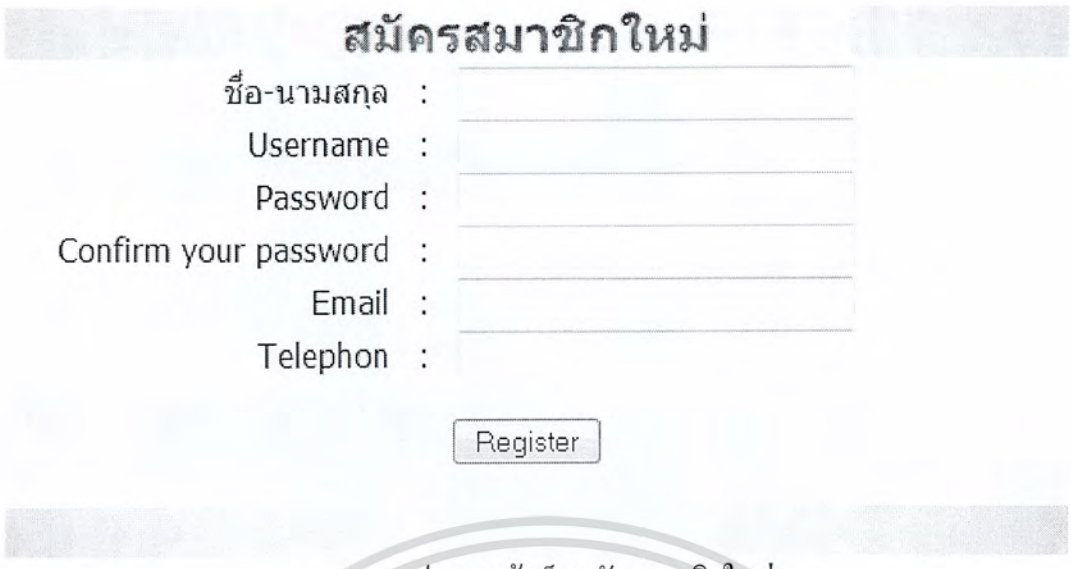
คอนโดมีเนียม	โฆษณาโดยสังเขป	รายละเอียดโฆษณา
สวัสดีทวีสุข	แบบห้อง 2 ห้องนอน ชนิดโฆษณา sale ราคา 5000 บาท	รายละเอียดโฆษณา (วันที่ลงโฆษณา 2011-2-8)
คอนโดเพื่อการทดสอบ	แบบห้อง 2 ห้องนอน ชนิดโฆษณา sale ราคา 5000	รายละเอียดโฆษณา (วันที่ลงโฆษณา 2011-2-7)

รูป ก.8 หน้าเว็บหลัก

ก.2 วิธีการใช้งานเว็บไซต์คอนโดมีเนียมที่ขับเคลื่อนด้วยฐานข้อมูล Relational Database

มีวิธีการใช้งานดังนี้

1) สมัครเป็นสมาชิกเว็บไซต์โดยการคลิกที่ลิงค์ “สมัครสมาชิกใหม่”

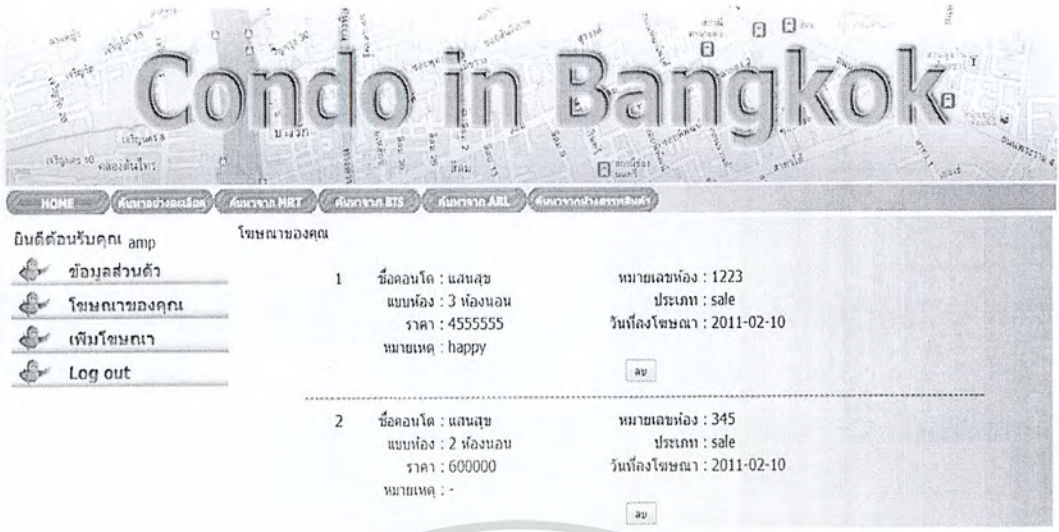


รูป ก.9 หน้าเว็บสมัครสมาชิกใหม่

2) กรอกข้อมูลแล้วกดลงทะเบียนแล้วทำการเข้าสู่ระบบโดยการกรอกชื่อผู้ใช้ และรหัสผ่าน เว็บไซต์จะพาไปหน้าสำหรับสมาชิก ซึ่งจะมีเมนูย่อยถ้าเมนูคลิก "ข้อมูลส่วนตัว" จะสามารถแก้ไขประวัติส่วนตัวได้

รูป ก.10 หน้าเว็บสำหรับแก้ไขประวัติส่วนตัว

3) หากต้องการดูหรือลบโฆษณาประกาศที่สมาชิกเว็บได้เคยประกาศเอาไว้ ให้คลิกเมนู "โฆษณาของคุณ"

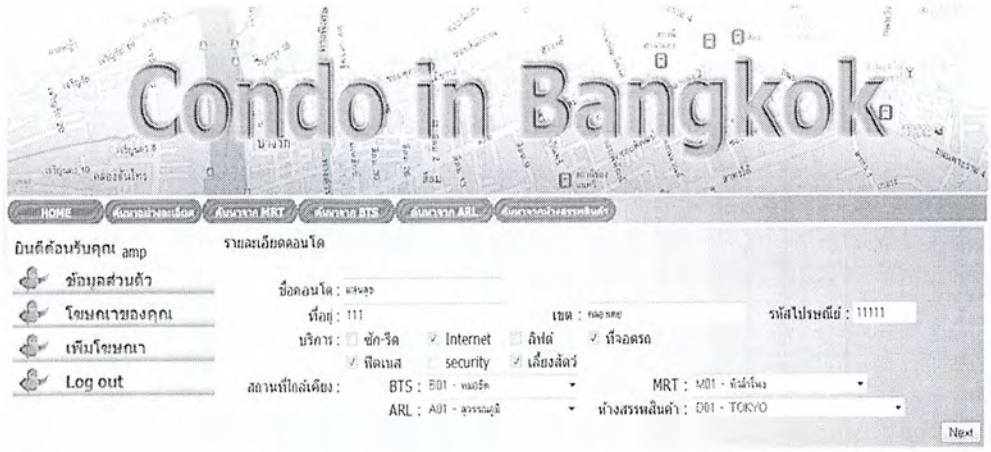


รูป ก.11 รายการโฆษณาคอนโดมีเนียมของสมาชิก

4) หากต้องการประกาศโฆษณาใหม่ลงในเว็บไซต์ให้คลิกเมนู “เพิ่มโฆษณา” เพื่อให้ผู้ใช้เลือกคอนโดมีเนียมที่ต้องการหากไม่มีคอนโดมีเนียมที่ต้องการ ให้เลือก “เพิ่มรายชื่อคอนโด” แล้วกดปุ่ม “ขั้นตอนต่อไป”

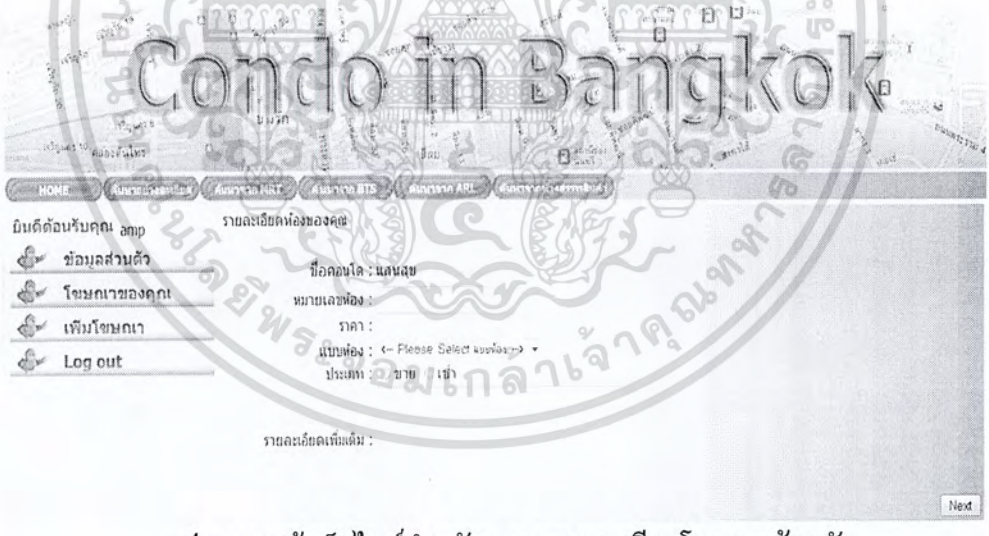
รูป ก.12 หน้าเว็บให้ผู้ใช้เลือกคอนโดมีเนียมที่ต้องการลงโฆษณา

5) จากนั้น กรอกรายละเอียดต่าง ๆ เกี่ยวกับคอนโดมีเนียม ถ้าเป็นคอนโดที่มีข้อมูลอยู่แล้วก็จะแสดงข้อมูลเดิมออกมาซึ่งสามารถปรับเปลี่ยนแก้ไขข้อมูลได้



รูป ก.13 หน้าเว็บรายละเอียดคอนโดมีเนียมแห่งหนึ่งที่สมาชิกต้องการลงประกาศโฆษณา

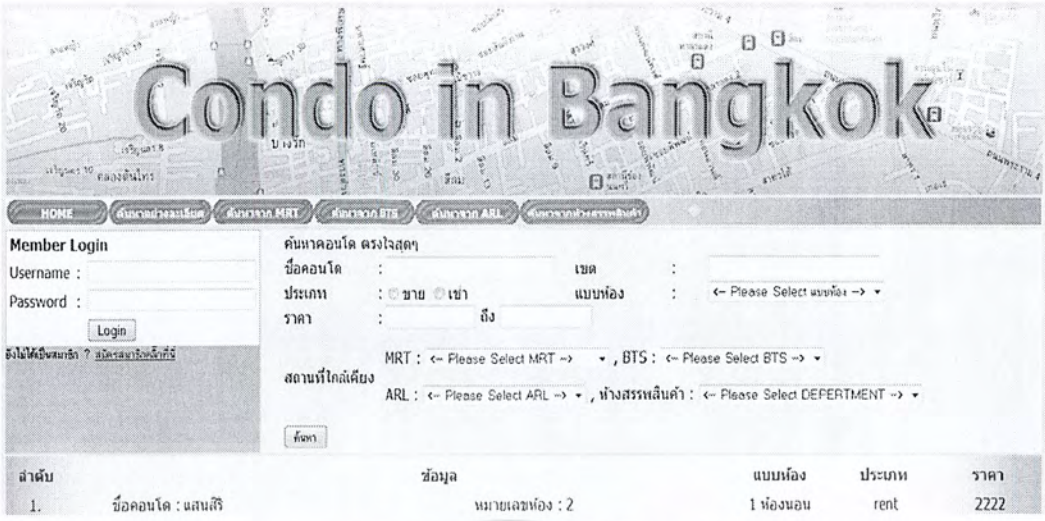
6) หลังจากกรอกรายละเอียดคอนโดมีเนียมเสร็จกดปุ่ม “ขั้นต่อไป” แล้วจะพาไปยังหน้าเว็บไซต์สำหรับกรอกข้อมูลรายละเอียดห้องภายในคอนโดมีเนียมที่เพิ่งเลือก หรือเพิ่งสร้างขึ้นมาใหม่ในขั้นตอนก่อนหน้านี้



รูป ก.14 หน้าเว็บไซต์สำหรับกรอกรายละเอียดโฆษณาห้องพัก

7) หลังจากกรอกรายละเอียดห้องเรียบร้อยแล้วกดปุ่ม “next” เพื่อดูประกาศโฆษณาของคุณ

8) ถ้าต้องการค้นหาคอนโดสามารถคลิกที่แถบเมนูด้านบน ซึ่งสามารถเลือกรูปแบบการสืบค้นได้ตามต้องการ

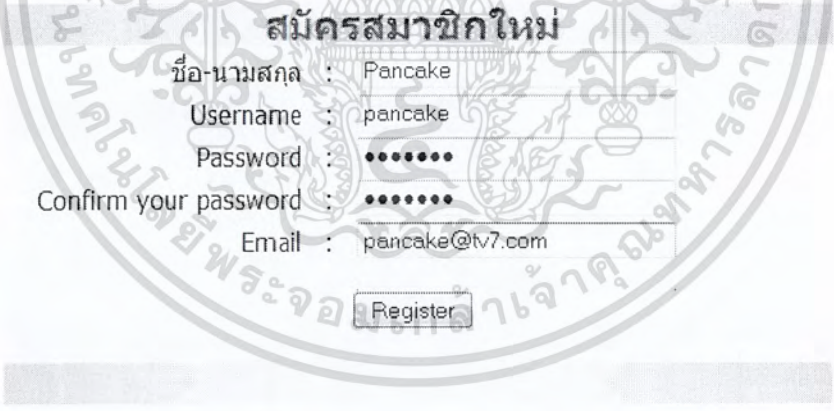


รูป ก.15 แถบเมนูที่ใช้สำหรับสืบค้นข้อมูล

ก.3 วิธีการใช้งานเว็บไซต์อพาร์ทเมนต์ที่ขับเคลื่อนด้วยฐานข้อมูล Relational Database

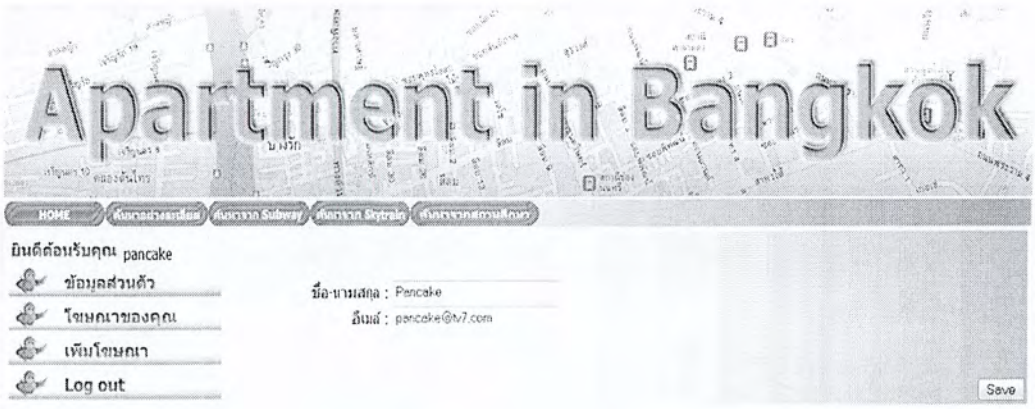
ประกอบด้วยวิธีดังนี้

1) สมัครเป็นสมาชิกเว็บไซต์โดยการคลิกที่ลิงค์ “สมัครสมาชิกใหม่”



รูป ก.16 หน้าเว็บสมัครสมาชิกใหม่

2) กรอกข้อมูลแล้วกดลงทะเบียนแล้วทำการเข้าสู่ระบบโดยการกรอกชื่อผู้ใช้ และรหัสผ่านเว็บไซต์จะพาไปหน้าสำหรับสมาชิก ซึ่งจะมีเมนูย่อยถ้าเมนูคลิก “ข้อมูลส่วนตัว” จะสามารถแก้ไขประวัติส่วนตัวได้

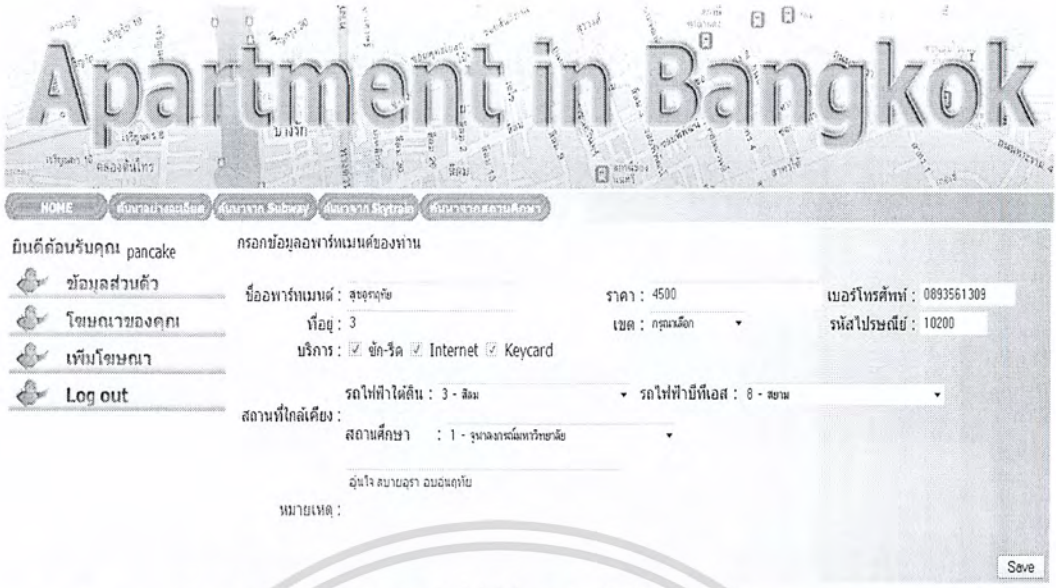


รูป ก.17 สมาชิกลงชื่อเข้าใช้งานระบบเพื่อประกาศโฆษณาอพาร์ทเมนต์

3) หากต้องการดูหรือลบโฆษณาประกาศที่สมาชิกเว็บได้เคยประกาศเอาไว้ ให้คลิกเมนู "โฆษณาของคุณ"

รูป ก.18 รายการโฆษณาอพาร์ทเมนต์ของสมาชิก

4) หากต้องการประกาศโฆษณาใหม่ลงในเว็บไซต์ให้คลิกเมนู "เพิ่มโฆษณา" เพื่อกรอกรายละเอียดต่าง ๆ เกี่ยวกับอพาร์ทเมนต์ ของสมาชิก



รูป ก.19 หน้าเว็บรายละเอียดอพาร์ทเมนต์แห่งหนึ่งที่สมาชิกต้องการลงประกาศโฆษณา

5) หลังจากกรอกรายละเอียดห้องเรียบร้อยแล้วกดปุ่ม “Save” เพื่อบันทึกประกาศโฆษณาของคุณ

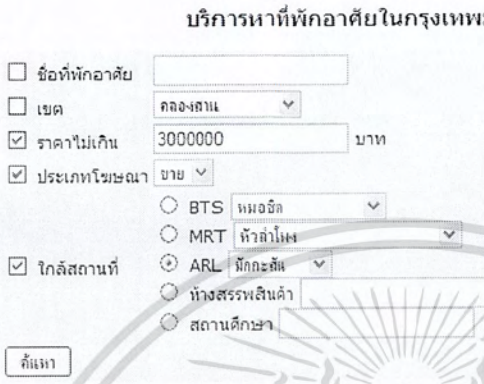
6) ถ้าต้องการค้นหาอพาร์ทเมนต์สามารถคลิกที่แถบเมนูด้านบน ซึ่งสามารถเลือกรูปแบบการสืบค้นได้ตามต้องการ

รูป ก.20 แถบเมนูที่ใช้สำหรับสืบค้นข้อมูล

ก.4 วิธีการใช้งานเว็บแอปพลิเคชันกลางที่ขับเคลื่อนด้วยฐานข้อมูล RDF

มีขั้นตอนวิธีใช้ดังนี้

1) กรอกข้อมูลคำค้นตามเงื่อนไขที่ต้องการ



รูป ก.21 หน้าหลักของเว็บแอปพลิเคชันกลาง

2) จากนั้นคลิกปุ่ม 'ค้นหา'

ที่อยู่ ใกล้สถานีแอร์พอร์ตลิงค์ เขตลาดกระบัง

ข้อมูลห้องโดยสังเขป	ราคา	ที่มาโฆษณา
หมายเลขห้อง 5555 แบบห้อง 2 ห้องนอน ประเภทโฆษณา sale	2800000 บาท	รายละเอียดโฆษณา

aaaaaaaaaa

ที่อยู่ aa เขตaa

ข้อมูลห้องโดยสังเขป	ราคา	ที่มาโฆษณา
หมายเลขห้อง 1313 แบบห้อง 2 ห้องนอน ประเภทโฆษณา sale	500000 บาท	รายละเอียดโฆษณา

รูป ก.22 ผลลัพธ์จากการสืบค้นข้อมูลของเว็บเอปพลิเคชันกลาง